Locality Sensitive Hashing

Lukas Bierling

September 2024

One of the downsides of restricting attention to a fixed size of tokens around the attention token is that important long range tokens which may capture more relevant information than the tokens in the direct neighborhood will be thrown away, hence leading to a significant loss of information. The Locality Sensitive Hashing (LSH) attention mechanism proposed by Kitaev et al. is another way to sparsify the attention matrix, however, instead of limiting attention to the direct neighborhood, it tries to compute attention only between somewhat "similar" tokens ([2]).

Since tokens are represented by high-dimensional embedding vectors inside the model, the goal is to formulate an algorithm that finds the l nearest neighbors in high-dimensional spaces. This is achieved by using a hashing function h which assigns each vector x to a hash $h(x)$. This function is 'locality-sensitive,' meaning vectors that are close to each other are likely to have the same hash, while distant ones do not ([2]).

Let $R \in R^{d_{model} \times \frac{b}{2}}$ be a random matrix and where b is the number of hashes. The hashing function h can then be described as:

$$h(x) = argmax(Concat_{row}(xR, -xR)) \tag{1}$$

Adoni et al. demonstrated that this is a practical and valid LSH schema, which is not only easy to implement but also effective for individual vectors and batches of vectors ([1]).

We can refer to equation 23 to understand how attention is computed in a sparsified attention matrix. The only difference is that our set S_i changes. Instead of containing the neighboring position token indices, it now consists of the tokens within the hashed bucket. However, the authors also omit for LSH-Attention the scaling factor $\sqrt{d_{model}}$ and set $Q = K$. Equation 23 then changes for LSH-Attention to:

$$\mathbf{S}_i^{\mathrm{w}}(q_i, k_{S_i}) = softmax(q_i {k_{S_i}}^{\mathrm{T}}) \tag{2}$$

for each token x_i while the rest of the equations 23-25 remain consistent. The set S_i can now be depicted as:

$$S_i = \{j : h(q_i) = h(k_j)\} \tag{3}$$

Meaning the set S_i consists of all positions j such that the hash value $h(q_i)$ equals the hash value of $h(k_j)$.

The process involves organizing queries by grouping them into buckets based on their hash values and arranging them in order within each bucket. To address the issue where a bucket might have a disproportionate number of queries compared to keys, the authors recommend normalizing each query vector q_j by its magnitude, setting $k_j = \frac{q_j}{||q_j||}$. This adjustment ensures that the hash values for the queries q_j and the keys k_j match, leading to an equal distribution of queries and keys in each bucket.

Once sorted, these query-key pairs are divided into segments, each containing $m$pairs. In cases where a bucket has more than m pairs, a unique attending rule applies: a query in one segment can refer to all keys in the same bucket from the previous segment, but it cannot refer to keys outside its current segment. This rule, however, doesn't apply to other queries in the same bucket; they are not allowed to refer to a key that's outside their current segment. This method is illustrated in Figure 1.

The authors state that their attention mechanism has a complexity of

$$\mathbf{S}^{\mathrm{w}} = \mathcal{O}(n \log n) \tag{4}$$

which can be more efficient than the local attention of the previous section depending on the sequence length n and the chosen window size w.

Finally, our hyperparameter set H can then be stated as: $\mathrm{H} = \{\mathrm{b}, \mathrm{m}\}$ Our set consists of the number of hashes b where in general more hashes mean the probability that similar tokens are grouped while the computational complexity increases and the chunk length m which stands for the size of the parts the sequence is chunked into after sorting it according to their buckets and positions.

References

[1] Alexandr Andoni et al. *Practical and Optimal LSH for Angular Distance*. 2015. arXiv: `1509.02897 [cs.DS]`.

[2] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: *CoRR* abs/2001.04451 (2020). arXiv: `2001.04451`. URL: `https://arxiv.org/abs/2001.04451`.

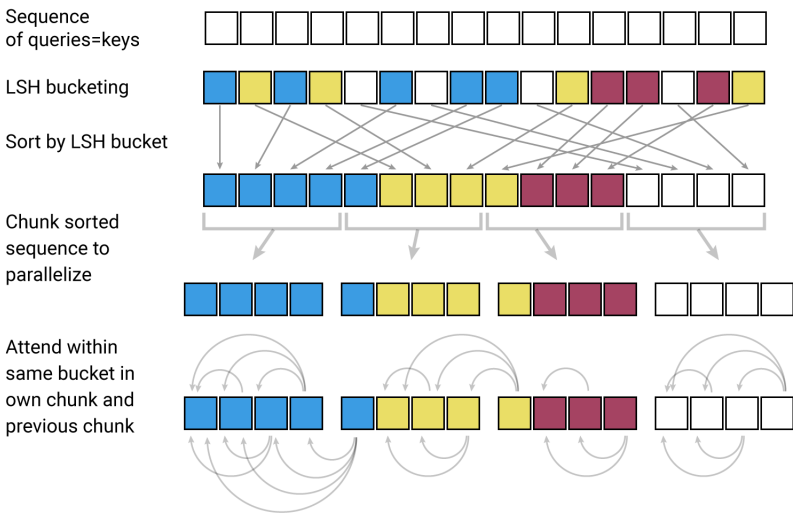

Figure 1: LSH attention algorithm: The sequence is hashed into buckets, then sorted according to their bucket number and position in the bucket. Following this the sequence is chunked into parts of length m. As shown queries outside a chunk can only attend backward.