Learned Indices : Point Index

Rahul Palamuttam

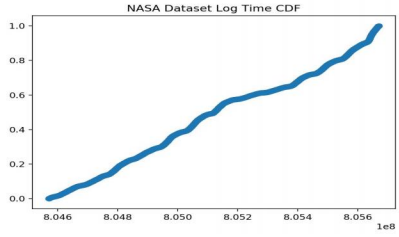
Introduction

Recently Deep Learning techniques have shown to advance the state of the art in tasks found commonly in Computer Vision, Natural Language Processing, and Speech Recognition. Traditionally, Neural Network approximators have been applied to domains outside of core computer systems. However, a recent paper “The Case for Learned Index Structures” makes a case for how indexes are really models which can be approximated by neural networks. Kraska et al. claims that because neural networks are universal approximators they can learn properties of the data being indexed to improve upon existing hashing schemes found in databases. The paper looks at how deep learning can be applied to achieve greater hash table utilization when storing range indices, point indices, and existence indices. An important point to note is that learned indices do have their drawbacks. For example, while neural networks can approximate the underlying cumulative distribution function the errors that commonly pop up in shallow models are due to the fact that the CDF function of real world datasets are very noisy upon closer inspection. So the authors employ a recursive index model to combat this which is primarily inspired by how B-Trees are organized. Each layer of the model essentially chooses which model-tree to use to compute the index.

Description of the dataset

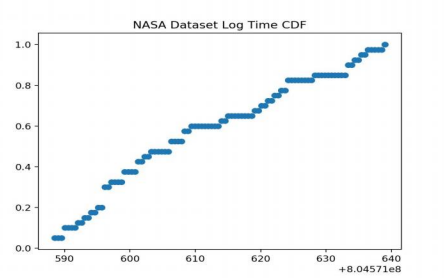
NASA Web Server logs

We decided to utilize the an open dataset from NASA - it specifically consisted of two traces containing two month’s worth of HTTP requests to the NASA Kennedy Space Center. The logs have the following columns : host, timestamp, request, HTTP reply code, and bytes in the reply. The log was collected starting on July 1 19995 upto August 31 1995. I decided to use this dataset as it was similar to the weblogs dataset that was used in the original paper by Kraska et al. It also had the distinct property of having an irregular cumulative distribution function. The figure below shows the CDF of the entire 14 million points in the dataset - at first glance it looks fairly easy to approximate with a linear function.



However, if we zoom in to a time range that is about 1 percent of the total span we see that the CDF is highly irregular. Thus a linear model would achieve fairly poor performance on the dataset.

Learned Indices : Point Index
Rahul Palamuttam



For comparison, Kraska's work utilized a dataset consisting of 200M web-server log records, rows from the OpenStreetMap database, and a synthetically generated lognormal dataset. The Weblogs dataset consisting of 200M log entry timestamps are noted to be highly irregular while the longitude and latitude keys in the map dataset is fairly linear with little irregularities.

Synthetic Datasets

Instead of utilizing a lognormal dataset, for debugging purposes I decided to create a simple linear and quadratic dataset. That is I randomly sampled 14M points from a linear function and quadratic function. Since the primary task at hand here is to approximate irregular CDF's I also randomly perturbed around 35% of the points in the synthetic datasets.

Data cleaning and preprocessing

While the synthetic datasets did not really need any type of cleaning, the weblogs dataset did. The dataset was stored in a csv file, which we read in using pandas. There were missing timestamps for certain days which had to be cleared out. Like in the original paper, we only wanted to deal with unique timestamps, so we had to filter out duplicate timestamps.

Explanation of choices and decisions

Unlike traditional goals of a model learning from a training set and generalizing well on a test set, the goals here were to literally memorize the training set in order to get optimal hash table utilization. For this reason, after each epoch I evaluated the model on the entire training set instead of having a separate test set.

Kraska et al. claims that the model itself trains in under an hour (minutes). However, after replicating the model as close to the paper as possible (clearly they utilize many more stages and more models per stage) the exact engineering effort to do this is missing from the paper details. Furthermore, recent blog posts by authors of the paper claim that there are missing details in the paper (such as what learning rate was used, what kind of optimizer was used to train the recursive index model, and if other architectures were tried). For this reason I stuck to solely utilizing the recursive index

model and utilized the adam optimizer with a learning rate of 0.001. Due to the computational challenges and lack of resources (even with a GPU) the recursive index model was fairly extensive to strain with large values of k (the number of nets to train per stage) and large values of l (the number of stages). For this reason I utilized a k value of 2 with 2 stages.

