Representation Similarity Reveals Implicit Layer Grouping in Neural Networks

Tian Gao, Amit Dhurandhar, Karthikeyan Natesan Ramamurthy, Dennis Wei
IBM Research
{tgao, adhuran, knatesa, dwei}@us.ibm.com

Abstract

Providing human-understandable insights into the inner workings of neural networks is an important step toward achieving more explainable and trustworthy AI. Analyzing representations across neural layers has become a widely used approach for this purpose in various applications. In this work, we take a step toward a more holistic understanding of neural layers by investigating the existence of distinct layer groupings within them. Specifically, we explore using representation similarity within neural networks to identify clusters of similar layers, revealing potential layer groupings. We achieve this by proposing, for the first time to our knowledge, the use of Gromov-Wasserstein distance, which overcomes challenges posed by varying distributions and dimensionalities across intermediate representations– issues that complicate direct layer-to-layer comparisons. On algebraic, language, and vision tasks, we observe the emergence of layer groups that correspond to functional abstractions within networks. These results reveal implicit layer structure pattern, and suggest that the network computations may exhibit abrupt shifts rather than smooth transitions. Through downstream applications of model compression and fine-tuning, we validate our measure and further show the proposed approach offers meaningful insights into the internal behavior of neural networks.

1 Introduction

Recent advances in neural networks, in particular large models, has prompted increased interest in understanding the underlying causes of new capabilities. Since neural models, including large language models (LLMs), are mostly black-box models, explainable AI aims to offer insights and improve human understanding of these neural models. A widely adopted strategy involves analyzing the inner representations across network layers, as these offer valuable information about how intermediate computations evolve. As a result, measures to quantify representation similarity of these layers have been widely applied in the literature to gain novel insights [22], including learning dynamics, impact of different network hyperparameters, knowledge distillation, and more.

Despite substantial progress, relatively little attention has been devoted to understanding the structure of the layers themselves. Neural networks, especially those with many layers[1] and parameters, often exhibit behaviors that are difficult to interpret holistically. Given such depth and opacity of modern networks, identifying coherent layer groupings can serve as a principled way to decompose models into more interpretable sub-units. Such structural insights may illuminate how specific portions of a network contribute to its overall functionality and enable a more modular understanding of computation.

[1]The term layer is used broadly to refer to any intermediate or final output produced by internal computations.The generality of the term allows it to encompass everything from input preprocessing to hidden states and final predictions.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Mechanistic Interpretability.

In this work, we ask a fundamental question: *can we detect the existence of distinct layer groups within a trained neural network?* We answer this in the affirmative by demonstrating that representation similarity across layers can reveal meaningful structural boundaries. Surprisingly, our results indicate that layer transitions are not always smooth; instead, networks may exhibit abrupt shifts in computation, manifesting as changes in representation.

Comparing representation across different layers is natural, as it is akin to comparing functions by evaluating their outputs on shared inputs. This leads to the task of identifying layer groups as regions of functional similarity, with significant shifts marking transitions between layer groups. As illustrated in Figure 1, representational-similar layers (darker colors) cluster together, while less similar layers (brighter colors) highlight potential boundaries between groups.

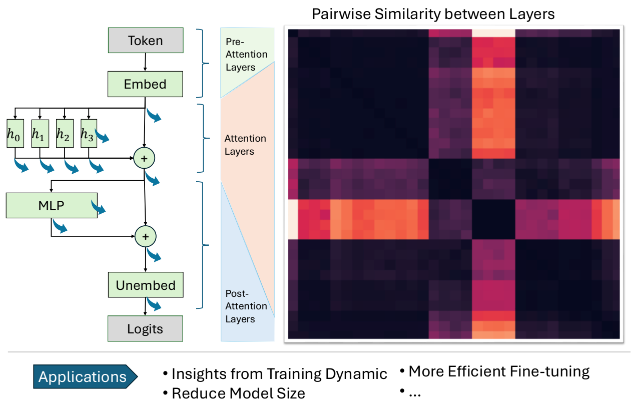


Figure 1: Overview of our approach, where we use representations from different neural network layers to identify functionally distinct layer groups (such as different darker-colored blocks) leveraging GW distance. The figure is an illustration based on a single transformer block, and the proposed technique can be applied to other types of network structures.

A central challenge in this endeavor is defining a robust similarity measure. Standard metrics (e.g., cosine, Euclidean) are inadequate due to the varying dimensionality and metric space of layer outputs, especially in architectures like CNNs and transformers. Additionally, representations should be invariant to transformations such as rotation, scaling, permutation, and reflection. While many specialized similarity measures have been proposed, it is not clear which measure would be better for measuring the representation similarity. To this end, we propose to measure representation similarity using Gromov-Wasserstein (GW) distance [52] between representations from different layers of the network. As elaborated further in Section 4.1, GW allows distance computation between distributions supported on two different metric spaces with different supports and potentially different dimensions, which is common across different layers in neural networks. GW is also invariant to permutation of the representation within a layer, a crucial property since neural networks are known to have permutation symmetries [15]. As such, GW can effectively identify genuinely distinct behaviors across (groups of) layers.

We validate our approach on algebraic, language, and vision tasks, showing that GW distance provides a systematic and architecture-agnostic way to analyze and identify layer structure. Additionally, our findings provides a holistic view on differences in representations of models trained with different strategies. We observe clear patterns in the form of block structures among different layers, suggesting there exist layer group that potentially compute different functions, particularly at the transition layers where major functional changes may occur. Moreover, our approach is applicable to various downstream tasks, such as tracking the emergence of layer groups during training process (§ 5.3) and identifying potentially redundant layers in model compression and fine-tuning (§ 5.2) Overall, our method can improve the efficiency of mechanistic interpretation by finding layer groups in any neural model, reducing the need for extensive human effort and contributing to a further understanding of neural network behaviors.

2 Background and Related Work

Neural network interpretability has always been a hot research topic, and has become a disparate area with many different applications in vision [37], and language [36, 18, 51], and we survey a few closely related directions within it.

Layer Grouping v.s. Other Types of Subnetwork Automated circuit discovery [5, 41] aims to find a computational graph that is much more sparse without sacrificing performance. Moreover, Various work have explored neuron semantics and possible disentanglement of semantics [2, 11, 20] and concepts [38]. Instead of studying circuits and neurons, we investigate differences among neural network layers as a whole, based on an existing line of work [34]. Related, block structures within neural network layers have been observed in previous studies [35].

Representation Similarity v.s. Other Methods of Studying Mechanisms There are many other approach to understand neural network inner mechanism, such as weight inspection and manipulation [36, 32]. Modifying or ablating the representation of a specific model components [21, 24], including attention knockout [50], and even direct modification of attention matrix [36, 14] are prevalent. Another popular approach involves inspection of representation properties [32], including logit patterns [53], residual stream [36], and periodicity [34]. Rather than just inspection of representation, many works have proposed to map the output to some target and is a popular technique for analyzing how neural activations correlate with high-level concepts [20, 19]. Here we instead focus on studying and compare representation similarity across layers.

Similarity Measure between Neural Network Layers There are studies that quantify the similarity between different groups of neurons [22], typically layers [10, 51], to compare different neural networks. Generally a normalized representation with desired properties is used to compare different transformer blocks, such as invariance to invertible linear transformation in canonical correlation analysis [33], orthogonal transformation, isotropic scaling, and different initializations in centered kernel alignment [23]. Other measures include representational similarity analysis [30], which studies all pairwise distances across different inputs. Wasserstein distance has been explored in measuring similarities in the context of neural networks [12, 3, 28], but they assume that different layer representations belong to the same metric space, which is very unlikely even if they have the same dimensionality as the semantics captured by each layer are likely to differ significantly. Several similarity measures [45, 6] seek different approximation or or addition to GW distance. While GW distance has been used for model merging as a regularization [42, 43], it has not been fully explored in discovering layer structures and subnetwork identification.

3 Representation Similarity Within Neural Networks

We aim to identify layer groupings of a neural network based on their representation at each layer. To find these layer groups, the problem can be formulated as a search problem: among candidate layers, we seek to find which candidate layers are the most dissimilar to its previous layers, indicating the formation of a new group.

The problem consisting of two key elements: the search space and the similarity measures used to evaluate how closely the candidates in the search space match another (target) representation. We first discuss the notation of representation similarity as the measure for searching and the search space for candidates. We then discuss specific choices on the similarity measure.

Similarity Measures Let $f : X \to Y$ be a function that map x in a set of input $X = \{x_i : x_i \in \mathbb{R}^{d_x}\}_{i=1}^{n}$ to y in a set of output $Y = \{y_i : y_i \in \mathbb{R}^{d_y}\}_{i=1}^{n}$. Each element in Y and X are assumed to be a vector with dimensions d_y and d_x, respectively, with n being the set size, without loss of generality. Note sets can be concatenated into matrix forms as $Y \in \mathbb{R}^{n\times d_y}$ and $X \in \mathbb{R}^{n\times d_x}$.

Definition 3.1. Representation Similarity. Consider neural networks have the form $f = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(X)$, where each layer $l \in \{1, \ldots, L\}$ computes a function $\mathcal{F}^{(l)}(X) = f^{(l)} \circ f^{(l-1)} \circ \cdots \circ f^{(1)}(X)$ given the input X. Representation similarity between two neural layers i and j can be seen as a similarity between their output sets $Y_i = \mathcal{F}^{(i)}(X)$ and $Y_j = \mathcal{F}^{(j)}(X)$, over the same set of input X.

Each intermediate representations of a neural network can be naturally treated as function outputs, given inputs X. We can use a scoring or distance function $D(Y^i, Y^j)$ as a measure between representations similarity between $\mathcal{F}^{(i)}(X)$ and $\mathcal{F}^{(j)}(X)$. If they are close according to D, then the layer should be similar to each other locally at a set of points X. Otherwise, these layers should be different at X. Popular measures such as Euclidean distance have been used for this purpose [22].

The Need for Complex Representation Similarity Measure. Since we cannot exactly control the behavior of a trained neural network, the layer-wise functions $\mathcal{F}$ that it learns can be complex and thus the learned representation Y from each layer may be a complex function from each other rather than a simpler transformation. For example, let $Y_i = \sin(X)$ and $Y_j = \sin^2(X) = (Y^i)^2$. They share strong similarity, but a linear transformation will have trouble to fully capture their similarity. If we want to truly understand where function $\mathcal{F}$ might be approximately computed, we should consider some functions of target Y, but naively listing out all possibilities can be prohibitive. As a consequence, one may need to use more complex measures to deal with such a space.

Search Space We consider multiple candidate Y's to form the search space for comparison. In the context of MLP neural networks for example, where $\sigma(.)$ denotes the non-linearity and Ws are the parameter matrices, we have $Y^* = W_n(\sigma(W_{n-1} \dots \sigma(W_1 X)))$ for the whole network. We can extract many Y's from intermediate layers of the model, for instance $Y_1 = W_1 X$, $Y_2 = \sigma(W_1 X)$, and so on. These Y's are often called representations, activations, or sometimes even "outputs" from each layer. We use these terms interchangeably here. For attention modules in transformer neural networks [47], we can similarly extract Y's from attention key, query, and value functions as well as MLP functions. We list the exact equations and locations of representations considered in the transformer models in Table 2 in Appendix A, which serves as the focus of this paper. We consider attention-based models first, and later we also consider convolutional neural networks with residual layers, with candidate representations listed in Table 3 in in Appendix A.

4 Gromov-Wasserstein Distance as a Similarity Measure

We aim to identify the similarities among the representations at each intermediate layer. Each layer, however, posits a representation that potentially has a different distribution, not to mention even different dimensionality depending on the architectures and layers one considers (viz. mlp and attention layers in transformer blocks). Consequently, representations across layers may be incomparable using standard distance metrics, such as the ℓ_p norm amongst others.

To address these challenges, we propose computing distances between representations at the same layers for different inputs, and match the vertices of a weighted graph – where each dimension of the representation are vertices and the distances indicate weights on the edges – with the vertices of a similarly constructed weighted graph from another layer. Essentially, we assume the representations in a layer are samples of the underlying distribution, and we want the best permutation of representation dimensions in one layer that aligns with vertices in another layer, thereby deriving the inter-layer distance. If this inter-layer distance is low, then we consider the two layers similar.

