Recommender Systems

Giovani Tavares
giovanitavares@outlook.com

University of Sao Paulo — February 17th, 2025

Motivation

I guess I could say that recommender systems is one of the ML topics I know the fewest about and that is quite the enough motivation I need to write this post.

1 What are Recommender Systems

Recommender systems are the models that ranks a set of items I given a set of users U. In other words, a recommender system is capable of scoring each item $i \in I$ for each user $u \in U$, resulting in an $M \times N$ matrix where M is the number of users in the set U and N the number of items in the set I.

$$Y = \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1N} \\ y_{21} & y_{22} & \cdots & y_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mN} \end{pmatrix}$$

Each y_{ui} is the score the user u gives to the item i. Recommender systems are capable of filling such matrix's entries for unseen pairs of (u, i). For example, *Netflix*'s recommender system is able to score a movie you have never seen and such score is used by you to decide whether to watch or ignore such movie.

Typically such matrix will be very large, but also very sparse: evidently for a large list of items, user will usually not have interacted with most of them, which will make most entries in the matrix unknown. We can also view this sparse matrix as a **bipartite graph** (a graph that can be separateed into two disjoint sets of vertices), where the weight of a $u - i$ edge is Y_{ui}. This reflects the fact that we are dealing with **relational data**, because the values of u and i have no intrinsic meaning (they are just arbitrary indices) unless when they are connected.

The sparsity of the matrix Y can be due to two main reasons:

- **Missing At Random**: a value y_{ui} is unknown simply because the user u has not yet interacted with item i, hence has not yet had the opportunity to give a score
- **Informative Missingness**: a value y_{ui} is unknown because the user u already knows they will dislike item i, so they deliberately chose not to interact with it

For simplicity, we will consider that Y is sparse due to **missing at random**.

1.1 Collaborative Filtering

This is the original and simplest approach for making recommender systems and is based on assigning y_{ui} based on the score similar users to u would have given to i. In this sense, *collaborative* means that other users' informations on the score of an item are used to *filter* the score u would give to such item themselves. Hence, we have the following definition:

Definition 1.1 (Collaborative Filtering). *For an item i, we assing the score user u would give to it as:*

$$\hat{Y}_{ui} = \sum_{u'=Y_{u',i\neq ?}} SimilarityScore(u', u) Y_{u',i} \tag{1}$$

I.e, $\hat{Y}_{ui}$ is simply the average of the scores other users have given to item i weighted by the similarity between each of such user to u.

Even though such approach might yield pretty satisfactory results in some scenarios, it has some downsights, specially the one related to the sparsity of Y: the computation of the collabrative filtering score might not be very meaningfull if users similar to u have not given any score to i. Hence, we need better approaches that will be described in the following sections.

1.2 Matrix Factorization

1. **Periodicity:** the model trained with sinusoidal embeddings should be able to capture the relative positions of tokens effectively, which means that the distances between the embeddings of a pair of clauses should not depend on their absolute position within the longer sentence. This is achieved by making the sinusoidal embeddings functions periodic, which was to be expected from the name of the function.

2. **Unique Representation (Injective Function):** this one is straight forward from the fact that sinusoidal embeddings must represent the positions of tokens within a sentence. This means that sets of tokens in different positions should not be mapped to the same output, otherwise different positions would have the same representation.

3. **Scale Invariance:** sinusoidal embeddings should represent tokens positions consistently regardless of the sequence length. This property is crucial for handling sequences of varying lengths in a transformer model including those that are longer than the ones the model were trained on. Essentialy, the scale invariance property says that the distance of two inputs x_t and x_{t-k} should be similar among different values of t. In other words, $x_t - x_{t-k}$ should not depend on t.

4. **Linearity:** this property is somewhat related to the scale invariance property. For the Sinusoidal Embedding function to be linear is good, because functions having such property are more easily learned by neural networks. Moreover, if the Sinusoidal Embedding Function is linear we also achieve the scale invariance property. This is true, because if such Sinusoidal Embedding function (SE) is linear, for any pair of sets of tokens separated by a distance of k, say X_t, X_{t+k}, there is a linear transformation M such that $M \times SE(X_t) = SE(X_{t+k})$. Hence, in order to represent X_{t+k}, the model must only learn the linear transformation M regardless of the value of k.

