Convolution LLM and void zones

“The void is the threshold of creation, full of the new, where the silence hums with potential and the unseen becomes the birthplace of all that is yet to be.” J. Campbell

Lyubimkov Dmitri

E-mail: loftlong@gmail.com

2025

Abstract

This article introduces a novel approach to knowledge discovery using a **Convolution Large Language Model (CLLM)** and the exploration of **void zones** in embedding space. The CLLM, built on a pre-trained transformer-based LLM, reduces high-dimensional token embeddings into compact text embeddings, enabling the identification of void zones—regions in the embedding space where no training data resides. By feeding embeddings from these void zones into the CLLM decoder, the system generates new text, potentially uncovering previously unseen knowledge or ideas. The article outlines the architecture of CLLM, emphasizing its role as an autoencoder that distills knowledge from token embeddings. A practical testing framework using culinary recipes is proposed to validate the method. If successful, this approach could revolutionize research and innovation by providing a systematic, computational mechanism for generating new knowledge. However, challenges such as the feasibility of void zones, dimensionality trade-offs, and the quality of generated outputs remain open questions. The article concludes by discussing the broader implications of this method, including its potential for AI self-improvement and cross-disciplinary applications.

Table of contents

1. Introduction

This article consists of two parts. The idea of void zones and how they can be used are described in the second part of the article. The first part about the Convolution Large Language Model (hereinafter CLLM) offers a possible scenario for preparing a neural network to find and use void zones. Another method or combination of methods can be used to prepare CLLM. The void zones themselves may also not exist or will not work in the way we need, but this can only be verified by preparing and testing CLLM.

2. CLLM

2.1. Initial LLM

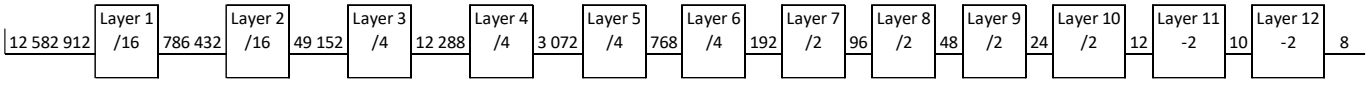
To prepare CLLM, you need a trained LLM based on transformers. This must be an LLM using both an encoder and a decoder. An LLM without an encoder will not create embeddings, and an LLM without a decoder will not allow the result to be verified and used. You can use any LLM, provided that you have access to its weight values. LLM after pre-training will be enough, instruction tuning and RLHF stages are not required and may even be a hindrance. Suppose the original LLM has a context size of 8192 tokens. Suppose each token generates embeddings of dimension 1536. This is a modern LLM at the time of writing. Thus, the input text generates up to 8192 * 1536 = 12,582,912 values (a matrix of size 8192 x 1536). This is a matrix of the meaning of tokens, their coordinates in the space of meanings.

2.2. CLLM

From the meaning of individual tokens, we need the coordinates of the meaning of the text as a whole. By CLLM, the author means a neural network that transforms tokens embedding into embedding of the text as a whole, significantly, very significantly reducing the dimensionality. The word “Convolution” does not mean the method used, but that the matrix of token meanings has collapsed into a vector of text meanings. The reader will say that this is impossible for several reasons at once. There are incredibly many meanings of texts and embedding for the text as a whole will be of large dimensionality and this will require computing resources comparable, and perhaps even greater than for training basic LLM. But for our purposes, CLLM will be trained on a limited corpus of texts, on a very limited one, on texts devoted to some one topic or several related topics. This will not require significant computing resources and will possibly reduce embeddings to a small dimensionality.

Let's imagine that the initial embeddings go through many layers, each of which reduces the dimensionality. If each layer reduces the dimensionality by only 2 times, then only 20 iterations will be required to reduce the dimensionality from 12,582,912 to 12. If each layer reduces the dimensionality by 4 times, then 9 iterations will be required. If each layer reduces the dimensionality by 16 times, then only 5 iterations will be required. It is currently impossible to predict what the dimensionality of the final embeddings vector should be. Moreover, this dimensionality will be different for different text corpora. Therefore, different layers will be needed, from reducing the dimensionality many times at the beginning of the chain to reducing the dimensionality by only a few units (up to 1) at the end of the entire chain. The vocabulary of modern LLMs is tens of thousands of tokens. We know that LLMs work well with such dimensionalities, so the task of the first layers is to reduce the dimensionality from tens of millions to tens of thousands, which is about 250 times. These are two layers of 16 or one of 256. The first layers can be made according to an architecture different from the rest to reduce the amount of calculations.

The following figure shows what it might look like to reduce the embeddings dimension to 8 in just 12 iterations.



2.3. CLLM layer architecture

There are different ways to reduce the dimensionality of LLM. The author thinks that an autoencoder based on transformers is best suited for our task. Firstly, we need both an encoder and a decoder to restore the original embeddings, secondly, transformers are well suited for finding dependencies and extracting knowledge. And we need to get distilled knowledge. The difference is that regular LLMs extract “knowledge” from a sequence of tokens, and CLLMs from a sequence of embeddings.

Thus, each CLLM layer consists in turn of several transformer layers. The number of layers should be determined experimentally. Modern LLMs have 100+ transformer layers. The number of CLLM layers is ~ 10. The author believes that several transformers per CLLM layer will be enough, so that the total number of transformers will be several dozen.