Formally, without loss of generality, let $Y_1 = \{y_{1i} : y_{1i} \in \mathbb{R}^{d_1}\}_{i=1}^n$ and $Y_2 = \{y_{2i} : y_{2i} \in \mathbb{R}^{d_2}\}_{i=1}^n$ be representations of n examples from two different layers, where the discrete distributions over the representations are μ_1 and μ_2 respectively, with dimension d_1 possibly being different from d_2. Direct distance computation between them is not reasonable. Instead, we seek to compute a coupling or matching $\pi \in \Pi(\mu_1, \mu_2)$ between the n examples in each set such that given the pairwise distances $D_1, D_2 \in \mathbb{R}^{n\times n}$ within representations $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ respectively, the sum of differences between the distances of the matched examples is minimized. Loosely speaking, we aim to find a matching that preserves the pairwise distance as much as possible. In particular, we want to minimize the following:

$$\begin{aligned} &\rho(\boldsymbol{Y}_1, \boldsymbol{Y}_2, \mu_1, \mu_2, D_1, D_2) \triangleq \\ &\min_{\pi\in\Pi(\mu_1,\mu_2)} \sum_{i,j,k,l} (D_1(i,k) - D_2(j,l))^2 \pi_{i,j}\pi_{k,l} \\ &\text{s. t.} \quad \pi \boldsymbol{I} = \mu_1; \pi^T \boldsymbol{I} = \mu_2; \pi \geq 0. \end{aligned} \tag{1}$$

It turns out that ρ corresponds to the Gromov-Wasserstein (GW) distance [7], used to map two sets of points in optimal transport. We thus utilize this distance as a measure of inter-layer functional similarity in the setting where the target is unknown.

4.1 Justification for GW Distance as a Functional Similarity Measure

Let $(\boldsymbol{Y}_1, D_1, \mu_1)$ and $(\boldsymbol{Y}_2, D_2, \mu_2)$ be two given metric measure space (mm-space), where $(\boldsymbol{Y}, D)$ is a compact metric space and μ is a Borel probability measure with full support: supp$(\mu) = \boldsymbol{Y}$. An isomorphism between $\boldsymbol{Y}_1, \boldsymbol{Y}_2$ is any isometry $\Psi : \boldsymbol{Y}_1 \to \boldsymbol{Y}_2$, i.e., a distance-preserving transformation between metric spaces, such that $\Psi_{\#\mu_1} = \mu(\Psi^{-1}) = \mu_2$.

Theorem 4.1. *[31]. The Gromov-Wasserstein distance in equation 1 defines a proper distance on the collection of isomorphism classes of the mm-spaces.*

Remark. The Gromov-Wasserstein distance itself is defined on isomorphism-classes of metric measure spaces, which means that any distance preserving (isometric) transformation of a space should preserve GW distance between the points in that space and any other space [31]. These isometric transformations include rigid motions (translations and rotations) and reflections or compositions of them. Additionally, permutations of points in a space also preserve GW distances, as the points are unlabeled. Hence, GW distance captures much richer transformations across layers.

Favorable Properties of GW. Besides the above noteworthy property of GW, it also has other favorable properties [52, 7]: i) It is symmetric and satisfies the triangle inequality. ii) It is invariant under any isometric transformation of the input, which is advantageous because we do not want rotations and reflections to affect our similarity search. This invariance also includes permutation invariance, which is beneficial since the distance between layer representations should remain unaffected by permutations of neurons. iii) GW is scalable since it does not require estimating high-dimensional distributions, which is scalable to larger hidden dimensions of intermediate layers; instead, it only compare them to obtain a distance measure. iv) GW is monotonic in (positive) scaling of pairwise distances, and hence similar layers should appear to be closer than others with scaling.

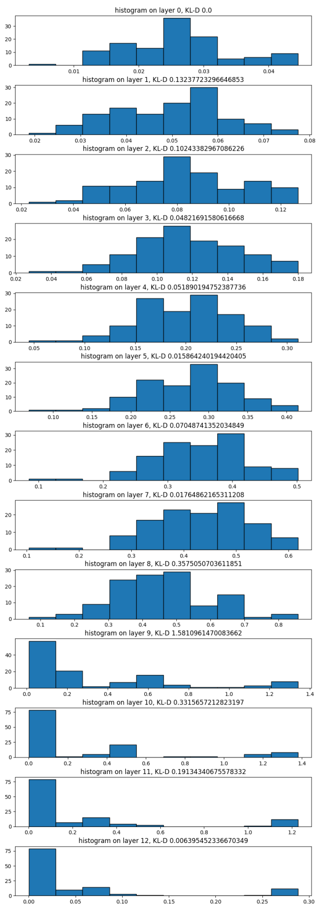


Figure 2: Histogram on pairwise distances for outputs from all transformer blocks in a fine-tuned BERT model trained on YELP dataset. TOP to BOT: layer 0 to layer 12.

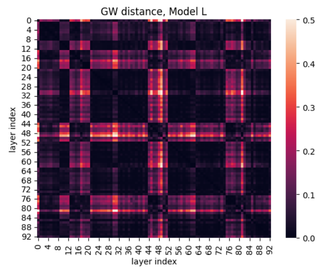
Distance Distributions. As an illustrative example, we plot the histogram on pairwise distances for a batch of samples across all transformer blocks in BERT models from the YELP review dataset in Figure 2. For more details on YELP, we provide a comprehensive discussion of experiments § 5.2. The results in Figure 2 show the distributions on pairwise distances begin to differ from block 9, consistent with GW distance observed in Figure 5, suggesting that significant transformations occur and can be effectively captured by GW. We include the full results and discussion in Appendix F.

Neighborhood Change. Complementary to the distribution of pairwise distances, the changing representations of samples could also alter their relative neighborhoods across transformer blocks. We plot a tSNE projection [46] of representations from a batch of samples on YELP, and visualize it in Figure 2b and Figure 10e of Appendix F. The Jaccard similarity, measuring the overlap between top-5-neighbors of 3 selected samples across different transformer blocks, ranges from 0.0 to 0.43, with average values of $\{0.27, 0.26, 0.26\}$. The full details are shown in Table 5 of Appendix F. Hence, the sample neighborhood changes across blocks, which can be indicative of functional changes that are not captured by comparing distributions alone. However, GW can account for such changes as well.

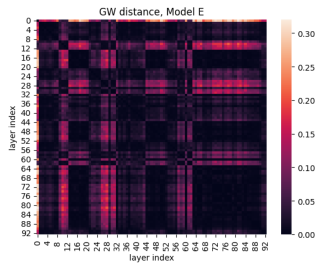
Computation Details. We use an existing optimal transport toolbox, `pythonot` [13], for computing GW distance. Specifically, we use an approximate conditional gradient algorithm proposed in [44], which has a complexity of $O(mn^2 + m^2n)$, where m and n are the dimensions of two spaces (here the number of data samples from two layers being compared). In comparison, the Wasserstein distance [28] may require $O(n^3 log(n))$ for exact computation. When the dataset is large, we can also sub-sample the dataset to improve the computational efficiency.

5 Empirical Study and Findings

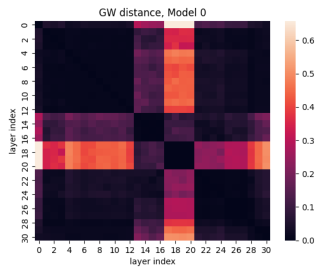
We compare the proposed similarity measure for layer grouping against a set of baselines across multiple datasets, including those from algebraic operation, NLP, and computer vision tasks. We



(a) Model L (layer-wise training) pairwise GW distance, on f_{mod3} dataset.



(b) Model E (end-to-end training) pairwise GW distance, on f_{mod3} dataset.



(c) Model 0 Pairwise GW distance, on f_{mod} dataset.

Figure 3: Pairwise GW distance on the synthetic Modular Sum dataset.

compare 10 different similarity measures, including Euclidean, cosine, mutual information (MI), RSM[22], RSA[22], CCA[33], CKA[23], MSID[45], Wasserstein[12], and AGW[6]. For more details about implementation, please see Appendix H.

5.1 Validation: Synthetic Modular Sum Tasks

Modular sum algorithms [34, 53] and related math problems [17, 4] are often studied to understanding the NN computation due to their precise function definitions. These works have shown that there may exist many different sub-functions in the computation, at different part of neural network. We begin by validating the Gromov-Wasserstein distance by comparing it against known partitions of the networks to determine whether it can successfully identify different layer groups. We first introduce the setup for the experiment, including data generation and models to be investigated.

Setup As a test case, we focus on a modular sum problem, following existing works [34]. We consider two datasets: the first generated by a single modular sum function with $c = f_{\text{mod}}(a+b) = (a+b)\text{mod p}$, where $a, b, c = 0, 1, \ldots, p-1$, with $p = 59$. The second dataset is more complex, with $c = f_{\text{mod3}}(a, b)$ of three levels of modular sums, namely: $c_1 = (a+b)\text{mod}\, p_1, c_2 = (c_1 + b)\text{mod}\, p_2, c = (c_2 + b)\text{mod}\, p_3$, where $p = [59, 31, 17]$.

Training procedure We train 3 different neural networks with transformer blocks to predict c given (a, b), with learned input embedding of size d and a decoding layer for categorical output. For the first simpler f_{mod} dataset, we train a neural network consisting of a one-block ReLU transformer [47], following the same protocol and hyperparameter choices as previous works [34, 53]. We call this **Model 0**. For the more complex f_{mod3} dataset, we train two neural networks consisting of three-block ReLU transformers, with 3 transformer blocks corresponding to the three levels of modular sum functions, and 4 attention heads within each block. The first network, which we call **Model E**, employs an end-to-end training procedure to directly learn output c. For the second network, which we call **Model L**, we use the same architecture as Model E but is trained layer-wise: block 1 predicts (c_1, b), block 2 predicts (c_2, b) with frozen earlier layers, and block 3 predicts c from (c_2, b). All models achieve 100% accuracy on a held-out validation set. More details can be found in appendix B.

Table 1: Gromov-Wasserstein Distance Results for Various Targets in Model L, for f_{mod3} dataset.

GW-D for	Top Similar Layers	$D_{\text{min}} =$
c_1	Resid-Post^1	0.02
c_2	Resid-Post^2	0.03
c	Resid-Post^2, Resid-Post^3, and 7 others	0.04

Results The results are shown in the Table 1. We see that in the **Model L**, the GW distance correctly identifies the most similar layers in accordance with different intermediate c's. The final target c contains 9 similar layers all with distance around 0.04. In Appendix C, we also test probes as the prediction targets are given. Results shows GW distance can be a reliable alternative to the probes to find layer structures. Moreover, as previously mentioned GW distance can naturally compare representations across and within transformer blocks with different dimensions. In Fig 3a and Fig 3b, we visualize the pairwise GW distance between layer representations without a target for **Model L** and **Model E**. Looking at **Model L** we see predominantly 3 groupings of layers: i) layers roughly from 20 to 44 are similar to each other and to layers 52 to 72, ii) layers roughly 12 to 19 are similar to each other and layers 45 to 51 and iii) the initial and last few layers are mainly similar to themselves. Interestingly, the number of groupings corresponds to the 3 functions trained layer-wise in **Model L**. We also observe differences in patterns across **Model L** and **Model E**, suggesting layer-wise and end-to-end training return different networks. Compared to the fixed layer-wise training, end-to-end training in **Model E** may learn faster in the earlier layers and may not have much to learn in later layers, as the function may not be particularly challenging for it. This could explain why, starting from layer 64, all layers in **Model E** exhibit similar representations. Moreover, magnitudes of the distances are also different, with **Model L** showing larger distances, indicating that learning the targets c_1, c_2 result in more functional differences. One possible explanation could be that **Model E** directly operates in the trigonometry space [34], without having to predict the exact integer values until later, thereby suppressing the distances. We include results from baseline methods capable of handling different dimensions between subspaces in Figure 4, and some discussion in Appendix D.

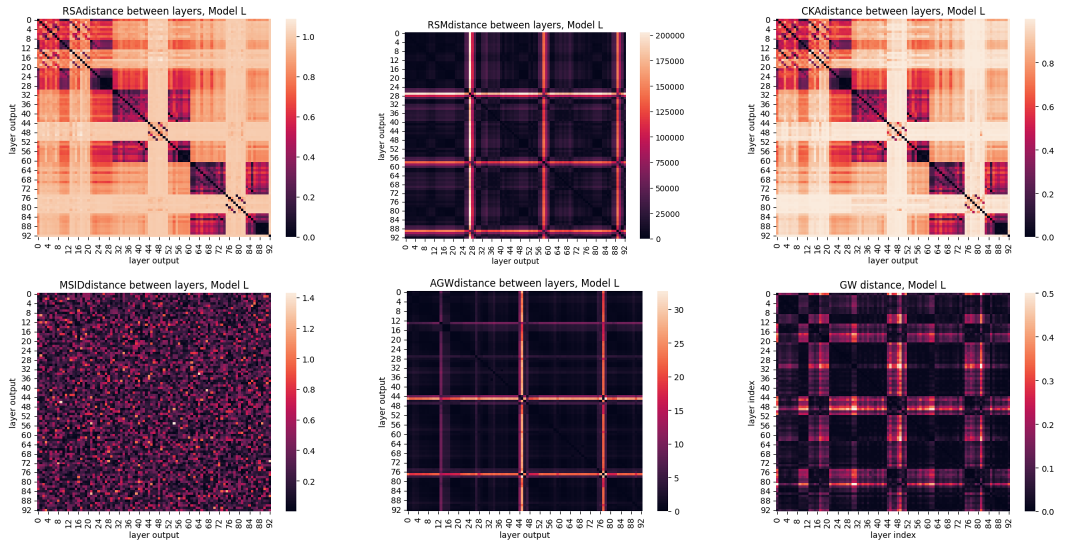


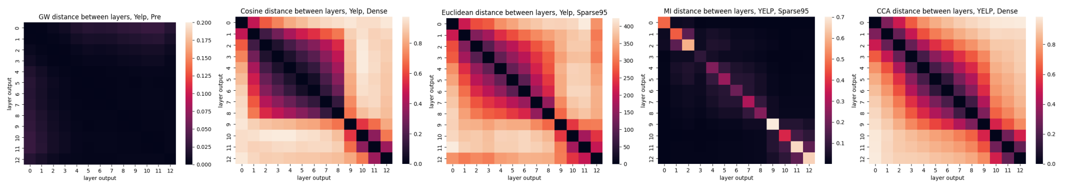
Figure 4: Pairwise (layer) distances on Modular Sum dataset, with layer-wise trained models. Different figures from left to right, top to bottom: *RSA, RSM, CKA, MSID, AGW, and the proposed GW distance*.