1.3 How are Sinusoidal Embeddings Defined?

The author's of the *Attention is All You Need* paper define the Sinusoidal Embeddings Function (SE) like the following.

Definition 1.2 (Sinusoidal Embeddings). *For a position pos in the sequence and a dimension i (where i ranges from 0 to $\frac{d}{2} - 1$), the embedding is given by:*

$$SE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{2}$$

$$SE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{3}$$

$$\tag{4}$$

This function can be vectorially expressed as the following:

$$\mathbf{SE}(pos) = \left[\sin\left(pos \cdot e^{-\frac{2i\ln(10000)}{d}}\right), \cos\left(pos \cdot e^{-\frac{2i\ln(10000)}{d}}\right)\right]_{i=0}^{\frac{d}{2}-1} \tag{5}$$

where:

- pos is the position in the sequence.
- i is the dimension index.
- d is the dimensionality of the embeddings.

Notice that SE is essentialy a function that outputs sin values to the even dimensions of an input x_t and cos for the odd ones with exponentially decreasing frequencies.

In the next section we will use the definition of SE function to verify the validity of the previous 4 properties it should have. Some verification will be done analitically and others will be done using $Python$.

2 Properties Verification

In this section, we will be checking each of the 4 presented properties of the Sinusoidal Embeddings using analytical techniques.

2.1 Peridiocity

This peridiocity of the SE function is straight-forward. Different dimensions i and $i+k$ of an input $x_t \in \mathbb{R}^d$ with position pos will have the same values output by the sin and cos because such functions are periodic.

Info: One question that a reader might have is whether the peridiocity property does not contradict the injectivity one. The answer for that is no. The peridiocity property is observed at a single dimension level, which means $SE_{(pos,2i)}$ that outputs a single value, is be periodic. The injectivity, on the other hand, is observed at the entire embedding level, which means $SE_{(pos)}$, that outputs a vector (embedding) of dimension d is injective.

2.2 Unique Representation (Injective Function)

We are gonna prove that two sequences from different positions are **not** mapped to the same vector/output, using a *proof by contradiction*. Let's assume that two sequences x_{t_1} and x_{t_2}, with different positions t_1 and t_2, respectivelly, are mapped to the same output using the previously SE function. This assumption implies the following **for any dimension i**

$$SE(x_{t_1}) = SE(x_{t_2}) \implies SE(t_1, 2i) = SE(t_2, 2i) \text{ and } SE(t_1, 2i+1) = SE(t_2, 2i+1) \tag{6}$$

$$\implies \sin\left(\frac{t_1}{10000^{\frac{2i}{d}}}\right) = \sin\left(\frac{t_2}{10000^{\frac{2i}{d}}}\right) \text{ and } \cos\left(\frac{t_1}{10000^{\frac{2i}{d}}}\right) = \cos\left(\frac{t_2}{10000^{\frac{2i}{d}}}\right) \tag{7}$$

For the last implication to be true, either the arguments of the sin and cos functions are the exact same or they are separated by $2\pi k$. We know they are not the same, because $t_1 \neq t_2$. Therefore, we're left with the condition:

$$|\frac{t_1}{10000^{\frac{2i}{d}}} - \frac{t_2}{10000^{\frac{2i}{d}}}| = 2\pi k \tag{8}$$

$$\implies |\frac{t_1 - t_2}{10000^{\frac{2i}{d}}}| = 2\pi k \tag{9}$$

$$\implies |t_1 - t_2| = 2\pi k 10000^{\frac{2i}{d}} \tag{10}$$

$$\tag{11}$$

Since t_1 and t_2 are integers that represent the sequences positions, and $10000^{\frac{2i}{d}}$ is a positive real number, the right side of the equation $|t_1 - t_2| = 2\pi k \cdot 10000^{\frac{2i}{d}}$ must also be an integer. However, $2\pi k \cdot 10000^{\frac{2i}{d}}$ is generally not an integer because 2π is an irrational number, which leads us to a contradiction.

