

Chatbot

- ✓ **Video:** Tasks with Long Sequences
2 min
- ✓ **Reading:** Tasks with Long Sequences
10 min
- ✓ **Reading:** Optional AI Storytelling
15 min
- ✓ **Video:** Transformer Complexity
3 min
- ✓ **Reading:** Transformer Complexity
10 min
- ✓ **Video:** LSH Attention
4 min
- 📖 **Reading:** LSH Attention
10 min
- 📖 **Reading:** Optional KNN & LSH Review
20 min
- 📅 **Lab:** Ungraded Lab: Reformer LSH
1h
- 🎥 **Video:** Motivation for Reversible Layers: Memory!
2 min
- 📖 **Reading:** Motivation for Reversible Layers: Memory!
10 min
- 🎥 **Video:** Reversible Residual Layers
5 min
- 📖 **Reading:** Reversible Residual Layers
10 min
- 📅 **Lab:** Ungraded Lab: Revnet
1h
- 🎥 **Video:** Reformer
2 min
- 📖 **Reading:** Reformer
10 min
- 📖 **Reading:** Optional Transformers beyond NLP
20 min
- 📖 **Reading:** Acknowledgments
10 min

Heroes of NLP: Quoc Le

Assignment

Course Resources

LSH Attention

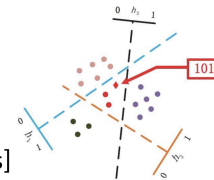
In **Course 1**, I explained how locality sensitive hashing (LSH) works. You learned about:

- KNN
- Hash Tables and Hash Functions
- Locality Sensitive Hashing
- Multiple Planes

Here are the steps you follow to compute LSH given some vectors. The vectors could correspond to the transformed word embedding that your transformer outputs. Attention is used to try which query (q) and key (k) are the most similar.

Compute the nearest neighbor to q among vectors $\{k_1, \dots, k_n\}$

- Attention computes $d(q, k_i)$ for i from 1 to n which can be slow
- Faster *approximate* uses locality sensitive hashing (LSH)
- Locality sensitive: if q is close to k_i :
 $\text{hash}(q) == \text{hash}(k_i)$
- Achieve by randomly cutting space
 $\text{hash}(x) = \text{sign}(xR) \quad R: [d, n_hash_bins]$



To do so, you hash q and the keys. This will put similar vectors in the same bucket that you can use. The drawing above shows the lines that separate the buckets. Those could be seen as the planes. Remember that the standard attention mechanism is defined as follows:

$$A(Q, K, V) = \text{softmax}(QK^T)V$$

Once you hash Q and K you will then compute standard attention on the bins that you have created. You will repeat the same process several times to increase the probability of having the same key in the same bin as the query.

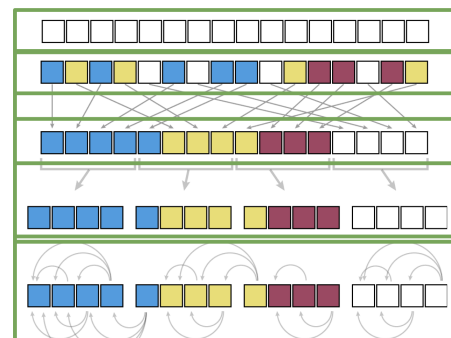
Sequence of Queries = Keys

LSH bucketing

Sort by LSH bucket

Chunk sorted sequence to parallelize

Attend within same bucket of own chunk and previous chunk



Given the sequence of queries and keys, you hash them into buckets. Check out Course 1 Week 4 for a review of the hashing. You will then sort them by bucket. You split the buckets into chunks (this is a technical detail for parallel computing purposes). You then compute the attention within the same bucket of the chunk you are looking at and the previous chunk. Why do you need to look at the previous chunk?

Mark as completed