Natural Language Processing

Lecture 17: LLM Tooling

Natabara Máté Gyöngyössy

Eötvös University, Department of Artificial Intelligence

2023

Augmenting Language Models

Motivations

From the early 2020s language models could be characterized by the following properties:

- Few-shot learners (Brown et al. 2020).
- Capable of inferring logical problems. (Chang et al. 2023)
- Prone to hallucinations (due to active knowledge gaps). (Zheng, Huang, and Chang 2023)
- Able to follow instructions in a step-by-step manner. (Wei et al. 2022)

Motivations

Knowledge can be injected in a few-shot manner, which could be interpreted to overcome hallucinations. With step-by-step processing, an augmented model can use low-complexity knowledge sources to answer complex questions.

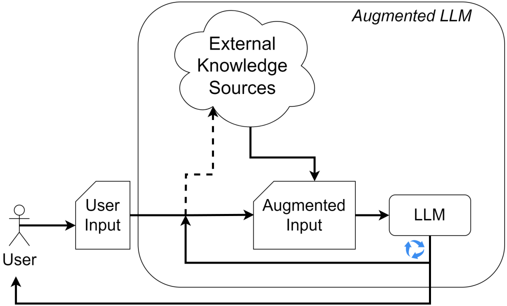

Figure 1: General augmented language model schema

Connection to prompting

There are two ways of injecting external information into a transformer-like Language Model:

- Vector sequence in the embedding space (cross-attn, prefix, etc.)
- Injecting text information to a prompt (special tokens, format etc.)

Important! Using proper prompting techniques should be considered alongside augmentation. Transformer-based models' context windows have a fixed length, which is a limitation!

Retrieval Augmentation

Retrieval

Easiest solution: Retrieval Augmented Generation (RAG).

Take an external knowledge base and query it. The resulting answer could then be utilized by the model.

Example prompt:

Answer the following question using the provided context only!

Question: <USER_INPUT>

Context: <RETRIEVED_CONTEXT>

Answer: <LLM>

How to retrieve information?

The most common methods for finding related information are:

- Keyword-based search (occurence, regex, etc.)
- **Vector-similarity search (TF-IDF, LM-embedding, etc.)**
- Relational queries
- Taxonomy-based search (lexicon, wiki, WordNet)
- Direct access (links, documents)

Vector-similarity based search methods

Let's assume that we have feature vectors (e^i) of certain documents ($i \in I$), where $||e^i||_2^2 = 1$.

The retrieval process should return the closest documents to the embedded user query e^q.

This is achieved by classical nearest-neighbor search. Assuming that $e \in \mathcal{R}^d$ and $|I| = N$ the complexity of retrieval is $O(Nd)$.

This scales hard with embedding size (quality) and the number of documents. Searching for the k nearest neighbors is the same.

Approximate nearest neighbor search

Prebuilt indices can reduce inference time, but memory and building time are still a limitation. Approximation is needed for storing and index building.

Possible solutions:

- Hashing
- Quantization
- Tree structure
- Graph-based

Hashing

Instead of returning an exact result bins are constructed with a hashing function. The family of LSH (Locality-Sensitive Hashing) functions is used as with them the probability of collision monotonically decreases with the increasing distance of two vectors.

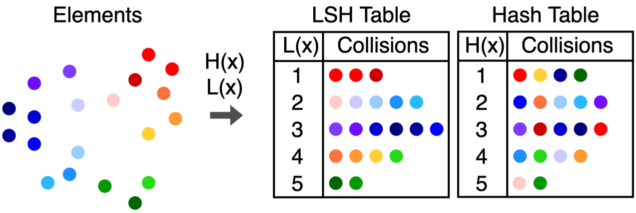

Figure 2: LSH hash properties. By Ben Coleman

Hashing

Complexity is reduced via binning. Fine-grained search is possible after finding the closest bins. For more advanced solutions refer to (Wang et al. 2021)!

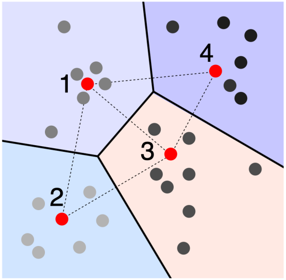

Figure 3: Using LSH clusters to ANN search. By Ben Coleman

Tree-based solutions

In tree structures, the branching factor b reduces the search complexity to $log_b(N)$.

In case of a binary KD-tree $b = 2$ a simple solution for building such a tree is just drawing a hyper-plane at the median orthogonal to the highest-variance data dimension. Then each half is split using the same principle. This continues until each node contains a single element only.