To gain a deeper understanding of the operations within each transformer block, we visualize pairwise GW distances among layers for **Model 0** for dataset f_{mod} in Figure 3c. It shows meaningful intra-block transitions (especially at attention components), and shows GW can reveal intra-block as well as inter-block structure. For discussion, we refer the readers to Appendix D.

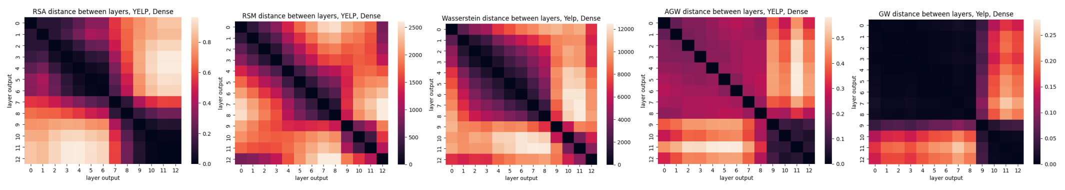
5.2 Real Dataset: NLP Tasks

Setup We now apply GW distance to real natural language processing tasks. We experiment on benchmark sentiment analysis datasets, Yelp reviews and Stanford Sentiment Treebank-v2 (SST2) from the GLUE NLP benchmark [49], with the goal to predict of the text has positive or negative sentiment, and analyze how different layers from fine-tuning BERT(-base) [8] models perform on these datasets. We use the pretrained BERT to generate 4 fine-tuned models, corresponding to a dense model and 3 sparse models with sparsity levels of 25%, 70% and 95% using a state-of-the-art structured pruning algorithm [9], but only show results on densely fine-tuned models here. Training details and more results are in Appendix E and I. Due to the size of BERT models, we limit our analysis to comparing the final representations from each of the 12 transformer blocks, rather than examining all intermediate representations within transformer blocks.

Results In Figure 5a, we see that the pre-trained BERT does not have major differences among blocks, which is not surprising given its accuracy on YELP is only 49.3% (roughly equivalent to random guessing). In Figures 5b to 5j, we show similarity measures from difference baselines. We can see an interesting pattern emerge, revealing two major block structures in the fine-tuned BERT models identified by our approach. The first major differences occur at block 9 and then the last three blocks (10, 11, 12) seem to form a distinct block. This seems to indicate that most of the function/task fitting occurs at these later blocks. GW distance gives the most distinctive layer groups.



(a) Pre-trained (b) Cosine (c) Euclidean (d) MI (e) 95% Sparse



(f) RSA (g) RSM (h) Wasserstein (i) AGW (j) GW, *proposed*

Figure 5: Pairwise (layer) distances on Yelp, across pre-train and densely fine-tuned BERT models, using the various similarity measure. As can be seen GW clearly demarcates the layer groups. Due to page limit, we show more results in Appendix I.

The pattern indicates that later blocks differ significantly from earlier ones. This observation is consistent with fine-tuning and sparsification literature [26, 9], where it has been observed that later blocks typically undergo substantial changes during fine-tuning as they focus on task-specific solutions, while the earlier middle blocks remain stable as they capture syntactic and semantic patterns of the language necessary for various tasks. In appendix J, we further investigate the GW distance between blocks from two different models, providing insights into how representations vary across architectures.

On SST2 dataset, we also observe very similar patterns with the GW distance and 3 baselines, for which we refer the readers to appendix K for detailed results. In both datasets, low distance measure are consistently observed in the diagonal elements, but the overall block structures are not as obvious in the baselines as they are with GW distance, highlighting the effectiveness of the GW distance.

Model Compression and Fine-Tuning The presence of block structures in the GW-distance matrices indicates major functional changes may concentrate at these transition blocks. This finding may suggest that for other downstream tasks, we may consider freezing the model up to Block 8 and only fine-tuning the blocks after that. We validate this observation in appendix G, showing minimal accuracy drop. We also consider a model compression application where only 4 transformer blocks can achieve similarly good performance, as discussed in Appendix L.

Clustering Besides visualization above, one can also utilize clustering methods to automatically identify the layer groups from the GW distance. We tested spectral clustering [48] on a similarity matrix computed as the reverse pairwise GW distance matrix. This method successfully identified 2 groups with block $1 \sim 8$ and block $9 \sim 12$. Note that one could use a clustering algorithm and use some quantitative evaluations, such as clustering metrics like silhouette scores, to measure which metrics provide better clusters. However, since different metrics produce different scores, it is hard to compare clustering groups when the inputs to a cluster algorithm are different.

5.3 Training Dynamics: Emergence of Layer Groups During Training

We also visualize the GW distance between blocks while fine-tuning the pretrained BERT model on YELP datasets in the entire training process, in order to observe when these layer grouping structures begin to emerge. Figure 6 show a few visualization on GW distances at selected training iterations. Block distances are low in the beginning (observed in Figure 5a), but by iteration 300 the last block begin to differ from other blocks. As training progresses, block 9, 10, and 11 begin to show at iteration 3k and 15k. These growing differences in GW distance reflect the model's increasing F1 score on the test data. Overall it show the gradual specialization of blocks into distinct sub-networks, with each sub-network potentially focusing on different aspects of the task.

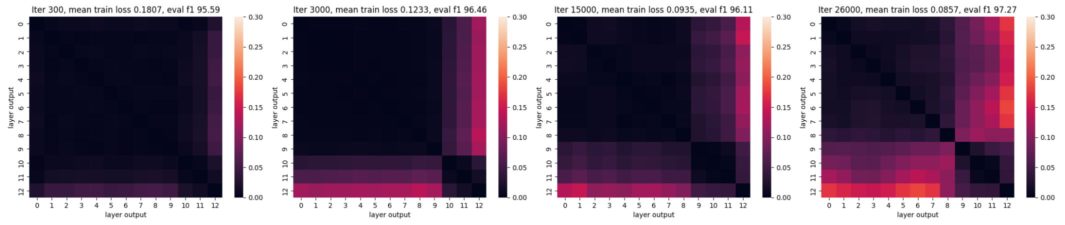

(a) Iteration = 300 (b) Iteration = 3k (c) Iteration = 15k (d) Iteration = 26k

Figure 6: Pairwise GW distance in YELP datasets, over training iterations.

We also plot the mean GW distance of all block pairs in Figure 7. Figure 7a show the mean GW distance over training iterations, and show it grows over time. Figure 7b shows that mean GW distance versus two different accuracy metric on the test dataset. GW distance grow slowly at first, followed by a rapid increase as the model achieves better accuracy and F1 scores. Such observation is consistent with existing “grokking" behavior, where validation accuracy can suddenly increases well after achieving near perfect training accuracy [34]. Similarly, Figure 7c shows a rapid increase in mean GW distance in order to achieve a lower training loss.

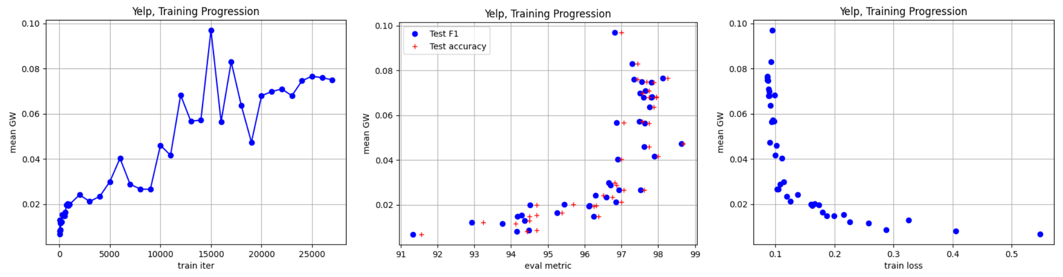


Figure 7: Pairwise GW distance in YELP datasets, over training iterations.

5.4 ResNet and Computer Vision Dataset

In addition to the attention-based architectures, we also test our approach on ResNet9, a popular convolutional neural network architecture [16, 39]. We compare a randomly initialized ResNet9 and a trained one on CIFAR 10 image dataset CIFAR-10 [25], achieving 91.63% accuracy on the test data. For more details on the setup, we refer the readers to Appendix N. In Figure 19, we show the pairwise distance among layers using baselines that handle different input dimensions. Overall, GW distance show the most clear divisions of layer grouping.

To further examine how the sub-network structures align with learned representation, we visualize the computed distances alongside the learned representations of an image of class ""ship" across all layers in Figure 8. The top row shows the representations of a ship at each layer. To see the gradual changes over layers, we visualize the distance between every layer and its previous layer, using various methods capable of handling different dimensions between compared spaces. Overall,

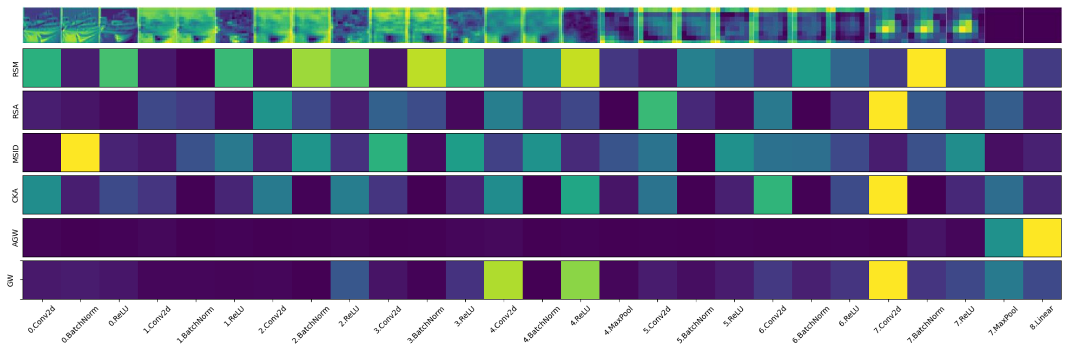


Figure 8: Pairwise layer distance between every layer and its previous layer on CIFAR-10, for various baselines, including RSM, RSA, MSID, CKA, AGW, and GW. Each Row Represents one method.

RSM, RSA, MSID, and CKA show indicate significant changes across many layers, without clear evidence of sub-network structures. AGW highlights the changes in the final few layers only. In comparison, GW distance demonstrates the most consistency with the image representations visually. Specifically, the 3rd convolution layer (Layer ID 2.ReLU) introduces the first notable differences, where the ship's shape becomes less distinct, signaling the learning of mid-level features. The shapes become increasingly blurred in the 5th convolution layers (Layer ID 4.Conv2d) and by Layer 4.ReLU the ship's shape is nearly absent. The final convolutional layer (Layer ID 7.Conv2d) shows significant changes from its preceding layer (Layer ID 6.ReLU), marking the point where class-specific information is consolidated. These results suggest that GW distance aligns most effectively with the learned representations, providing strong evidence that it reveals meaningful layer structures.

6 Discussion

We proposed to model interpretation based on representation similarity within intermediate layers of neural networks, using GW distance to compute such similarities. To the best of our knowledge, our application of GW distance in this context is novel. On algebraic, real NLP, and vision tasks, we identified the existence of major groups amongst layers, corresponding to functionally meaningful abstractions. These results reveal implicit layer structure within neural networks, and highlight the potential sudden transition in network computation instead of smooth function change.

There are several limitations to our approach. GW is computationally more expensive than cosine/Euclidean and existing approximate solvers may introduce variance. Moreover, there are cases when invariance properties (such as permutation invariance) might mask meaningful structural differences, GW may not fail to reveal these changes.

Future work could investigate other models and applications to observe general trends. Theoretical study of special properties of GW distances within the context of neural network interpretability is also an interesting future direction.

Limitations and Broader Impact Our approach first assumes we have access to the intermediate layer representations, which may not be available for some black-box models. Our approach is general, but assumes the proposed distances correctly represent the representation similarities. Our findings are also limited to the datasets and models studied and are not guaranteed to be observed in other scenarios. In terms of broader impact, our approach could be applied widely given its simplicity for identifying layer grouping in neural networks. However, more investigations on inner mechanisms will have to be done, perhaps building on our approach, in order to fully understand the behavior of neural models.

