Using Deep Learning Models to Replace Large Materialized Views in Relational Database

Jia Zou
Arizona State University
jia.zou@asu.edu

Materialized views are "materialized" result sets of database queries. It is widely used to accelerate query executions. However, "no gain is with no pains", the memory and storage space used to materialize a large view (e.g., the join output of two large tables) could be expensive and thus sometimes it is inhibitive to have such large materialized views. In the recent decades, with the advances in heterogeneous hardware such as Graphic Processing Unit (GPU), deep learning has demonstrated its capability in accurately modeling non-linear relationships with acceptable training and inference costs. If we see a query as a function of mapping individual table cells to the position in the output table, indexed by attributes and row keys, the question is whether we can learn this relationship using a neural network model? If so, we may utilize the trained models to replace materialized views. Each of the subsequent queries against the views can thus be transformed into a set of inference tasks that request to map the individual source table cells to the positions in the view, as illustrated in Fig. 1.

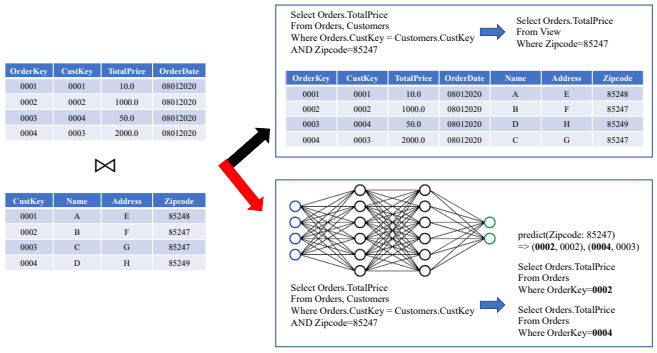

Figure 1: Learning-based materialization view

This work discusses two critical questions: (1) What are the benefits and costs of using deep learning models to replace materialized views? (2) How is it possible to train a neural network model to replace a materialized view?

11th Annual Conference on Innovative Data Systems Research (CIDR '21) January 10-13, 2021, Chaminade, USA.

Benefits vs. Costs. Reduction in storage overhead is a main benefit. Deep learning models usually have significantly smaller sizes compared to today's open datasets. For example, the size of the largest known model (i.e. GPT3 has 175 billion parameters) is significantly smaller than many enterprise datasets that are at petabytes' level. Among the text embedding models that are available on TensorFlow hub, the largest dataset used is common crawl that has 220 Terabytes in size, and 2.6 billions of tuples. However, the embedding model trained on the common crawl dataset, using the language-agnostic BERT sentence embedding model, is just 1.63 Gigabytes in size. There is significant saving in storage overhead.

The costs associated with the deep learning approach are mainly in its training overhead and the inference latency. We find that the training data can be automatically created from the view results, and the training overhead can be amortized to many inferences. In addition, the inference requests regarding individual source table cells are independent with each other and can be concurrently served using a distributed set of GPUs. Moreover, there is no physical join operations required anymore, thus there is no need to handle explosive intermediate data and expensive shuffling operations. Another issue is that errors may be brought by the deep learning approach. However many Big Data applications are error-tolerable, as long as the majority of source cells are correctly mapped to the output table.

Preliminary Representation and Modeling Design. For the feature representation, each cell is annotated using the cell's context, such as attribute name and row key in its source table, forming a sequence of tokens, which is then transformed into a sequence of vectors through a pre-trained embedding.

Regarding the design of the label space, logically each position in the output view can be indexed by the attribute name, and row key. It is easy to derive the set of attributes from the query, and the set of row keys for join operations that are with 1-1 mapping (i.e., two tables join on the shared key, like joining populations and coronavirus cases on the county attribute); and 1-N mapping (i.e., two tables join on a foreign key, e.g., joining customers and orders on custkey). While it requires a join to obtain a set of possible keys for the M-N join cases, we only need to perform it once and the join results can be discarded after training. Then the label space is formulated as a probability distribution over attributes and row keys. We can determine the position(s) of an individual cell by sampling the probability distribution.

The view can be discarded once the model is trained, thus the expensive memory and storage space will be saved.