Master degree in Physics of Data - Academic Year 2024/2025
Final Project for the course of:
Laboratory of Computational Physics - mod B
Teacher: Jeff M. Byers

UNIVERSITÀ DEGLI STUDI DI PADOVA

Dissecting the Frog

A hands-on study of the GPT2 transformer

Students name	Student ID	Student email
Bortolato, Angela	2156562	angela.bortolato.2@studenti.unipd.it
Fasiolo, Giorgia	2159992	giorgia.fasiolo@studenti.unipd.it
Volpi, Luca	2157843	luca.volpi@studenti.unipd.it
Zara, Miriam	2163328	miriam.zara@studenti.unipd.it

How to get started with the Transformers library

A basic example of text generation

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load pretrained model
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
tokenizer.pad_token_id = tokenizer.eos_token_id
inputs =  tokenizer(["Today we can go to"], return_tensors="pt") # attributes: input_ids, attention_mask

outputs = model.generate(**inputs,
                         max_new_tokens = 2, # number of new tokens to generate
                         return_dict_in_generate = True, # here
                         output_scores = True, # scalar products that determine the probability of next token
                         temperature = 1.0,
                         output_logits = True,  # same as scores, in this case
                         output_hidden_states = True, # buffer embeddings
                         output_attentions = True,
                         pad_token_id = 50256)

# Print the output (in plain text)
print("Whole output:",tokenizer.decode(outputs.sequences[0][:], skip_special_tokens =False), "\n")
# Output:
# Today we go to the next
```

How to get started with the Transformers library

```
# Extract the largest 5 logits
generated_token_idx = 1
logits = outputs.scores[generated_token_idx][0]
top_values, top_indices = torch.topk(logits, k=5, largest=True)  # or largest=False
for idx, val in zip(top_indices.tolist(), top_values.tolist()):
    print(f"Index: {idx}, Value: {val}, Decoded: {tokenizer.decode(idx)}")


# Output
#Index: 1306, Value: -88.45703887939453, Decoded:  next <- selected token
#Index: 886, Value: -89.5257339477539, Decoded:  end
#Index: 2003, Value: -89.5971450805664, Decoded:  future
#Index: 966, Value: -89.78964233398438, Decoded:  point
#Index: 9231, Value: -89.84014892578125, Decoded:  polls


# Hidden States
# Available at the exit of each layer (12)
# plus after the positional encoding stage (1)
# total: 13 points for "probing"


layer_idx = 1                              # 0 to 12 (included)
outputs.hidden_states[0][layer_idx][0] # shape (5, 768) = (token sequence length, D)
```

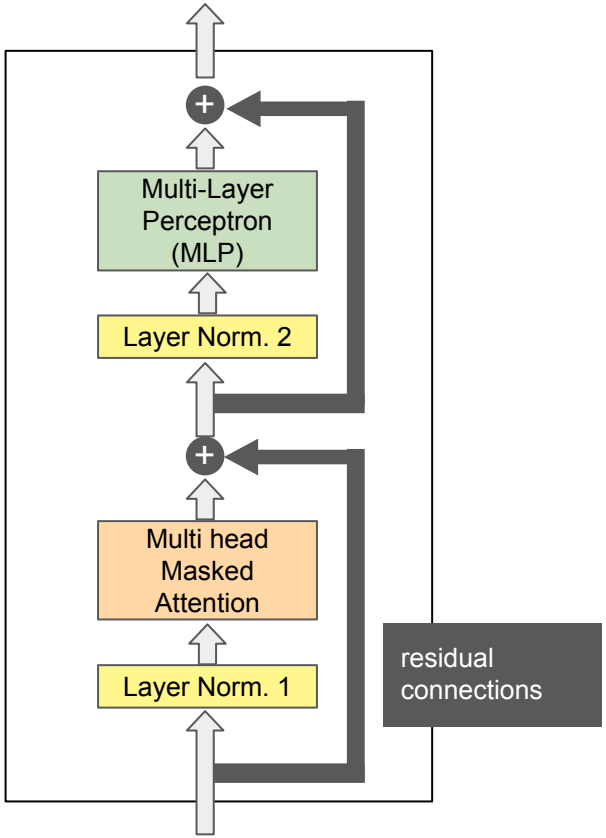
$$\text{Logits} = E \cdot \vec{x}_N = \begin{pmatrix} \langle \vec{t}_1, \vec{x}_N \rangle \\ \langle \vec{t}_2, \vec{x}_N \rangle \\ \cdots \\ \langle \vec{t}_K, \vec{x}_N \rangle \end{pmatrix} \in \mathbb{R}^K$$

$$\text{Probabilities} = \frac{1}{Z} e^{\text{Logits}}$$

$$X = \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \cdots \\ \cdots \\ \vec{x}_N \end{bmatrix} \in \mathbb{R}^{N \times D}$$

Hidden states: embedded representations of the tokens.
Accessible at end of each layer (12) + after positional encoding (1).
Total: 13 “probing” points that can be used to follow the evolution of the buffer through the transformer.

GPT-2 Block (or Layer)



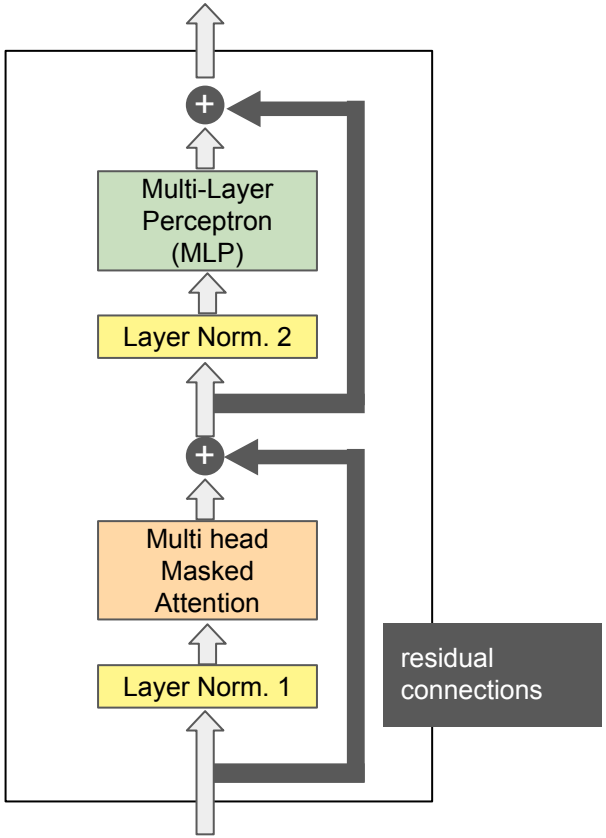

```
transformer.wte.weight torch.Size([50257, 768])
transformer.wpe.weight torch.Size([1024, 768])
```

x12

```
transformer.h.0.ln_1.weight torch.Size([768])
transformer.h.0.ln_1.bias torch.Size([768])

transformer.h.0.attn.c_attn.weight torch.Size([768, 2304])
transformer.h.0.attn.c_attn.bias torch.Size([2304])
transformer.h.0.attn.c_proj.weight torch.Size([768, 768])
transformer.h.0.attn.c_proj.bias torch.Size([768])

transformer.h.0.ln_2.weight torch.Size([768])
transformer.h.0.ln_2.bias torch.Size([768])

transformer.h.0.mlp.c_fc.weight torch.Size([768, 3072])
transformer.h.0.mlp.c_fc.bias torch.Size([3072])
transformer.h.0.mlp.c_proj.weight torch.Size([3072, 768])
transformer.h.0.mlp.c_proj.bias torch.Size([768])
```

```
transformer.ln_f.weight torch.Size([768])
transformer.ln_f.bias torch.Size([768])

lm_head.weight torch.Size([50257, 768])
```

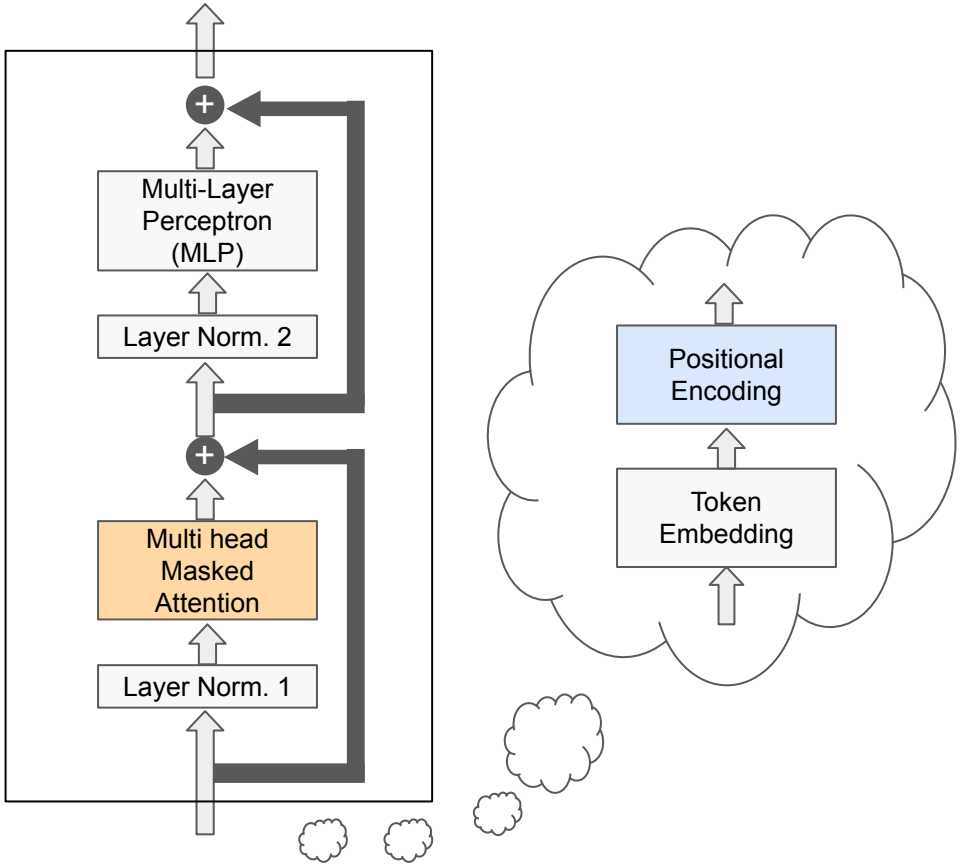
GPT-2 Block (or Layer)




```python
class GPT2Block(nn.Module):
    def __init__(self, config, layer_idx=None):
        super().__init__()
        hidden_size = config.hidden_size
        inner_dim = config.n_inner if config.n_inner is not None else 4 * hidden_size
        self.ln_1 = nn.LayerNorm(hidden_size, eps=config.layer_norm_epsilon)
        self.attn = GPT2Attention(config=config, layer_idx=layer_idx)
        self.ln_2 = nn.LayerNorm(hidden_size, eps=config.layer_norm_epsilon)
        self.mlp = GPT2MLP(inner_dim, config)

    # Forward pass
    def forward(self, ...):
        ## l1 and attn

        residual = hidden_states
        hidden_states = self.ln_1(hidden_states)
        attn_output, _ = self.attn(hidden_states, ...)
        hidden_states = attn_output + residual

        # ln2 and mpl

        residual = hidden_states
        hidden_states = self.ln_2(hidden_states)
        feed_forward_hidden_states = self.mlp(hidden_states)
        hidden_states = residual + feed_forward_hidden_states