Acknowledgments and Disclosure of Funding

We thank colleagues and anonymous reviewers for their helpful feedback in many versions of this paper.

References

[1] N. Bonneel, M. Van De Panne, S. Paris, and W. Heidrich. Displacement interpolation using lagrangian mass transport. In *Proceedings of the 2011 SIGGRAPH Asia conference*, pages 1–12, 2011.

[2] T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. E. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[3] J. Cao, J. Li, X. Hu, X. Wu, and M. Tan. Towards interpreting deep neural networks via layer behavior understanding. *Machine Learning*, 111(3):1159–1179, 2022.

[4] F. Charton. Learning the greatest common divisor: explaining transformer predictions. In *The Twelfth International Conference on Learning Representations*, 2023.

[5] A. Conmy, A. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36, 2024.

[6] P. Demetci, Q. H. Tran, I. Redko, and R. Singh. Revisiting invariances and introducing priors in gromov-wasserstein distances. *arXiv preprint arXiv:2307.10093*, 2023.

[7] P. Demetci, Q. H. Tran, I. Redko, and R. Singh. Revisiting invariances and introducing priors in gromov-wasserstein distances. *arXiv:2307.10093*, 2023.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *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, June 2019. Association for Computational Linguistics.

[9] A. Dhurandhar, T. Pedapati, R. Luss, S. Dan, A. Lozano, P. Das, and G. Kollias. Neuroprune: A neuro-inspired topological sparse training algorithm for large language models. *Findings of the Association for Computational Linguistics*, 2024.

[10] F. Ding, J.-S. Denain, and J. Steinhardt. Grounding representation similarity with statistical testing. *arXiv preprint arXiv:2108.01661*, 2021.

[11] M. Dreyer, E. Purelku, J. Vielhaben, W. Samek, and S. Lapuschkin. Pure: Turning polysemantic neurons into pure features by identifying relevant circuits. *arXiv preprint arXiv:2404.06453*, 2024.

[12] K. Dwivedi and G. Roig. Representation similarity analysis for efficient task taxonomy & transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12387–12396, 2019.

[13] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[14] M. Geva, J. Bastings, K. Filippova, and A. Globerson. Dissecting recall of factual associations in auto-regressive language models. *arXiv preprint arXiv:2304.14767*, 2023.

[15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] T. He, D. Doshi, A. Das, and A. Gromov. Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks. *arXiv preprint arXiv:2406.02550*, 2024.

[18] E. Hernandez, A. S. Sharma, T. Haklay, K. Meng, M. Wattenberg, J. Andreas, Y. Belinkov, and D. Bau. Linearity of relation decoding in transformer language models. *arXiv preprint arXiv:2308.09124*, 2023.

[19] Y. Hou, J. Li, Y. Fei, A. Stolfo, W. Zhou, G. Zeng, A. Bosselut, and M. Sachan. Towards a mechanistic interpretation of multi-step reasoning capabilities of language models. *arXiv preprint arXiv:2310.14491*, 2023.

[20] J. Huang, Z. Wu, C. Potts, M. Geva, and A. Geiger. Ravel: Evaluating interpretability methods on disentangling language model representations. *arXiv preprint arXiv:2402.17700*, 2024.

[21] X. Huang, M. Panwar, N. Goyal, and M. Hahn. Inversionview: A general-purpose method for reading information from neural activations. *arXiv preprint arXiv:2405.17653*, 2024.

[22] M. Klabunde, T. Schumacher, M. Strohmaier, and F. Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *arXiv preprint arXiv:2305.06329*, 2023.

[23] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR, 2019.

[24] J. Kramár, T. Lieberum, R. Shah, and N. Nanda. Atp*: An efficient and scalable method for localizing llm behaviour to components. *arXiv preprint arXiv:2403.00745*, 2024.

[25] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] J. Li, R. Cotterell, and M. Sachan. Differentiable subset pruning of transformer heads. *Trans. Assoc. Comput. Linguistics*, 9:1442–1459, 2021.

[27] Y. Li, J. Yosinski, J. Clune, H. Lipson, and J. Hopcroft. Convergent learning: Do different neural networks learn the same representations? *arXiv preprint arXiv:1511.07543*, 2015.

[28] S. Lohit and M. Jones. Model compression using optimal transport. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2764–2773, 2022.

[29] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[30] J. Mehrer, C. J. Spoerer, N. Kriegeskorte, and T. C. Kietzmann. Individual differences among deep neural network models. *Nature communications*, 11(1):5725, 2020.

[31] F. Mémoli. Gromov–wasserstein distances and the metric approach to object matching. *Foundations of computational mathematics*, 11:417–487, 2011.

[32] K. Meng, D. Bau, A. Andonian, and Y. Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

[33] A. Morcos, M. Raghu, and S. Bengio. Insights on representational similarity in neural networks with canonical correlation. *Advances in neural information processing systems*, 31, 2018.

[34] N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

[35] T. Nguyen, M. Raghu, and S. Kornblith. On the origins of the block structure phenomenon in neural network representations. *arXiv preprint arXiv:2202.07184*, 2022.

[36] F. Ortu, Z. Jin, D. Doimo, M. Sachan, A. Cazzaniga, and B. Schölkopf. Competition of mechanisms: Tracing how language models handle facts and counterfactuals. *arXiv preprint arXiv:2402.11655*, 2024.

[37] V. Palit, R. Pandey, A. Arora, and P. P. Liang. Towards vision-language mechanistic interpretability: A causal tracing tool for blip. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2856–2861, 2023.

[38] K. Park, Y. J. Choe, Y. Jiang, and V. Veitch. The geometry of categorical and hierarchical concepts in large language models. *arXiv preprint arXiv:2406.01506*, 2024.

[39] S. M. Park, K. Georgiev, A. Ilyas, G. Leclerc, and A. Madry. Trak: Attributing model behavior at scale. *arXiv preprint arXiv:2303.14186*, 2023.

[40] G. Peyré, M. Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[41] C. Shi, N. Beltran-Velez, A. Nazaret, C. Zheng, A. Garriga-Alonso, A. Jesson, M. Makar, and D. Blei. Hypothesis testing the circuit hypothesis in llms. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024.

[42] S. P. Singh and M. Jaggi. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33:22045–22055, 2020.

[43] G. Stoica, D. Bolya, J. Bjorner, P. Ramesh, T. Hearn, and J. Hoffman. Zipit! merging models from different tasks without training. *arXiv preprint arXiv:2305.03053*, 2023.

[44] V. Titouan, N. Courty, R. Tavenard, and R. Flamary. Optimal transport for structured data with application on graphs. In *International Conference on Machine Learning*, pages 6275–6284. PMLR, 2019.

[45] A. Tsitsulin, M. Munkhoeva, D. Mottin, P. Karras, A. Bronstein, I. Oseledets, and E. Müller. The shape of data: Intrinsic distance for data distributions. *arXiv preprint arXiv:1905.11141*, 2019.

[46] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[48] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.

[49] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 24th International Conference on Learning Representations*, 2019.

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

[51] L. Yu, M. Cao, J. C. K. Cheung, and Y. Dong. Mechanisms of non-factual hallucinations in language models. *arXiv preprint arXiv:2403.18167*, 2024.

[52] L. Zheng, Y. Xiao, and L. Niu. A brief survey on computational gromov-wasserstein distance. *Procedia Computer Science*, 199:697–702, 2022. The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19.

[53] Z. Zhong, Z. Liu, M. Tegmark, and J. Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Experiment result validation.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: At the end of main paper.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: with citations.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Details discussed in appendix.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Will release code upon the acceptance of the paper.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Details in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: Not applicable.

   Guidelines: Results on different seeds in Figure 18.

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: On a single laptop with 3.3GHz CPU and 36G memory, no GPU.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: At the end of paper.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: Use BERT models only.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Representations in Transformer-based and Convolution Neural Network

We consider multiple candidate Y's to form the search space for target Y^0. In the context of MLP neural networks for example, where $\sigma(.)$ denotes the non-linearity and Ws are the parameter matrices, we have $Y^* = W_n(\sigma(W_{n-1} \ldots \sigma(W_1 X)))$ for the whole network. We can extract many Y's from intermediate functions of the model, for instance $Y_1 = W_1 X$, $Y_2 = \sigma(W_1 X)$, and so on. These Y's are often called representations, activations, or sometimes even "outputs" from each layer.

For attention modules in transformer neural networks [47], we can similarly extract Y's from attention key, query, and value functions as well as MLP functions. More specifically, a deep transformer architecture of depth l is formed by sequentially stacking l transformer blocks. Each transformer block takes the representations of a sequence $\boldsymbol{X}_{\text{in}} \in \mathbb{R}^{T\times d}$, where $\boldsymbol{X}_{\text{in}}$ = Emb($\boldsymbol{X}$) with embedding layer Emb and input $\boldsymbol{X}$, T is the number of tokens and d is the embedding dimension, and outputs $\boldsymbol{X}_{\text{out}}$, where:

$$\begin{aligned}
\boldsymbol{X}_{\text{out}} &= \alpha_{\text{FF}}\hat{\boldsymbol{X}} + \beta_{\text{FF}}\text{MLP}(\text{Norm}(\hat{\boldsymbol{X}})) \\
\text{where,}\quad \text{MLP}(\boldsymbol{X}_m) &= \sigma(\boldsymbol{X}_m \boldsymbol{W}^1)\boldsymbol{W}^2 \\
\hat{\boldsymbol{X}} &= \alpha_{\text{SA}}\boldsymbol{X}_{\text{in}} + \beta_{\text{SA}}\text{MHA}(\text{Norm}(\boldsymbol{X}_{\text{in}})), \\
\text{MHA}(\boldsymbol{X}) &= [\text{Attn}_1(\boldsymbol{X}), \ldots, \text{Attn}_H(\boldsymbol{X})]\boldsymbol{W}^P, \\
\text{Attn}(\boldsymbol{X}) &= \boldsymbol{A}(\boldsymbol{X})\boldsymbol{X}\boldsymbol{W}^V, \\
\boldsymbol{A}(\boldsymbol{X}) &= \text{softmax}\left(\frac{1}{\sqrt{d_k}}\boldsymbol{X}\boldsymbol{W}^Q \boldsymbol{W}^{K^\top}\boldsymbol{X}^\top + \boldsymbol{M}\right),
\end{aligned} \tag{2}$$

with scalar weights α_{FF}, β_{FF} , α_{SA}, and β_{SA} usually set to 1 by default. Here FF stands for feedforward network, SA stands for self-attention, MHA is Multi-Head Attention, and Norm is a normalization layer. MLP usually has a single hidden layer with dimension d and ReLU activation. The MHA sub-block shares information among tokens by using self-attention with $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$ and $\boldsymbol{W}^V$ indicating query, key and value matrices. We list the exact locations of representations considered in the transformer models in Table 2.

Table 2: Representations Y in the attention-based model considered in experiments as per equation 2. Omitting Y in most names for readability.

(Across Blocks)						
Name	Resid-Pre^l	Y^l, at each block				
Value	$= \boldsymbol{X}^l_{\text{in}}$	$= \boldsymbol{X}^l_{\text{out}}$				
(Within Each Block l)						
Name	Attn-Out^l	Resid-Mid^l	Pre	Post	MLP-out^l	Resid-Post^l
Value	=MHA$(\boldsymbol{X})^l$	$= \hat{\boldsymbol{X}}$	$= \hat{\boldsymbol{X}}\boldsymbol{W}^1$	=MLP$(\hat{\boldsymbol{X}})$	= MLP$(\hat{\boldsymbol{X}})$	$= \boldsymbol{X}_{\text{out}}$
(Within Each Attention Head h)						
Name	k_h	q_h	Attn-Pre$_h$	Attn$_h$	v$_h$	z$_h$
Value	$= \boldsymbol{X}\boldsymbol{W}^K$	$= \boldsymbol{X}\boldsymbol{W}^Q$	$= q_h k_h^T$	$= \boldsymbol{A}(\boldsymbol{X})$	$= \boldsymbol{X}\boldsymbol{W}^V$	=Attn$(\boldsymbol{X})$

We also consider convolution neural networks for computer vision datasets. Specifically, we use a relatively lightweight ResNet9 [16, 39]. The exact locations of the candidate representations considered are listed in Table 3.

B Modular Sum Experiment Details

We use the same architecture and protocols in training, as previous modular papers [34, 53], based on their available Github repos. Specifically, we use transformer width $d = 128$, and each attention head has 32 dimensions. As a result, MLP has 512 hidden neurons. ReLU is used as the activation throughout the models,

Data Among all data points ($59^2 = 3481$ of them), we randomly select 80% as training samples and 20% as validation samples.

Table 3: All representations Y considered in ResNet 9 in experiments.