If the very first layers cannot be organized according to the transformer architecture due to the volume of calculations, then they can be replaced with whatever works, even a perceptron. Perhaps this will also work.

3. Void zones

3.1. Trained CLLM

So, we have the original LLM, which transforms a text up to the maximum allowed context size into an array of embeddings. Having run a small set of texts on a certain topic through this original, we obtained a set of such arrays for training CLLM. After that, we trained CLLM as an autoencoder, the encoder of which reduces the dimensionality by transforming embeddings tokens into embeddings of the entire text with a small dimensionality. The decoder, in turn, is able to restore the original embeddings array. From which the decoder of the original LLM is able to restore the original text/tokens.

After the CLLM is trained, we run it on the entire text corpus used for training and store them in vector database.

3.2. Zones without embeddings or void zones

Let's now look at the resulting vector database with text embeddings. It may turn out, the author hopes that it will turn out, that in N-dimensional space the coordinates of these vectors are not evenly distributed. They can form some areas of concentration. And they can also form voids - zones where no embeddings fall. We will call such zones void zones.

What happens if we feed embeddings from void zones or at least from areas near embeddings existing in the vector database to the CLLM decoder input? The decoder will restore this vector to array of token embeddings. Now let's feed this sequence of token embeddings to the input of the original LLM decoder to get a human-readable text. What will be in this text? Some kind of nonsense? A repetition of an existing text? Or new, previously unseen texts with new knowledge and ideas?

The basic idea of searching for void zones is that when convolving embeddings, transformer layers try to find connections and dependencies between tokens of embeddings. However, not all variants of the resulting vectors are found in the source texts. By selecting embeddings from void zones, we select possible combinations of connections and dependencies that were learned from the training dataset, but have never been encountered before. What is this if not new knowledge and ideas based on already learned knowledge and ideas? Some embeddings from void zones will lead to duplicates of existing texts. Some may generate meaningless text. And only some will give new knowledge. Finding such interesting zones and embeddings can be an additional difficult task. But the reward for success will be a universal mechanism for finding something new not with the help of reasoning or intuition, but with the help of only calculations according to a clear algorithm and clear criteria.

3.3. Research and inventions based on void zones

If the presented idea works, it will change or significantly complement the way humanity currently conducts research and invents new things. And this does not necessarily apply to some technical inventions. Everything depends on what text corpus we build CLLM on, in which we then look for void zones. If we use descriptions of goods from some stores, we can end up with descriptions of new, not yet existing goods. If we build CLLM on texts from several subject areas, it may be possible to obtain new knowledge at the intersection of several areas.

The void zones method is not a magic wand, it is just another tool that did not exist before. For its success, it is necessary to correctly select a corpus of texts for training CLLM, learn to search for such void zones and filter the obtained results by their usefulness. Its strength is in its versatility - we take a large LLM, train a thematic CLLM, collect vectors from void zones, and select the most suitable texts obtained from them. The found and selected new data can be used to retrain the basic LLM. And then repeat the process. If we exclude a person from this chain, we get a process of self-development and self-improvement of AI. A large neural network trains small networks that search for new knowledge, the most useful of which are selected by another neural network. And on the resulting new corpus of texts (or even on arrays of token embeddings without translation into text), AI retrains itself.

3.4. CLLM dimension

In order to compute something new, void zones must first arise. If the CLLM dimension (the size of the resulting embedding) is too large, there will be too much empty space for trial embeddings. If the dimension is too small, the decoder will lose the ability to correctly restore the original token embeddings and/or the entire space of the resulting CLLM embedding will be filled and void zones will not arise. The dimension for each text corpus must be selected individually. As such neural networks are researched and used, an empirical formula for the embedding vector size may be found depending on the size of the original text corpus, the number of tokens in it, and their distribution.

It is possible that when training CLLM the resulting vectors will be distributed uniformly, preventing the void zones method from being applied. In this case, another term will need to be added to the lost function, which would grow with a uniform distribution of vectors and decrease when the embeddings vectors tend to be closer to each other.

4. Testing

The author suggests a simple and understandable way to test the method. Let's take culinary recipes as a text corpus. They have a short length, so we can work with a small token embeddings array, which will reduce the amount of calculations. The options for ingredients, methods, and modes of their processing and serving are limited. Their combinations are also limited, so it will be possible to build the resulting CLLM. We divide all recipes into two parts. In the second part, we select one recipe, for example, all pizzas. The first part includes all recipes except those included in the second part. On the first corpus of texts, we train CLLM1, which knows nothing about the recipes from the second set of texts. Now we train CLLM1 on the texts of recipes from the second part. As a result, we get CLLM2.

Now we feed the recipes from the second part of the texts to the input of CLLM2. The resulting embeddings should not be found in CLLM1, because CLLM1 has not seen these recipes. That is, there should be emptiness in their place. Now we feed the embeddings of CLLM2 from the texts from the second set of texts to the input of the CLLM1 decoder and then feed the resulting array of token embeddings to the input of the original LLM decoder. If the author is right and we did everything correctly, then at the output we will get pizza recipes from the second set of text, despite the fact that CLLM1 trained only on the first set has never seen them.