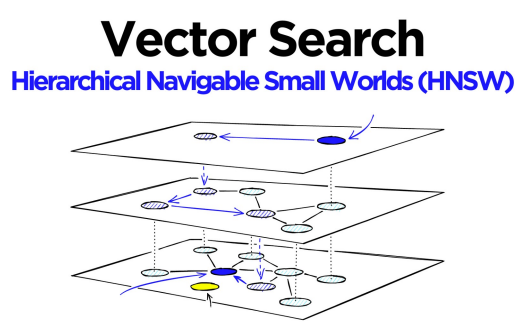
Assignment 3 - HNSW

24 February 2022 14:04

MOTIVATION

Deep learning algorithms rely on mathematical operation such as dot product or L2-distance to compute similarity between two or more items. For example, if Google wants to recommend most relevant news to a user, it would compute the distances between a user embedding $\hat{x}$ and each news embedding $\hat{z}_l$ ($l = 1..L$). You can think of $\hat{x}$ as a representing user preferences and $\hat{z}_l$ representing the features of news item l. This is also known as one-vs-all (OVA) method. However, computing OVA distances for every user, in billions, to each news article, in millions, is prohibitive. To speed up this task, many approximate nearest neighbour search (ANNS) have been proposed in the literature. ANNS are designed to reduce the prediction complexity of naïve algorithms. However, with ever-growing internet and social media platforms, news content has increased exponentially. The increase in news content has led to the problem of large model sizes, which cannot be maintained on a single system. We need to design an efficient parallel and distributed mechanism so that ANNS can be distributed on multiple machines and still be efficient for real-time predictions. In this assignment we will explore one such ANNS called HNSW.

HNSW Stands for Hierarchical Navigable Small World graphs. HNSW is designed to reduce the prediction complexity of OVA from $O(L)$ to $O(\log(L))$. This reduction in complexity is achieved by learning a hierarchical graph structure. To know about the learning of HNSW, please refer to the figure below and this (https://www.pinecone.io/learn/hnsw/) blog.



(source:Hierarchical Navigable Small Worlds (HNSW) | Pinecone)

Objective:

Your implementation should be able to distribute HNSW based prediction algorithm among N nodes and should make use of C cores on each node to predict top K most relevant items, for each user represented in an input file. The pre-trained HNSW graph is also given in a file.

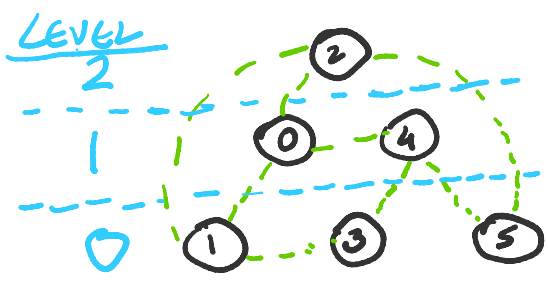
Pre-trained HNSW data: (location:/scratch/cse/phd/anz198717/TA/COL380/A3/to_students on HPC)

You will be given pre-trained HNSW with the following items to do the predictions. Each is in a separate text file.

- max_level
  - a single value denoting height of hierarchy in graph (single integer)
- ep
  - Id of Entry Point vertex in graph from where we begin traversing the graph (single integer)
- level
  - level[i] of a node i in the graph (Whitespace-delimited integers in the order of i, starting at i=0)
- index
  - list of integer values to determine nearest vertices in the graph structure (Whitespace-delimited)
- indptr
  - WS-delimited list of integers, s.t. index[indptr[i]:indptr[i+1]] gives the range of neighbours of node i (Range is open at the end.)
- level_offset
  - Gives the range of nearest neighbours at each level: index[indptr[i]+level_offset[l] : indptr[i]+level_offset[l+1]] are the labels of neighbours of node i at level l. Some slots may be –1, if edge to that node at level l from node i does not exist. (WS-delimited ints)
- vect
  - dense vector representation of items ($\mathbb{R}^{L\times D}$) (Each line has *1* news embedding and has D **real** values, space-delimited.)