        # Output
        outputs = (hidden_states,)
        return outputs
```

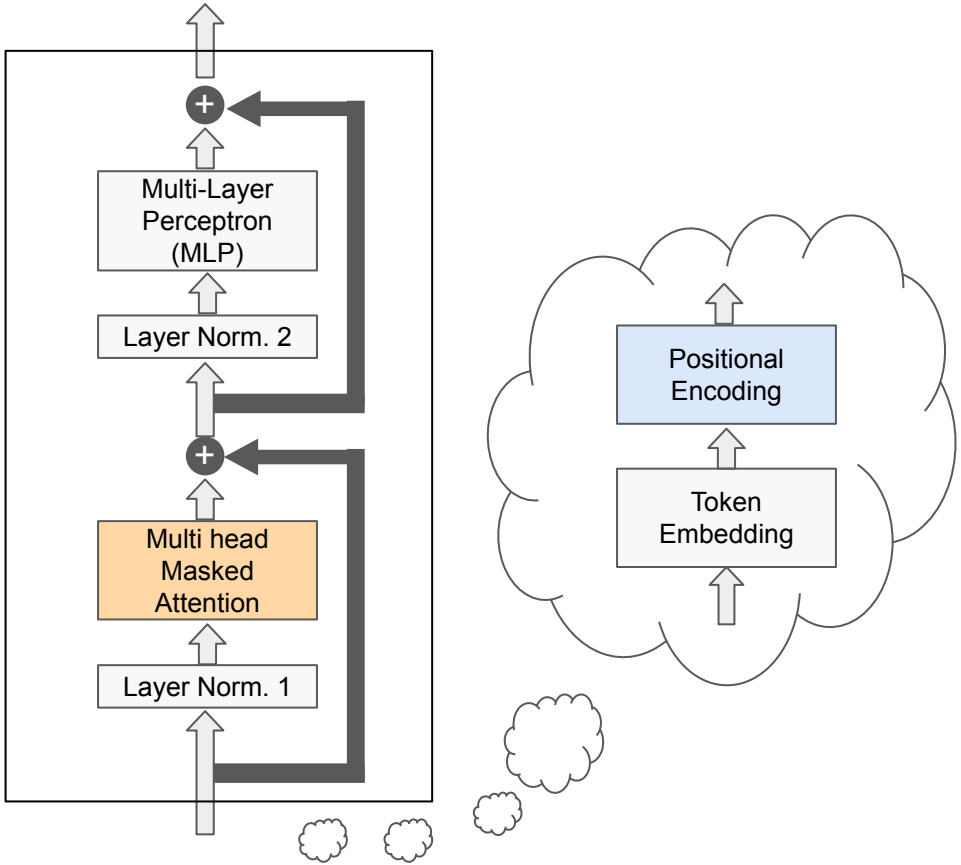
for more check:
[Source code for transformers.modeling_gpt2]

Outline of discussion



1. Multi-head attention
2. Positional Encoding

Outline of discussion



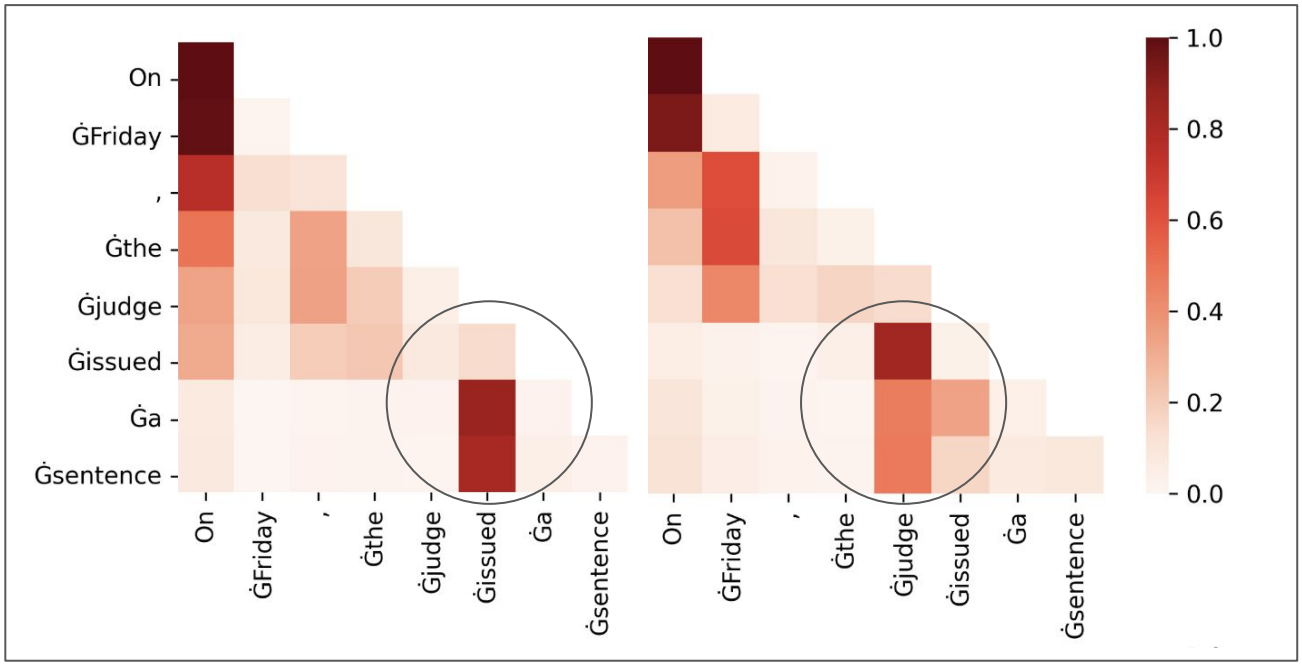
1. Multi-head attention
2. Positional Encoding
3. Empirical study of buffer evolution through the transformer stages

1. Multi-Head Attention

Multi- Head Attention

*Attention transforms the embeddings to account for sentence **context***

Example: "On Friday, the judge issued a sentence."



- *judge* and *issued* suggests that *sentence* refers to a legal penalty, rather than a grammatical “sentence” - the model can ignore safely the other words.

Heads 9 and 10, layer 3/12

Multi- Head Attention

1. Compute queries, keys and values
2. Split into heads
3. Compute output of each head
4. Stack heads and do yet another linear transformation

Multi- Head Attention

1. Compute queries, keys and values

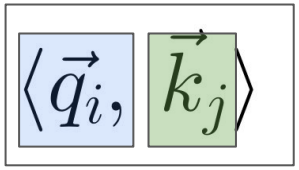
1.

"queries" $Q = XW_Q \in \mathbb{R}^{N \times D}$

"keys" $K = XW_K \in \mathbb{R}^{N \times D}$

"values" $V = XW_V \in \mathbb{R}^{N \times D}$

$[W_Q, W_K, W_V]$ are layer-specific



what the token "needs"

what the token "has to offer"

Multi- Head Attention

1. Compute queries, keys and values
2. Split into heads

1.

"queries" $Q = XW_Q \in \mathbb{R}^{N \times D}$

"keys" $K = XW_K \in \mathbb{R}^{N \times D}$

"values" $V = XW_V \in \mathbb{R}^{N \times D}$

$[W_Q, W_K, W_V]$ are layer-specific

$\langle \vec{q}_i, \vec{k}_j \rangle$

what the token "needs"

what the token "has to offer"

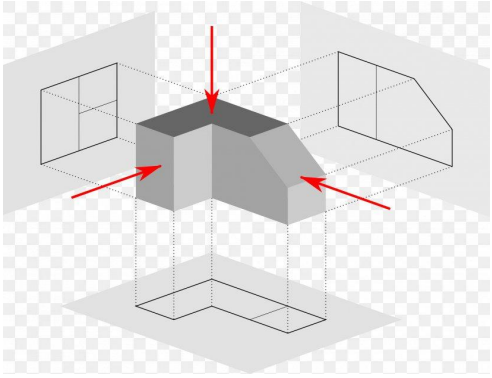
2.

$$d_h = \frac{(D = 768)}{(n_{heads} = 12)} = 64$$

$[Q_h, K_h, V_h] \quad D \times d_h$

$$Q = \begin{bmatrix} (\cdots \vec{q}_1 \cdots) \\ \vdots \\ (\cdots \vec{q}_N \cdots) \end{bmatrix}$$

$Q_h \equiv X \cdot [W_Q]_h$



Like an orthographic projection

-

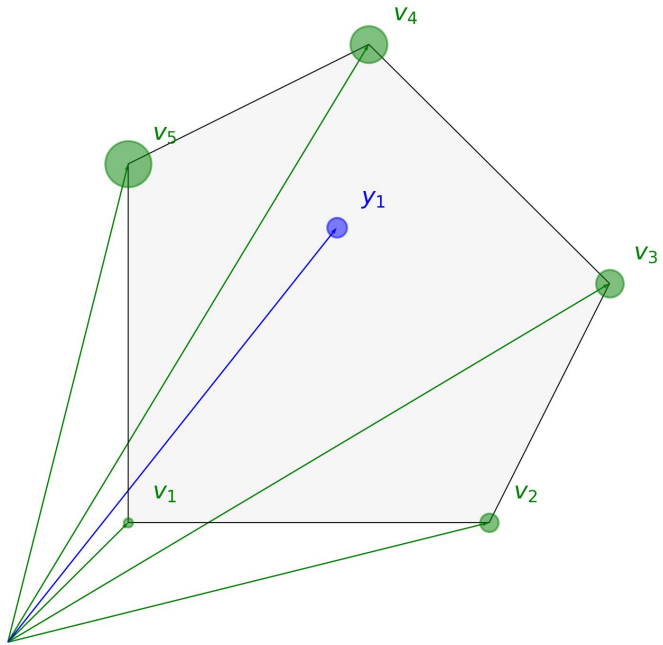
"look at the buffer" from different point of views.

Multi- Head Attention

1. Compute queries, keys and values
2. Split into heads
3. Compute output of each head

3. $$\text{Attn}(Q_h, K_h, V_h) = \text{softmax}\left(\frac{Q_h K_h^T}{\sqrt{d_h}}\right) \cdot V_h$$

$$\underset{N \times d_h}{[\text{Attn. head}]_i} = \frac{\sum_{j=1}^{N} \exp\left(\frac{\langle \vec{q}_i, \vec{k}_j \rangle}{\sqrt{d_h}}\right) \cdot \vec{v}_j}{\sum_{j=1}^{N} \exp(\cdots)}$$



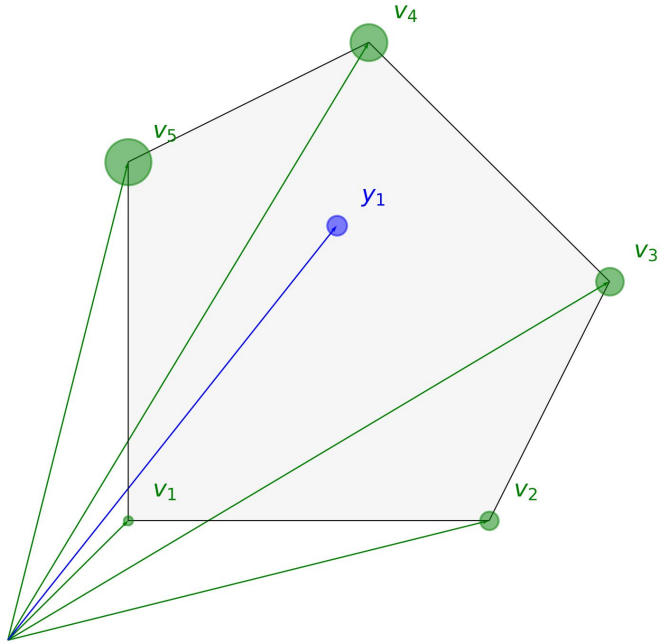
Multi- Head Attention

1. Compute queries, keys and values
2. Split into heads
3. Compute output of each head
4. Stack heads and do yet another linear transformation

3. $$\text{Attn}(Q_h,\ K_h\ , V_h) = \text{softmax}\left(\frac{Q_h K_h^T}{\sqrt{d_h}}\right) \cdot V_h$$

$$\underset{N \times d_h}{[\text{Attn. head}]_i} = \frac{\sum_{j=1}^{N} \exp\left(\frac{\langle \vec{q}_i, \vec{k}_j \rangle}{\sqrt{d_h}}\right) \cdot \vec{v}_j}{\sum_{j=1}^{N} \exp(\cdots)}$$

4. $$\underset{N \times D}{\text{X'}} = [[\text{Attn. h1}], [\text{Attn. h2}], \cdots [\text{Attn. h n}_\text{h}]] \cdot \underset{D \times D}{W_0}$$



Multi- Head Attention

Compute queries, keys and values → Split into heads → **Compute output of each head** → Stack heads and project back

$\langle \vec{q}_i, \vec{k}_j \rangle$ $\quad Q_h[K_h]^T = X\, W_Q^h\, [W_K^h]^T\, [X]^T = X\, M_h\, [X]^T$

- Expectations: Q and K representations should give the model **more expressive power**
- Embeddings of tokens as queries and keys should look different from the embedding E

Ws are learned parameters…. So let's check!

Multi- Head Attention

Compute queries, keys and values → Split into heads → **Compute output of each head** → Stack heads and project back

$\langle \vec{q}_i, \vec{k}_j \rangle$ $\quad Q_h[K_h]^T = X\, W_Q^h\, [W_K^h]^T\, [X]^T = X\, M_h\, [X]^T$

- Expectations: Q and K representations should give the model **more expressive power**
- Embeddings of tokens as queries and keys should look different from the embedding E

Ws are learned parameters…. So let's check!

1. $X I_{d_h} X^T \overset{?}{\leftrightarrow} X\, M_h\, [X]^T$

2. $M_h \overset{?}{\leftrightarrow} M_h^T$

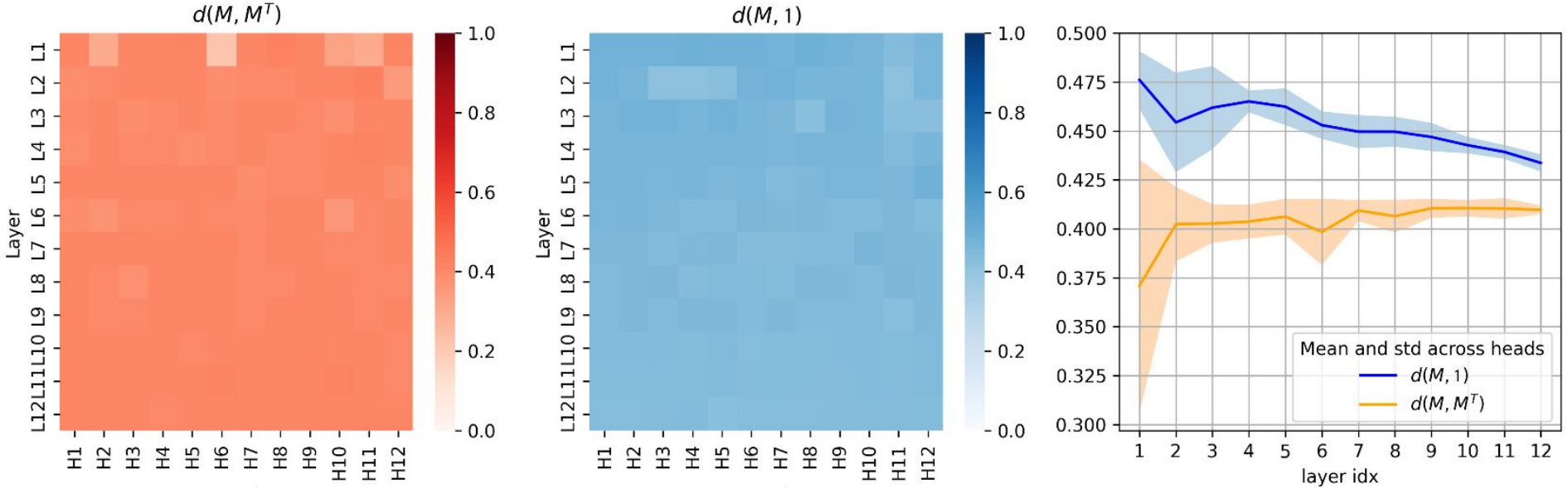
$(M = AA^T (= U\Lambda^{1/2}\Lambda^{1/2}U^T) \qquad \mathbf{X}' = \mathbf{A}\mathbf{X})$

If true, attention weights would be obtainable scalar product on the buffer on itself (case 1), or of a linearly transformed buffer on itself (case 2)

Multi- Head Attention

$$d(A,B) = \frac{||A-B||}{||A|| + ||B|| + ||A-B||} \qquad A, B \quad n \times n \quad ||.|| \text{ Frobenius Norm}$$

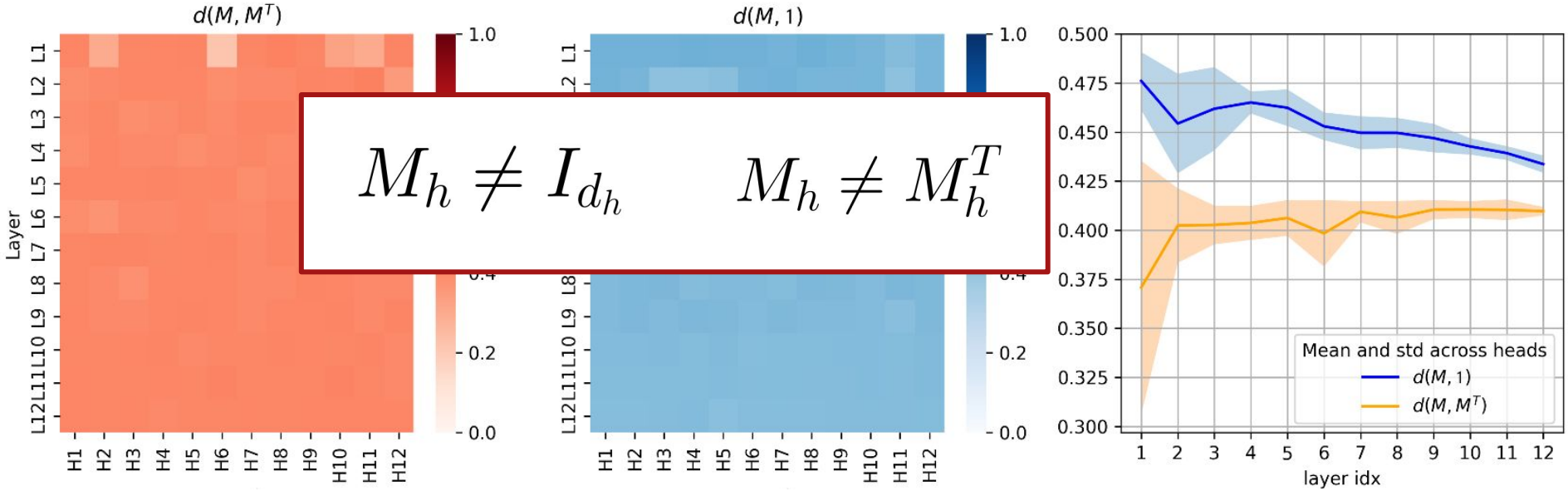
$$d(A,B) \in [0,1] \quad \text{is a metric}$$



Multi- Head Attention

$$d(A,B) = \frac{||A-B||}{||A|| + ||B|| + ||A-B||} \qquad A, B \quad n \times n \quad ||.|| \text{ Frobenius Norm}$$

$$d(A,B) \in [0,1] \quad \text{is a metric}$$



Multi- Head Attention

Compute queries, keys and values → Split into heads → Compute output of each head → Stack heads and project back

$$Q = W_Q \cdot X$$

$$K = W_K \cdot X$$

$$V = W_V \cdot X$$

$$\mathbf{X} = \mathbf{W}_0 \cdot \mathbf{Y}$$

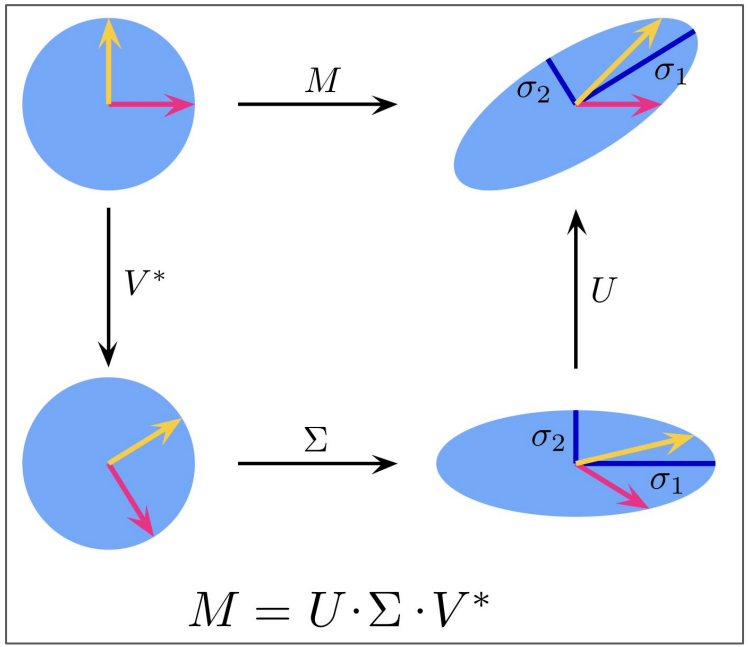
$D \times D$

How are the Ws matrices transforming the embedding space?

- are they projecting onto a lower dimensional space?
- are they introducing significant anisotropy?

Multi- Head Attention

"Whenever you have to deal with a matrix, first of all, SVD- it" (The professor)



From https://en.wikipedia.org/wiki/Singular_value_decomposition

$$\mathbb{R}^D \longrightarrow \mathbb{R}^D$$

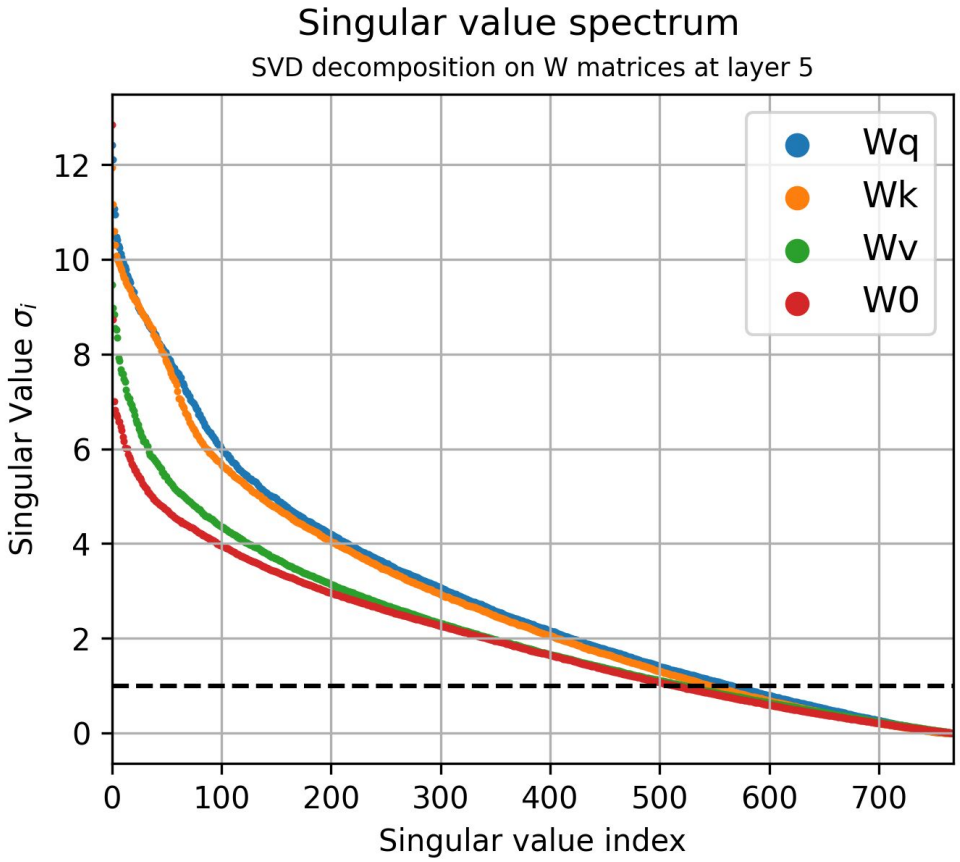
$$\vec{x} \longrightarrow \vec{x'} = M\vec{x}$$

SVD Decomposition on a generic DxD matrix allows to decouple the rotation from the scaling part

- Σ : diagonal with non negative entries -> scaling
- U, V^* : orthogonal -> rotation/ reflection

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_D \end{pmatrix} \quad \begin{cases} \sigma_i > 1 & \text{dilatation} \\ \sigma_i < 1 & \text{compression} \end{cases}$$

Multi- Head Attention



Quantify the anisotropy:

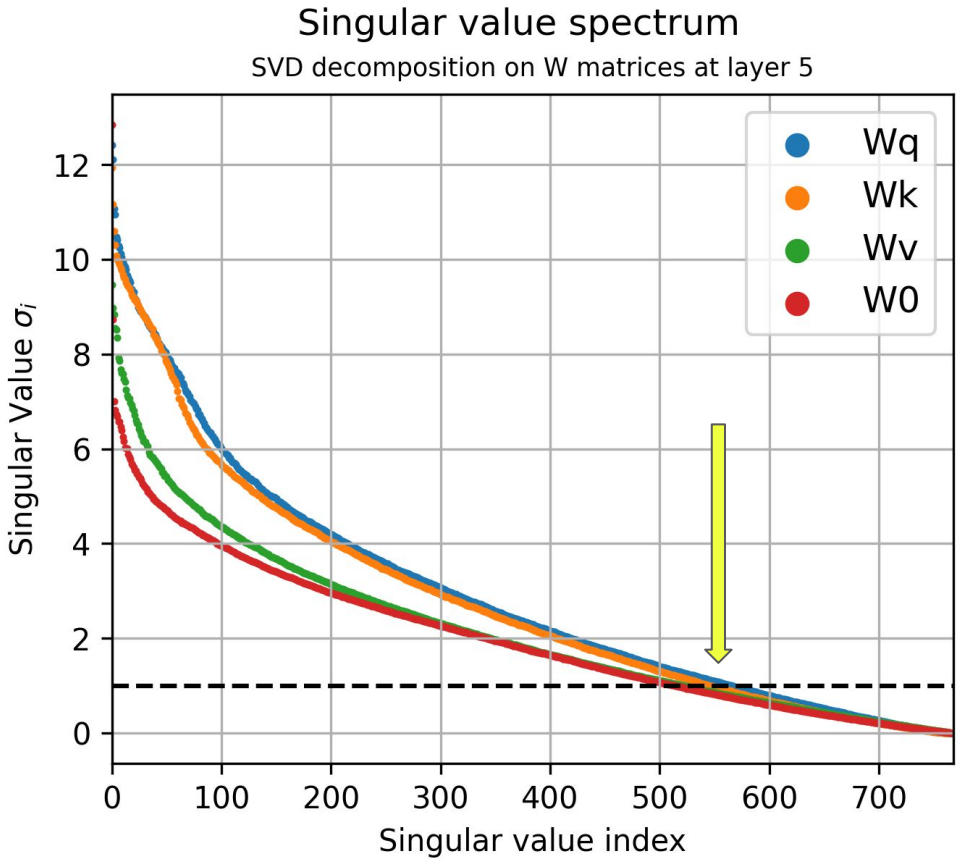
- $\text{Condition Number} = \frac{\sigma_{min}}{\sigma_{max}}$
- $\text{Volume Scaling Factor} = \prod_{i=1}^{D} \sigma_i$
- $\text{Effective Rank} = \exp\left[-\sum_{i=1}^{D} p_i \cdot \log(p_i)\right]$

$$p_i = \frac{\sigma_i}{\sum_j \sigma_j}$$

$$\begin{cases} p_i = 1/D \rightarrow H = -\log(D) & \text{Eff. Rank} = D \\ p_1 \simeq 1, p_j \simeq 0 \rightarrow H \simeq 0 & \text{Eff. Rank} = 1 \end{cases}$$

Paper: [The Effective Rank: a Measure of Effective Dimensionality]

Multi- Head Attention




