Analysis of GPT-2

https://github.com/massimilianoviola/gpt2-unraveled/

Massimiliano Viola

October 2024

1 Introduction

The generative pre-trained transformer 2 (GPT-2) by OpenAI [1], when released in February 2019, represented a significant advancement in natural language processing technologies. It showcased the potential of large-scale transformer models to generate coherent and contextually rich text even in zero-shot task transfer and provided empirical validation for their performance scaling laws. GPT-2's ability to translate, summarize, and answer questions without task-specific training demonstrated the effectiveness of the simple next token prediction objective when the model is pre-trained with massive amounts of data. This report focuses on understanding the inner workings of GPT-2 from a geometric perspective, starting with an exploration of its embedding space and then examining the layer normalization, attention, and multi-layer perceptron (MLP) blocks. The goal is to provide insights into how these components process input tokens within a high-dimensional space, applying a series of geometric transformations to generate the next token based on the input context sequence.

2 Architecture Overview: The Residual Stream

The GPT-2 family of models (ranging from 124M to 1.5B parameters) closely follows the original transformer architecture proposed in *Attention is All You Need* [2], with a few key adjustments that nearly all modern transformer models have since adopted. First, a decoder-only transformer is used, processing input sequences in an autoregressive manner, predicting one token at a time while attending to all previous tokens. Second, the layer normalization position is changed from post-norm (applied between the residual blocks, after attention and fully connected layers) to pre-norm (applied inside the residual blocks, before the attention and fully connected layers). This modification makes gradients well-behaved at initialization, improving stability and requiring significantly less training time and hyper-parameter tuning [3]. Third, a final layer normalization is also added before the output linear layer, which uses the transpose of the word embedding matrix to map the hidden states back to vocabulary logits for token prediction.

The updated architecture can be seen in Fig. 1. In a nutshell, we start with a token embedding, followed by a series of identical blocks with residual connections, and finally a token unembedding. Each block consists of an attention layer, followed by an MLP layer, with normalization before each. Each attention layer consists of multiple heads, which operate in parallel. The GPT-2 variant we will analyze, the smallest version with 124M parameters [1], uses 12 layers with 12 heads each.

Although a transformer is often seen as a sequence of stacked layers with residual connections, similar to how layers are structured in a ResNet [4], it can be more insightfully understood through the concept of its *residual stream* [5]. This view emphasizes a continuous flow of information – a communication channel – where the input token representations are progressively transformed across layers, with input and output living in the same space and tokens being moved around via additions. Both the attention and MLP layers each read their input from the residual stream by performing a linear projection, then write their result to the residual stream by adding a linear projection back in. Different layers can process different information by storing it in different subspaces, and this is especially important in the

[1] https://huggingface.co/openai-community/gpt2

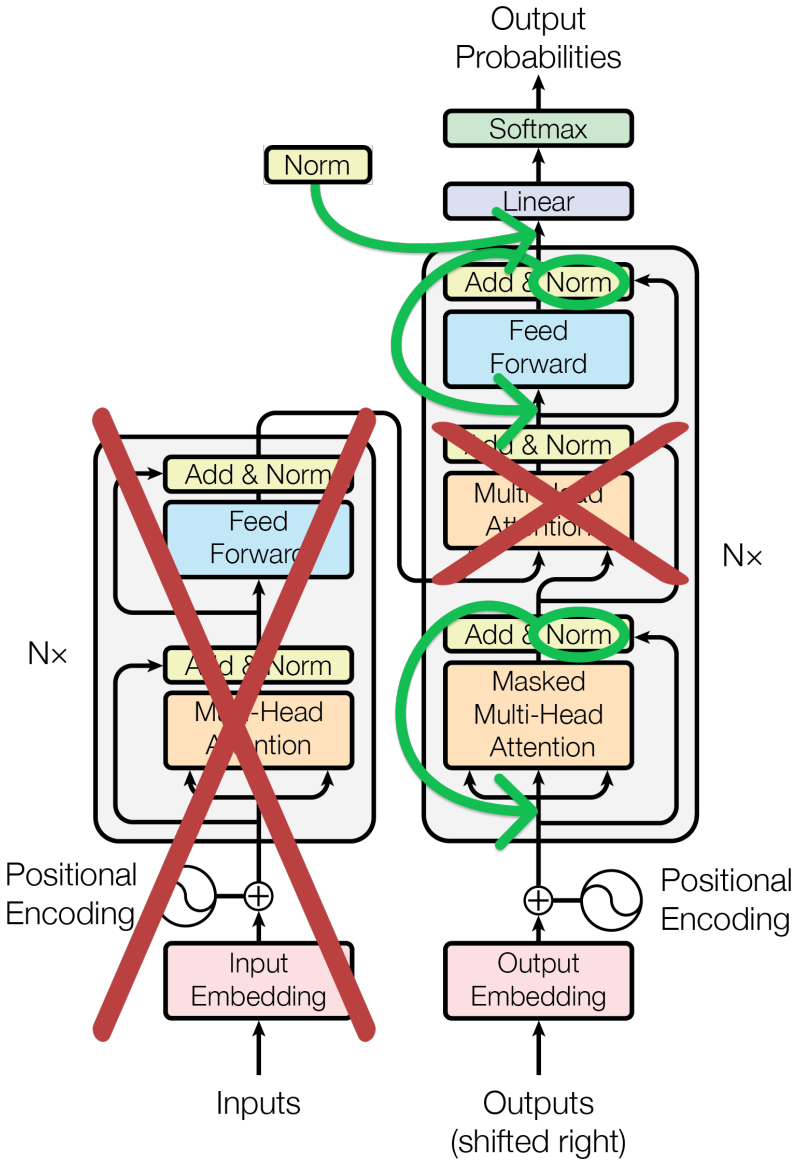

Figure 1: **Decoder-only transformer**, (masterfully) adapted from [2].

case of attention heads, since every individual head operates on comparatively small subspaces, likely not interacting with each other. This deeply linear structure comes with the property that there is no privileged basis in the residual stream, and we could rotate it by rotating all the matrices interacting with it, without changing the model behavior. In this setting, it makes sense to start by examining the properties of the embedding space used by GPT-2.

3 Embeddings Exploration

All token processing happens in a high-dimensional vector space, which is accessed via a token embedding matrix. This matrix maps one-hot vectors, representing specific tokens in the vocabulary, to dense vectors that correspond to their learned representations, capturing various semantic and syntactic properties. Our GPT-2 variant uses an embedding space of 768 dimensions and a vocabulary of 50257 elements. These tokens are derived using Byte-Pair Encoding (BPE), specifically a byte-level variant. Unlike traditional tokenizers that rely on Unicode characters, the GPT-2 tokenizer operates on bytes. This approach keeps the size of the base vocabulary small at 256 items, while still ensuring that all possible characters, including

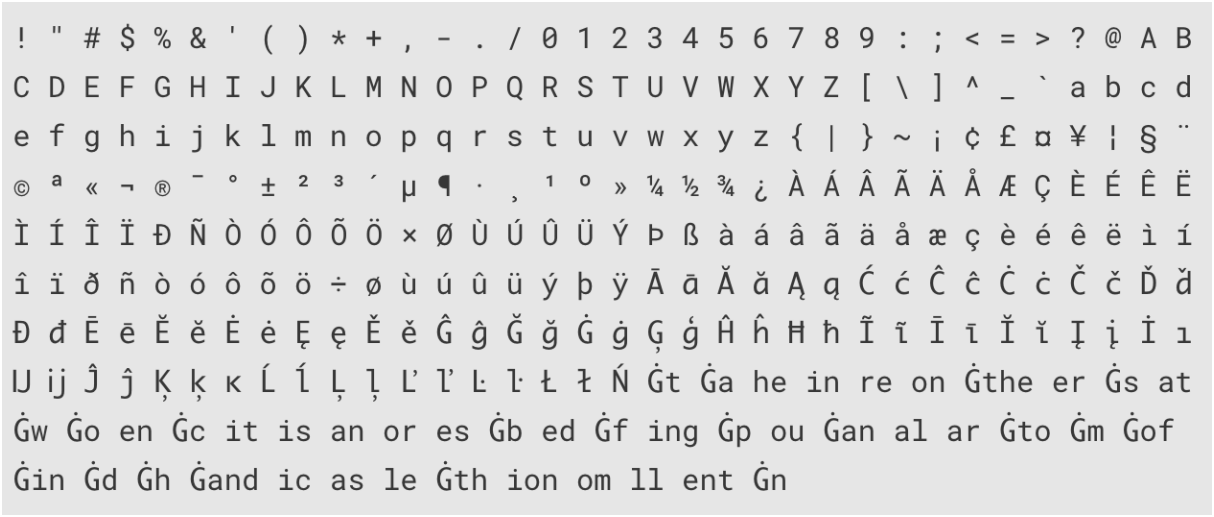

Figure 2: **The first 300 tokens** in the GPT-2 vocabulary.

rare or special symbols, are represented without needing an unknown token `<UNK>`. The vocabulary size of 50257 corresponds to the 256 bytes as base tokens, a special end-of-text token `<endoftext>`, and the symbols learned with 50000 BPE merges. Spaces are converted to a special character (the Ġ) in the tokenizer before BPE splitting to avoid digesting spaces. We can see the first 300 tokens in Fig. 2.