Hence, the initial assumption $SE(x_{t_1}) = SE(x_{t_2})$ must be false, which let's us say that **two sequences x_{t_1} and x_{t_2}, with different positions t_1 and t_2, respectivelly, are not mapped to the same output using the previously SE function.**

2.3 Linearity & Scale Invariance

As previously mentioned, the scale invariance property is a consequence of $SE(pos)$'s linearity. Hence, by proving that $SE(pos)$ is linear we also prove that such function is scale invariant.

We need to find $M \in \mathbb{R}^{d\times d}$ such that $M \times SE(x_t) = SE(x_{t+k})$. We have the following system of equations:

$$\begin{bmatrix} m_{00} & m_{01} & \cdots & m_{0(d-1)} \\ m_{10} & m_{11} & \cdots & m_{1(d-1)} \\ \vdots & \vdots & \ddots & \vdots \\ m_{(d-1)0} & m_{(d-1)1} & \cdots & m_{(d-1)(d-1)} \end{bmatrix} \begin{bmatrix} sin(\omega_0 t) \\ cos(\omega_0 t) \\ \vdots \\ sin(\omega_{(d-2)/2} t) \\ cos(\omega_{(d-2)/2} t) \end{bmatrix} = \begin{bmatrix} sin(\omega_0 (t+k)) \\ cos(\omega_0 (t+k)) \\ \vdots \\ sin(\omega_{(d-2)/2} (t+k)) \\ cos(\omega_{(d-2)/2} (t+k)) \end{bmatrix} \tag{12}$$

$$m_{00}\sin(\omega_0 t) + m_{01}\cos(\omega_0 t) + \cdots + m_{0(d-1)}\cos(\omega_{d-1} t) = \sin(\omega_0 t)\cos(\omega_0 k) + \sin(\omega_0 k)\cos(\omega_0 t) \tag{13}$$

$$m_{10}\sin(\omega_0 t) + m_{11}\cos(\omega_0 t) + \cdots + m_{1(d-1)}\cos(\omega_{d-1} t) = \cos(\omega_0 t)\cos(\omega_0 k) - \sin(\omega_0 k)\sin(\omega_0 t) \tag{14}$$

$$\vdots$$

$$m_{(d-1)0}\sin(\omega_0 t) + m_{(d-1)1}\cos(\omega_0 t) + \cdots + m_{(d-1)(d-1)}\cos(\omega_{(d-2)/2} t) = \cos(\omega_{(d-2)/2} t)\cos(\omega_{(d-2)/2} k) - \sin(\omega_{(d-2)/2} k)\sin \tag{15}$$

This system has a solution:

$$m_{00} = \cos(\omega_0 k) \tag{16}$$
$$m_{01} = \sin(\omega_0 k) \tag{17}$$
$$m_{02} = m_{03} = \ldots = m_{0(d-1)} = 0 \tag{18}$$
$$m_{10} = -\sin(\omega_0 k) \tag{19}$$
$$m_{11} = \cos(\omega_0 k) \tag{20}$$
$$m_{12} = m_{13} = \ldots = m_{1(d-1)} = 0 \tag{21}$$
$$m_{22} = \cos(\omega_1 k) \tag{22}$$
$$m_{23} = \sin(\omega_1 k) \tag{23}$$
$$m_{20} = m_{21} = m_{24} = \ldots = m_{2(d-1)} = 0 \tag{24}$$
$$m_{32} = -\sin(\omega_1 k) \tag{25}$$
$$m_{33} = \cos(\omega_1 k) \tag{26}$$
$$m_{30} = m_{31} = m_{34} = \ldots = m_{3(d-1)} = 0 \tag{27}$$
$$\vdots$$
$$m_{(d-2)(d-2)} = \cos(\omega_{(d-2)/2} k) \tag{28}$$
$$m_{(d-2)(d-1)} = \sin(\omega_{(d-2)/2} k) \tag{29}$$
$$m_{(d-2)0} = m_{(d-2)1} = m_{(d-2)2} = \ldots = m_{(d-2)(d-3)} = 0 \tag{30}$$
$$m_{(d-1)(d-2)} = -\sin(\omega_{(d-2)/2} k) \tag{31}$$
$$m_{(d-1)(d-1)} = \cos(\omega_{(d-2)/2} k) \tag{32}$$
$$m_{(d-1)0} = m_{(d-1)1} = m_{(d-1)2} = \ldots = m_{(d-1)(d-3)} = 0 \tag{33}$$
$$\tag{34}$$