Approach:

The first step was to mimic the neural network architecture presented in the paper on a subset of the dataset and overfit. The baseline architecture is simple as it consists of two hidden layers each of which is of size 32. The final output activation layer is of size 1. For the purposes of the milestone I decided to forgo the recursive model index loss and used a simple sum-squared loss shown below.

$$L_0 = \sum_{(x,y)} (f_0(x) - y)^2$$

Kraska et al. makes an interesting choice in setting up the experiment with regards to the number of hash slots (the labels) and the number of unique keys being hashed - they are the same. In essence, the work on approximating point indices seems to solely be devoted to learning a hash function that does a one-to-one remapping from the key space to the index space. The hash table utilization is measured in terms of the number of keys mapped successfully to a unique slot versus the total number of keys. It's important to point out that for computing point index predictions the ground truth indices are hand picked to range from 0 to the length of the dataset.

We use mini batch gradient descent with an entire epoch consisting of 500 steps and a batch_size of 100. We use also use the Adam Optimizer initialized with a learning rate of 0.1.

The finer details of processing the dataset involved ingesting the tsv file using pandas, filtering out missing time stamps and manually generating index values as the labels.

Recursive Model Index

$$L_\ell = \sum_{(x,y)} (f_\ell^{(\lfloor M_\ell f_{\ell-1}(x)/N \rfloor)}(x) - y)^2 \qquad L_0 = \sum_{(x,y)} (f_0(x) - y)^2$$

Learned Indices : Point Index
Rahul Palamuttam

The recursive index model is crucial part of the learned indices paper and the results of the milestone show that this needs to be implemented. Here each layer consists of several shallow networks. Each layer essentially computes the index of the network to use in the following layer. As noted by Kraska et al. each of the models makes a prediction about the general location of the key. The subsequently selected model is then used to make a more localized prediction from the general location derived before it. An interesting point to note is that the Learned Indices paper does not address learning how many layers of models the network really needs nor do they adequately distinguish the difference between this architecture and a conventional deep neural network architecture.

Results, metrics and discussions

Data	1 Layer	RMI
Weblog	10%	55%
Linear	65%	45%
Quad	45%	35%

The first experiment looked at a 1 layer neural network and its performance on learning mappings for the weblog timestamps, and the linear and quadratic datasets. Due to the irregular nature of the CDf in the NASA weblog dataset the 1 layer net could only achieve 10% accuracy. Interestingly, the 1 layer net performed well on the synthetically generated linear and quadratic datasets. Of course, it could not achieve 100 percent accuracy since I randomly perturbed 35% of the points in the synthetic datasets.

The recursive index model as expected performed well in mapping the timestamps in the weblog dataset. However, it is surprising that it was not able to achieve the results shown in the paper by Kraska (which was claimed to be 100 % in the alternative baselines section). I believe this is because of the limited range of k and l hyperparameters I could play with when fine tuning the recursive index model. The actual paper did use two stages in the RMI but used a k value of 10k. Furthermore, the comparison is a slightly apples to oranges comparison in that their hash table implementation is slightly different and doesn't solely rely on a one to one mapping. The

table used a separate chaining approach for every skipped item except and utilized the remaining free slots with offsets as pointers.

Conclusion:

A crucial motivation of this work is the ability to improve the performance of real world computer systems using deep learning. While the learned-indexes paper looked at how it might be possible to approximate database indices with neural networks the overlooked fact is that the the distribution of keys change with time. What's really missing from a practical standpoint is the ability to online-learn the recursive model index to support inserts and deletes. This is a crucial part of any real world index structure which I hope to address towards the end of the project.

A strong case is made that neural networks could be used as index approximators in databases. While the work was inspired by the idea that indexes are really models, it could also be said that models are also indexes. By this reasoning we should be able to handle deletes as an operation that "undoes" a gradient update for a set of items being removed from the hashtable. Apart from asking the authors what exactly their hyperparameters were, I think for learned indexes to be of practice use there needs to be a way to account for inserts and deletes in a learned index model. This could be a promising next research goal for subsequent work and could really push the limits of conventional transaction workloads.

References and related work

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis: The Case for Learned Index Structures. SIGMOD Conference 2018: 489-504

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, Jeff Dean: Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.

http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html