In addition to token embeddings, GPT-2 uses learned position embeddings, also of dimension 768, for the model to make sense of the sequential nature of language and capture relations at long distances. Each token is assigned a position embedding based on its location in the sequence, ranging from 0 to 1023, as the model can accommodate a maximum of 1024 tokens in its context window. This position embedding is then added to its corresponding token embedding before the input is fed to the first block. Let's proceed with a detailed analysis of each component involved in this embedding architecture, and their interaction. Lots of these analysis are repeated from [6].

By plotting a random chunk of the token embedding matrix in Fig. 3, we can notice an interesting phenomenon: the horizontal stripes are dimensions of the latent space where the token embeddings are systematically larger or smaller than the average. In other words, not all dimensions in the embedding space are centered around zero, and indeed, some dimensions exhibit significant offsets.

We can also generate histograms from the rows of the token embedding matrix, examining the mean

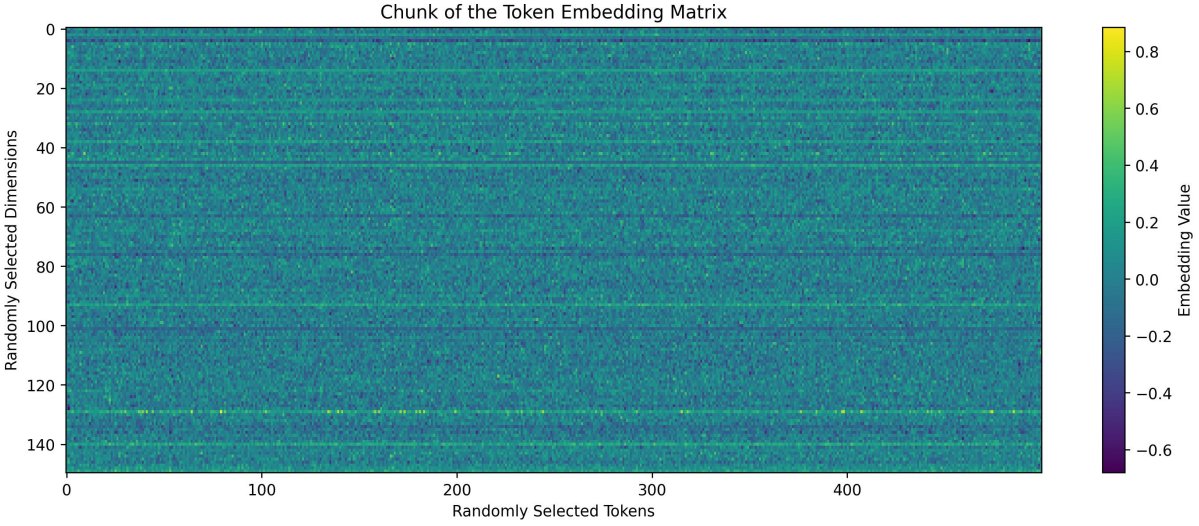

Figure 3: **Chunk of the Token Embedding matrix** for a random subset of tokens and dimensions.

and standard deviation of token values across different dimensions, visible in Fig. 4. We notice two things: the average mean in a dimension is most of the time zero, with a few dimensions having significantly larger absolute values as glimpsed in the matrix plot; the standard deviation also has a typical value of 0.125 in the majority of cases, with a few dimensions with a much smaller deviation from their mean visible in the heavy left-tail.

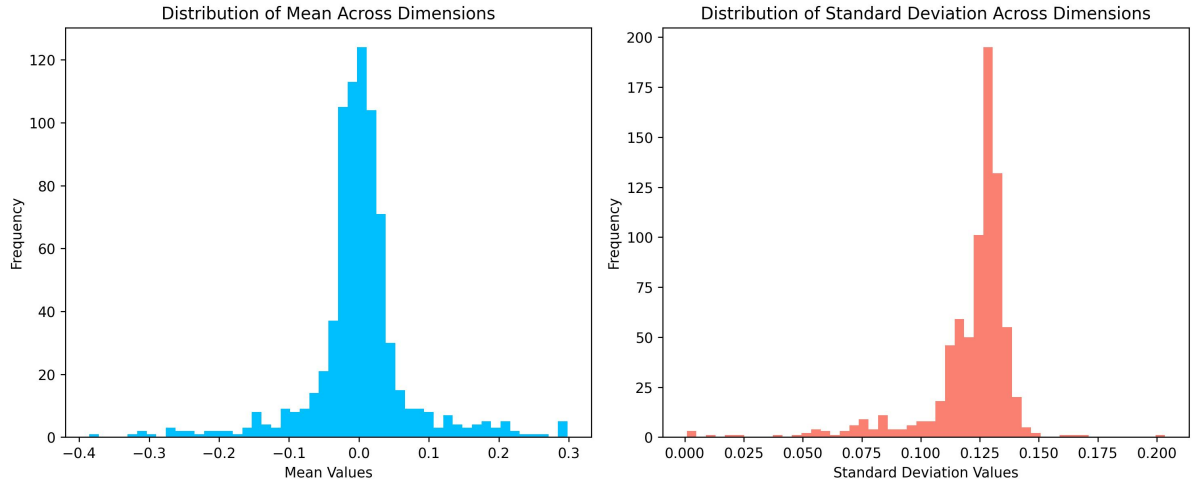

Figure 4: **Mean and Standard Deviation Distributions** across dimensions.

When we plot the histogram of the embedding values per token for a small subset of dimensions (Fig. 5 left), we observe indeed that many channel dimensions follow roughly the same distribution, while a few have smaller variance and larger means. Do these factors correlate? Apparently, this is the case: dimensions with more extreme mean values tend to be associated with the most extreme standard deviation values, suggesting some type of relationship (Fig. 5 left). We'll come back to this in a second.

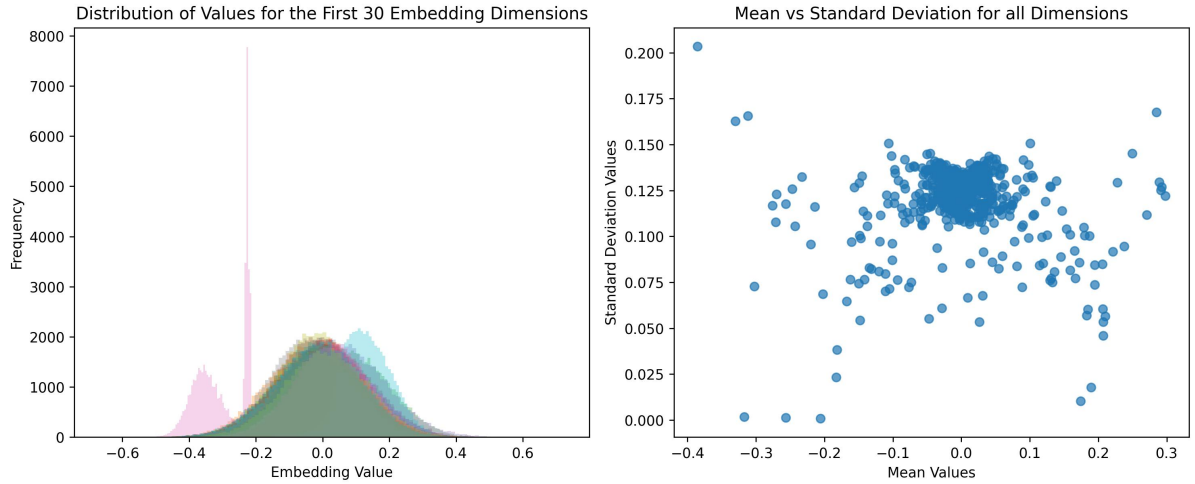

Figure 5: **Word Embedding Distributions** for the first 30 dimensions and scatter of mean and standard deviations.