Which lets us define M as:

$$M = \begin{bmatrix} \cos(\omega_0 k) & \sin(\omega_0 k) & 0 & 0 & \cdots & 0 & 0 \\ -\sin(\omega_0 k) & \cos(\omega_0 k) & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos(\omega_1 k) & \sin(\omega_1 k) & \cdots & 0 & 0 \\ 0 & 0 & -\sin(\omega_1 k) & \cos(\omega_1 k) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos(\omega_{(d-2)/2} k) & \sin(\omega_{(d-2)/2} k) \\ 0 & 0 & 0 & 0 & \cdots & -\sin(\omega_{(d-2)/2} k) & \cos(\omega_{(d-2)/2} k) \end{bmatrix} \tag{35}$$

We found a matrix $M \in \mathbb{R}^{d \times d}$ such that $M \times SE(x_t) = SE(x_{t+k})$. Hence, SE is a linear function. Moreover, notice how **M does not depend on t**, only on k. This is what gives us the scale invariability property.

3 Visualizing the Sinuspoidal Embeddings Properties With *Python*

In this section, we will write some $Python$ functions and classes to visualize the 4 cited properties of Sinusoidal Embeddings. Visualization is a great way to grasp such function's behaviour without having to necessarily prove it (even though you can always come back to this post with you're interested in the proofs).

3.1 Sinusoidal Embedding Definition With Pytorch

The sinusoidal embeddings module will store a multi-embedding tensor with $shape = (max\_pos, embed\_dim)$, where max_pos represents the maximum position we are interested in representing. This way, each of such tensor's row represents a the sinusoidal embedding for a position pos such that $SE(pos), 0 \leq pos < max\_pos$.

```python

import pytorch.nn as nn

class SinusoidalEmbeddings(nn.Module):

        def __init__(self, max_pos:int, embed_dim: int):
                super().__init__()
                # Returns a tensor with shape (time_steps, 1).
                positions = torch.arange(max_pos).unsqueeze(1).float()

                # Creates a tensor with shape (embed_dim//2,). We just need
                # half of the dimensions of the input embeddings to compute all
                # of the sinusoidal embeddings frequencies
                dimensions = torch.arange(start = 0, end = embed_dim, step = 2).float()

                # Compute the frequencies vector
                frequencies = torch.exp(dimensions * -(math.log(10000.0) / embed_dim))

                # Initialize the embeddings tensor with shape (time_steps, embed_dim)
                embeddings = torch.zeros(time_steps, embed_dim, requires_grad=False)

                # Apply sin to even indices (0, 2, 4, ...) of the input embeddings
```

```
                embeddings[:, 0::2] = torch.sin(positions * frequencies)

                # Apply cos to odd indices (1, 3, 5, ...) of the input embeddings
                embeddings[:, 1::2] = torch.cos(positions * frequencies)

                self.embeddings = embeddings

        def forward(self, x, t):
                embeds = self.embeddings[t].to(x.device)
                return embeds[:, :, None, None]
```
Listing 1: Sinusoidal Embedding Module Definition

3.2 Sinusoidal Embeddings Periodicity

As previously mentioned, the periodicity of Sinusoidal Embeddings is observed in each of its dimensions. Hence, we need to plot its dimensions values for different positions to see their periodic behavior.

```

max_pos = 100

# Our embeddings will only have 4 dimension
embed_dim = 4
sinusoidal_embeddings = SinusoidalEmbeddings(max_pos, embed_dim)

# Generate embeddings for a range of time steps
embeddings = sinusoidal_embeddings.embeddings

# Convert embeddings to numpy for plotting
embeddings_np = embeddings.numpy()

# Plot the sunosoidal embeddings for different time steps
plt.figure(figsize=(14, 8))
for i in range(embed_dim):
        plt.plot(embeddings_np[:, i], label=f"Dim {i} (i = {i//2})")

plt.title("SE(pos, 2i)")
plt.xlabel("pos")
plt.ylabel("Value")
plt.legend(loc="upper right", bbox_to_anchor=(1.15, 1))
```
Listing 2: Generating the plot of the embedding's dimensions for in different positions