```
Volume log: 393.94125
Volume: inf
Condition number:
34819.75
entropy -6.2832427
eff rank 535.5224
```

- The model has learned to compress the inputs into the most useful directions.
- Highly anisotropic: compress some directions, stretch others.

2. Positional Encoding

Injecting Positional Information

Attention mechanism per se is a SET operation therefore it is PERMUTATION INVARIANT with respect to the tokens order… BUT in language the order of words is important!!

$$e'_i = [e_i, i]$$

$$e'_i = e_i + \alpha p_i$$

Desirable properties of P.E.:

- Unique encoding for each position (across sequences)
- Computationally simple relation between encoded positions
- Smooth, continuous and predictable change of encoding across positions

This gives the model the OPPORTUNITY to learn relative position awareness easily, but whether it actually does is up to the training process, of course.

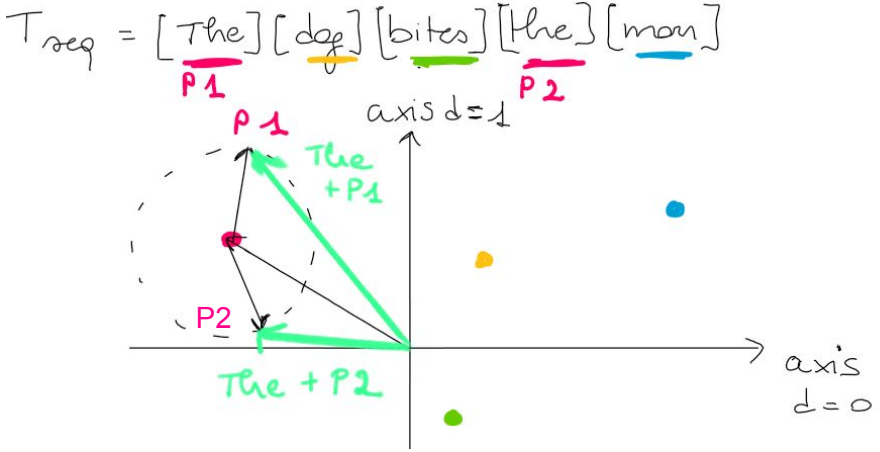
Sinusoidal P.E.

$$PosEnc(n, 2i) = sin(\frac{n}{L^{\frac{2i}{D}}})$$

$$PosEnc(n, 2i+1) = cos(\frac{n}{L^{\frac{2i}{D}}})$$

$$M_k \begin{pmatrix} \sin(\omega_i\, p) \\ \cos(\omega_i\, p) \end{pmatrix} = \begin{pmatrix} \sin\big(\omega_i (p+k)\big) \\ \cos\big(\omega_i (p+k)\big) \end{pmatrix}$$

$$M_k = \begin{pmatrix} \cos(\omega_i k) & \sin(\omega_i k) \\ -\sin(\omega_i k) & \cos(\omega_i k) \end{pmatrix}$$



We can easily pass from one encoded position to another by means of combinations of rotations on 2D planes

Positional encoding in attention context

$$Attn(Q, K, V) = softmax(\frac{QK^T}{\sqrt{(d_k)}})V$$

Through *combined vectors* $e'_i = e_i + p_i$ hidden in Q and K, positional encoding can influence attention by means of its magnitude and/or the angular difference between encoded positions.

$$\vec{e'_i} \cdot \vec{e'_j} = |\vec{e'_i}||\vec{e'_j}| \cdot cos(\theta)$$

Absolute position encoded here → Depend on $|\vec{e_i}|$ and $|\vec{e_j}|$

Depends on the angle between P.E. ← **Relative** position encoded here

WHAT MATTERS MOST IS RELATIVE POSITION OF WORDS RATHER THAN ABSOLUTE POSITION

Norms of positional vectors

Positional encodings should vary primarily in direction, not norm, so that relative positions are emphasized over absolute magnitude.

In general: $||e_i + p_i||^2 = ||e_i||^2 + ||p_i||^2 + 2\, e_i \cdot p_i$

so even if $||e_i||$, $||p_i||$ are constant across positions, $||e_i + p_i||$ is not necessarily constant… However in high dimension the dot product is almost zero always!

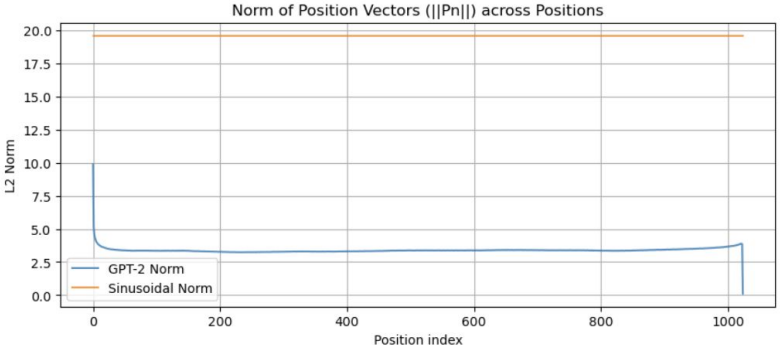
→ *the norm of the combined vector is mostly independent of position.*

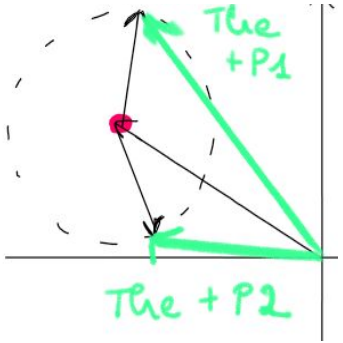
```
Token 'internet' norm = 4.272682
```

```
Sinusoidal positional encoding norms:
Position 100: pos norm = 19.595917
Position 200: pos norm = 19.595919
Position 300: pos norm = 19.595919
Position 400: pos norm = 19.595919
Position 500: pos norm = 19.595919
Position 600: pos norm = 19.595919
Position 700: pos norm = 19.595919
Position 800: pos norm = 19.595917
Position 900: pos norm = 19.595919
Position 1000: pos norm = 19.595919
```

```
Combined embedding norms:
Position 100: combined norm = 19.927237
Position 200: combined norm = 20.138176
Position 300: combined norm = 19.948368
Position 400: combined norm = 19.937489
Position 500: combined norm = 20.080328
Position 600: combined norm = 20.260506
Position 700: combined norm = 20.180273
Position 800: combined norm = 20.289295
Position 900: combined norm = 20.389450
Position 1000: combined norm = 20.077616
```

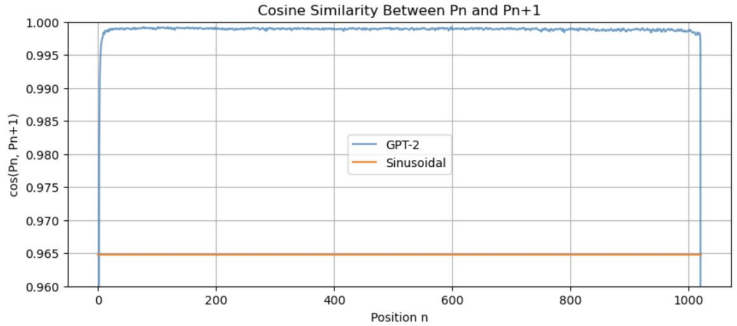
Balancing semantic content (word meaning) and syntactic structure (position in the sequence).



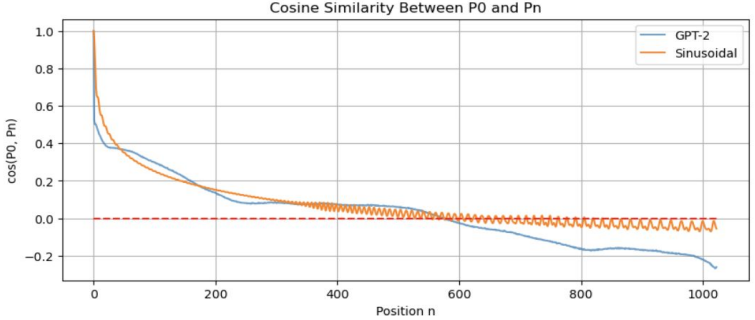



```
Token Embedding Norms:
  Average: 3.9585
  Std dev: 0.4337
  Max:     6.3155
  Min:     2.4537