Moving to the position embedding matrix, we notice some interesting patterns too, resembling sinusoidal waves that change signs at various frequencies across the context positions (Fig. 6). All these are learned, not fixed like in the original transformer paper [2], so it is definitely impressive. However, most dimensions seem to remain close to zero all the time, indicating that only a subset of dimensions actively contribute to encoding positional information while the rest stay relatively inactive. Nowadays, learned added positional embeddings with a fixed vector for each position, like those used in GPT-2, have largely

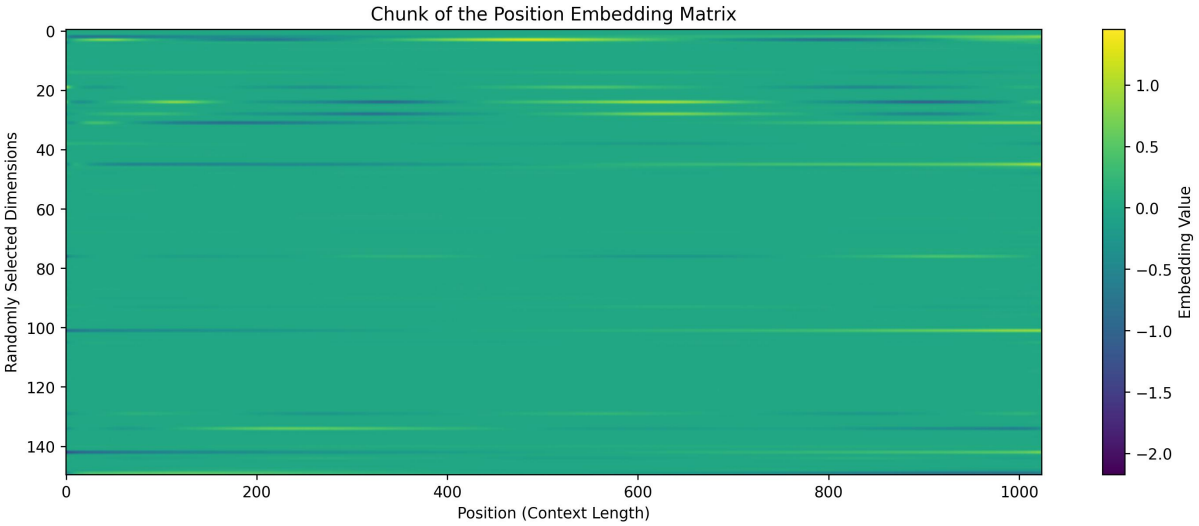

Figure 6: **Chunk of the Position Embedding matrix** for a random subset of dimensions.

been replaced by Rotary Positional Embeddings [7] (RoPE), applied directly into the attention mechanism by rotating the query and key vectors. RoPE emphasizes learning the relative distances between tokens, providing more flexibility in capturing long-range dependencies and allowing for the sequence length to be increased beyond its original limit. Looking at the signals as waves changing with the context position, we notice the first position is an outlier and that indeed they look like a bunch of sine waves (Fig. 7 left). Looking at their standard deviation per dimension, we can identify a very distinct cluster of dimensions where the position embeddings are doing something, with the rest of the dimension being near-zero all the time (Fig. 7 right). This is a strong indication of a split in the embedding dimensions, where certain dimensions may primarily encode positional information, while others focus more on capturing the semantic meaning of words.

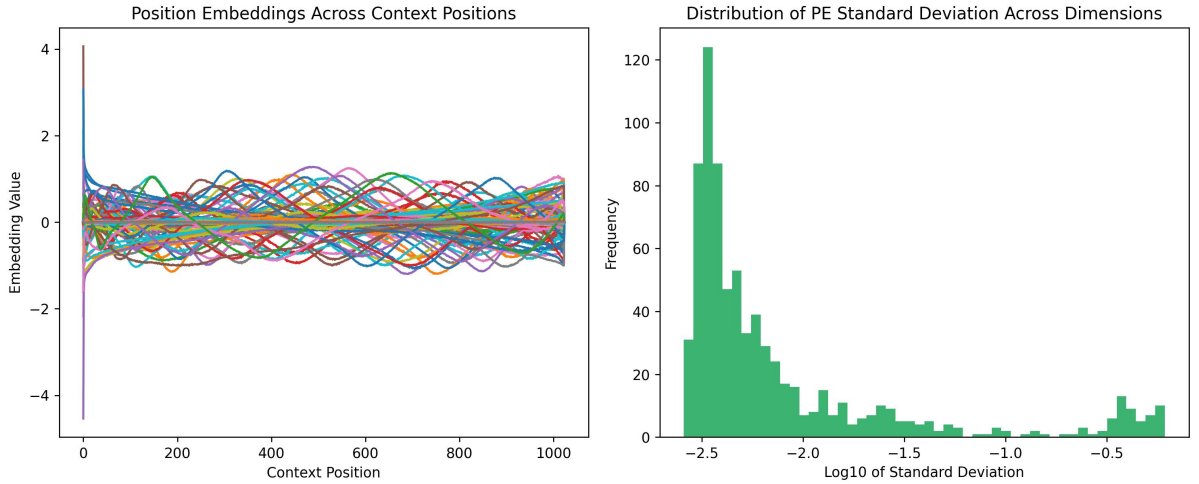

Figure 7: **Position Embedding Distributions** over context position and their standard deviations.

We can observe this relationship by plotting the mean and standard deviation for both word and position embeddings in a scatter plot in Fig. 8. On the x-axis, we place the standard deviation of the position embeddings, with lower values on the left indicating not much position embedding activity and higher values on the right reflecting a lot of position embedding activity. Several things can be read from the plot: (i) First, the low standard deviation (green) and high mean (blue) token embeddings are mostly in the top right cluster where dimensions have high means and standard deviations for position

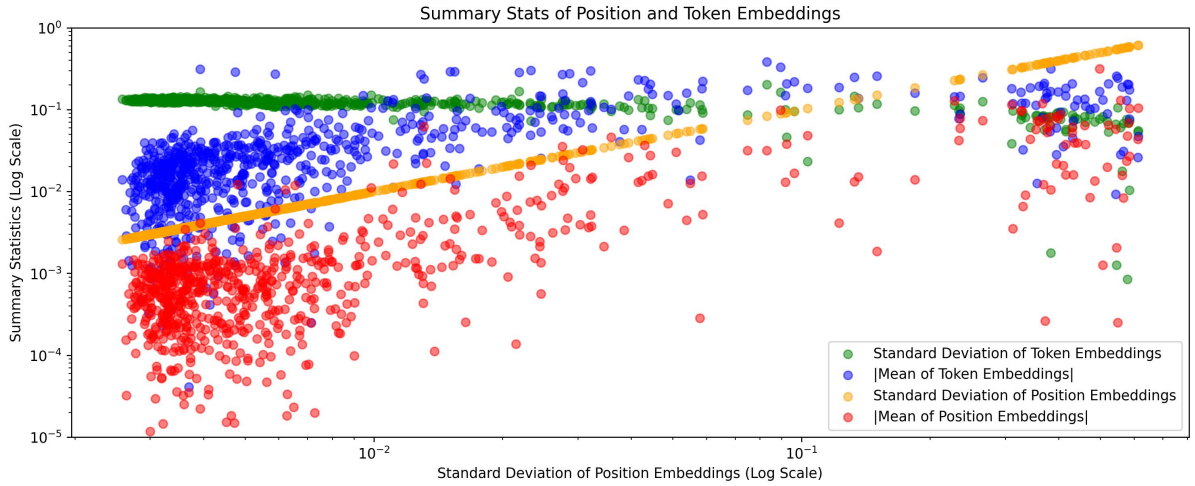

Figure 8: **Position Embedding Activity** against Summary Statistics for word and position embeddings.

embeddings; (ii) Second, there is a negative correlation (green) between token and position standard deviations, meaning the two typically don't have large variations at the same time; (iii) Third, position embeddings are constantly centered near zero and their means are a bit smaller than their standard deviations (red < orange), whereas normal word token embeddings are centered near zero (blue < green) but the weird ones are not (blue > green).

We can also plot pairs of dimensions and see this in practice in Fig. 9, with the token (black) and position embeddings (red). There are 1024 red points in each scatter plot, one for each position, and 50257 black, one for each token. We can see in some cases that token embeddings are significantly different from zero and very active, while position embeddings are not active, or vice versa token embeddings are almost inactive and we have a lot of position embedding variation instead. It is curious to see some strong correlation in one case, suggesting that the true intrinsic dimension of the token embedding cloud might not be the full 768 dimensions.

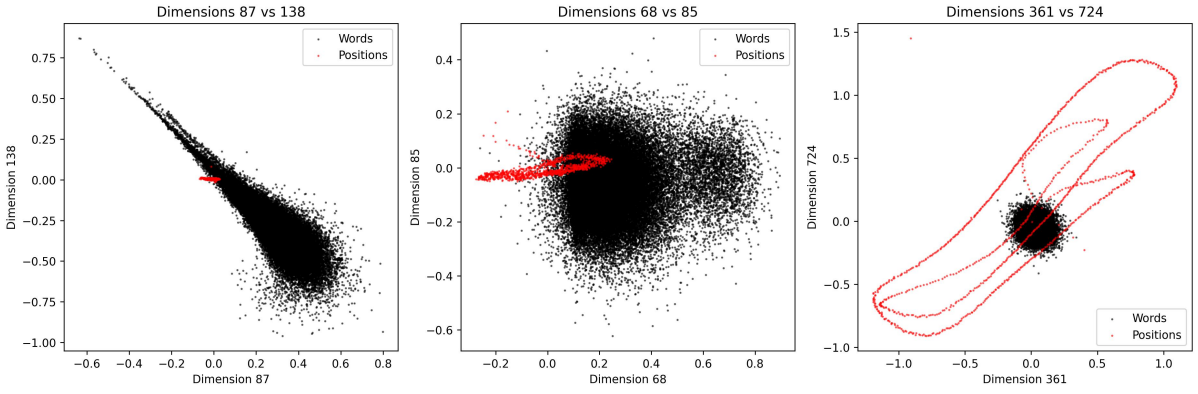

Figure 9: **Word and Position Embeddings Scatter** for various pairs of dimensions, exhibiting different behaviours.