As noticed in the plot above, the value of the function $SE(pos, 2i)$ defined in 1.2 repeats itself in a frequency that is inversely proportional to i, which is the dimension being represented (an index in the multidimensional embedding). As a consequence, **the higher the dimension of our model's embeddings (previously called d and called $embed\_dim$ in the Sinusoidal Embedding module), the less SE values vary.**

Intuitively, such consequence means that embeddings close to each other (they represent sentences in not very distant positions), will have their differences captured in lower dimensions, because their higher dimensions are likely to be very similar. The exact opposite is true for embeddings that represent sentences that are far away from each other. Let's visualize that by plotting different embeddings' heatmaps.

```

import torch

# We'll create embeddings with many dimensions to better see the
# frequency decay effect
```

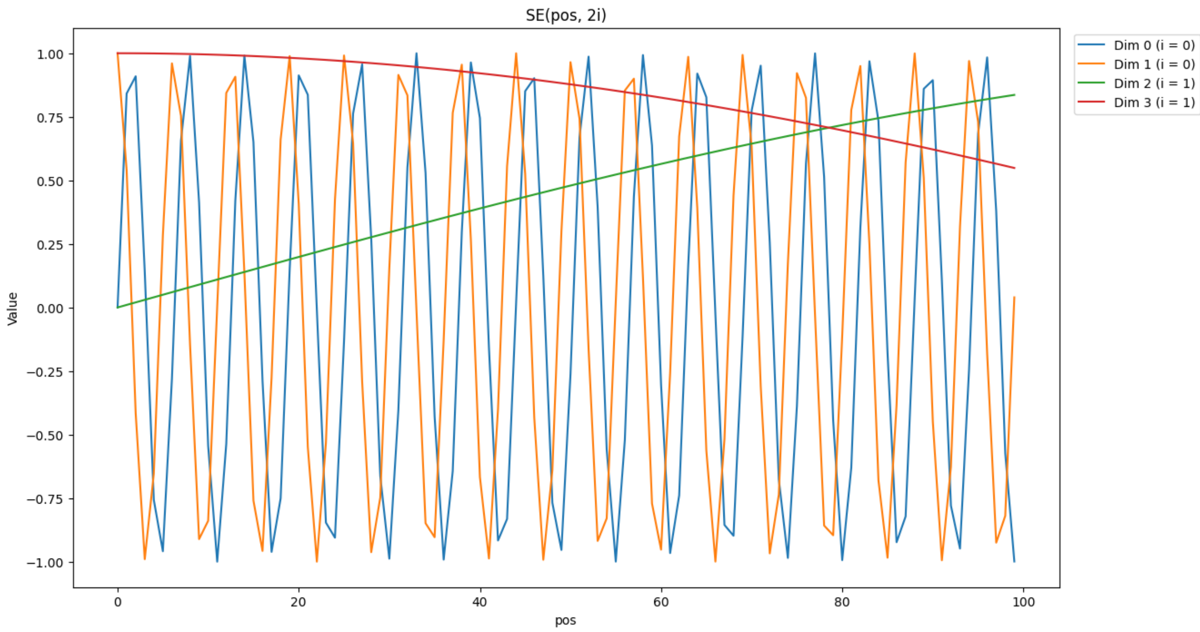

Figure 1: The embedding's dimensions values for the SE function

```
max_pos = 100
embed_dim = 128
sinusoidal_embeddings = SinusoidalEmbeddings(max_pos, embed_dim)

x = torch.zeros(embed_dim)
results = torch.zeros(max_pos, embed_dim)
for pos in range(max_pos):
results[pos] =  x + sinusoidal_embeddings.embeddings[pos]

tensor_np = results.numpy()
# Plot the heatmap using matplotlib
plt.figure(figsize=(16, 6))
plt.imshow(tensor_np, aspect='auto', cmap='RdBu')
plt.colorbar(label='')
plt.xlabel('2i')
plt.ylabel('pos')
plt.title('SE(pos, 2i)')
plt.show()
```

Listing 3: Generating the plot of the sinusoidal embeddings for different positions

In the plot above, we see that embeddings with close positions (two close values in the pos axis) have different values of $SE(pos, 2i)$ for very small $2i$ (lower dimensions) while their values for higher dimensions are similar. On the other hand, if we pick two embeddings with very distant pos, they might differ from each other only in higher dimension. What this means is that in order to represent longer sentences (that contain positions very distant from each other), our model needs to have more dimensions. **The longer the input sentences, the higher the model's dimensions need to be.**

3.3 Unique Representativiness (SE is an injective function)

Figure 2 shows us that no two rows are the same because of the exponential decay of the frequencies with the increase of the dimension (increase of $2i$). As each row represents embeddings with different positions, what the figure is essentialy showing us is that two embeddings with different positions will never have the same representation, i.e., SE is injective.