Along with this, you will be given U user embeddings ($q \in \mathbb{R}^{U\times D}$) in user.txt for which you need to make top-K predictions.
Here U $<$ 1M, L≈1M and D≈786

Consider a HNSW graph given below for better understanding



Edges are shown in green

(location: /scratch/cse/phd/anz198717/TA/COL380/A3/dummy_data)
M = 3, L=6, D=5, max_level = 3, ep = 2

Argument	Value
level	[1, 0, 2, 0, 1, 0]
level_offset	[0, 3, 5, 6]
indptr	[0, 5, 8, 14, 17, 22, 25]
index	[1, -1, -1, 4, -1, 3, -1, -1, 1, 5, -1, 0, -1, -1, 1, -1, -1, 3, 5, -1, 0, -1, -1, -1,-1]
vect	[[0.2, 0.4, 0.1, 0.4, 0.3], [0.1, 0.2, 0.8, 0.3, 0.1], [0.7, 0.3, 0.9, 0.3, 0.2], [0.8, 0.4, 0.1, 0.6, 0.3], [0.0, 0.2, 0.8, 0.5, 0.1], [0.4, 0.3, 0.9, 0.3, 0.7]]

Neighbours of node 2 at level 0
Index offset is indptr[2] i.e. 8
Level offset start is level_offset[0] i.e. 0
Level offset end is level_offset[1] i.e. 3
Neighbours = index[8+0:8+3] = index[8:11] = [1, 5, -1]

Neighbours of node 2 at level 1
Index offset is indptr[2] i.e. 8
Level offset start is level_offset[1] i.e. 3
Level offset end is level_offset[2] i.e. 5
Neighbours = index[8+3:8+5] = index[11:13] = [0, -1]

Prediction Algorithm: Pseudocode for HNSW prediction

```
// BSF search in the layer for best candidates
def SearchLayer(q, candidates, indptr, index, level_offset, lc, visited, vect):
    top_k = copy(candidates)
    while(candidates.len>0){
        ep = candidates.pop()
        start = indptr[ep] + level_offset[lc]
        end = indptr[ep] + level_offset[lc+1]
        for px in index[start:end]{
            if visited.find(px) or px == -1
                continue
            visited.set(px)
            _dist = cosine_dist(q, vect[px])
            if _dist > top_k.max() and not top_k.have_space :
                continue
            top_k.insert(<px, _dist>) // insert the new node
            top_k.trim() // keep only top k nodes
            candidates.push(<px, _dist>)
        }
    }
    return top_k

// Outputs top k most relevant items
def QueryHNSW(q, topk, ep, indptr, index, level_offset, max_level, vect):
    top_k = heap(topk=topk, largest=False)
    top_k.push(<ep, cosine_dist(q, vect[ep])>)
    visited = hash_map() // mark nodes visited in the graph
    visited.set(ep)
    L = max_level
    for level in L … 0:
        top_k = SearchLayer(q, top_k, indptr, index, level_offset, level, visited, vect)
    return top_k
```

TEST STRUCTURE: You will be required to submit the following files in the given data directory format.

- <ENTRY_NUMBER>
  - compile.sh
  - DataSetup.sh
  - HNSWpred.sh
  - Rest of your work

Description of each file is as follows:

- compile.sh
  - This should compile your code for further use
- DataSetup.sh <in_path> <out_path>
  - Convert raw data to the required format. This takes in two arguments
    - Data directory for pre-trained HNSW with given file names
    - Output directory where you can dump the files after converting to any format of your choice
- HNSWpred.sh <out_path> <int> <user_file> <user_output_file>
  - Given a query it should predict top K relevant items/vertices in descending order. Takes in 4 arguments
    - Output directory same as DataSetup.sh
    - K (top-K predictions to be made by the HNSW)
    - User representation file (A text file same as vect, each line corresponds to 1 user with D dimension)
    - Output file - where HNSW will write space separated vector Ids (Id is the line number in the vect file, starting at 0) in descending order, one line per user. Users are in the order of the user representation file.

Sample run would work as follows

```
./compile.sh
./DataSetup.sh /data/path/hnsw/given /data/path/hnsw/converted
./HNSWpred.sh /data/path/hnsw/converted 5 users.bin user_prediction.txt
```

This should generate user_prediction.txt file where each line corresponds to the user predictions in descending order.

REPORT:

- Output index of top K items for each user in sorted order (descending) in a text file such that each entry in a row corresponds to the entry in file user.txt
- Report the accuracy of HNSW and average prediction time (pred. time / user) with max memory usage for N = [2, 5, 10], C = [5, 10, 24], K = [5, 10, 15]