To confirm that, while also analyzing the interaction between word and position embeddings further, we apply Principal Component Analysis (PCA) for dimensionality reduction. Starting with token embeddings, in Fig. 10 we can see that around 550 components are enough to explain 90% of the variance of the entire word embedding cloud, but when we restrict our interest to random subsets of 1000 neighboring tokens, the 90% variance threshold drops to 250 principal components. This suggests that tokens within a localized region of the embedding space, likely with a very similar semantic sense, live in a much lower dimensional space than 550 and 768, and probably they tend to form clusters or manifolds. In contrast,

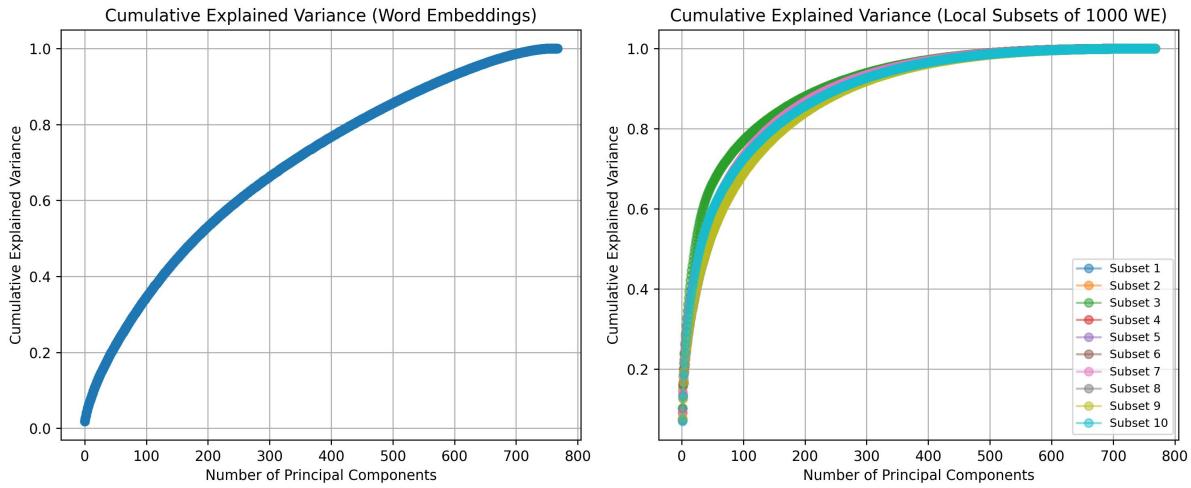

Figure 10: **PCA Analysis of Word Embeddings** for the entire vocabulary and random subsets of 1000 neighbors.

the entire embedding space contains a wider variety of tokens with more complex relationships, requiring more components to capture the diversity of the whole vocabulary.

When we do the same for position embeddings, we find that only 3 components are enough to capture 90% of the variability there, so basically position embeddings are extremely low dimensional and all the other dimensions are a constant linear transformation of the three main PCA components. Projected in 3D, we get a spiral [8], with most of the tokens living on it except positions 0 and 1023 which are clear outliers. It remains unexplained to me why this helical structure is the most natural way for GPT-2 to express position.

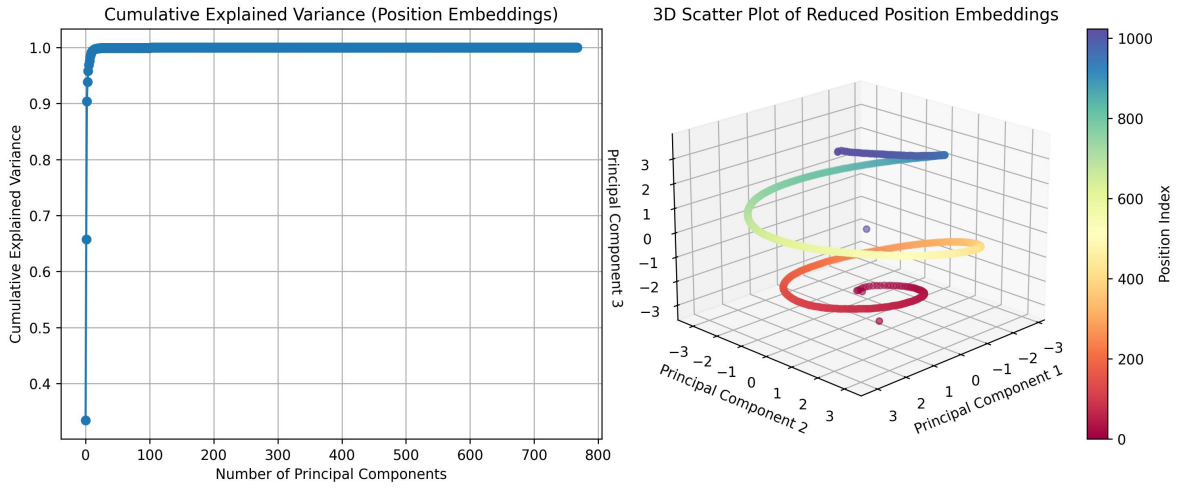

Figure 11: **PCA on Position Embeddings** and their projection in 3D.

It is pretty interesting to see the cosine similarities between rows of the positional embedding matrix creating a beautiful sine wave pattern, explained by the helical structure (Fig. 12 left). Finally, if we project in the 3 main PCA components for position embeddings also all the word embeddings, we clearly see the two are pretty much disjoint, or at least there is not much activity in these dimensions from the word embeddings.

One way to interpret it is that there is no correlation or dependency between their meaning and their position within sequences, so word embeddings contain intrinsic semantic properties that are not influenced by their position. Furthermore, if we believe that neural networks store information along

directions, one way to think of this is that the space is reserving 3 dimensions for positional information, and 765 for the semantic content of the tokens.

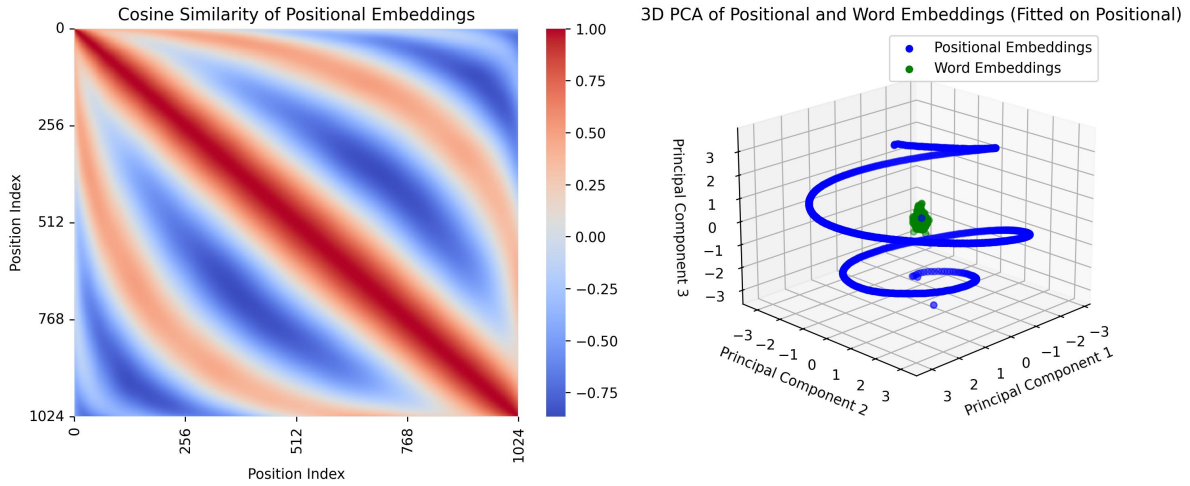

Figure 12: **Cosine Similarity Between Position Embeddings and Combined PCA** analysis of word and position embeddings.

4 Layers Exploration

Now that we understand the space we operate in much better, we can try to comprehend from a geometric perspective what happens in a layer normalization, in an attention layer, and in an MLP. What follows is largely inspired by [9], [8], [6], and [10].