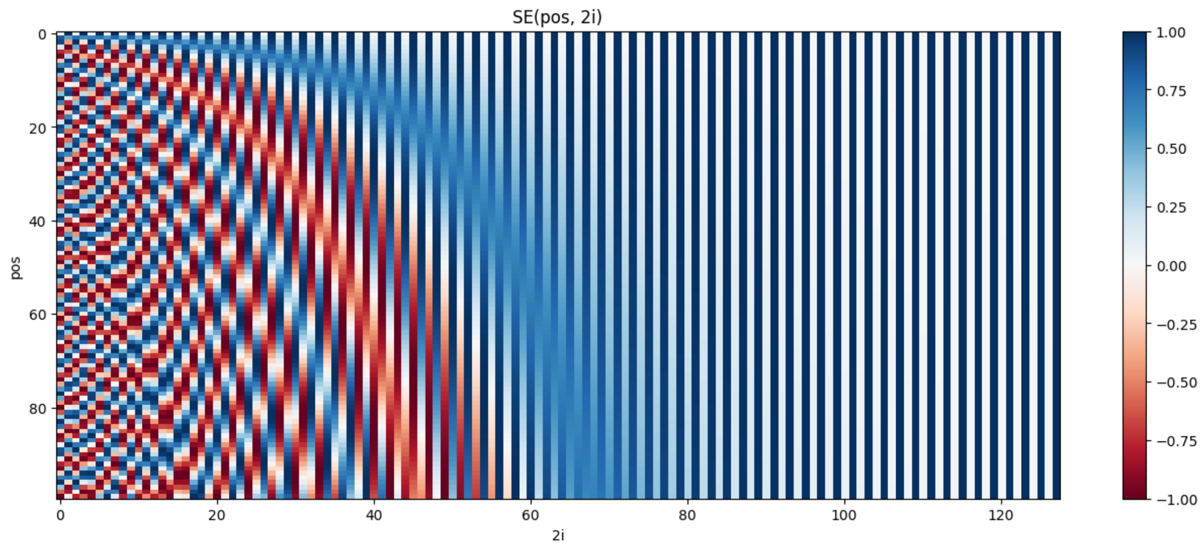

Figure 2: The SE function's frequency gets lower as the dimension grows

3.4 Linearity & Scale Invariance

As previously mentioned, to be scale invariant, SE must be such that the distance between $SE(pos)$ and $SE(pos+k)$ must be the same as the one between $SE(0)$ and $SE(k)$. In order to check that, we'll calculate the distance between every two sinusoidal embeddings of our module and see the linear property of such distance. Hence, all the rows will have to be subtracted from the first one, from the second one and so on up until the last one.

```python

import torch

# We'll create embeddings with many dimensions to better see the
# frequency decay effect
max_pos = 1000
embed_dim = 1000


sinusoidal_embeddings = SinusoidalEmbeddings(max_pos, embed_dim).embeddings

# A single column tensor where each row's single element contains an
# entire sinusoidal embeddings tensor
T_2 = sinusoidal_embeddings[:, None, :]

# A single row tensor where column's single element contains an
# entire sinusoidal embeddings tensor
T_1 = sinusoidal_embeddings[None, :, :]

# By broadcasting, this operation will save the desired differences
# in the tensor's last dimension
differences = T_2 - T_1

# As the differences were saved in the last dimension, we need
# to calculate the norm with respect to it, which is indicated
# by dim=-1
distances = torch.norm(differences, p=2, dim=-1)