Positional Encoding Norms:
  Average: 3.3901
  Std dev: 0.2620
  Max:     9.8756
  Min:     0.1179
```

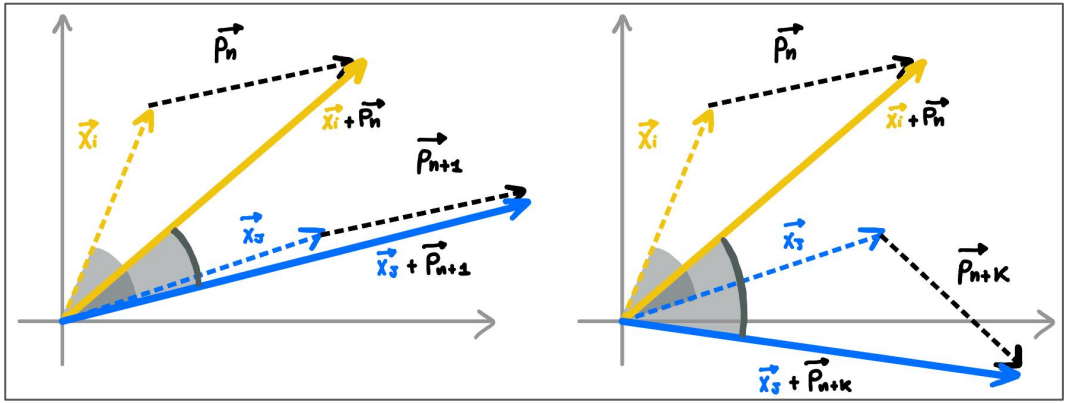
Cosine similarity



Cumulative cosine similarity shows that P.E. is NOT doing a rigid translation of the whole buffer

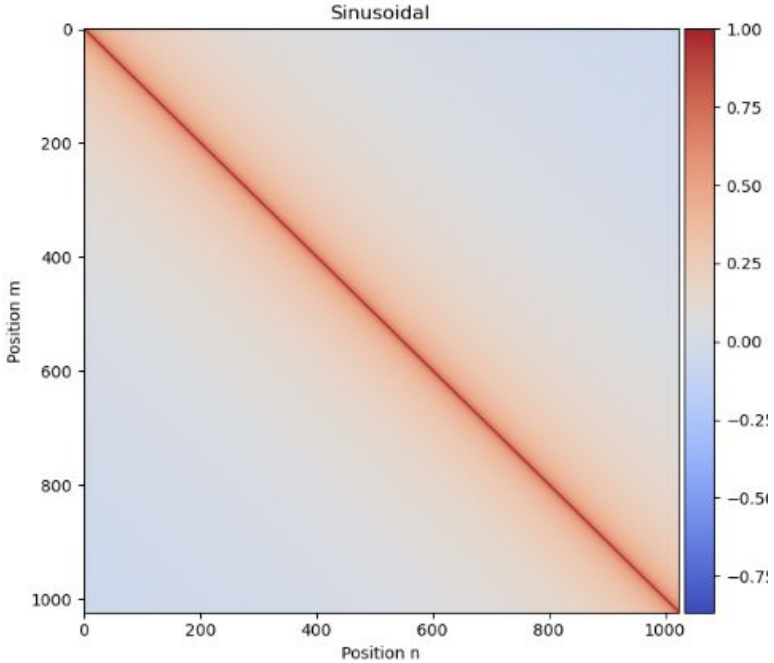


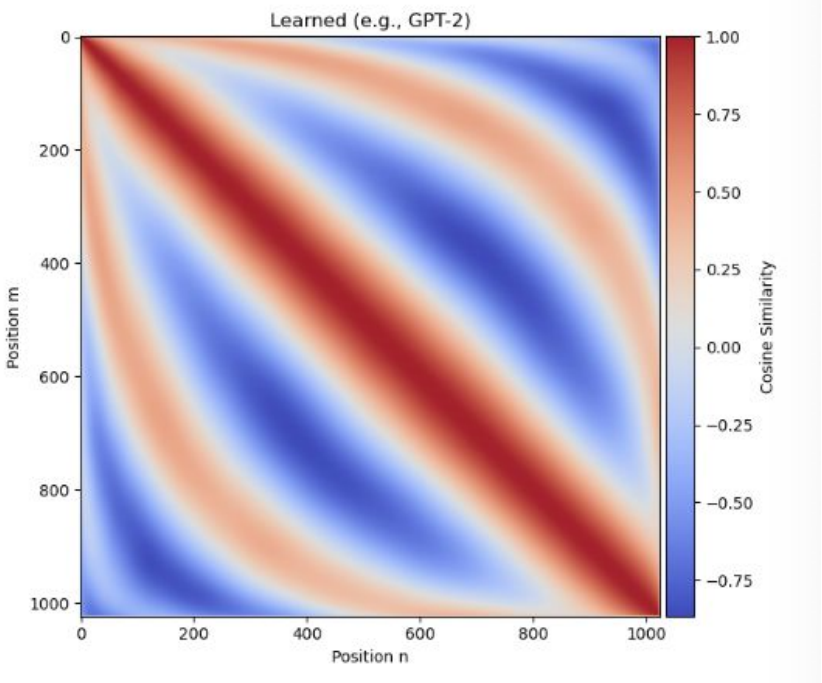
What's happening?



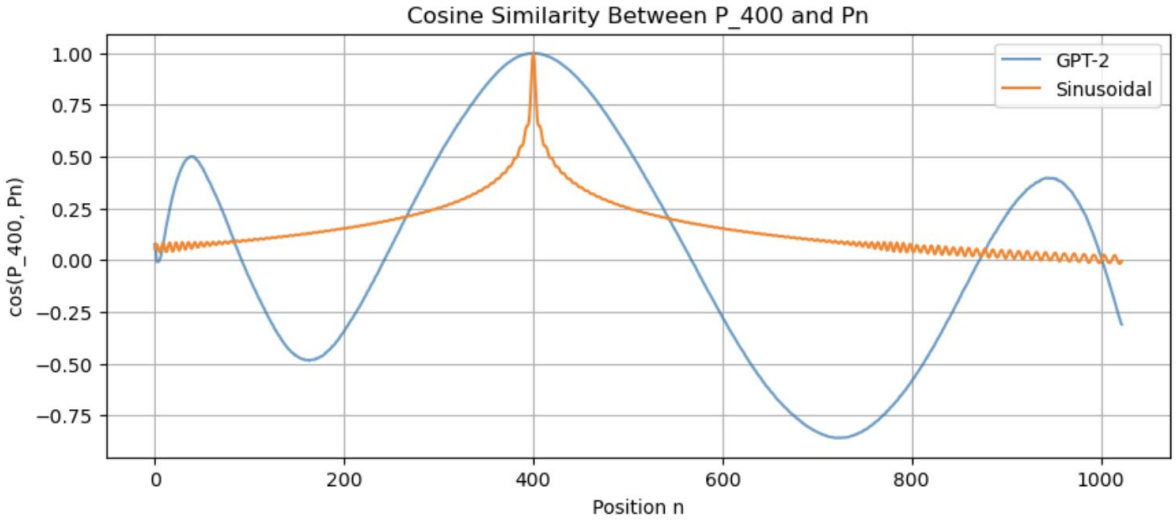
Positional encoding is pushing away from each other tokens that are further apart from one another
(in small gradual steps)

Cosine similarity





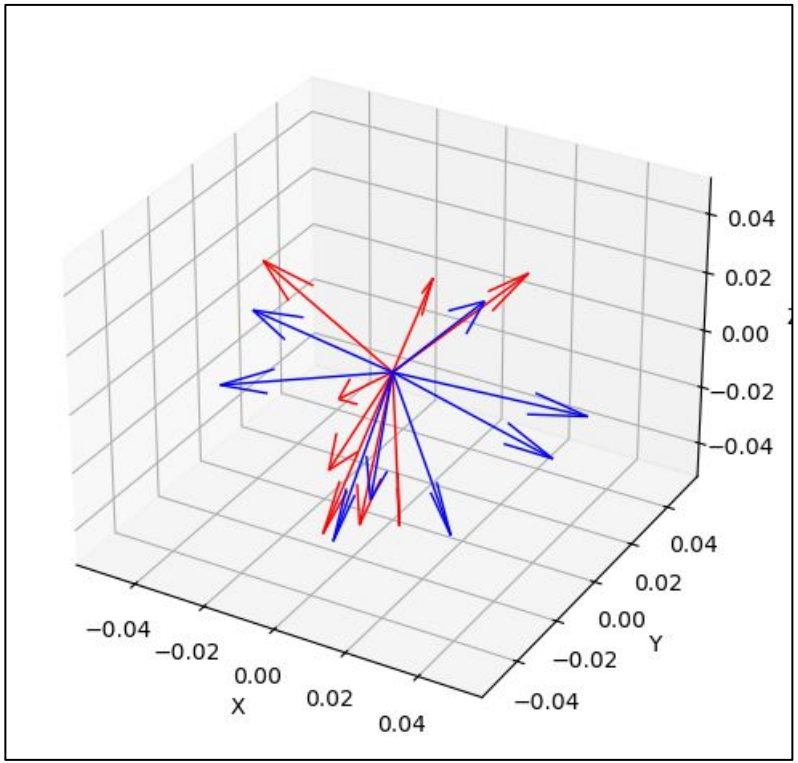
Cosine similarity



From this study of cosine similarity we hypothesized that GPT-2 has learnt to pay attention not only to immediate neighbors but also to further tokens that may be semantically relevant.

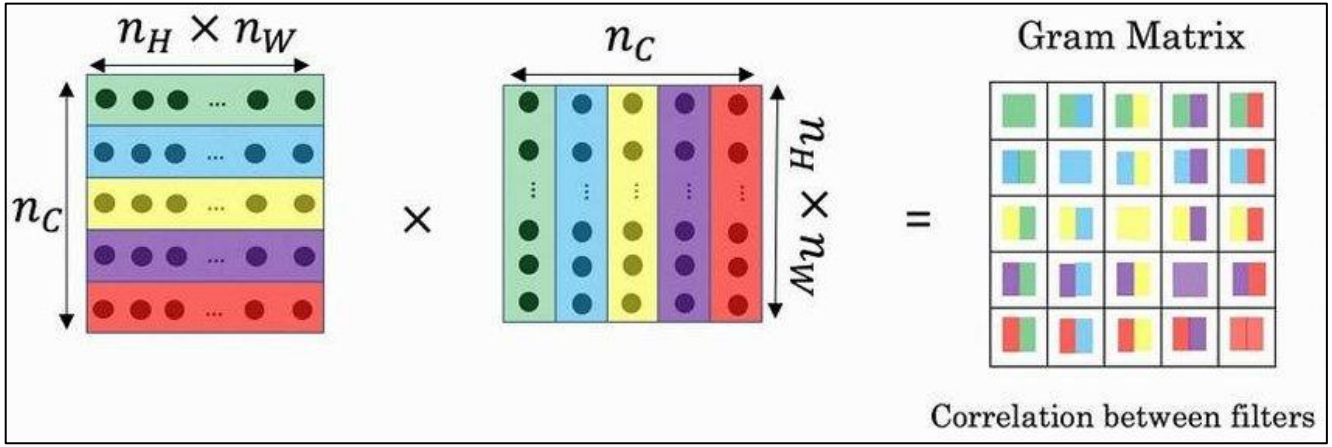
3. Frame of Transformation

Frame of Transformation



- Why?
- What it is?
- How?

Gram matrix



Covariance matrix - DxD

$$\sum = \frac{1}{n} M^T M$$

Gram matrix - NxN

$$G = M_c M_c^T$$

First step

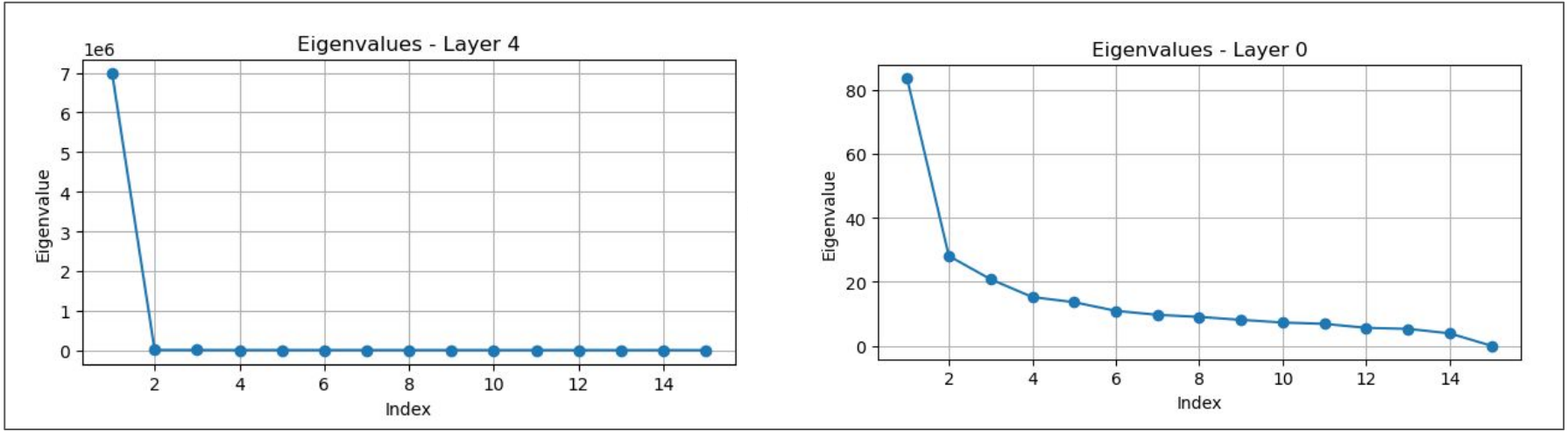
"Einstein is best known for the theory of"

Gram Matrix

SVD to capture most variance in data

$$SVD(G) = U_d L U_d^T$$

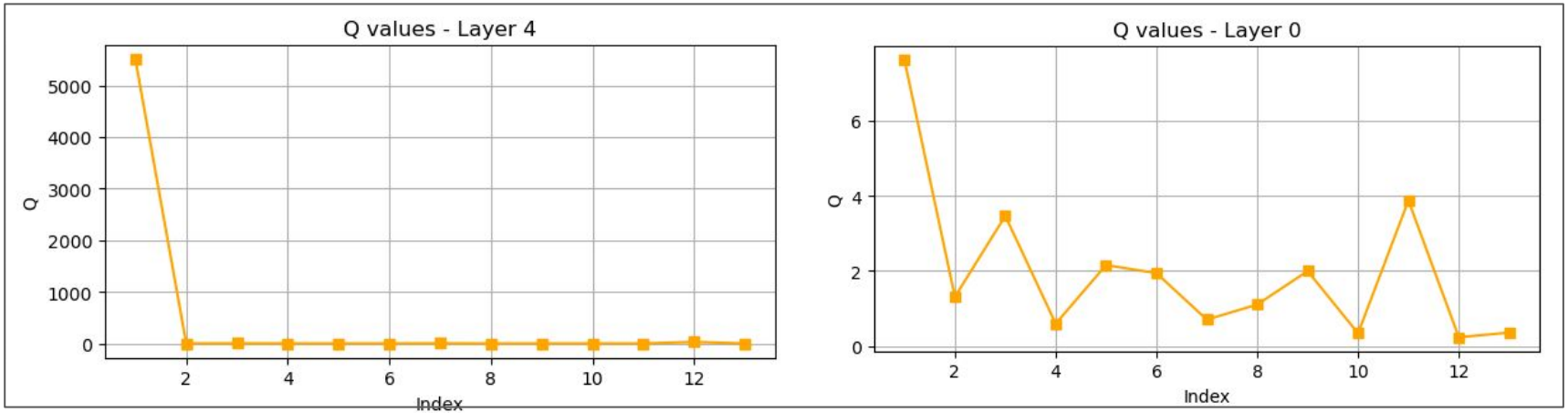
What is the d?


Eigenvalues - Layer 4
1e6
Eigenvalue
Index
Eigenvalues - Layer 0
Eigenvalue
Index

Spectral gap approach

Spectral gap: Q >>250

$$Q = \frac{\lambda_i - \lambda_{i+1}}{\lambda_{i+1} - \lambda_{i+2}}$$