4.1 Layer Normalization

We begin with the layer normalization function, which can be shown to project d-dimensional input features onto the surface of a $(d1)$ dimensional hyper-sphere. As introduced in [11], layer normalization is expressed in terms of the mean μ and standard deviation σ of an input feature vector $X \in \mathbb{R}^d$:

$$\text{LayerNorm}(X) = \frac{X - \mu}{\sigma} \tag{1}$$

Both the mean and standard deviation are taken along the feature dimension d, such that:

$$\begin{aligned} \mu &= \frac{1}{d}\sum_{i}^{d} x_i \\ \sigma &= \sqrt{\frac{1}{d}\sum_{i}^{d}(x_i - \mu)^2} \end{aligned} \tag{2}$$

As noted in [12], if we think of the numerator in Eq. (1) as an operation between X and the vector $\boldsymbol{\mu} = [\mu, \mu, \ldots, \mu] \in \mathbb{R}^d$, the resulting vector $X - \boldsymbol{\mu}$ is orthogonal to the $\vec{\mathbb{1}} \in \mathbb{R}^d$ vector. Therefore, layer normalization can be interpreted as a projection of X onto the hyperplane defined by the normal vector $\vec{\mathbb{1}}$, consisting of all points whose components sum to zero. Furthermore, the division by σ acts as a

scaling factor that modifies the norm of $(X - \boldsymbol{\mu})$ to be $\sqrt{d}$:

$$\begin{aligned}\sigma &= \sqrt{\frac{1}{d}\sum_{i}^{d}(x_i - \mu)^2} \\ &= \frac{1}{\sqrt{d}}\sqrt{\sum_{i}^{d}(x_i - \mu)^2} \\ &= \frac{1}{\sqrt{d}}\|X - \boldsymbol{\mu}\|_2\end{aligned} \tag{3}$$

Putting all together, layer normalization projects a vector $X \in \mathbb{R}^d$ to the hyperplane perpendicular to $\vec{\mathbb{1}} \in \mathbb{R}^d$, and normalizes the projection such that it lies on the surface of a $d1$ dimensional hyper-sphere $\mathcal{H}_S$ of radius $\sqrt{d}$. A visualization of this process for $d = 3$ is shown in Fig. 13, where all data points lie within the plane perpendicular to the $\vec{\mathbb{1}}$ vector. In practice, layer normalization includes two additional

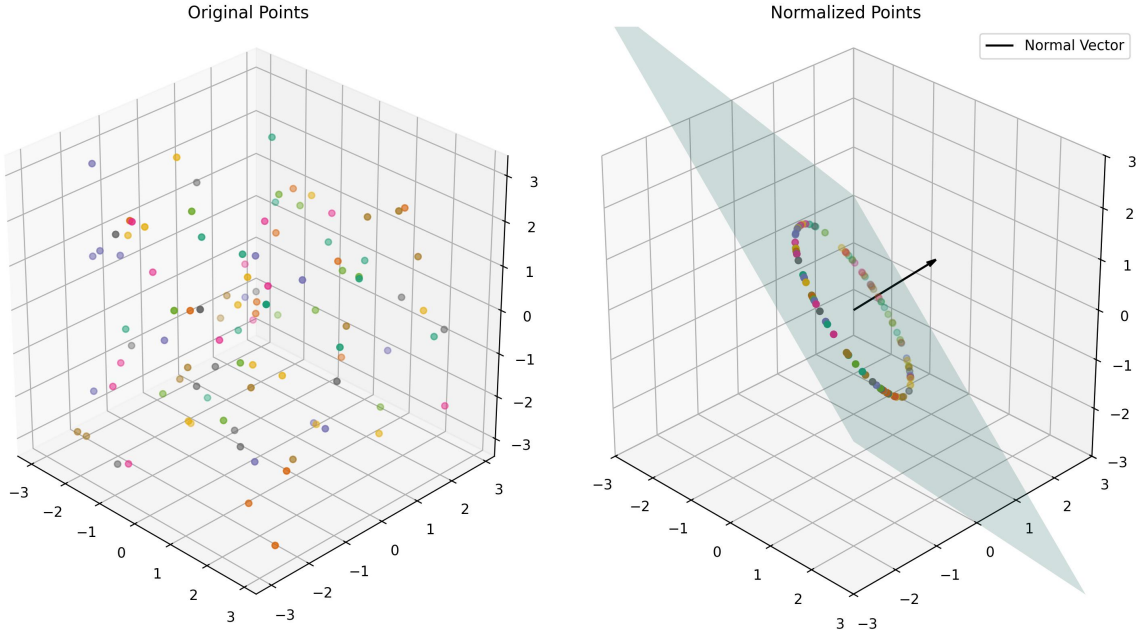

Figure 13: **Layer Normalization on 3D Data** with original points on the left and after layer normalization (default parameters) on the right.

learnable affine transform parameters: a scaling factor γ (initialized as 1) and a bias term β (initialized as 0), both vectors of size d. The vector γ scales each coordinate axis of $\mathbb{R}^d$ independently, transforming the hyper-sphere into a hyper-ellipsoid, and the bias term β shifts the center of said ellipsoid away from the origin. The highly non-linear behavior that arises from this operation might be overlooked at first, but there is evidence that it is crucial for the attention mechanism to attend to all keys equally. When we plot the γ weights, responsible for the rescaling of each dimension, against position embedding activity across all 12 layers for both layer normalization before attentions (Fig. 14) and before MLPs (Fig. 15), we notice an interesting pattern. In the first layer, the model seems to reduce the influence of dimensions overly affected by positional embeddings scaling them down, while the later layers mostly mess with a few outlier points. This adjustment helps balance positional information and semantic content, preventing positional data from overwhelming the model. If we look closer at these outliers, they are recurring dimensions that appear over and over, so it looks like GPT-2 is paying attention to that set of dimensions more than others by scaling them up every time. From deeper analysis [6], some of these like dimension 138 seem to encode information about the unigram frequency of tokens in the training data and norm of embedding vectors, crucial for the model to focus on as first guesses for the importance of a token.

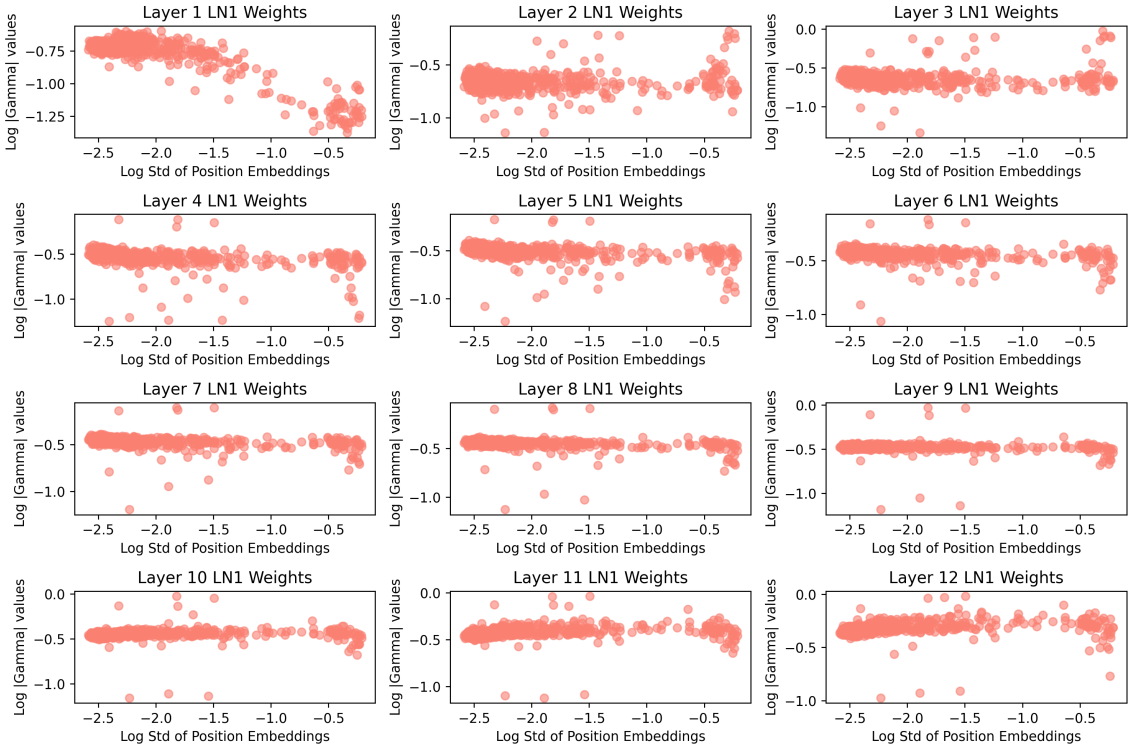

Figure 14: **Layer Normalization Gammas Before Attentions** by position embedding activity.