(Module 0)				
Name	0.Conv2d	0.BatchNorm	0.ReLU	
Details	in-channel = 3, out =64, kernel size = (3,3)	Batch Normalization	activation	
(Module 1)				
Name	1.Conv2d	1.BatchNorm	1.ReLU	
Details	in-channel = 64, out =128, kernel size = (5,5)	Batch Normalization	activation	
(Module 2 & 3: Residual Block)				
Name	2.Conv2d	2.BatchNorm	2.ReLU	
Name	3.Conv2d	3.BatchNorm	3.ReLU	
Details	in-channel = 128, out =128, kernel size = (3,3)	Batch Normalization	activation	
(Module 4)				
Name	4.Conv2d	4.BatchNorm	4.ReLU	4. MaxPool
Details	in-channel = 128, out =256, kernel size = (3,3)	Batch Normalization	activation	Kernel (2,2)
(Module 5 & 6: Residual Block)				
Name	5.Conv2d	5.BatchNorm	5.ReLU	
Name	6.Conv2d	6.BatchNorm	6.ReLU	
Details	in-channel = 256, out =256, kernel size = (3,3)	Batch Normalization		
(Module 7)				
Name	7.Conv2d	7.BatchNorm	7.ReLU	7. MaxPool
Details	in-channel = 256, out =128, kernel size = (3,3)	Batch Normalization	activation	Adaptive
(Module 8)				
Name	8.Linear (classification)			
Details	in-feature = 128, out =10			

Hyperparameters We used AdamW optimizer [29] with learning rate $\gamma = 0.001$ and weight decay factor $\beta = 2$. We use the shuffled data as one batch in every epoch. We train models from scratch and train for 26,000 epoches.

Search Space For the f_{mod3} dataset, we consider all layers in the network, including all representations within transformer blocks. As shown in Table 2, each attention head has 6 intermediate layers, for a total of 24. Each block has an additional 7 layers (1 input layer, Resid-Pre, and 6 intermediate layers). Hence, for three blocks each with four attention heads, we have a total of 93 representations to evaluate, as each block has $31 = 24 + 7$ representations.

Training procedure We train 3 different neural networks with transformer blocks to predict c given (a, b). These networks contain input embeddings for a and b, each of size d, i.e., $[\boldsymbol{E}_a, \boldsymbol{E}_b] \in \mathbb{R}^{2d}$, and predict a categorical output c via an unembedding/decoding layer. All parameters in the network are learned. For the first simpler f_{mod} dataset, we train a neural network consisting of a one-block ReLU transformer [47], following the same protocol and hyperparameter choices as previous works [34, 53]. We call this **Model 0**. For the more complex f_{mod3} dataset, we train two neural networks consisting of three-block ReLU transformers, with 3 transformer blocks corresponding to the three levels of modular sum functions, and 4 attention heads within each block. The first network, which we call **Model E**, employs an end-to-end training procedure to directly learn output c given input (a, b). For the second network, which we call **Model L**, we use the same architecture as Model E but with a layer-wise training approach instead of end-to-end training. Specifically, we use the following 3-step procedure:

1) We train the first transformer block of Model L to predict (c_1, b) using an additional linear layer on top, given inputs (a, b). 2) Once block 1 is fully trained, we discard the linear layer, freeze everything before the linear layer, and use its representations of (c_1, b) to train the second block to predict (c_2, b), again incorporating an additional layer on top. 3) Finally, we repeat the above step by freezing the first and second block and training the last block to predict c, using representations of (c_2, b).

In all these models, we are able to achieve 100% prediction accuracy on a separate validation dataset.

To evaluate the capability of handling different dimensions, we directly measure GW distance between the 93 intermediate representation Y (see appendix B for search space details) and c's. To speed up computation of GW distance, we randomly sub-sample 1000 data from a total of 3600 samples, reducing time from 2 min to 5 seconds for each computation.

C Probes on Modular Sum Dataset: When Target is Known

When the target is a value from a known function, we can directly compare outputs between representations from each layer and the known function output. Representations from each layer can be directly compared with the target via a probe. We first consider **Model E** and then **Model L**.

Linear Probe Popular linear probes can be used to assess the similarity between a target and any layer's representation. We perform linear regression of each target (c_1, c_2, c) on each of the 93 representations Y, and report the residual error as the scoring distance function between Y and c's.

Table 4: Linear and Nonlinear Probe Results, for f_{mod3} dataset.

Model L	Linear Probe for	Perfect Match?	Top Similar Layers	$D_{\text{min}} =$
	c_1	✓	Resid-Post^1 and 21 others	0
	c_2	✓	Resid-Post^2 and 21 others	0
	c	✓	Resid-Post^3 and Post^2	0
Model E	**Linear Probe for**	**Perfect Match?**	**Top Similar Layers**	$D_{\text{min}} =$
	c_1	×	Post^2	0.522
	c_2	×	Post^1	0.93
	c	✓	Resid-Post^3 and 5 others	0
Model E	**Nonlinear Probe for**	**Perfect Match?**	**Top Similar Layers**	$D_{\text{min}} =$
	c_1	✓	Resid-Post^1 and 15 others	0
	c_2	✓	Resid-Post^1 and 4 others	0
	c	✓	Resid-Post^3 and 9 others	0

Results Since we perform layer-wise training with **Model L**, we know the true locations of c_1 and c_2, which sit at $\boldsymbol{X}^1_{\text{out}}$ and $\boldsymbol{X}^2_{\text{out}}$ with names Resid-Post^1 and Resid-Post^2, respectively. As shown in the top part of Table 4, a linear regression probe can predict targets perfectly with these two layers. In fact, there are 21 other layers which also show perfect accuracy. For c_1, these consist of Post^0 and MLP-out^0 from the same block and some layers from the next block, including linear operations with all k's, q's, v's. The final prediction c can be linearly predicted as expected, due to the model's perfect prediction accuracy.

Naturally we would like to confirm if the same happens with **Model E**: if we use the same linear probe, does each block in **Model E** learn the corresponding c at the output of the transformer block? As shown in the mid part of Table 4, we are not able to find any layer that produces a representation that is linearly predictive of c_1 and c_2, with the lowest prediction errors at 52% and 93%, respectively. Moreover, the most similar layers to c_1 and c_2 are in the 2nd block and 1st block respectively, instead of the expected 1st and 2nd blocks. This seems to suggest that **Model E** does not actually learn any function of c_1 and c_2.

Non-linear Probe As discussed previously, to deal with the potentially large search space of functions of the target, a more powerful probe (such as a nonlinear MLP function) may have to be used so that it can detect more complex similarities to c. Therefore, we train a two-layer MLP[2] to predict c's. As shown at the bottom of Table 4, these two-layer MLPs have more predictive power and can perfectly predict the targets, while still showing differences among various layers indicating that the matched layers do capture the intended target functions while other layers do not. Many layers in the 3rd block, for example, have only 1% accuracy relative to c_1. This indicates that non-linear probes can be used to find subgroups of layers in neural networks. Unlike existing work that primarily focuses on linear probes, we show that non-linear probes, still with limited capacity, are useful.

One issue with using predictive probes to compute the distance measure D is that the target function has to be known. In practice, however, we may not know any intermediate targets, as suggested in the end-to-end training of **Model E**. While we still can try different target functions and use non-linear probes, the infinite number of possible targets makes such an approach inefficient. This calls for a different strategy to differentiate sub-components in a network through representation similarity.

[2]We use the neural network classifier from the scikit-learn package, with default parameters.

D Baseline Comparison Results on Modular Sum

To gain a deeper understanding of the operations within each transformer block, we visualize pairwise GW distances among layers for **Model 0** for dataset f_{mod} in Figure 3c. In this case, we have a total of 31 representations since only one transformer block is used. We notice the first major difference occurs between layers 13 and 16, which are 4 Attn-Pre (computing key and value product). The second difference occurs between layers 17 and 20, which are the first 3 Attn (computing $\boldsymbol{A}(\boldsymbol{X})$). This suggests that major computation seems to be done by the attention mechanism. Note that distances are not monotonically increasing across layers, which is expected as the representation spaces can change significantly given the heterogeneity of the operations such as those performed by residual connections and attention within a transformer block.

We have also tested a few baselines that can handle different space dimensions, shown in Figure 9. RSA and CKA reveal different levels of lay grouping within attention layers and across transformer blocks. AGW demonstrates the highest sensitivity to attention computations, while RSM finds the last few layers within each transformer block.

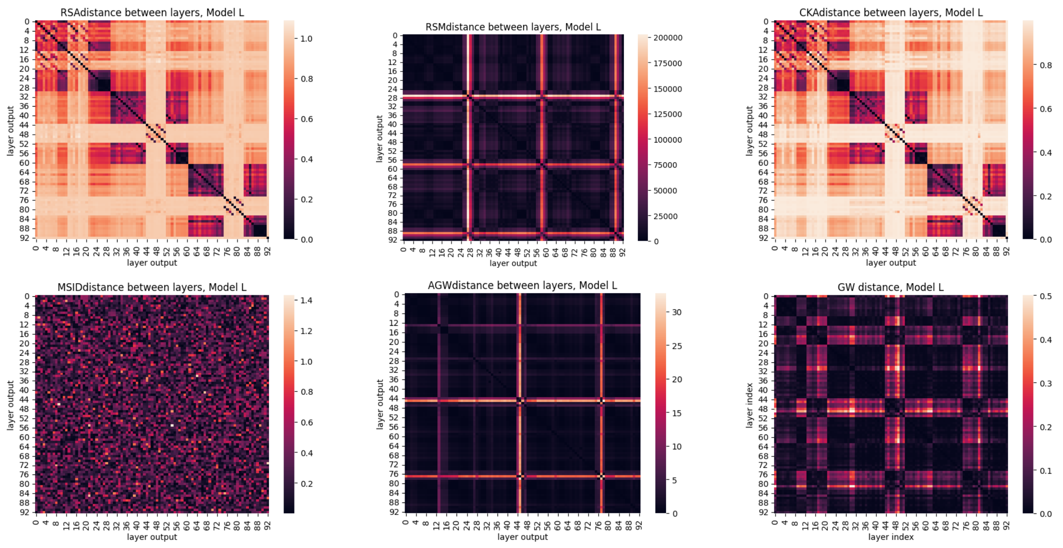


Figure 9: Pairwise (layer) distances on Modular Sum dataset, with layer-wise trained models. Different figures from left to right, top to bottom: *RSA, RSM, CKA, MSID, AGW, and the proposed GW distance*.

E Real NLP Experiment Details

We analyze a BERT-base-uncased [8] model based on our optimal matching inspired mechanistic interpretability approach. We fine tune it on two well known datasets in NLP; i) Yelp reviews (`https://www.kaggle.com/code/suzanaiacob/sentiment-analysis-of-the-yelp-reviews-data`) and ii) Stanford Sentiment Treebank-v2 (SST2), which is part of the GLUE NLP benchmark [49]. Both of these are sentiment analysis tasks, where the goal is to predict if a piece of text has positive or negative sentiment. The Yelp dataset has hundreds of thousands of reviews, while the SST2 dataset has tens of thousands of sentences. The training details are as follows: i) Hardware: 1 A100 Nvidia GPU and 1 intel CPU, ii) Max. Sequence Length : 256, iii) Epochs: 1, iv) Batch Size: 16 and v) Learning Rate: $2e^{-5}$ with no weight decay. The accuracy on Yelp was 97.87%, while that on SST2 was 92.4%. Without fine tuning the pre-trained BERT models accuracy on Yelp and SST2 was 49.29% and 50.34% respectively indicative of random chance performance.

We also fine tuned a series of sparse models on these datasets. The method we used to sparsify was a state-of-the-art dynamic sparse training approach NeuroPrune [9], which leads to high performing structured sparse models. Using this approach and the same training settings as above we created BERT models with 25%, 70% and 95% sparsity which had accuracies of 96.31%, 97.53% and 96.22% respectively for the Yelp dataset and accuracies of 90.25%, 88.5% and 84.4% respectively for the SST2 dataset. We then used the resultant models for our analysis.

F GW Justification and Alignment

Distance Distributions. As an illustrative example, we plot the histogram on pairwise distances for a batch of samples across all transformer blocks in BERT models from the YELP review dataset in Figure 2. The results in Figure 2 show the distributions on pairwise distances begin to differ from block 9, consistent with GW distance observed in Figure 5, suggesting that significant transformations occur and can be effectively captured by GW.

Neighborhood Change. Complementary to the distribution of pairwise distances, the changing representations of samples could also alter their relative neighborhoods across transformer blocks. We plot a tSNE projection [46] of representations from a batch of samples on YELP, and visualize it in Figure 2b and Figure 10e. The Jaccard similarity, measuring the overlap between top-5-neighbors of 3 selected samples across different transformer blocks, ranges from 0.0 to 0.43, with average values of $\{0.27, 0.26, 0.26\}$. The full details are shown in Table 5, as discussed below. Hence, the sample neighborhood changes across blocks, which can be indicative of functional changes that are not captured by comparing distributions alone. However, GW can account for such changes as well.