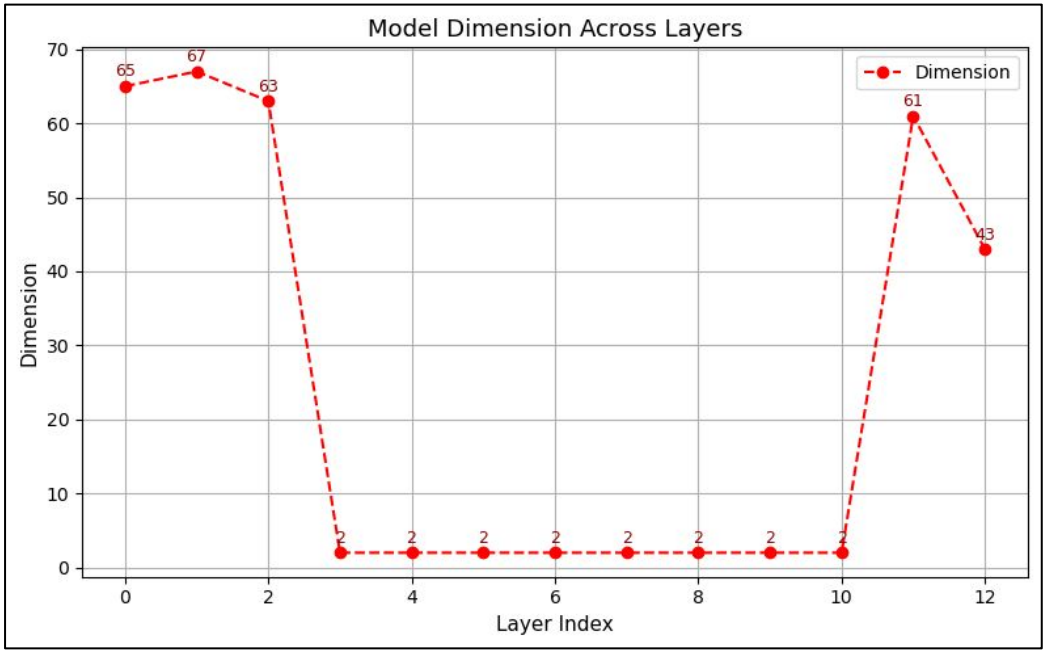
Marchenko – Pastur law

$$M = \frac{N_{sample}}{N_{feature}}$$

LIMIT

$$\lambda_i > \sigma^2 \left(1 + \sqrt{\frac{1}{M}}\right)^2$$

medium prompt (150 tokens)



$$\check{G} = U_d\ (i) \cdot U_d^T\ (i+1)$$

$$SVD\,(\check{G})$$

To obtain the single values:

$$\sigma_1, \ldots,\ \sigma_{D^{\cdot}}$$

$$D^{\cdot} = \min\ (D,\ N-1)$$

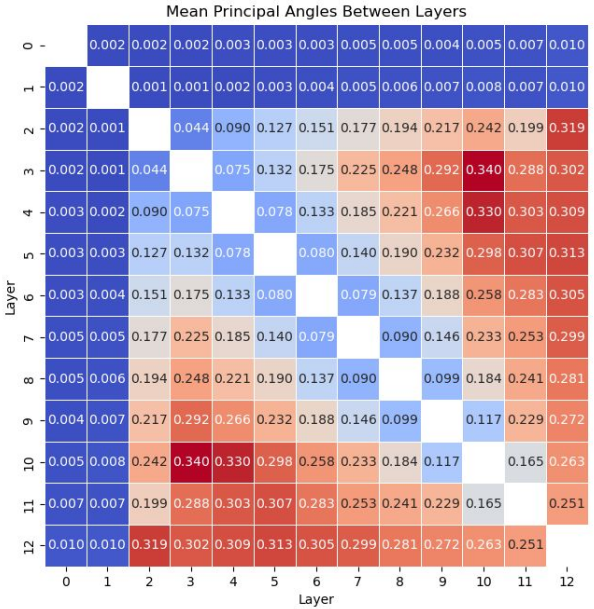
$$\theta_i = \arccos\,(\sigma_i)$$

$$PA = \sum_{1}^{D^{\cdot}} \theta_i$$

Results

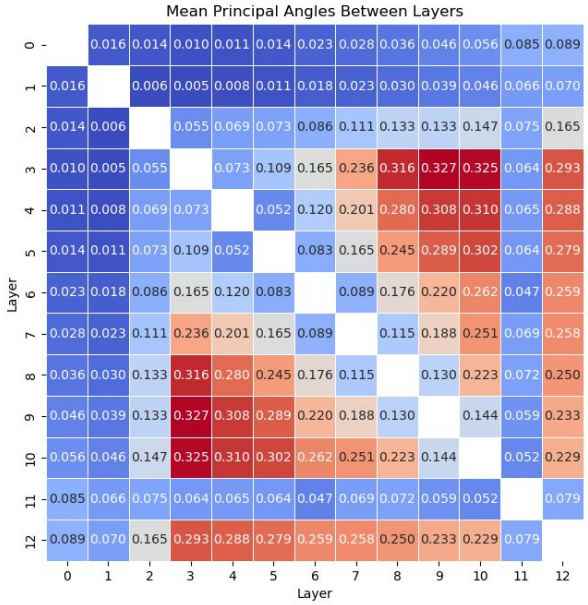
SHORT PROMPT

$$N_{token} = 15$$



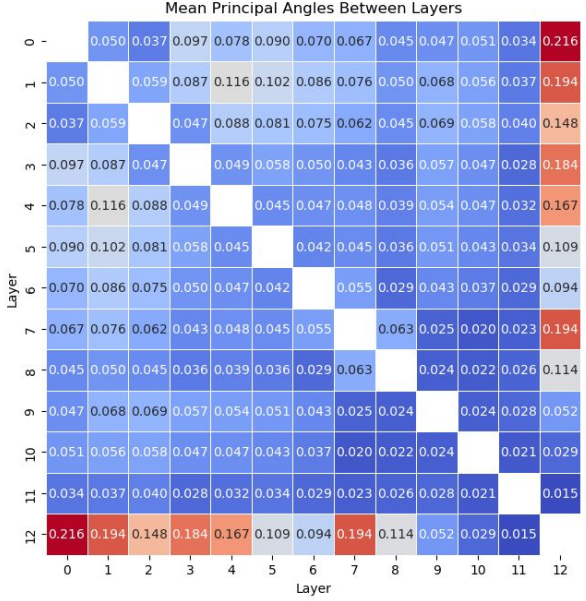
MEDIUM PROMPT

$$N_{token} = 150$$

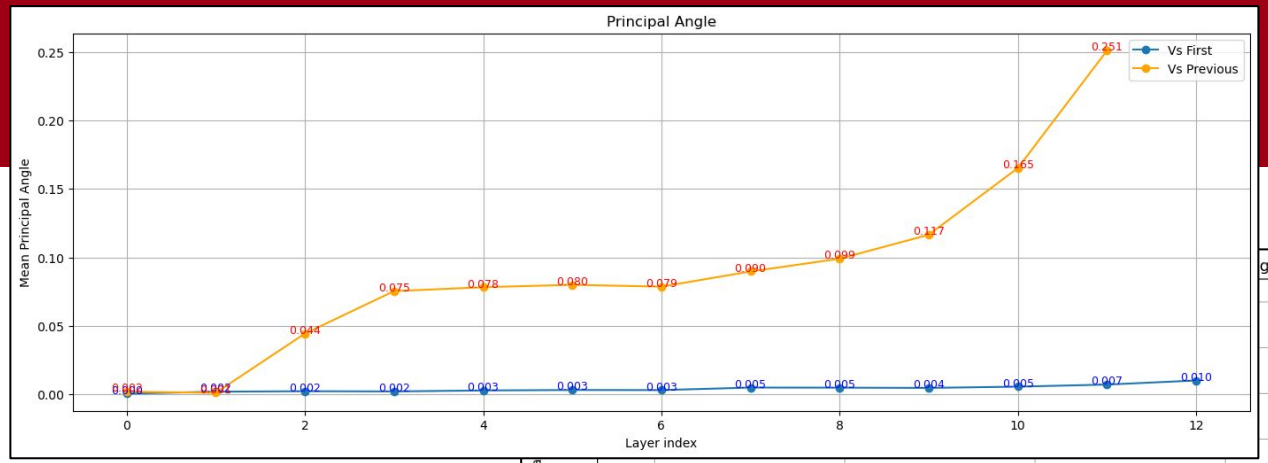


LONG PROMPT

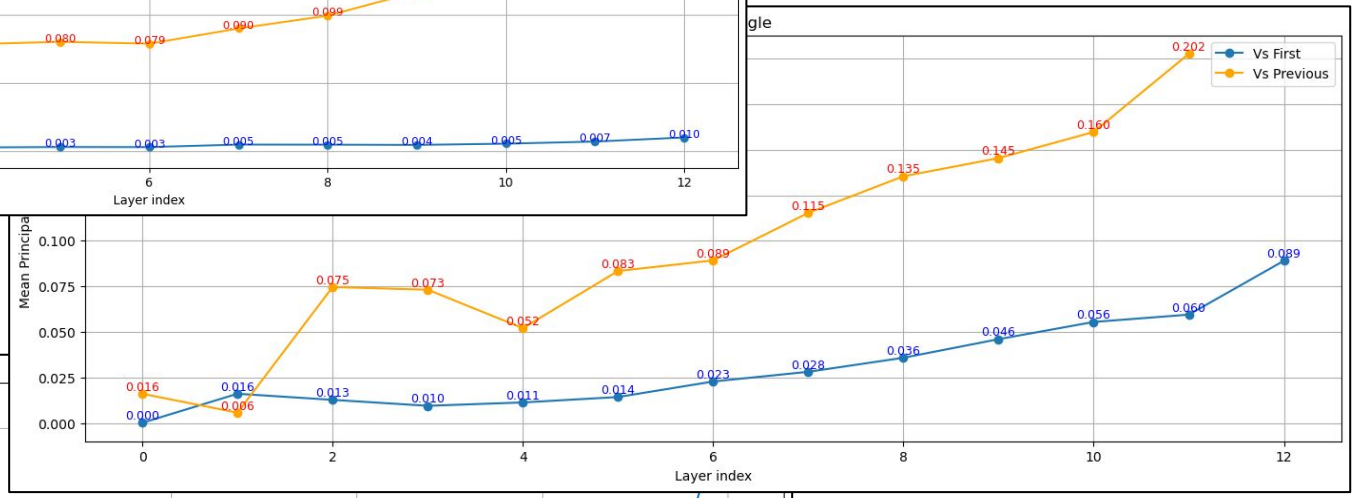
$$N_{token} = 500$$

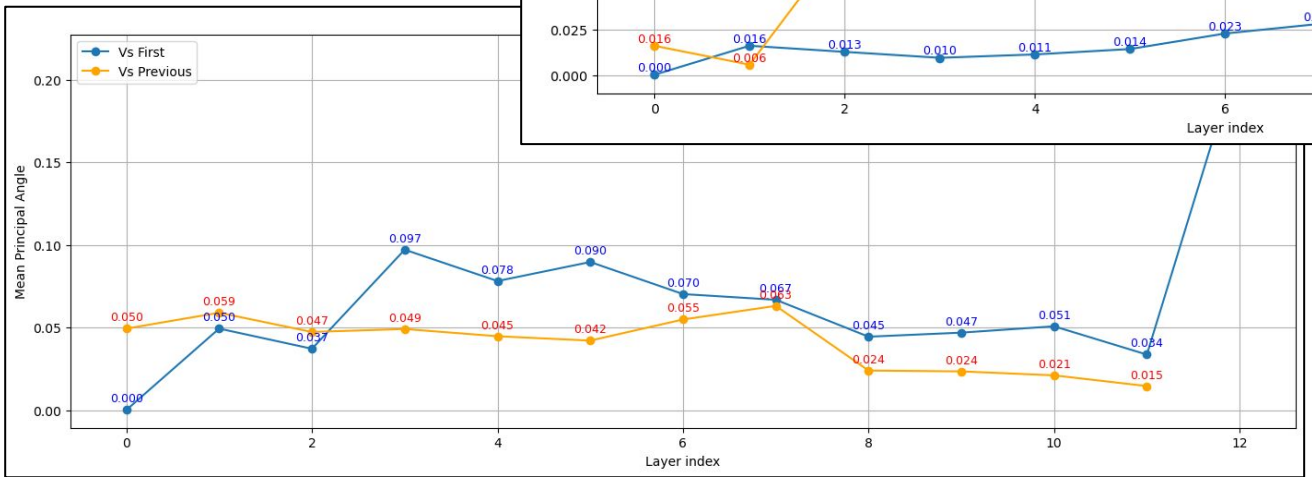


SHORT PROMPT



MEDIUM PROMPT



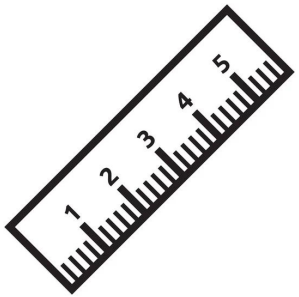


LONG PROMPT

4. Exploration of Subspace

Goal

- How diverse are the signal subspaces?
- How much of the ambient space is explored?



DIFFERENT METRICS **& DIFFERENT BUFFER SIZES...**

- Mean Pairwise Principal Angles
- Dispersion Score (Average normalized angle)
- Subspace Overlap
- Effective Dimensionality
- Union Subspace Rank (Dimensionality of Combined Subspace)
- Subspace Dispersion via MDS embedding

Mean Angle and Dispersion Score

Higher avg angle → subspaces are more diverse → space is more explored