```

```
# Plot the resulting 2D tensor as a heatmap
plt.figure(figsize=(8, 6))
plt.imshow(distances.numpy(), aspect="auto", cmap="PuRd")
plt.colorbar(label="L2 Norm (Euclidean Distance)")
plt.xlabel("pos")
plt.ylabel("pos")
plt.title("Euclidean Distances Between Different Position Sinusoidal
    Embeddings")
plt.show()
```

Listing 4: Generating the plot of the module of the difference between every two different positions sinusoidal embeddings

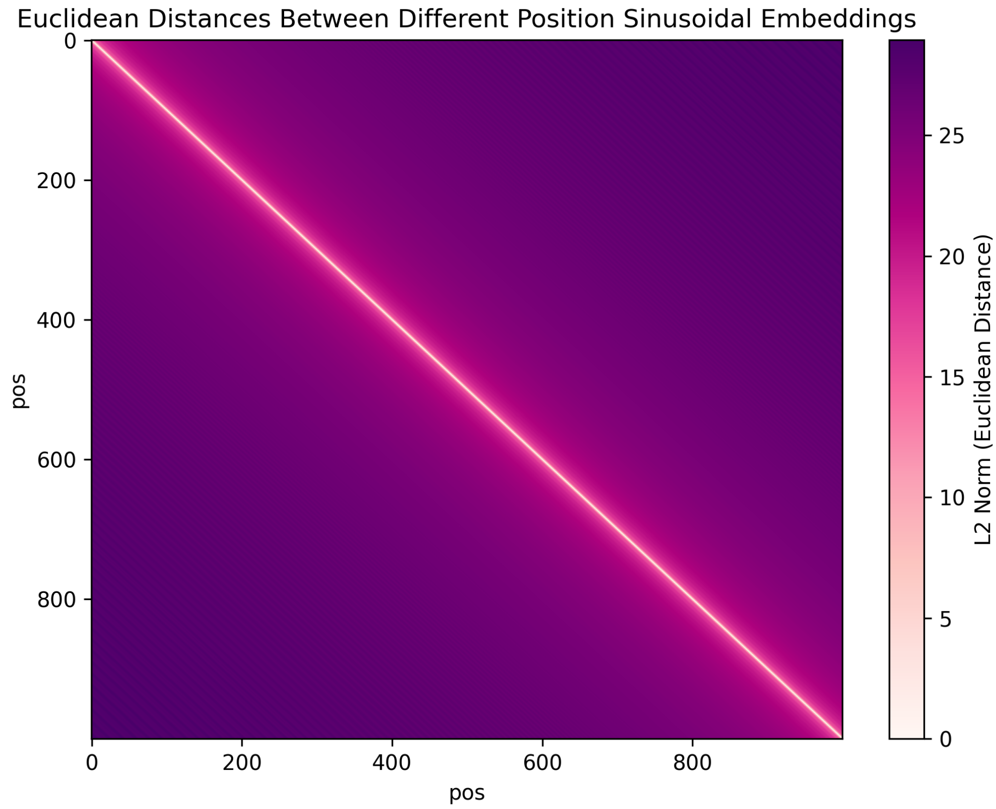

Figure 3: The sinusoidal embeddings difference vector's modules decrease linearly with the distance between the subtracted vectors

Figure 3 above shows us how the distance between sinusoidal embeddings of different positions decreases smoothly and linearly for embedidding with $d = 1000$.

Question 1

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

(a) Do this.

(b) Do that.

(c) Do something else.

3.5 Algorithmic issues

In malesuada ullamcorper urna, sed dapibus diam sollicitudin non. Donec elit odio, accumsan ac nisl a, tempor imperdiet eros. Donec porta tortor eu risus consequat, a pharetra tortor tristique. Morbi sit amet laoreet erat. Morbi et luctus diam, quis porta ipsum. Quisque libero dolor, suscipit id facilisis eget, sodales volutpat dolor. Nullam vulputate interdum aliquam. Mauris id convallis erat, ut vehicula neque. Sed auctor nibh et elit fringilla, nec ultricies dui sollicitudin. Vestibulum vestibulum luctus metus venenatis facilisis. Suspendisse iaculis augue at vehicula ornare. Sed vel eros ut velit fermentum porttitor sed sed massa. Fusce venenatis, metus a rutrum sagittis, enim ex maximus velit, id semper nisi velit eu purus.