Then combined tree and embedding space search algorithms could be used to find nearest neighbors. For example: priority search.

Priority search

First, the node (or cell) containing the query is selected, then the closest neighboring tree nodes are visited bounded by a maximal embedding space distance initialized by the distance between the query and the embedding vector in the query's cell.

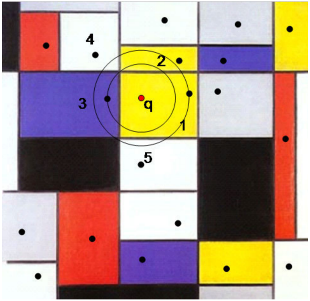

Figure 4: Geometric visualization of priority search. From (Silpa-Anan and Hartley 2008)

Quantization

Given a codebook defined by centroids $\mathcal{C} = c_i | i \in I$ where $I = 0, 1, ...m - 1$ is finite.

We map $q(.)$ each real vector to the closest centroids. The set of real vectors mapped to c_i is the Voronoi cell of it denoted by V_i.

Meaning that $q(x) = \arg\min_{c_i \in C} d(x, c_i)$, where $d(.)$ is the distance function.

$c_i = E_x[x|i] = \int_{V_i} p(x) \cdot xdx$, then should be defined as the center of the Voronoi cell.

Product Quantization

Simple quantization is still inefficient as cluster centers are to be calculated using demanding algorithms such as k-means (complexity $O(dm)$). In the case of a simple 1 bit/component 128-dimensional quantized vector, it would take $m = 2^{128}$ centroids to calculate and store.

That's too much!

Solution: We should factor the vector into multiple segments (similar to MHA).

Product Quantization

In case of a vector split into L segments, each can be quantized by its specific quantizer. That means $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \ldots \times \mathcal{C}_L$ and $I = I_1 \times I_2 \times \ldots \times I_L$ should be decomposed into the Cartesian-product of the sub-quantizers and sub-indices.

In this case the complexity is reduced to $O(dm^{\frac{1}{L}})$ according to (Jegou, Douze, and Schmid 2010).

Distances between quantized values of each segment can be calculated and stored for the search step.

Product Quantization

Using pre-computed tables of $d(c_i, c_j)$, we can easily calculate the distance of the full vectors e^i and e^q. Which, in the Euclidean distance case equals:

$$d(e^i, e^q) = d(q(e^i), q(e^q)) = \sqrt{\sum l \in Ld(q_l(e^i), q_l(e^q))}$$

This results in an average search complexity of N comparisons plus looking up and summing the corresponding distances in the L lookup tables. This boils down to $O(N + L\ logL \cdot log\ logN)$ if $N >> L$ according to (Jegou, Douze, and Schmid 2010).

Graph-based

Graph methods excel in

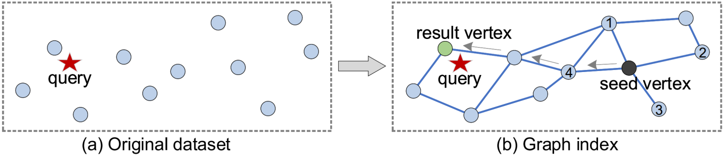

Figure 5: How graph-based ANN search works: (Wang et al. 2021)

Approximate nearest neighbor search

Approximation is needed to successfully capture the graph construction and search problem effectively.

Classical search

Embedding models

RAG

Entity-knowledge base

RAG pre-trained models (Retro-style)

Tooling

AutoGPT (Inner monologue)

API calling

Tool-finetuned models

References

References I

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. "Language Models Are Few-Shot Learners." https://arxiv.org/abs/2005.14165.

Chang, Yupeng, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, et al. 2023. "A Survey on Evaluation of Large Language Models." *arXiv Preprint arXiv:2307.03109*.

Jegou, Herve, Matthijs Douze, and Cordelia Schmid. 2010. "Product Quantization for Nearest Neighbor Search." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33 (1): 117–28.

Silpa-Anan, Chanop, and Richard Hartley. 2008. "Optimised KD-Trees for Fast Image Descriptor Matching." In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 1–8. IEEE.

References II

Wang, Mengzhao, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. "A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search." *arXiv Preprint arXiv:2101.12631*.

Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." *Advances in Neural Information Processing Systems* 35: 24824–37.

Zheng, Shen, Jie Huang, and Kevin Chen-Chuan Chang. 2023. "Why Does ChatGPT Fall Short in Providing Truthful Answers?" https://arxiv.org/abs/2304.10513.