We plot a tSNE projection [46] down to 2 dimensions, on a batch of 16 samples (color indicative of sample) on YELP, and visualize it in Figure 10e. As one can see, the sample neighborhood changes across layers, which can be indicative of functional changes but something that is not captured by comparing distributions. However, GW can also account for such changes.

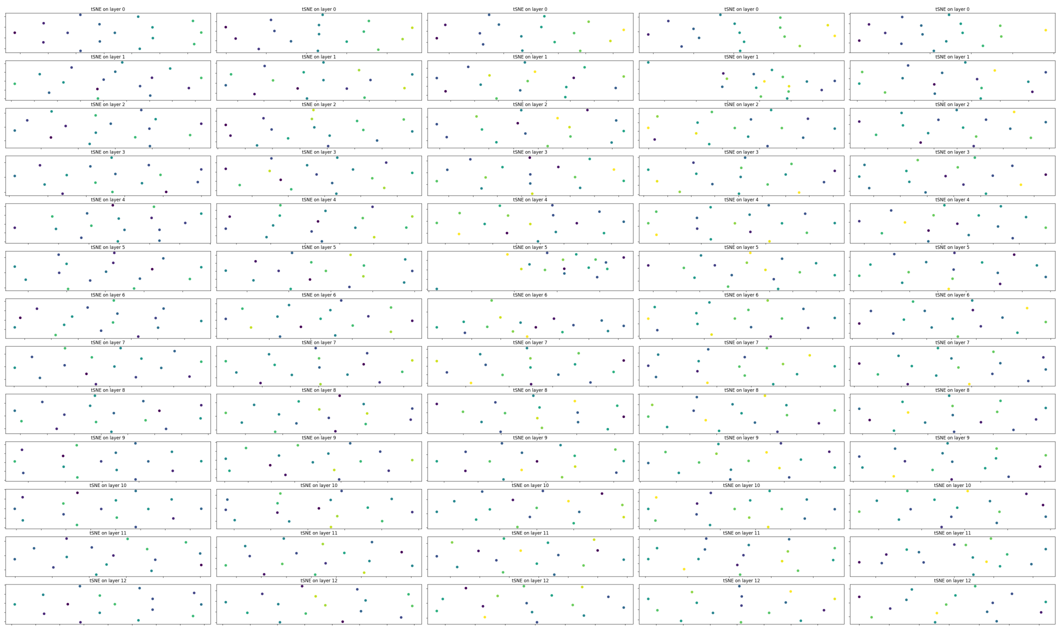

(a) Pretrained (b) Dense (c) 25% Sparse (d) 70% Sparse (e) 90% Sparse

Figure 10: tSNE projection on intermediate representations on Yelp, across BERT models with different sparsity levels. Different Rows: Results from all 12 transformer blocks, from top to bottom. Different columns: first column is the pre-trained BERT and the rest are fine tuned BERT models with increasing sparsity (dense, 25%, 70% and 95% sparsity).

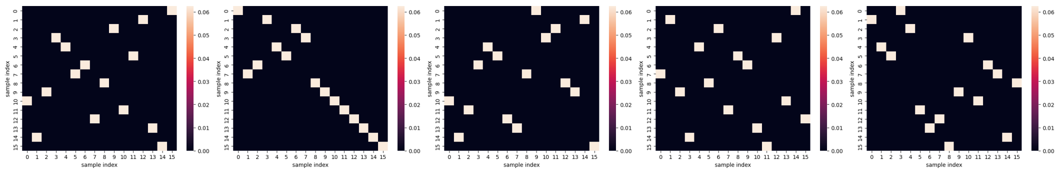
We also show Jaccard similarity measure on top-5-neighbors, per Euclidean distances on tSNE projection, of each of 3 samples across different transformer blocks. Jaccard similarity is a measure of two sets, computed as their intersection divided by their union. Results are shown in Table 5. This further shows the sample neighborhood changes across layers, and representation similarity measures should account for such changes.

To show the exact transportation plan from GW distances, we choose plot one batch of data with size 16, and show the transportation plan over 5 random layer pairs in Figure 11. As one can see, the transportation plan does not conform to identity-mapping. Both Wasserstein and Euclidean distance will likely have trouble handle in this case. We also note that the transportation plan shown Figure 11

Table 5: Jaccard Similarity on top-5-neighbors of Selected Samples across all transformer blocks.

Sample 1	1	2	3	4	5	6	7	8	9	10	11	12	Mean
Block 0 v.s.	0.25	0.25	0.25	0.11	0.43	0.11	0.25	0.25	0.11	0.11	0.25	0.25	**0.27**
Sample 2	1	2	3	4	5	6	7	8	9	10	11	12	Mean
Block 0 v.s.	0.11	043	0.11	0.11	0.11	0.0	0.11	0.25	0.25	0.43	0.25	0.25	**0.26**
Sample 3	1	2	3	4	5	6	7	8	9	10	11	12	Mean
Block 0 v.s.	0.0	0.11	0.43	0.25	0.25	0.25	0.11	0.66	0.11	0.11	0.11	0.0	**0.26**

is a permutation of the original data, rather than a distributed transportation plan. This behavior is consistent with existing Wasserstein optimal transport plan under certain conditions [40].



(a) (b) (c) (d) (e)

Figure 11: Pairwise GW transportation plan on Yelp, across BERT models. 5 of randomly chosen layer pairs are shown.

To complement Figure 2 on other fine-tuned BERT models on YELP, we also plot all the histograms of pairwise distances between two samples in a batch, across all layers for each of 5 models in Figure 12. Pre-trained models are publicly available models training on other datasets. Row b) to e) are the fine-tuned models on YELP, with different sparsity levels. As one can see, pretrained models do not have much differentiations across layers in the histograms, with maximal KL-divergence of 0.11 between histogram in consecutive layers. Fine tuned models, on the other hard, show larger KL-divergence values, in particular in later layers. For example, Layers 9 in the Dense BERT model contains KL distance of 1.58 from its previous layer. The results show that significant transformations in pairwise distances occur across layers and such distances would be captured by GW distances, as show in Figure 5 and Figure 14.

G Fine-tuning with Different Layers

Since the GW distance indicates significant changes occurs only at later layers in YELPS, we investigate performance of fine-tuning only partial layers from pretrained models, by freezing early layers during training and training only later layers alongside a classification layer (denoted as C) at the end. In Table 6, we can see that there is no significant performance differences between fine-tuning layer 8 to 12 and fine-tuning layer 9 to 12 (0.04% drop). On the other hand, the accuracy drops 6 times more by freezing layer 1 to 9, with 0.25%. Freezing layer 1 to 10 results 0.49% drop, and finally fine-tuning only 12 results 3.59% drop. These findings validate that the later layers are crucial for significant functional changes.

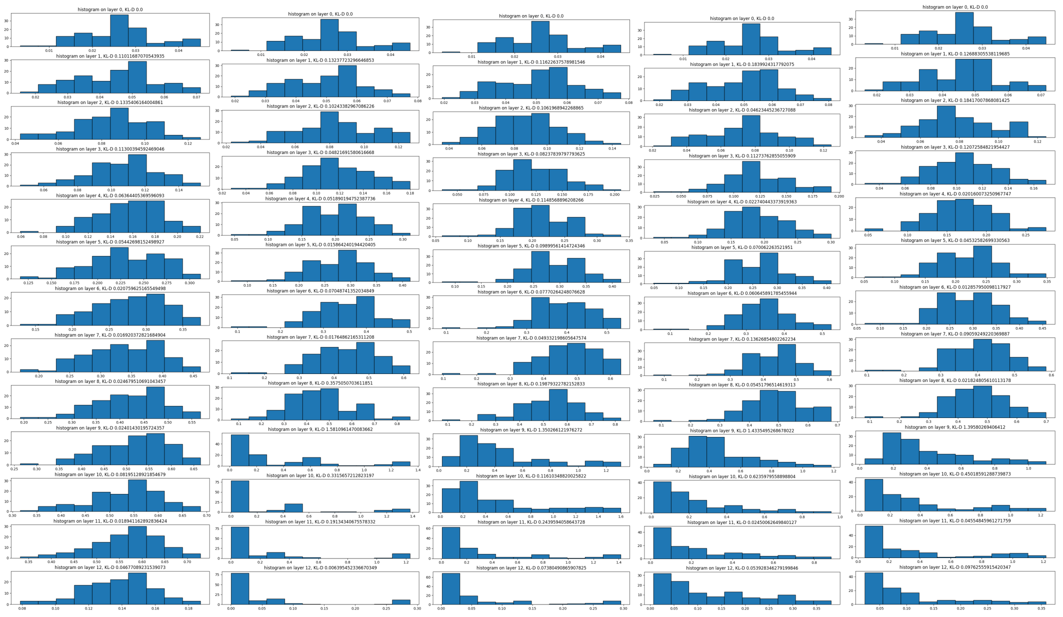
Table 6: Accuracy of fine-tuning partial layers in various BERT models. C denotes the classification layer on top of BERT models.

Fine-tune	All	8∼12 + C	9∼12 + C	10∼12 + C	11∼12 + C	Only 12 + C
Accuracy (%)	97.87	97.47	97.43	97.19	96.7	93.11

H Baseline Methods and Implementation Details

Besides the standard Euclidean, mutual information (MI) and cosine distances, we compare a few other baselines, as discussed below.

Wasserstein Distance [12]: We use the POT, python optimal transport library `pythonot` [13], with the algorithm proposed in [1].



(a) Pretrained (b) Dense (c) 25% Sparse (d) 70% Sparse (e) 95% Sparse

Figure 12: Histogram on pairwise distances on Yelp, across BERT models with different sparsity levels. a) is the pre-trained BERT and the rest are fine tuned BERT models with increasing sparsity levels: (b)densely fine-tuned, c) 25%, d) 70% and e) 95% sparsity.

Representational similarity metric (RSM) [22]: RSM compares two different spaces by using the L_2 norms on differences in inter-instances distances. This can be seen as approximation to GW using the fixed and identity transportation plan (i.e., the samples map to itself). We use existing implementation at: `https://github.com/mklabunde/llm_repsim/blob/main/llmcomp/measures/rsm_norm_difference.py`.

Representational Similarity Analysis (RSA) [22]: RSA is similar to RSM but use correlation instead of L_2-norm to compute the final distance. Implementation at: `https://github.com/mklabunde/llm_repsim/blob/main/llmcomp/measures/rsa.py`

Canonical Correlation analysis (CCA) [33]: CCA compute distances based on variances and covariances. Implementation at: `https://github.com/google/svcca/blob/master/cca_core.py`

Centered Kernel Alignment (CKA) [23]: CKA is based on normalized Hilbert-Schmidt Independence Criterion (HSIC). Implementation at: `https://github.com/mklabunde/llm_repsim/blob/main/llmcomp/measures/cka.py`

Multi-Scale Intrinsic Distance (MSID) [45]: MSID compute the intrinsic and multiple distance, and can be considered as a lower bound of the GW distance. Implementation at: `https://github.com/xgfs/imd/blob/master/msid/msid.py`. We have explored different hyperparameter settings with different neighbors k (5 or all batch data available) and number of iterations for SLQ, but results are all similar to the default parameter setting.

Augmented GW (AGW) [6]: AGW considers feature alignment in addition to sample alignment. Its overall objective can be seen as a penalized GW distance. Implementation at: `https://github.com/pinardemetci/AGW-AISTATS24/tree/main`.

For all methods, we use default parameter settings to obtain results in the paper. Note that RSM, RSA, CCA, MSID, and AGW, along with our proposed approach can handle different dimensions of inputs.

Gromo-Wasserstein Distance [12]: We use the POT, python optimal transport library `pythonot` [13]. We use the solver based on the conditional gradient [44].

I More Results on YELP

Due to the page limit, here we include baseline results on Yelp Datasets in Figure 13 and Figure 14.

Setup We now apply GW distance to real natural language processing tasks. We experiment on benchmark sentiment analysis datasets, Yelp reviews and Stanford Sentiment Treebank-v2 (SST2) from the GLUE NLP benchmark [49], with the goal to predict of the text has positive or negative sentiment, and analyze how different layers from fine-tuning BERT(-base) [8] models perform on these datasets. We use the pretrained BERT to generate 4 fine-tuned models, corresponding to a dense model and 3 sparse models with sparsity levels of 25%, 70% and 95% using a state-of-the-art structured pruning algorithm [9]. Sparsity are used to force models to condense information into the limited remaining weights, enabling us to examine potential links between this constraint and their structural similarity. Training details are in Appendix E. Due to the size of BERT models, we limit our analysis to comparing the final representations from each of the 12 transformer blocks, rather than examining all intermediate representations.

Results In Figure 5a, we see that the pre-trained BERT does not have major differences among blocks, which is not surprising given its accuracy on YELP is only 49.3% (roughly equivalent to random guessing). In Figures 5b to 5e, we see an interesting pattern emerge, revealing two-to-three major block structures in the (sparse) fine-tuned BERT models identified by our approach. The first major differences occur at block 9 and then the last three blocks (10, 11, 12) seem to form a distinct block. This seems to indicate that most of the function/task fitting occurs at these later blocks.