Algorithm 1: `FastTwoSum`

Input: (a, b), two floating-point numbers
Result: (c, d), such that $a + b = c + d$

if $|b| > |a|$ **then**
    exchange a and b ;
end
$c \leftarrow a + b$;
$z \leftarrow c - a$;
$d \leftarrow b - z$;
return (c, d) ;

Fusce varius orci ac magna dapibus porttitor. In tempor leo a neque bibendum sollicitudin. Nulla pretium fermentum nisi, eget sodales magna facilisis eu. Praesent aliquet nulla ut bibendum lacinia. Donec vel mauris vulputate, commodo ligula ut, egestas orci. Suspendisse commodo odio sed hendrerit lobortis. Donec finibus eros erat, vel ornare enim mattis et.

Question 2 *(with optional title)*

In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.

Mauris interdum porttitor fringilla. Proin tincidunt sodales leo at ornare. Donec tempus magna non mauris gravida luctus. Cras vitae arcu vitae mauris eleifend scelerisque. Nam sem sapien, vulputate nec felis eu, blandit convallis risus. Pellentesque sollicitudin venenatis tincidunt. In et ipsum libero. Nullam tempor ligula a massa convallis pellentesque.

4 Implementation

Proin lobortis efficitur dictum. Pellentesque vitae pharetra eros, quis dignissim magna. Sed tellus leo, semper non vestibulum vel, tincidunt eu mi. Aenean pretium ut velit sed facilisis. Ut placerat urna facilisis dolor suscipit vehicula. Ut ut auctor nunc. Nulla non massa eros. Proin rhoncus arcu odio, eu lobortis metus sollicitudin eu. Duis maximus ex dui, id bibendum diam dignissim id. Aliquam quis lorem lorem. Phasellus sagittis aliquet dolor, vulputate cursus dolor convallis vel. Suspendisse eu tellus feugiat, bibendum lectus quis, fermentum nunc. Nunc euismod condimentum magna nec bibendum. Curabitur elementum nibh eu sem cursus, eu aliquam leo rutrum. Sed bibendum augue sit amet pharetra ullamcorper. Aenean congue sit amet tortor vitae feugiat.

In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.

hello.py

```python
#! /usr/bin/python

import sys
sys.stdout.write("Hello World!\n")
```

Fusce eleifend porttitor arcu, id accumsan elit pharetra eget. Mauris luctus velit sit amet est sodales rhoncus. Donec cursus suscipit justo, sed tristique ipsum fermentum nec. Ut tortor ex, ullamcorper varius congue in, efficitur a tellus. Vivamus ut rutrum nisi. Phasellus sit amet enim efficitur, aliquam nulla id, lacinia mauris. Quisque viverra libero ac magna maximus efficitur. Interdum et malesuada fames ac ante ipsum primis in faucibus. Vestibulum mollis eros in tellus fermentum, vitae tristique justo finibus. Sed quis vehicula nibh. Etiam nulla justo, pellentesque id sapien at, semper aliquam arcu. Integer at commodo arcu. Quisque dapibus ut lacus eget vulputate.

Command Line

```
$ chmod +x hello.py
$ ./hello.py

Hello World!
```

Vestibulum sodales orci a nisi interdum tristique. In dictum vehicula dui, eget bibendum purus elementum eu. Pellentesque lobortis mattis mauris, non feugiat dolor vulputate a. Cras porttitor dapibus lacus at pulvinar. Praesent eu nunc et libero porttitor malesuada tempus quis massa. Aenean cursus ipsum a velit ultricies sagittis. Sed non leo ullamcorper, suscipit massa ut, pulvinar erat. Aliquam erat volutpat. Nulla non lacus vitae mi placerat tincidunt et ac diam. Aliquam tincidunt augue sem, ut vestibulum est volutpat eget. Suspendisse potenti. Integer condimentum, risus nec maximus elementum, lacus purus porta arcu, at ultrices diam nisl eget urna. Curabitur sollicitudin diam quis sollicitudin varius. Ut porta erat ornare laoreet euismod. In tincidunt purus dui, nec egestas dui convallis non. In vestibulum ipsum in dictum scelerisque.

Notice: In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.