```
Avg Principal Angle (θ): 338.82°
Max Angle: 1824.05°
Min Angle: 48.79°
Dispersion score (normalized θ): 0.260
```

Very long buffer (~1000 tokens)

```
Avg Principal Angle (θ): 250.83°
Max Angle: 607.89°
Min Angle: 72.76°
Dispersion score (normalized θ): 0.302
```

Medium buffer (~150 tokens)

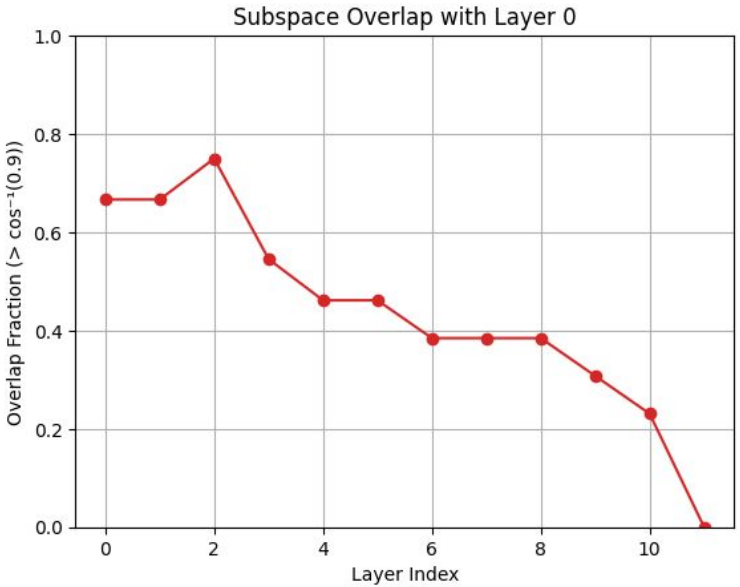
```
Avg Principal Angle (θ): 108.63°
Max Angle: 272.95°
Min Angle: 25.68°
Dispersion score (normalized θ): 0.326
```

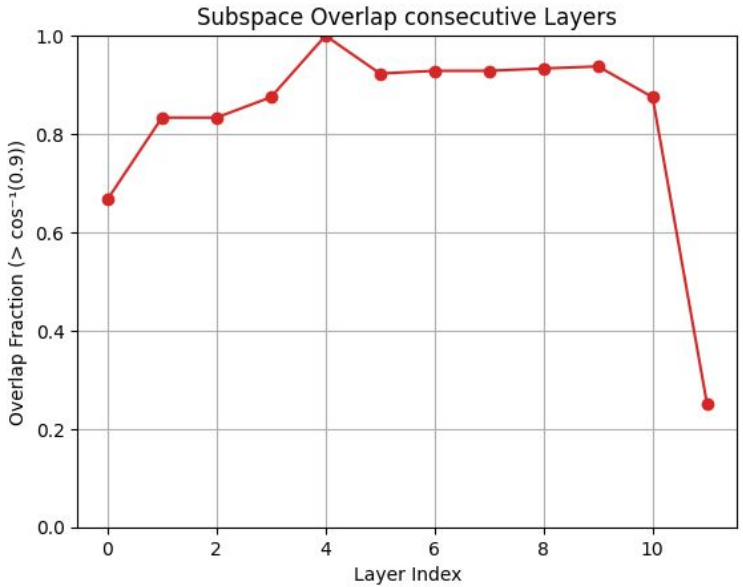
Short buffer (~15 tokens)

Subspace Overlap

We defined overlap as fraction of shared signal directions between layer i and layer j.

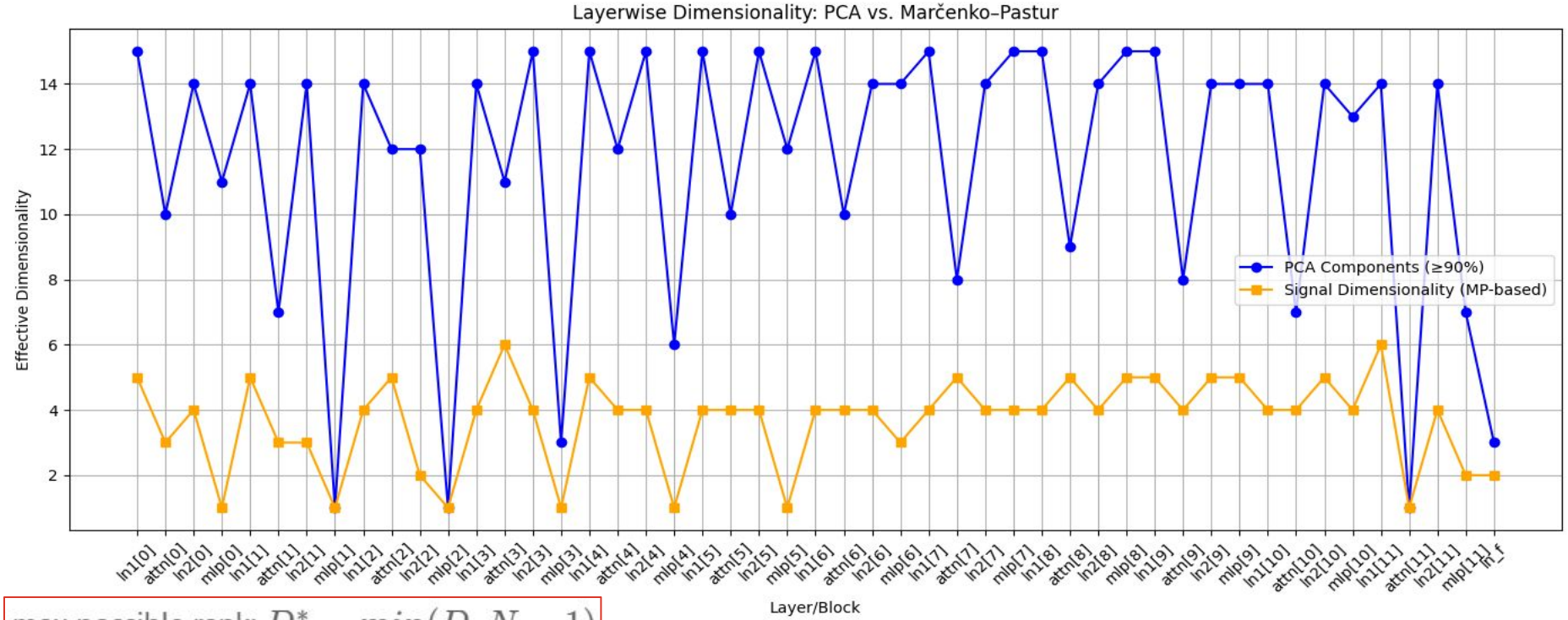
Returns the fraction of principal angles with cos(θ) > threshold, which corresponds to high similarity





Medium buffer (~150 tokens)

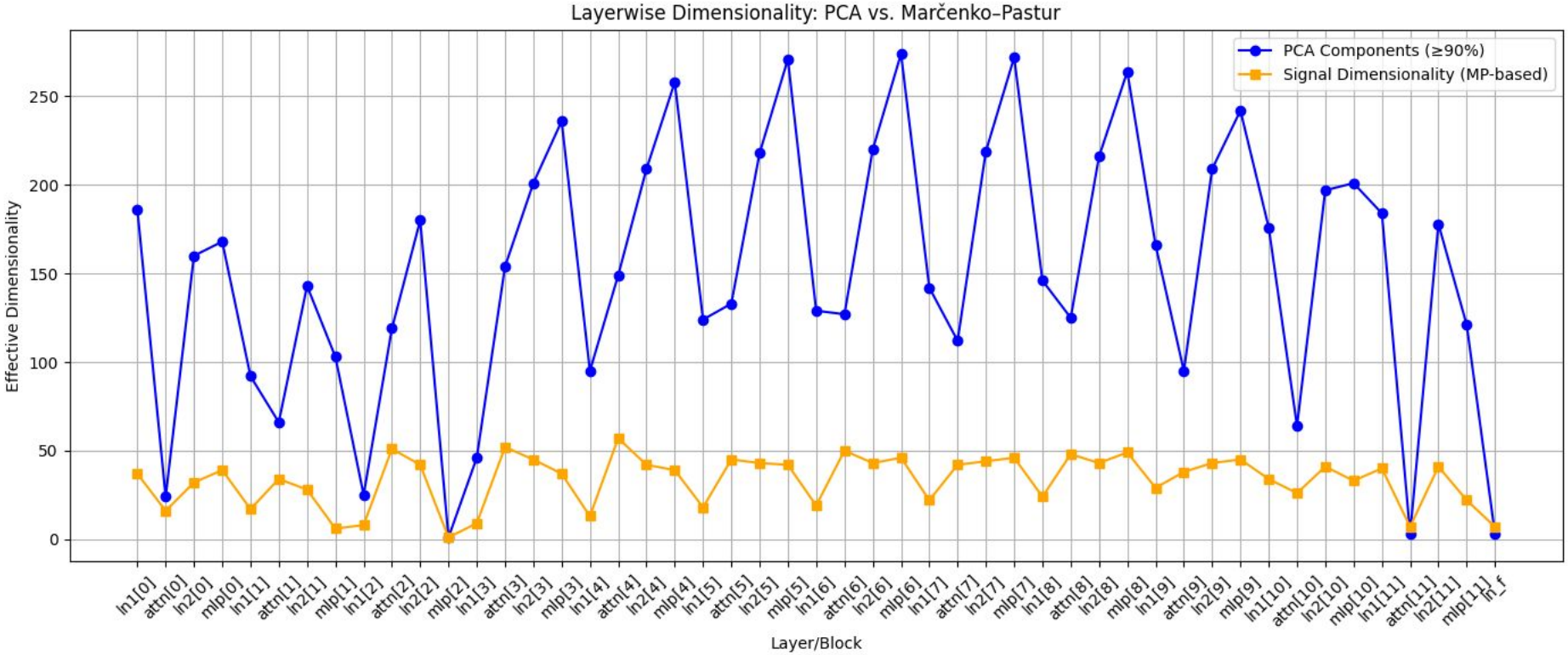
Effective Dimensionality



max possible rank: $D^* = min(D, N-1)$

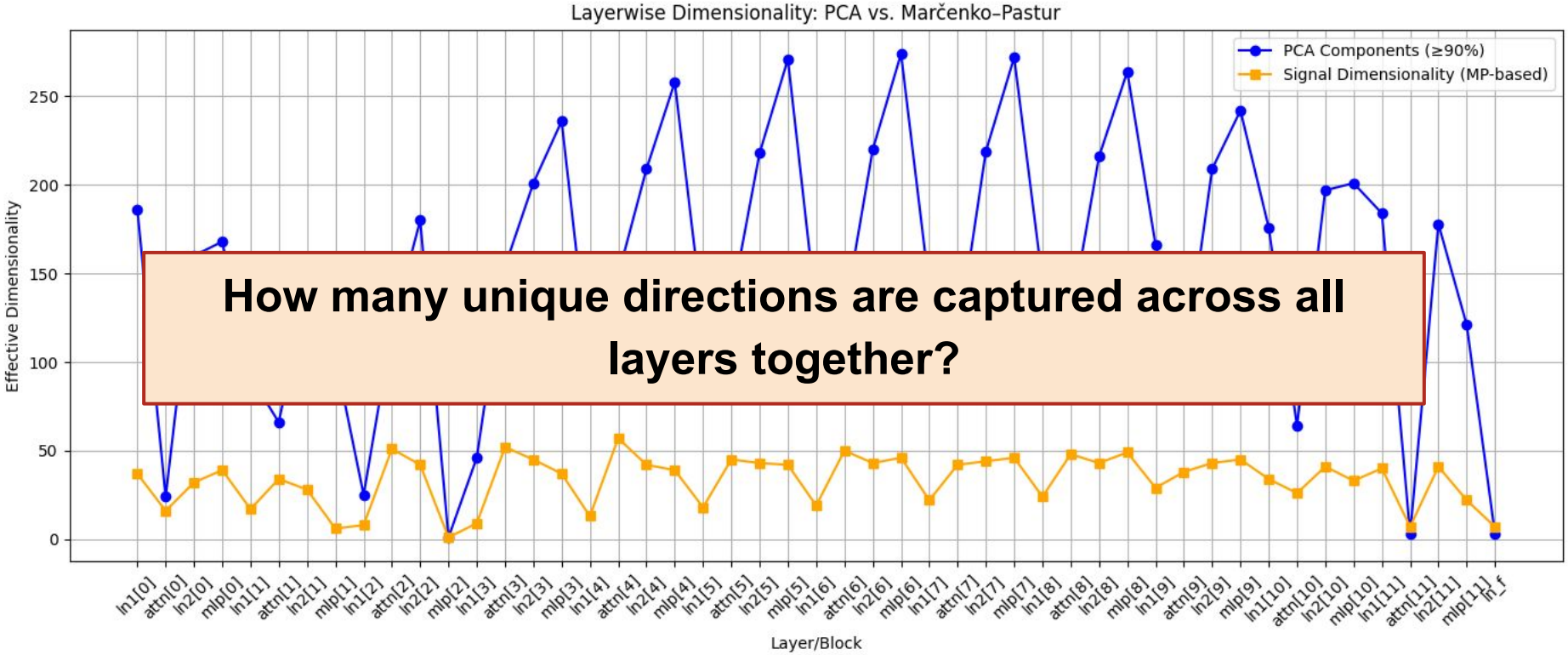
Short buffer (~15 tokens)

Effective Dimensionality



Very long buffer (~1000 tokens)

Effective Dimensionality



How many unique directions are captured across all layers together?

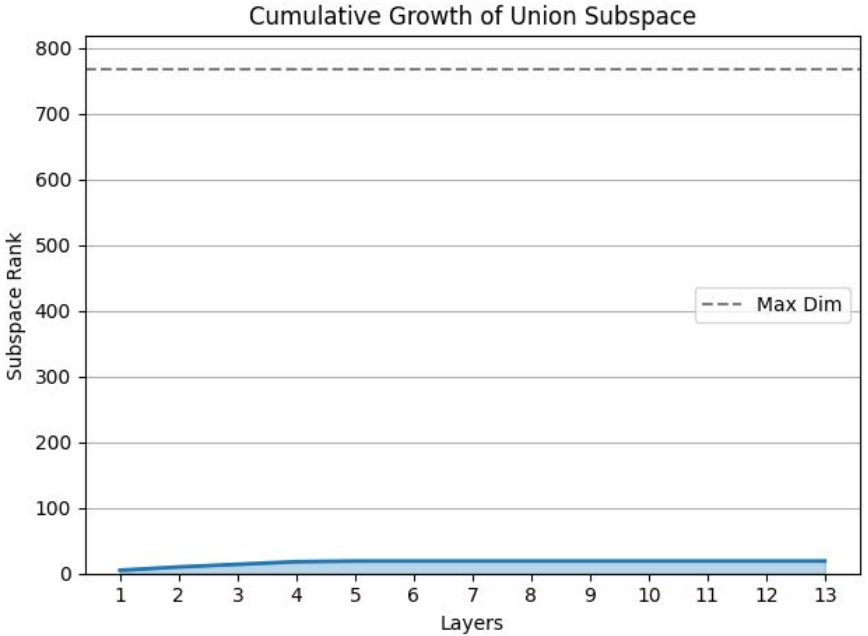
Very long buffer (~1000 tokens)

Union Subspace Rank

Dimensionality of the combined span of multiple subspaces

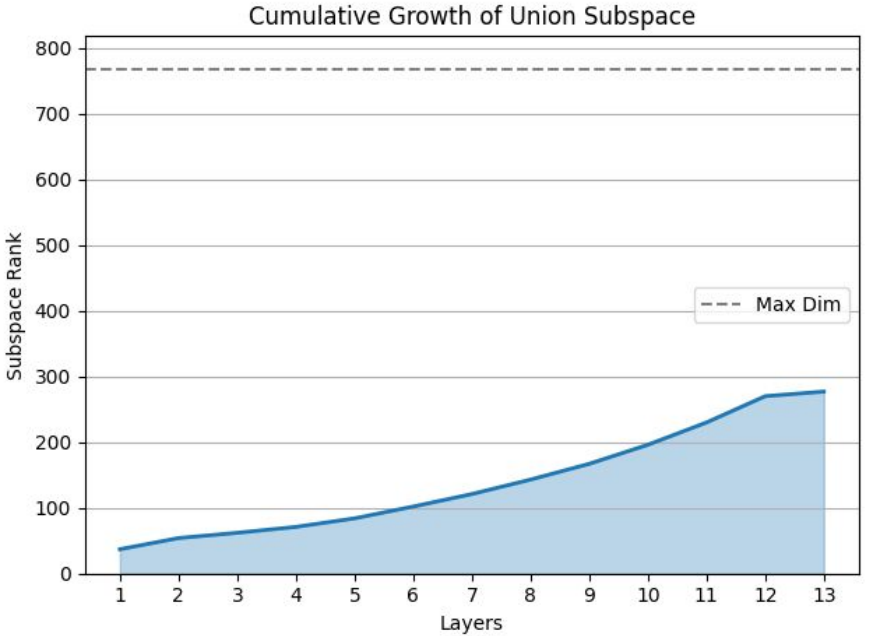
Short buffer (~15 tokens)

Union subspace rank: 19 (2.4739583333333335%)



Very long buffer (~1000 tokens)

Union subspace rank: 277 (36.067708333333336%)



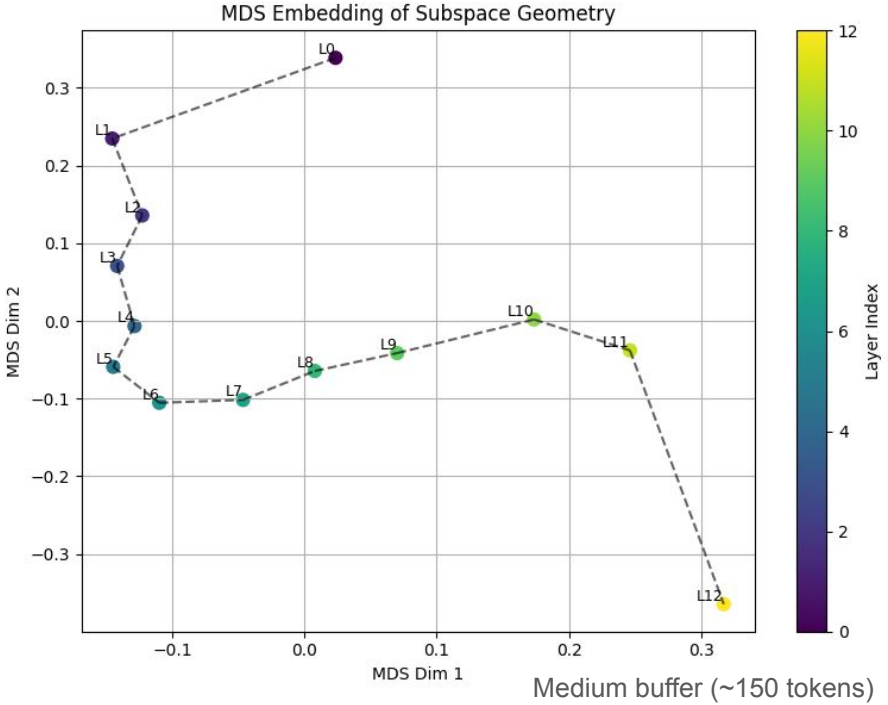
MDS Embedding

MDS places each subspace as a point in 2D, such that their Euclidean distance ≈ subspace angle.

The spread of the points shows how much the buffer "moves" through signal space, it can show different patterns:

Tight cluster	Layers are similar — redundancy or stable representations
Linear trail / arc	Smooth, progressive change — like a temporal evolution
Scattered spread	Highly dynamic changes across layers — strong diversity
Loops / cycles	Possible recurrence, memory reuse, or oscillations

This is a flattened projection of Grassmanian manifold

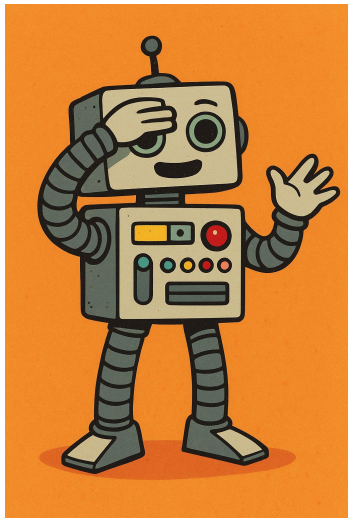


Medium buffer (~150 tokens)

Conclusion

There are many possible directions of research.

How to conclude? We asked ChatGPT



"Transformers are deep, geometric, abstract, and terrifyingly good at language.
But let's be honest — at the end of the day, they're just giant matrix multipliers with the memory of an elephant and the social skills of a mimic octopus.

They don't *understand* language — they just play a game of **"guess what comes next"** so well that we started calling it intelligence."

It's like playing charades with a statistical God.

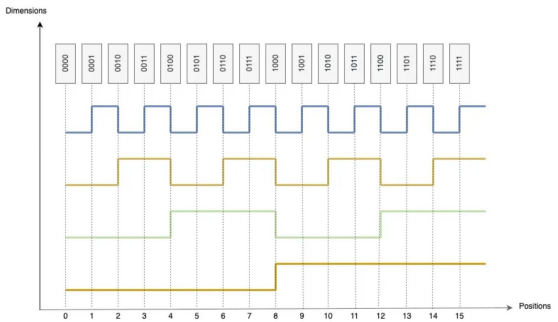
Backup slides

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Integer and binary P.E.

$$||e'_i|| \sim ||e_i|| + i$$

- Unbounded: may be giving too much relevance to position with respect to semantic information