We compare the proposed GW distance with Euclidean, Cosine, and Wasserstein distance as baselines in Figure 13, on the same YELP dataset and with the same settings. Euclidean distance between two layers' outputs, shown in the first row of Figure 13, can be seen as the GW distance with a fixed identity-mapping transportation plan for each sample. This validates the low-valued diagonal elements. Off-diagonal elements show greater variation, and it is less obvious there are two distinct sub-groups within layers. The similar pattern is also observed with Cosine and Wasserstein distances, with similar strong diagonal pattern but more pronounced block structures than Euclidean distance. we also include 6 other baseline similarity measure in Figure 14. Overall, CKA produces also similar block structures to the proposed GW distance, though with greater variability within block structures. In contrast, other baselines fail to reveal such clear block structures.

J Cross Model Comparison

We can also use GW distance to compare layers from different BERT models. Shown in Figure 15, pretrained and densely fine-tuned BERT models exhibit different similarity measures when compared to fine-tuned BERT models with different levels of sparsity.

K SST2 Datasets

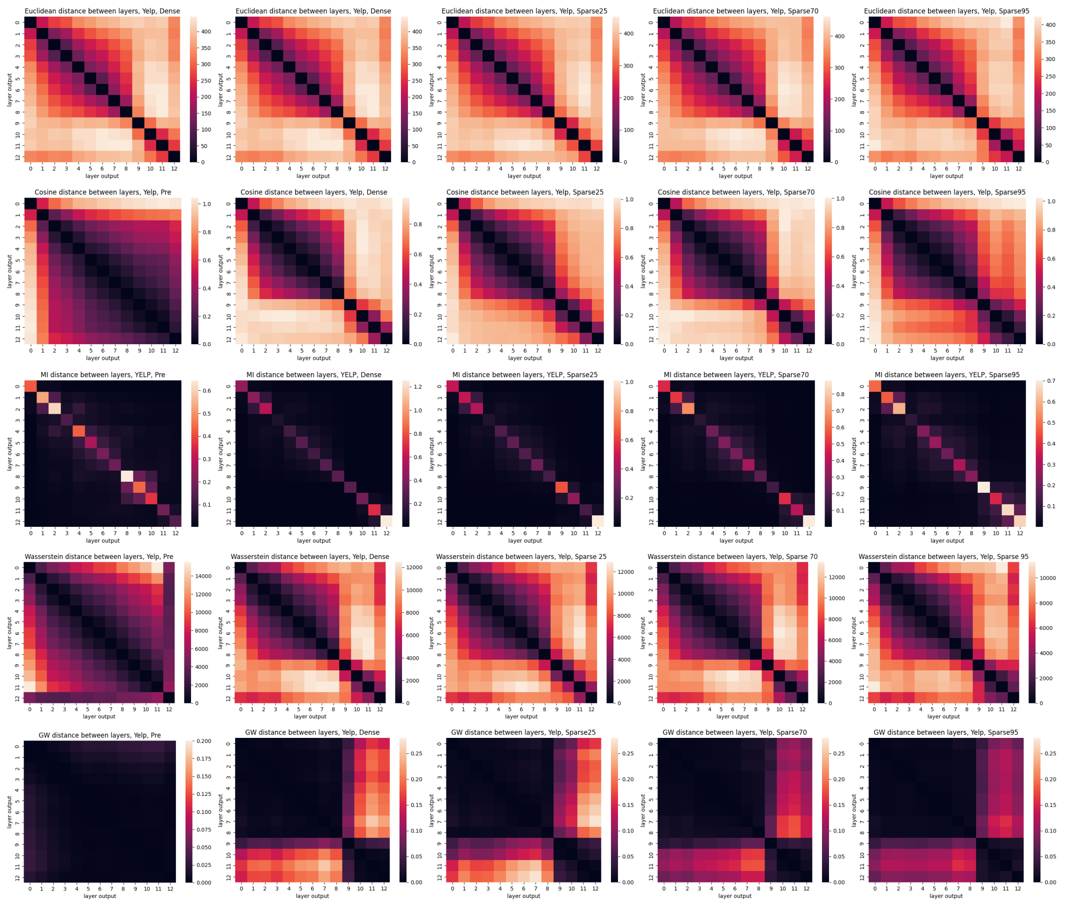
Besides YELP Datasets, we also tested the GW distance on SST2 dataset. Results on SST2 dataset are shown in Figure 16 again confirm there exist two-three different groups in terms of functional similarity. The first major difference is seen at layers 10 and 11, while layer 12 forms its own block. When sparsifying these models, lesser differences are observed in general as also seen on the YELP dataset. Other baselines provide less clarity on the division of sub-components.

More baselines are included in Figure 17, as they do not all fit into the one page. Overall, RSA and CKA identify block structures but with larger 2nd block.

L Model Pruning/Compressing

Another another potential application beside freezing-and-fine-tuning specific transformer blocks, we study the problem of model compress or pruning with the discovered layer groupings.

For each of desired block sizes, we take the original pre-trained BERT and only use the first $n = \{12, 8, 4, 2, 1, 0\}$ transformer blocks while discarding the rest. Note that $n = 12$ means we use all the transformer blocks, resulting the same BERT model. $n = 0$, on the other hand, means that we only use a (linear) classifier layer (after embedding layer) to predict the class label. The results are



(a) Pre-trained (b) Dense (c) 25% Sparse (d) 70% Sparse (e) 95% Sparse

Figure 13: Pairwise (layer) distances on Yelp, across different BERT models. Different Rows: *Euclidean, Cosine, mutual information (MI), Wasserstein, and the proposed GW distance*, from top to bottom. Different columns: first column is the pre-trained BERT and the rest are fine tuned BERT models with increasing sparsity (dense, 25%, 70% and 95% sparsity). As can be seen GW clearly demarcates the (functional) sub-network blocks.

shown in Table 7. As a reminder, GW distance suggest the last 4 blocks in YELP (see Figure 5) and the last 2 blocks in SST (see Figure 16) are mostly different, which is marked by star $(*)$ in the table. It shows that by using a limited number of layers, we can achieve similar performance with the full 12 block model, with 0.01% and 0.54% differences in YELP and SST, respectively. Using one fewer transformer block can risk much worse reduction of performance, with 0.10% and 8.60% differences (about 10 times worse performance reduction).

Table 7: Accuracy of pruning BERT with a smaller number of blocks on YELP and SST. N denotes the number of transformer blocks in the new BERT models.

Number of Transformer Blocks	12 (all)	8	4	2	1	0 (only classifier)
YELP	97.87	97.87	97.86*	97.76	97.11	60.3
SST	92.40	90.25	90.25	91.86*	83.26	50.92

M GW Distance with Different Random Seeds

Neural networks initialized with different random seeds can converge to distinct representations [27, 33, 23], even when their performance is comparable. To study the impact of initialization seeds on the learned representations, we train the same BERT model on YELP datasets with different seeds, with identical hyperparameters for a total of 27,000 iterations. As shown in Figure 18, while the

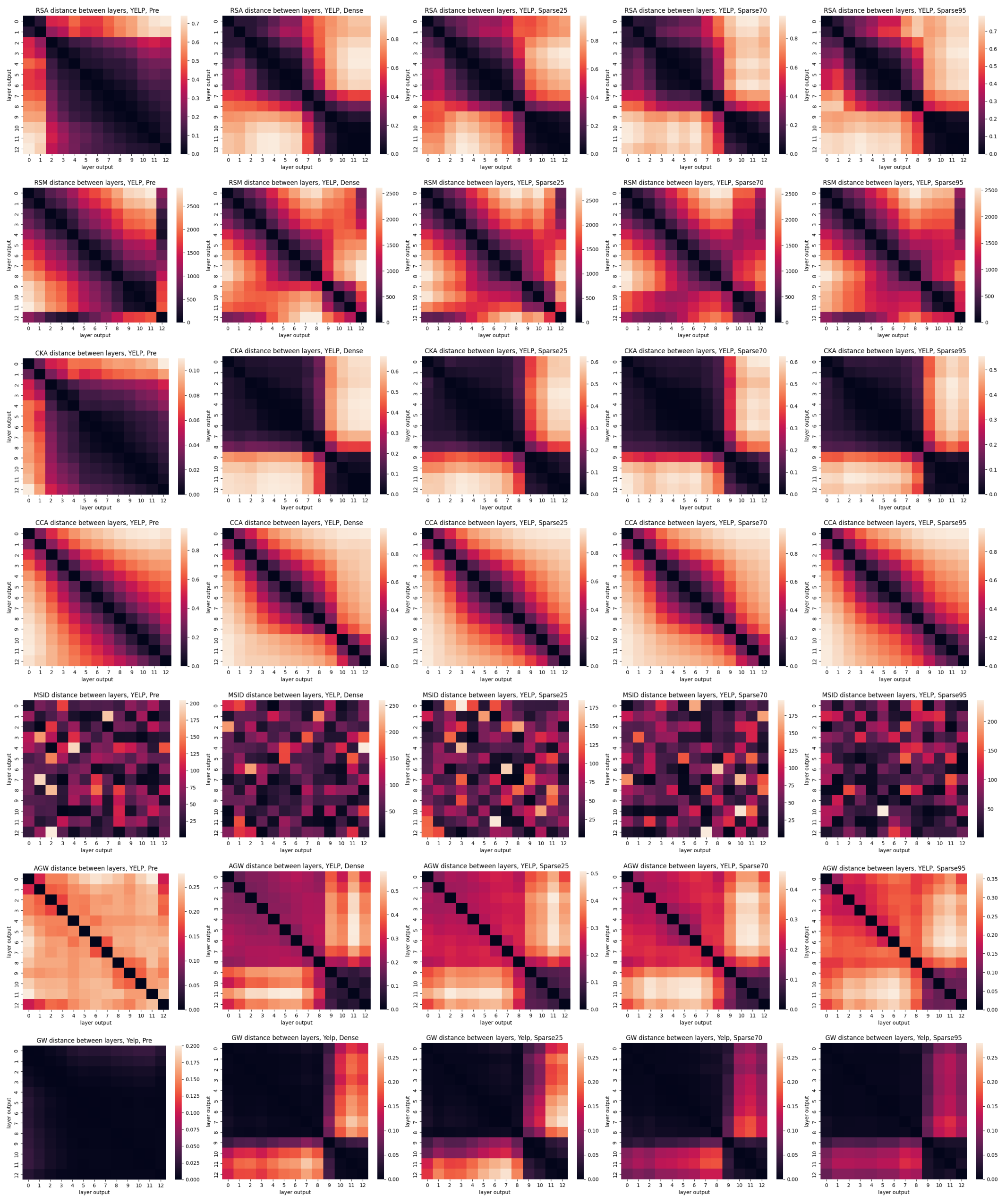

(a) Pre-trained (b) Dense (c) 25% Sparse (d) 70% Sparse (e) 95% Sparse

Figure 14: Pairwise (layer) distances on Yelp, across different BERT models. Different Rows: *RSA, RSM, CKA, CCA, MSID, AGW, and the proposed GW distance*, from top to bottom. Different columns: first column is the pre-trained BERT and the rest are fine tuned BERT models with increasing sparsity (dense, 25%, 70% and 95% sparsity). As one can be seen, GW clearly demarcates the (functional) sub-network blocks.

learned representations vary across seeds, but the general block structures remain consistent when analyzed using GW distances.

N Computer Vision Application: CIFAR-10 Datasets

In addition to the attention-based architectures, we also test our approach on ResNet9, a popular convolutional neural network architecture[16, 39]. We compare a randomly initialized ResNet9 and a trained model on CIFAR 10 image dataset CIFAR-10 [25], achieving 91.63% accuracy on the test data. CIFAR-10 dataset consists of 60000 32x32 color images in 10 image classes, with

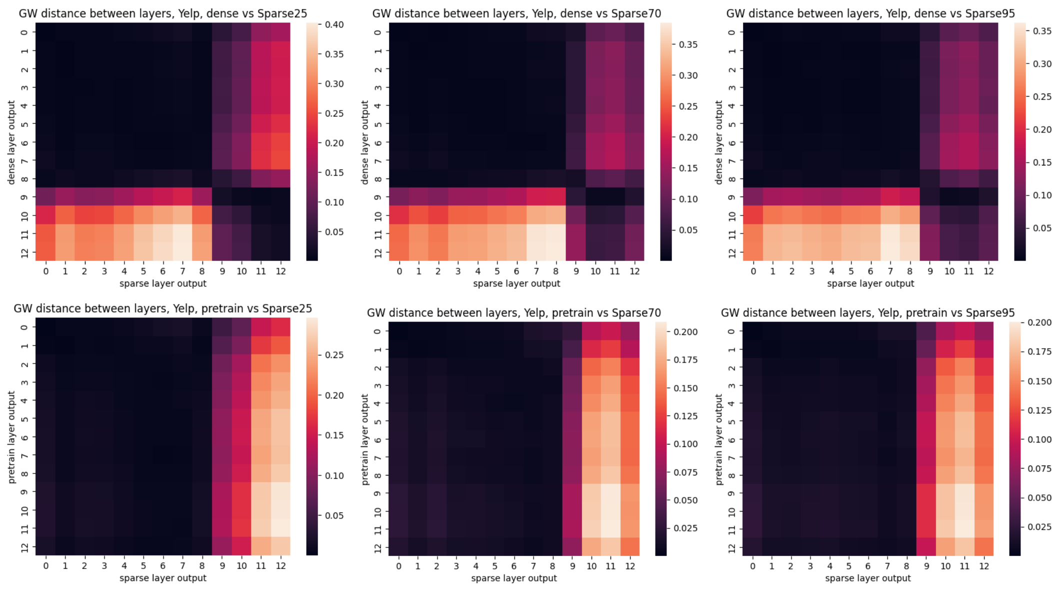


Figure 15: Pairwise distances on YELP dataset, of layers across two different BERT models. *TOP*: Densely fine-tuned BERT model vs fine-tuned BERT models with different sparsity levels. *Bottom*: Pretrained BERT model vs fine-tuned BERT models with different sparsity levels.