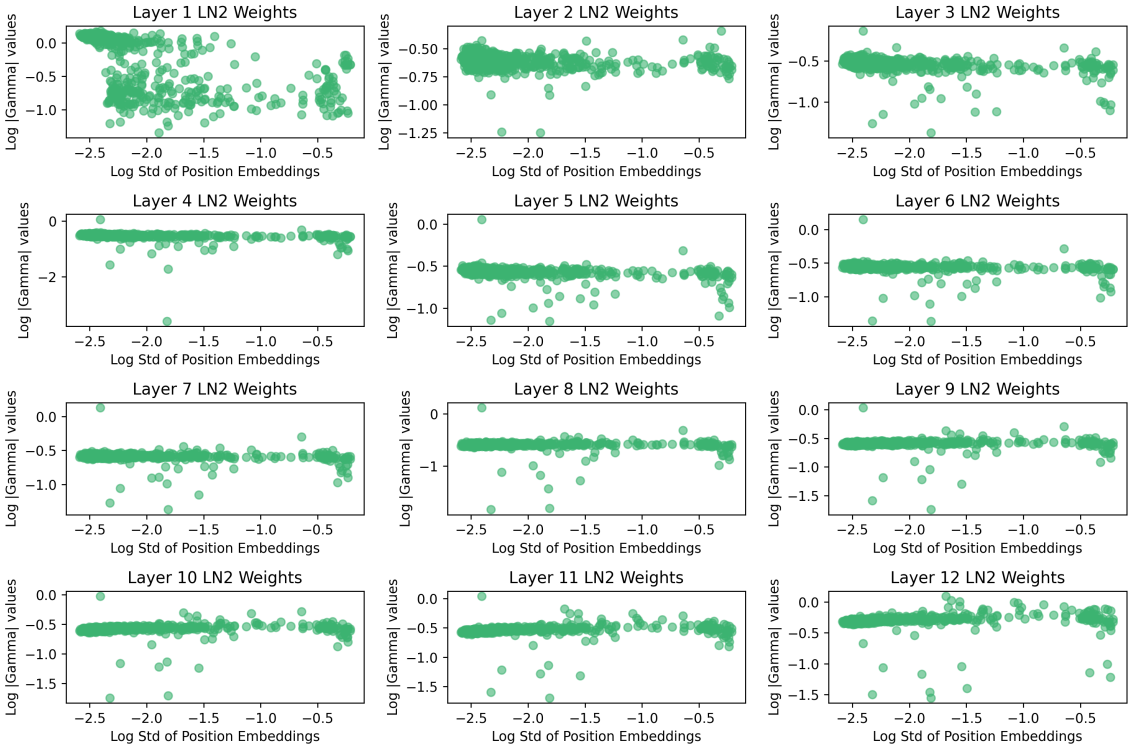

Figure 15: **Layer Normalization Gammas Before MLPs** by position embedding activity.

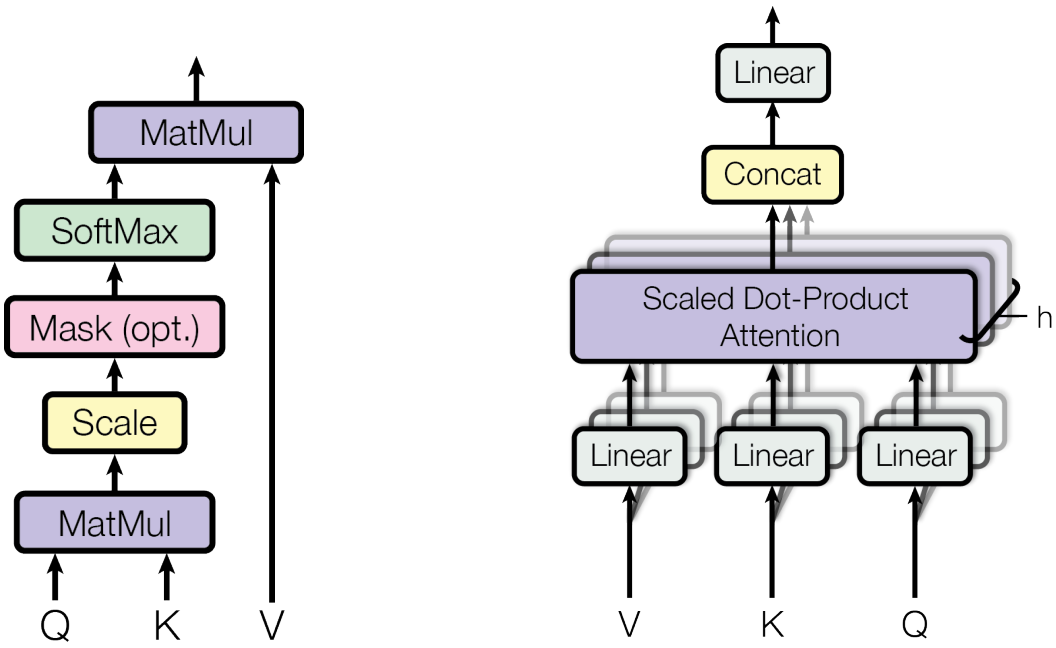

Figure 16: **Single and Multi-Head Attention** structure, taken from [2].

4.2 Attention

Now, let's analyze what happens inside an attention block. As we said already, there are 12 identical blocks in our GPT-2 variant, and the two layer normalizations in each block force the data to be constrained on the surface of a hyper-sphere, potentially unique to each layer. However, thanks to the residual nature of the model, all intermediate layer representations share the same vector space and thus are projecting features onto the same hyper-sphere $\mathcal{H}_S$ at least at the beginning. Additionally, since layer normalization is also applied before the classification softmax, the model maximizes the dot product between points living on the hypersphere $\mathcal{H}_S$ and the word embeddings in $\mathbb{R}^d$. This process, much facilitated in high dimensions, links the normalized representations to word semantics in the embedding space.

Although multi-head attention is used, we'll first revisit the self-attention module [2] with a single head. For a given input sequence $X \in \mathbb{R}^{s\times d}$ of length s, the self-attention mechanism (Fig. 16 left) is defined as follows:

$$\text{SelfAttention}\left(X, W_Q, W_K, W_V\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{4}$$

where

$$\begin{aligned} Q &= XW_Q \\ K &= XW_K \\ V &= XW_V \end{aligned} \tag{5}$$

with $W_Q \in \mathbb{R}^{d\times k}, W_K \in \mathbb{R}^{d\times k}$ projection matrices from the original model dimension d to an intermediate dimension k, and $W_V \in \mathbb{R}^{d\times d}$ square matrix to return vectors with the same dimensionality as the input. In practice, it is convenient to factorize W_V as a product of two smaller matrices of dimensionality $\mathbb{R}^{d\times k}$, so that the number of parameters in W_V matches the sum of those in W_Q and W_K. This low-rank transformation performs the update in two steps, first mapping the embedding vectors down to a much smaller dimension and then mapping them back up to the original dimension. This approximation is especially relevant in the context of multi-head attention.

For multi-head attention (Fig. 16 right), multiple projection matrices W_Q^i, W_K^i, W_V^i are considered, one for each head $i \in [1, \ldots, h]$ with h being the number of heads. In GPT-2 $h = 12$ parallel attention layers are used. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. The value dimension is commonly set equal to k and an output matrix $W_O \in \mathbb{R}^{hk\times d}$ is used to combine information from all heads, mimicking the two-step

transformation explained above with intermediate dimensionality of $k = d/h = 768/12 = 64$ in our case:

$$\begin{aligned} \text{MultiHead}(X) &= \text{Concat}\left([\text{head}_1, \ldots, \text{head}_h]\right) W_O \\ \text{where head}_i &= \text{SelfAttention}\left(X, W_Q^i, W_K^i, W_V^i\right) \in \mathbb{R}^{s \times k} \end{aligned} \tag{6}$$

The fundamental action of attention heads is moving information. Keeping in mind the residual stream view of our transformer, they read information from the residual stream of one token and write it to the residual stream of another token.

So, what is going on here, wrapped into a convoluted but very parallelizable matrix multiplication form, is h fully independent attention patterns proposing h different updates to each vector in a certain position between 1 and s in the context $X \in \mathbb{R}^{s \times d}$. The embedding in that position takes the sum of all those proposed changes and adds the result to the original embedding, refining the embeddings with the contextual information coming from many different ways of processing it.

Attention is very hard to interpret from a geometric perspective like we did for layer normalization, so we just see some interesting pattern for now. Plotting a attention matrices for heads at different layers in GPT-2 for an example sentence in Fig. 17, we notice that a lot of focus is always on the first token. If we looked at all heads in a layer, however, we will very likely spot some more active ones that display