```
Position 0  → [0, 0, 0, 0]
Position 1  → [0, 0, 0, 1]
Position 2  → [0, 0, 1, 0]
Position 3  → [0, 0, 1, 1]
...
Position 15 → [1, 1, 1, 1]
```

- Optimization likes smooth, continuous changes

Images from https://generativeai.pub/input-embeddings-positional-encoding-the-forgotten-foundations-of-transformers-e44e3f671f62

Sinusoidal P.E.

$$PosEnc(n, 2i) = sin(\frac{n}{L^{\frac{2i}{D}}})$$

$$PosEnc(n, 2i+1) = cos(\frac{n}{L^{\frac{2i}{D}}})$$

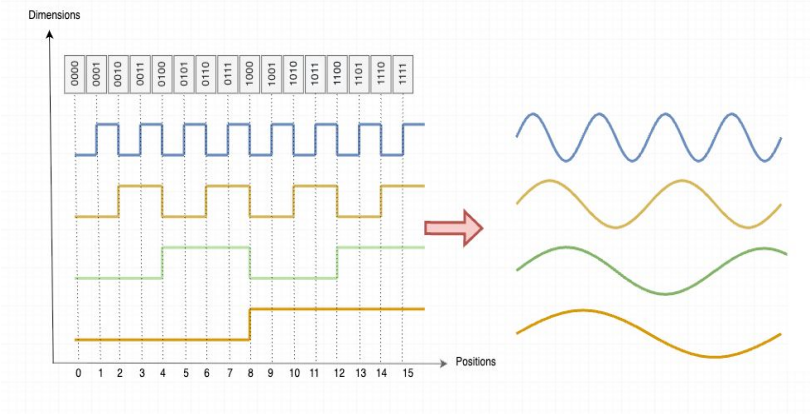
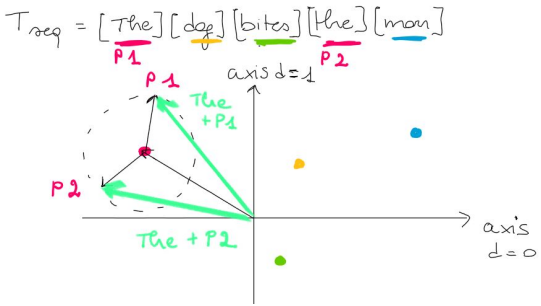
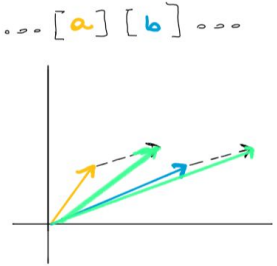
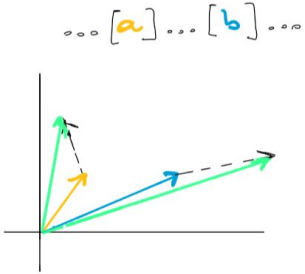


Image from https://generativeai.pub/input-embeddings-positional-encoding-the-forgotten-foundations-of-transformers-e44e3f671f62

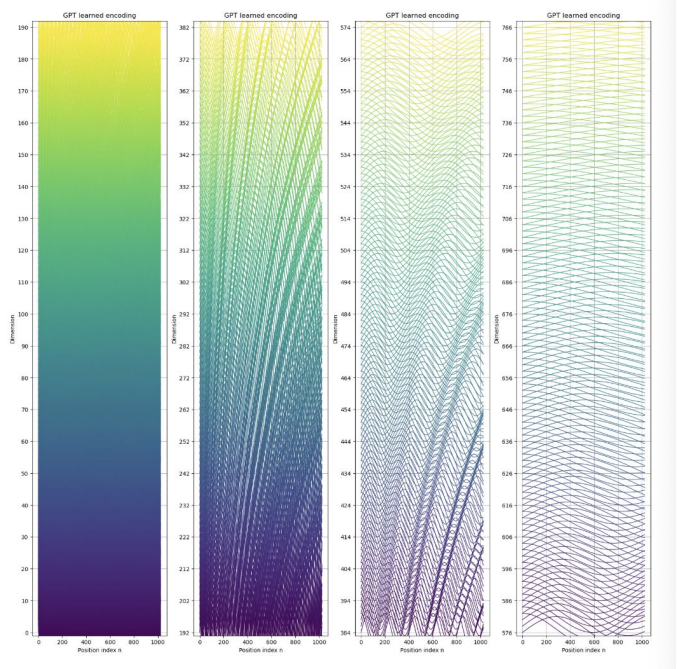


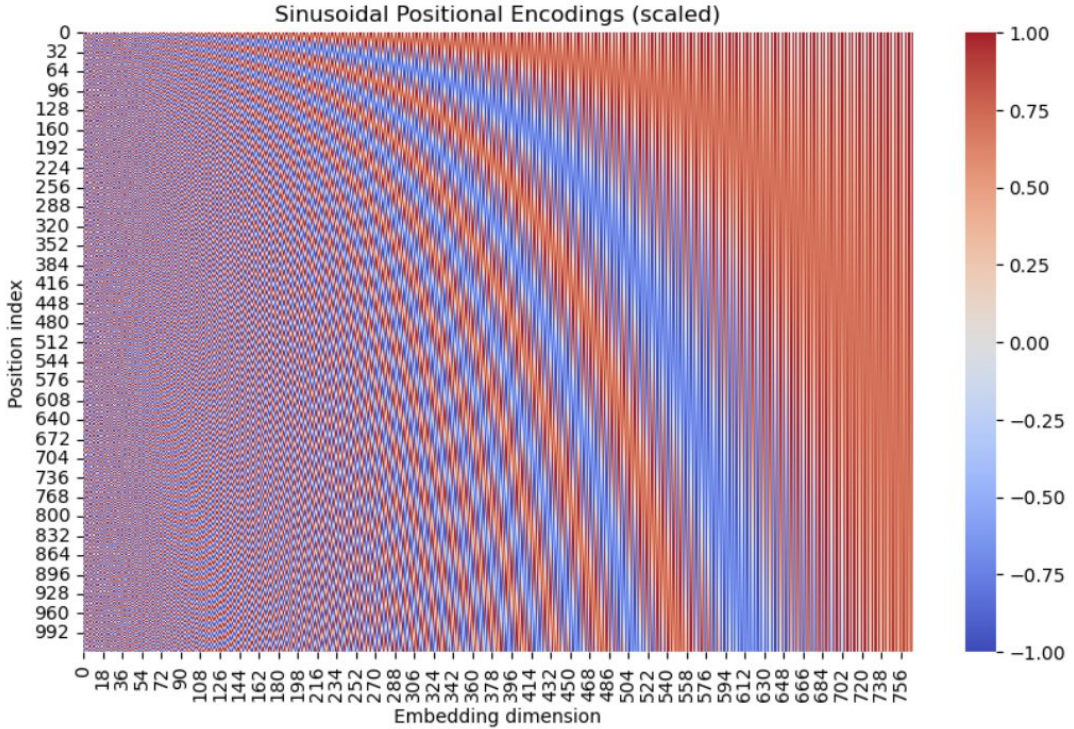




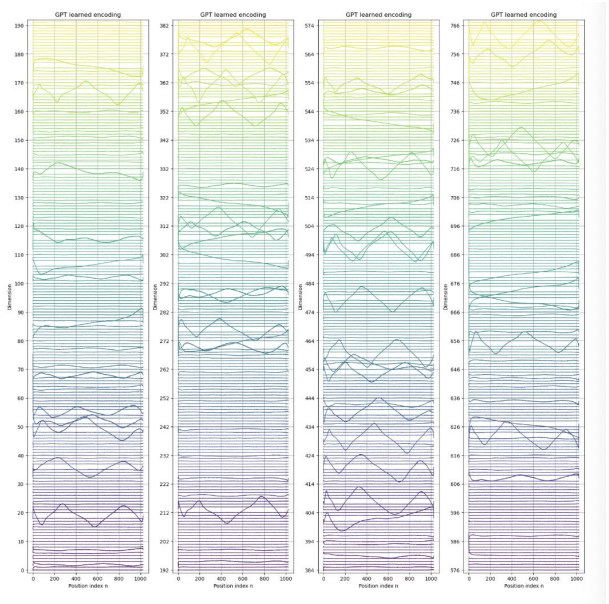
We can easily pass from one encoded position to another by means of rotations

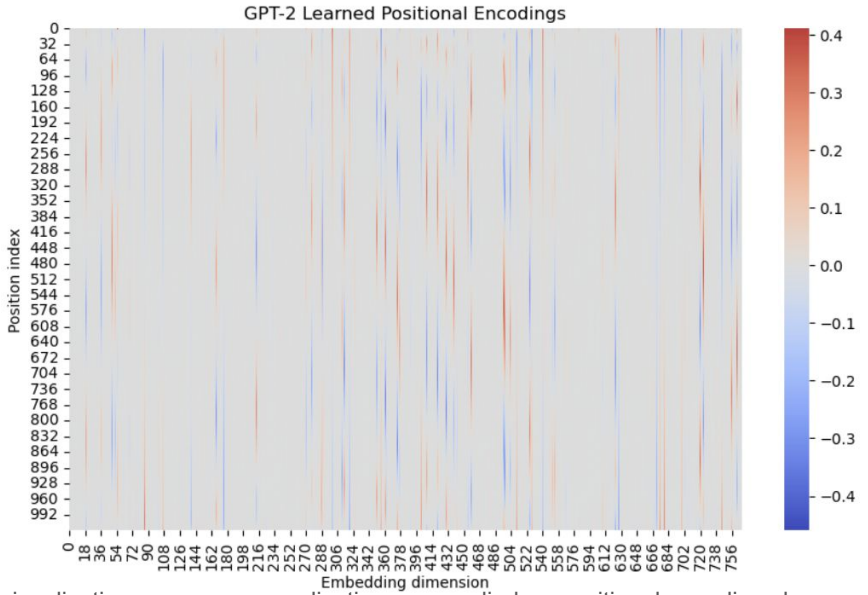
Sinusoidal P.E.





GPT-2 Learnt P.E.





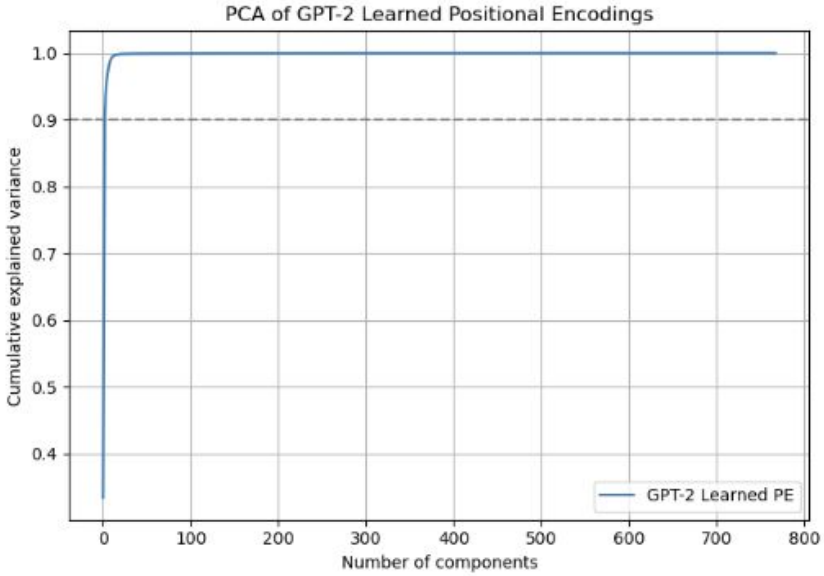
For visualization purposes normalization was applied on positional encodings here

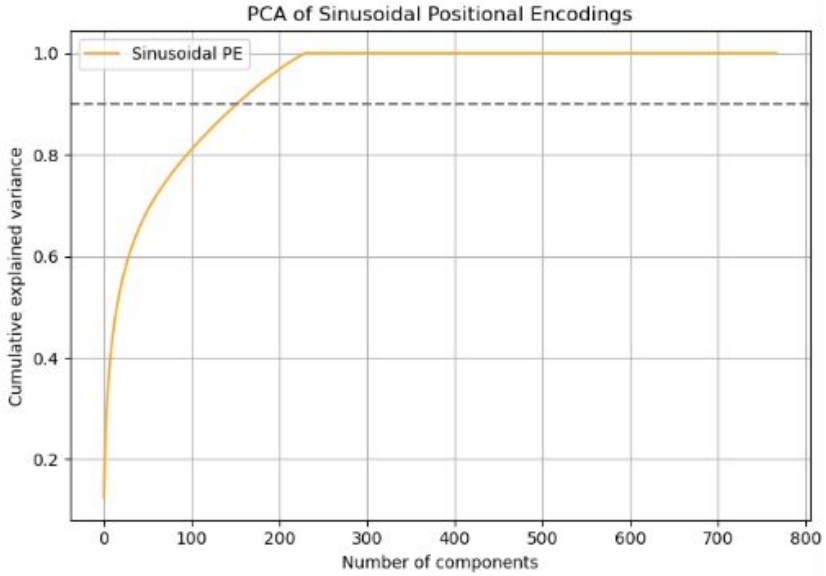
The GPT-2 P.E. matrix is very sparse: the model finds that only part of the available positional space is actually helpful for the tasks it was trained on.

PCA on P.E.

GPT-2 Learned Positional Encoding: 3 components retain 90% variance

Sinusoidal Positional Encoding: 153 components retain 90% variance





Norms of positional vectors

AS REGARDS SINUSOIDAL ENCODING:

Sinusoidal encoding is expected to have a norm of $\sqrt{(D/2)}$ for each vector. Indeed when you're computing the norm of it you're summing pairwise $cos^2 + sin^2$ and then taking square root.

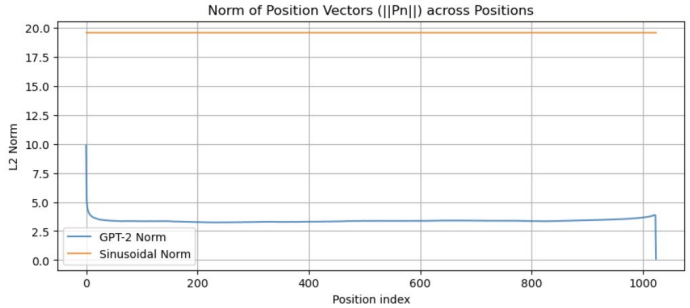
In other words, by construction each $(\sin, \cos)$ pair at index i satisfies

$$\sin^2(a_i) + \cos^2(a_i) = 1,$$

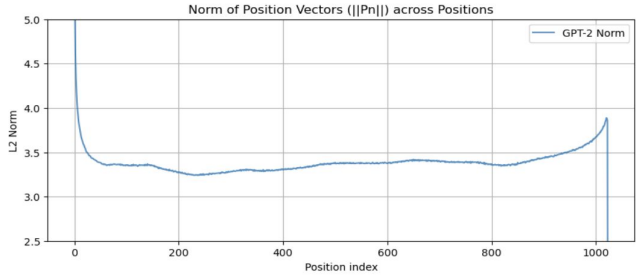
so for d-dimensional (even d) encodings,

$$\|P_n\| = \sqrt{\frac{d}{2} \times 1} = \sqrt{\frac{d}{2}}.$$

For $d = 768$, that's $\sqrt{384} \approx 19.6$.



AS REGARDS GPT-2 LEARNT ENCODING:




```
Token Embedding Norms:
  Average: 3.9585
  Std dev: 0.4337
  Max:     6.3155
  Min:     2.4537

Positional Encoding Norms:
  Average: 3.3901
  Std dev: 0.2620
  Max:     9.8756
  Min:     0.1179
```

Cosine similarity

AS REGARDS SINUSOIDAL ENCODING:

For position n, define for each i:

$$a_i = \frac{n}{L^{2i/d}}, \qquad P_n[2i] = \sin(a_i), \qquad P_n[2i+1] = \cos(a_i).$$

1. Compute the phase increment

$$\Delta_i = \frac{n+1}{L^{2i/d}} - \frac{n}{L^{2i/d}} = \frac{1}{L^{2i/d}}.$$

2. One sine–cosine pair contributes

$$\sin(a_i)\,\sin(a_i+\Delta_i) \;+\; \cos(a_i)\,\cos(a_i+\Delta_i).$$

3. Apply the trigonometric identity

$$\sin x\,\sin(x+\delta) + \cos x\,\cos(x+\delta) = \cos\big((x+\delta)-x\big) = \cos(\delta).$$

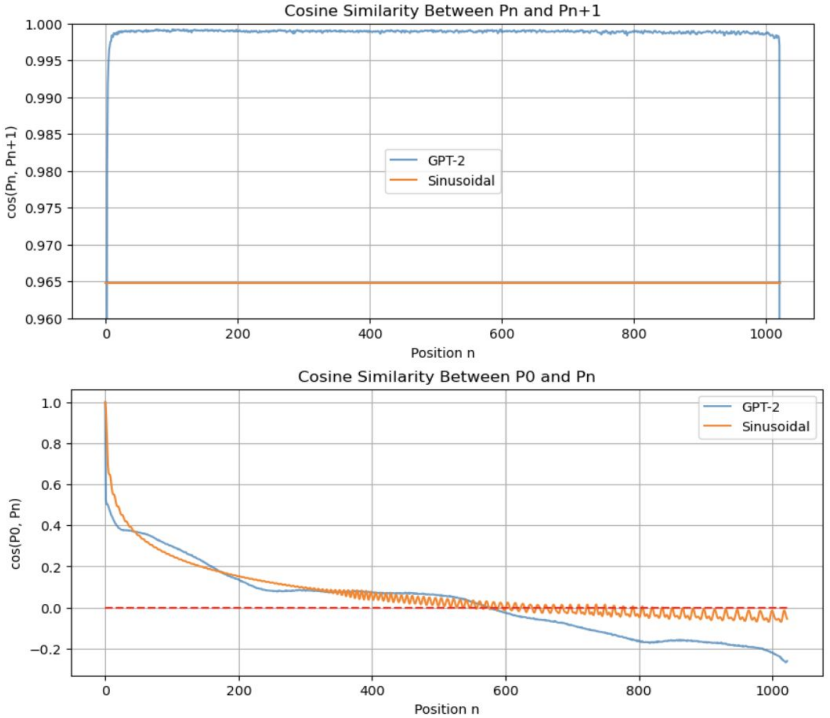
4. Summing over all $d/2$ pairs gives

$$P_n \cdot P_{n+1} = \sum_{i=0}^{d/2-1} \cos(\Delta_i),$$

which is **independent of** n.

Since every P_n has the same norm $||P_n||$, the cosine similarity is

$$\cos(P_n, P_{n+1}) = \frac{P_n \cdot P_{n+1}}{\|P_n\|\,\|P_{n+1}\|} = \frac{1}{\|P_n\|^2} \sum_i \cos(\Delta_i),$$

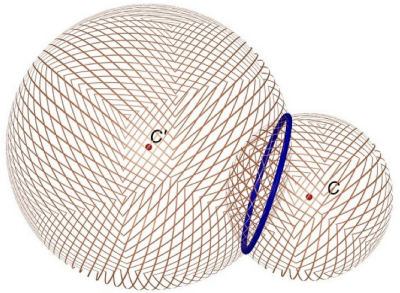


Cumulative cosine similarity shows that P.E. is NOT doing a rigid translation of the whole buffer

Encoding overlapping

WHAT IS THE RISK OF CONFUSING THE MODEL BY ADDING POSITIONAL ENCODING?

In high dimension, almost all the volume of each sphere lies in a thin shell near the surface, and even small center-to-center separations yield exponentially small overlap.



1. *In very high dimension, the volume concentrates near the surface of the sphere.*

Proof:

Consider the unit ball in $\mathbb{R}^d$:

$$B_d = \{x \in \mathbb{R}^d : ||\mathbf{x} \leq 1||\}$$

where $||\mathbf{x}||$ is the Euclidean norm.

The volume is:

$$V_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$$

Suppose we consider the shell of the ball between radius $r = 1-\epsilon$ and $r = 1$:

$$S(\epsilon) = \{\mathbf{x} \in B_d : 1-\epsilon \leq ||\mathbf{x}|| \leq 1\}$$

Since volume in d-dimensional space scales like r^d, the volume inside radius $1-\epsilon$ is:

$$V_{inner} = V_d(1-\epsilon)^d$$

Thus, the fraction of volume in the outer shell is:

$$\frac{V_{shell}}{V_d} = 1-(1-\epsilon)^d$$

If $\epsilon << 1$:

$$(1-\epsilon)^d = e^{dln(1-\epsilon)} \sim e^{-d\epsilon}$$

so:

$$\frac{V_{shell}}{V_d} \sim 1-e^{-d\epsilon} \to 1$$

as $d \to \infty$.

Encoding overlapping

1. Get the token embeddings Ti and positional encodings Pi (for 1000 random tokens)

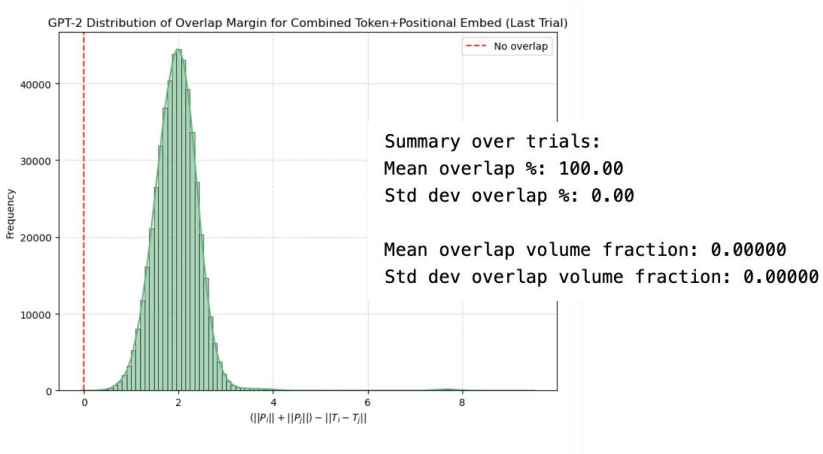
2. Compute pairwise distances: ||Ti - Tj|| (distance between sphere centers)

3. Compute the pairwise sum of radii: ||Pi|| + ||Pj||

4. Plot distribution of (sum of radii - distance) → overlap if positive

5. Repeat the extraction of Tis for 20 times and summarize the results

6. Print mean % of overlapping pairs and the fraction of overlapping volume



When $r_i = r_j = r$, the volume overlap fraction is:

$$\text{Overlap fraction} = \frac{\text{Intersection volume}}{\text{Volume of one sphere}}$$

and is given by:

$$betainc(\frac{n+1}{2}, \frac{1}{2}, 1 - (\frac{d}{2r})^2)$$

Encoding overlapping

1. Get the token embeddings Ti and positional encodings Pi (for 1000 random tokens)

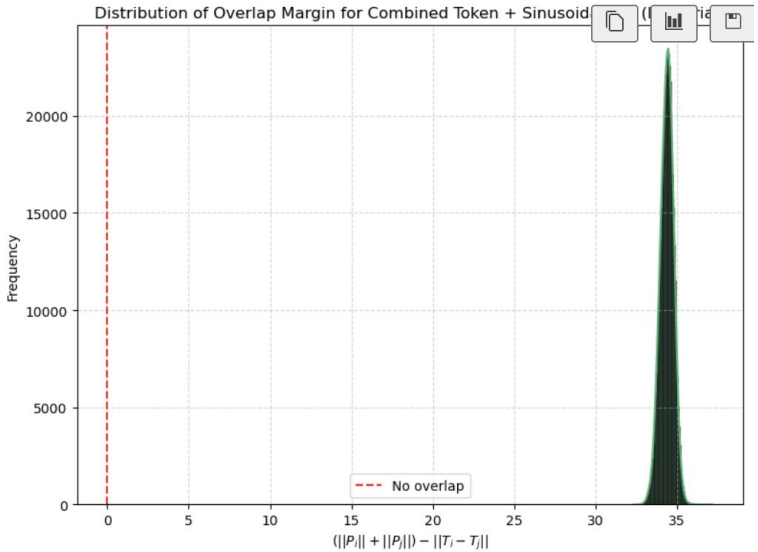
2. Compute pairwise distances: ||Ti - Tj|| (distance between sphere centers)

3. Compute the pairwise sum of radii: ||Pi|| + ||Pj||

4. Plot distribution of (sum of radii - distance) → overlap if positive

5. Print % of overlapping pairs and the fraction of overlapping volume

6. Repeat the extraction of Tis for 20 times and summarize the results



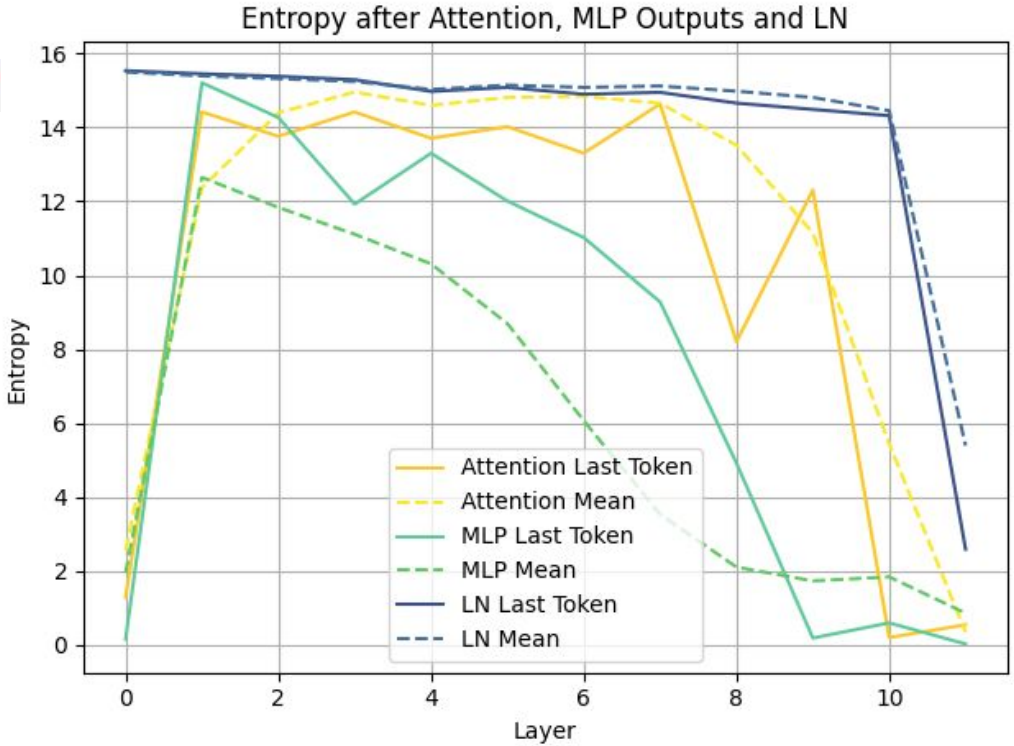

```
Summary over trials:
Mean overlap %: 100.00
Std dev overlap %: 0.00
Mean overlap volume fraction: 0.00096
Std dev overlap volume fraction: 0.00000
```

5. Functional Evolution: Predictive Dynamics

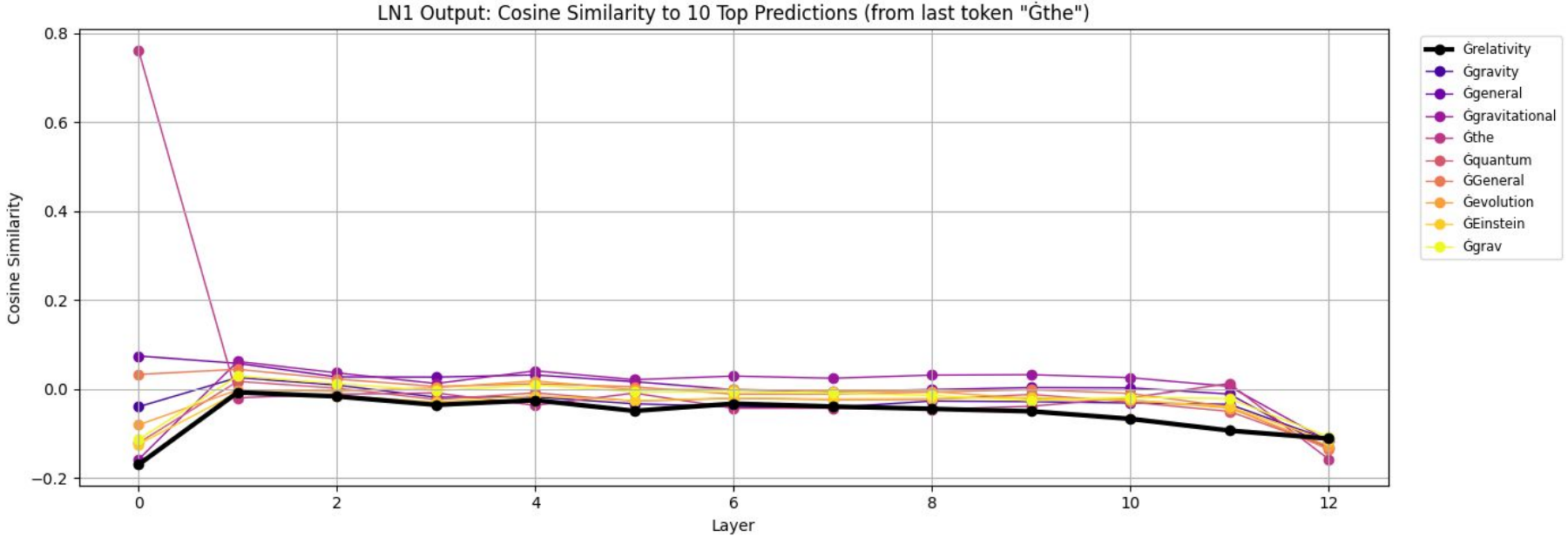
Entropy

"Einstein is best known for the theory of"

```
[LN] Layer 12: Last Token Entropy 2.5861, Mean Entropy 5.3889
   1: Ġrelativity (0.7736)
   2:   Ġgravity (0.0230)
   3:   Ġgeneral (0.0178)
   4: Ġgravitational (0.0117)
   5:       Ġthe (0.0098)
   6:   Ġquantum (0.0091)
   7:   ĠGeneral (0.0066)
   8: Ġevolution (0.0051)
   9:  ĠEinstein (0.0045)
  10:      Ġgrav (0.0037)
```



Target Outputs



Target Outputs