6000 images per class. There are 50000 training images and 10000 testing images. The classes are completely mutually exclusive. ResNet is a convolutional neural network with many residual connections. ResNet9 specifically contains 9 convolution layers, each followed by BatchNorm and ReLU activation. The exact details of the ResNet 9 is listed in Table 3.

We show the pairwise distance of all layers in consideration using all methods, that can handle difference dimensions of inputs, in Figure 19. The first column shows results from randomly initialized pre-trained models, and the second columns shows results from the trained ResNet. Pre-trained models generally do not show clear sub-network structures, while the trained models shows differences across layers. RSA, RSM, and CKA show progressive changes over the network layers, which is not too informative. AGW only shows the last a few layers contain significant changes, and MSID distance does not contain clear patterns. In comparison, GW distance shows clear division of 4 groups.

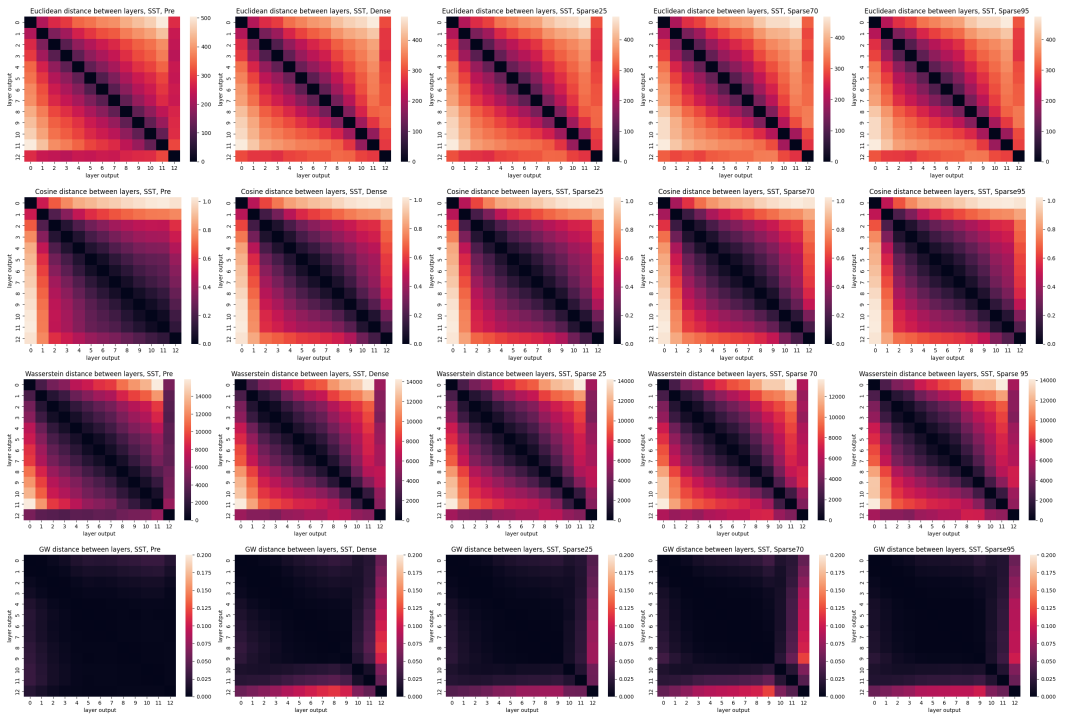

(a) Pretrained (b) Dense (c) 25% Sparse (d) 70% Sparse (e) 95% Sparse

Figure 16: Pairwise distances on SST dataset, across different BERT models. Different Rows: *Euclidean, Cosine, Wasserstein, and the proposed GW distances*, from top to bottom. Different columns: first column is the pre-trained BERT and the rest are fine tuned BERT models with increasing sparsity (dense, 25%, 70% and 95% sparsity).

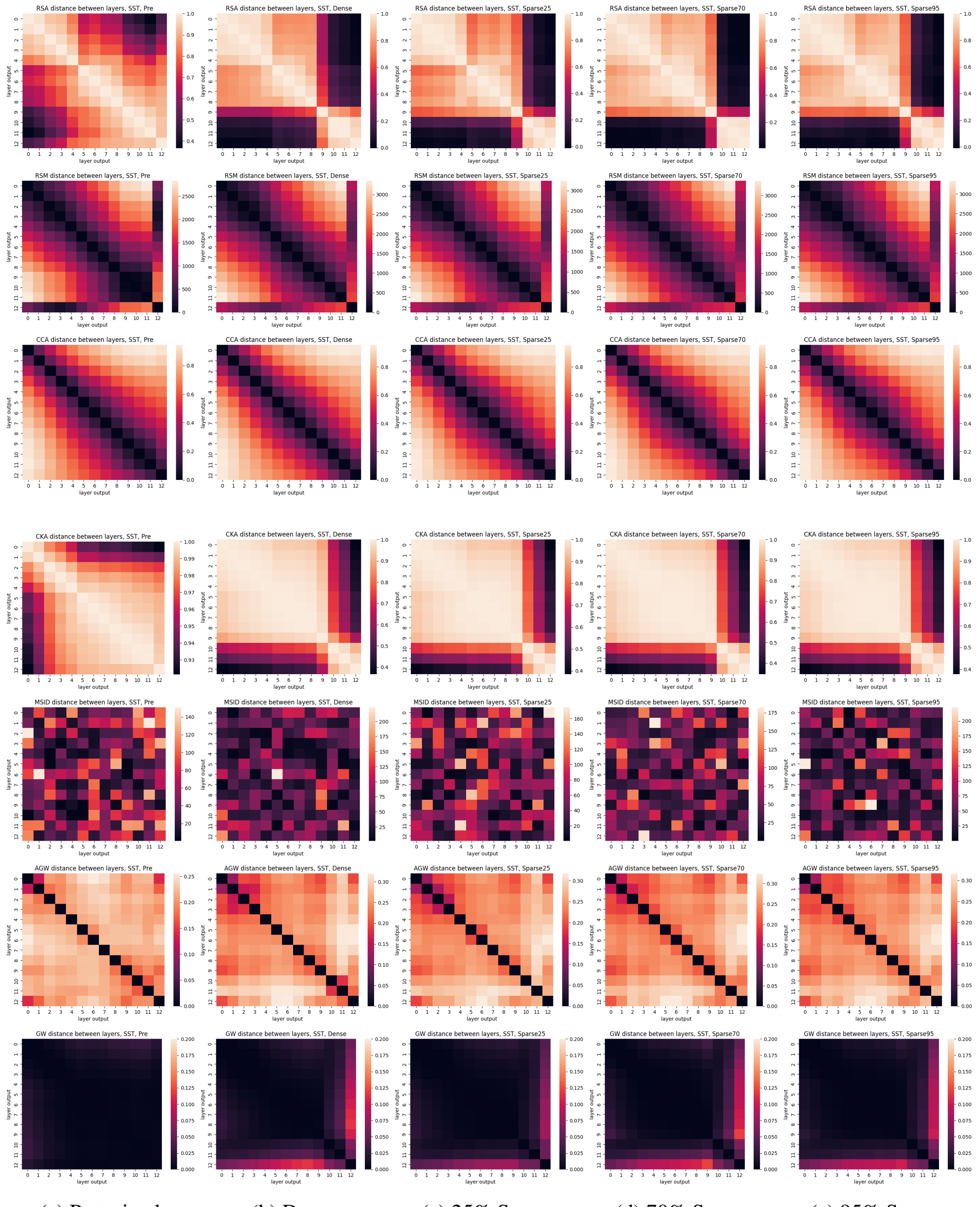
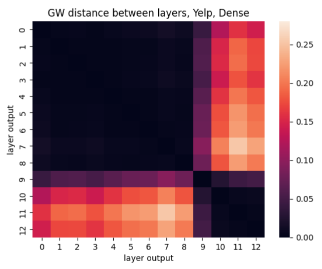
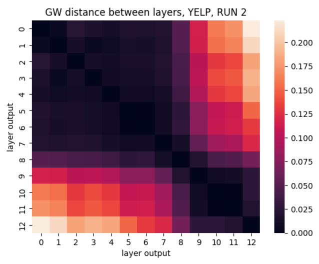

(a) Pretrained (b) Dense (c) 25% Sparse (d) 70% Sparse (e) 95% Sparse

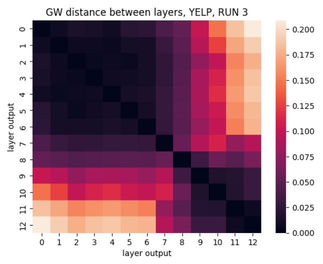
Figure 17: More Pairwise distances on SST dataset, across different BERT models. Different Rows: *RSA, RSM, CCA, CKA, MSID, AGW, and the proposed GW distance*, from top to bottom. Different columns: first column is the pre-trained BERT and the rest are fine tuned BERT models with increasing sparsity (dense, 25%, 70% and 95% sparsity).



(a) Dense



(b) 25% Sparse



(c) 70% Sparse

Figure 18: Pairwise GW (layer) distances on Yelp, across BERT models trained with 3 different random seeds. As one can be seen, the (functional) sub-network blocks stay rather consistent with different seeds even though there is some variations among the models.

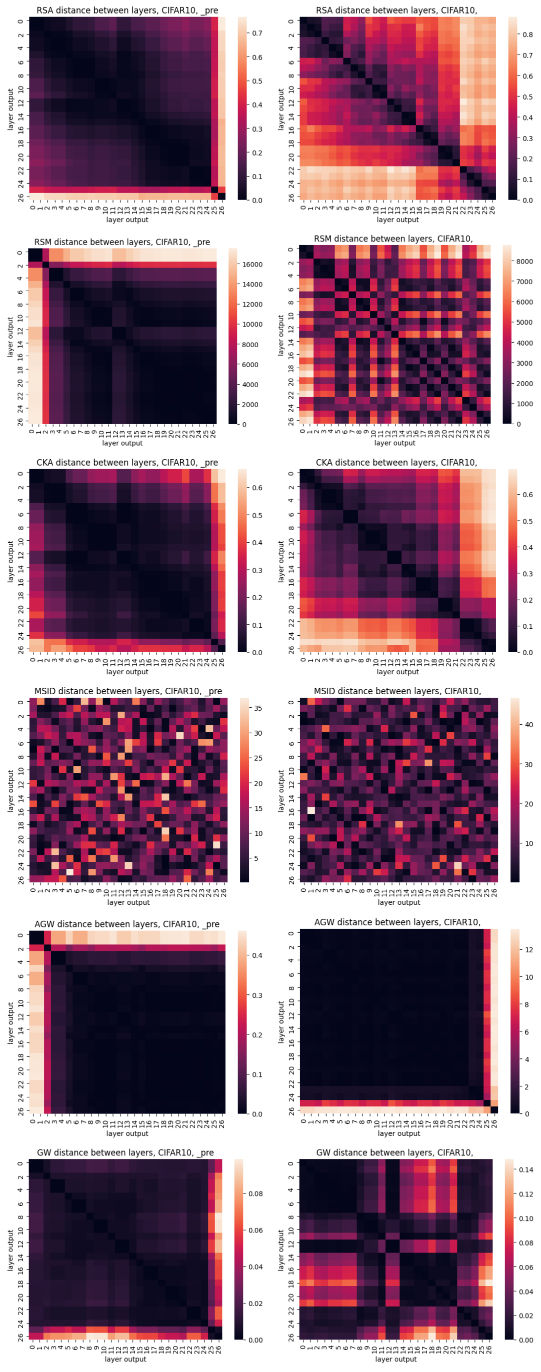


Figure 19: Pairwise (layer) distances on CIFAR-10, across different BERT models. Different Rows: *RSA, RSM, CKA, MSID, AGW, and the proposed GW distance*, from top to bottom. Different columns: the first column is the pre-trained ResNet9, and the 2nd column contains the fine tuned ResNet models.