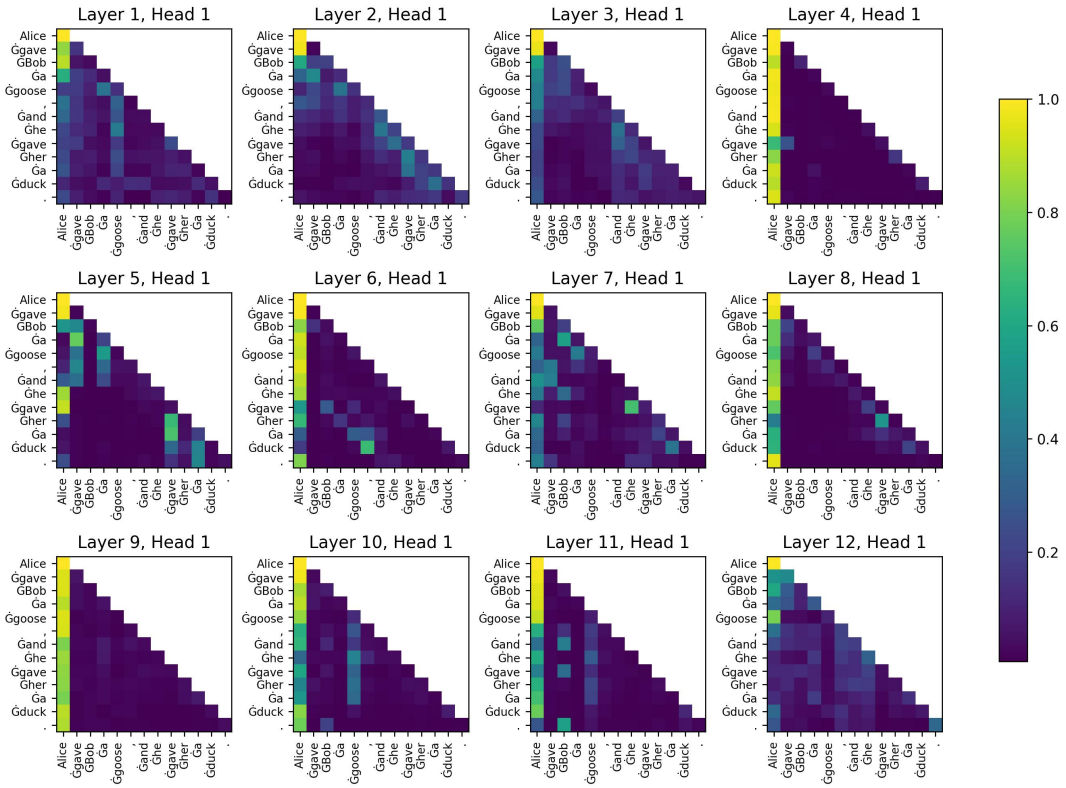

Figure 17: **Visualization of attention matrices** for the first head of each layer.

a small number of highly defined relations, focusing on the current token or targeting a few others, as pointed out in [13].

We can also see that although very prominent, attention to the first token by subsequent ones decays with context length layer by layer, while the attention that the last token pays to the first one remains very high (Fig. 18), also for longer sequences.

4.3 Multi-layer Perceptron

The MLP is conceptually very simple, processing each vector independently of the others, by applying two linear transformations with a non-linear activation function in between and proposing the update to

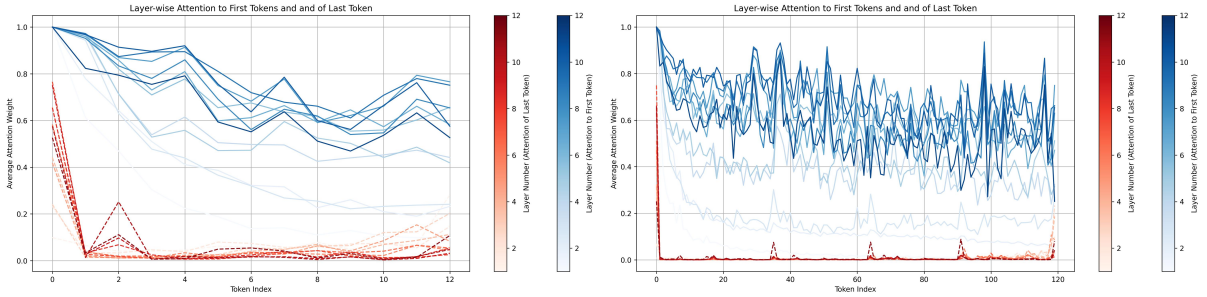

Figure 18: **Attention weights to the first position (blue) and of the last one (red)** averaged over all heads, layer by layer.

the residual stream. The hidden size is typically four times the base embedding size, 3072 for GPT-2, providing a huge number of dimensions to store data.

Accounting for the majority of the parameters in the transformer, and with no exchange of information between tokens happening at this stage, there is evidence [14] that it is where the model stores general knowledge and acts as a fact lookup table queried by the contextualized inputs. In other words, MLPs are linear representations of attributes, and extraction of these facts seems to be refined layer by layer in parallel to the embedding being more and more contextualized.

Rows of the first matrix scaling up dimensions can be thought of as directions in this larger space, the activation of each neuron tells how much a given input vector aligns with a certain direction, and columns of the second one tell what will be added to the result if that neuron is active. Individual neurons very rarely represent facts and clean features, and this finds evidence in the idea of superposition [15], also explaining in part why scaling LLMs works so well, and why the number of facts learned scales superlinearly in the number of neurons. If we associate independent concepts to perpendicular direction into a N-dimensional space, we can only have N vectors perpendicular to each other. But if we relax that constraint a little bit and tolerate some small deviations from 90 degrees, the number of nearly perpendicular vectors we can fit in that same space grows exponentially [10], as demonstrated in Fig. 19 where we optimize ten times more vectors than the number of dimensions in a 100-dimensional space for maximum near perpendicularity, and many more could fit. It is possible to store many more ideas than dimensions in this space, using a combination of different canonical dimensions, i.e. a superposition.

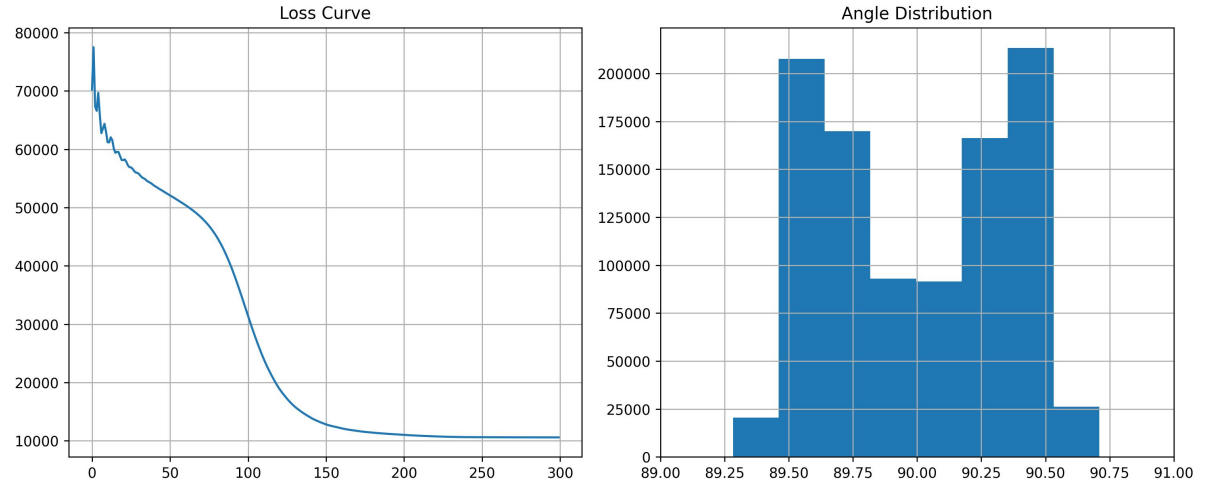

Figure 19: **Optimization process to test superposition** in a high dimensional space, with many more vectors than dimensions.

5 Journey Through the Layers

As we know at this point, all tokens proceed in parallel through the transformer layers and their representation gets updated with contextual information in attention layers and lookup facts in MLPs. During training, the model is asked to predict the next token for each position simultaneously by consuming all partial sequences of ground truth tokens as inputs, using teacher forcing. At inference-time, where the model autoregressive consumes its previous outputs as inputs, all we care about is the prediction coming from the the last token. We will try to understand what happens to the entire cloud while the last token makes the prediction.

First of all, we can imagine that all tokens move quite a lot from their starting position. If we project in the same 3D space the original token embeddings, the final position of one of these, and its trajectory, we get an intuition of how much a single token moves in Fig. 20 while information circulates and it gets contextualized.

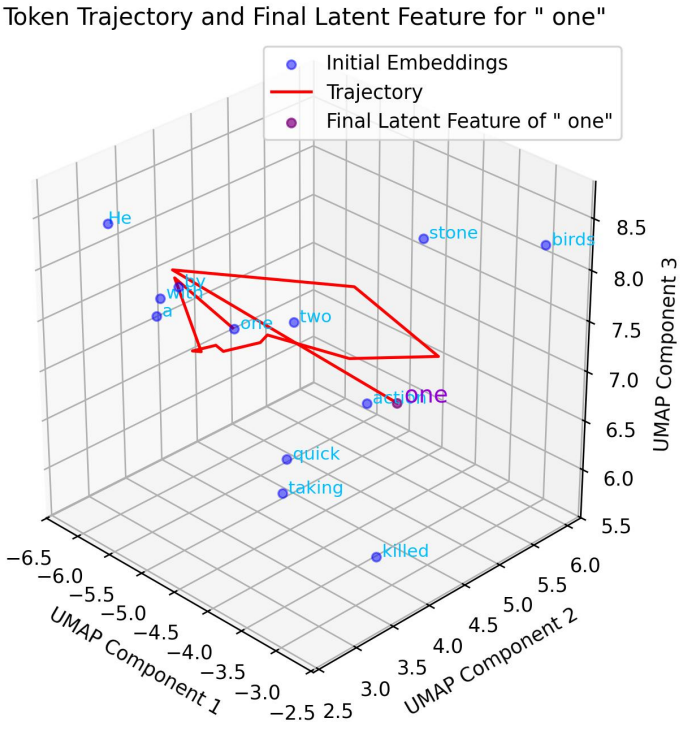

Figure 20: **Journey of a token** through all the transformer layers while it gets contextualized, projected in 3D.

Now, we might ask two questions: (i) do tokens tend to go towards the next word in sequence, and (ii) how far are they from the original position? Neglecting positional embeddings, in Fig. 21 we plot the cosine similarity of each contextualized token layer by layer with the non-contextualized token embedding of the next token and of itself, for an example phrase. As we can see, the first measure seems to stay relatively stationary and drops in the final layer, while we see that distance from the original position increases constantly. So tokens don't seem to go toward the next word, but also they seem to move significantly from their starting point. So where are they going?

If we let the model run, and get what its prediction would be, we can track the distance of all tokens to this prediction (non-contextualized), and also monitor the distance of all tokens to the last value of the sequence (contextualized, layer by layer), the one producing the prediction itself (Fig. 22). We can notice a clear pattern for our example: all tokens, layer by layer, seem to get closer and closer to the last one while that moves too, basically grouping all together. This last token now contextualized, however, does

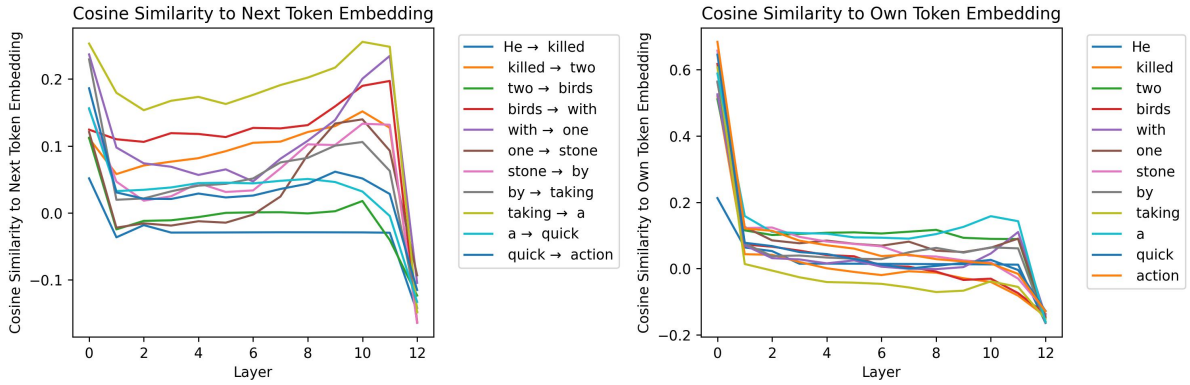

Figure 21: **Similarity of all tokens to their own starting embedding and the embedding of the next word in the sequence**, taken non-contextualized ignoring position.

not align necessarily with the representation of the predicted next word, and it can even have negative cosine similarity with it. It seems counter-intuitive, but the key point here is that the model doesn't directly use the cosine similarity between the hidden state of the last token and the non-contextualized embeddings of the vocabulary tokens for prediction, but a learned mapping through an additional linear transformation and passed through a softmax layer.

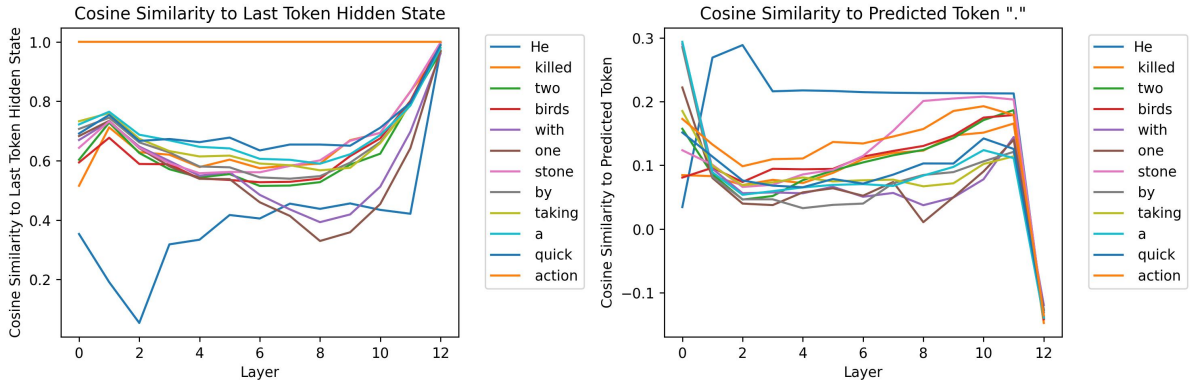

Figure 22: **Similarity of all tokens to the final token in the sequence and to the prediction**, the first taken layer by layer and the second taken non-contextualized.

We can then analyze, for a few sentences, some statistics of the cloud like average distance to the centroid and covariance. We notice once again some surprising pattern: both average distance, in terms of the norm of the distance vectors to the centroid, and variability, in terms of the trace of the covariance matrix, start small, then increase by a lot, and eventually plummet in the last layer (Fig. 23).

Putting it all together, from these analyses, it seems like the cloud of tokens at the beginning remains relatively compact, while each token gets contextualized and moves away from the original direction on top of the already present positional embeddings. Then, the tokens start to increase their magnitude and the cloud opens up, with each token still following the last token from the distance. At the final layers, the next token prediction objective kicks in hard, and the cloud condenses around the final token first in direction and then in distance. This is observed at various lengths of the input sequence.

As a side note, we always need to remember that although the cloud is free to move around in space, the layer normalization before each operation acts as a prior to performing operations in a canonical space which is the surface of a hypersphere. In conclusion, even a simple model like GPT-2, seems very hard to interpret and has some very non-monotonic behavior which personally left me very intrigued and confused at the same time.

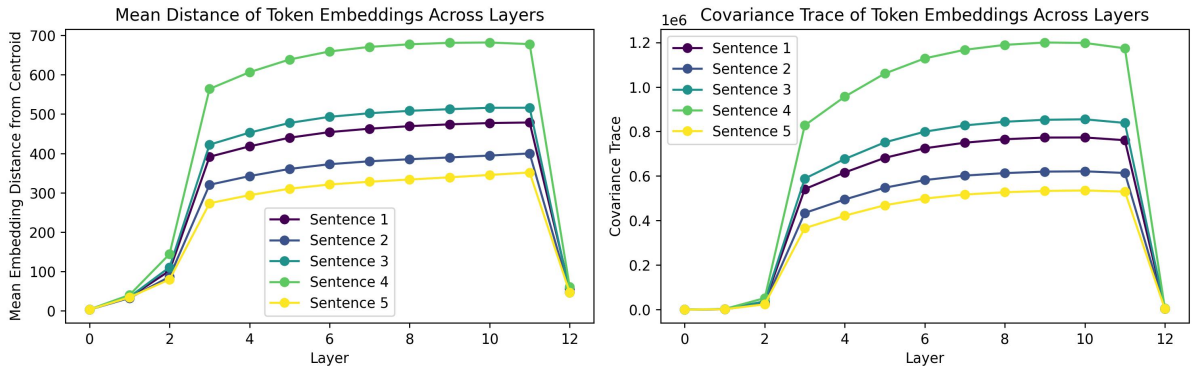

Figure 23: **Mean distance to centroid and covariance** of the token cloud layer by layer.

References

[1] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *Technical report*, 2019.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[3] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. `https://transformer-circuits.pub/2021/framework/index.html`.

[6] Adam Scherlis. An exploration of gpt-2's embedding weights. `https://www.lesswrong.com/posts/BMghmAxYxeSdAteDc/an-exploration-of-gpt-2-s-embedding-weights`, 2022. Accessed: 2024-10-01.

[7] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[8] Adam Yedidia. Gpt-2's positional embedding matrix is a helix. `https://www.lesswrong.com/posts/qvWP3aBDBaqXvPNhS/gpt-2-s-positional-embedding-matrix-is-a-helix`, 2023. Accessed: 2024-10-01.

[9] Raul Molina. Traveling words: A geometric interpretation of transformers, 2023.

[10] Grant Sanderson. Neural networks series. `https://www.3blue1brown.com/topics/neural-networks`, 2024. Accessed: 2024-10-01.

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

[12] Shaked Brody, Uri Alon, and Eran Yahav. On the expressivity role of layernorm in transformers' attention, 2023.

[13] Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model, 2019.

[14] Neel Nanda, Senthooran Rajamanoharan, Janos Kramar, and Rohin Shah. Fact finding: Attempting to reverse-engineer factual recall on the neuron level, Dec 2023.

[15] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. `https://transformer-circuits.pub/2022/toy_model/index.html`.