TensorDB: Database Infrastructure for Continuous Machine Learning

F. Liu, A. Oehmichen, J. Zhang, K. Sun, H. Dong, Y.Mo and Y. Guo
Data Science Institute, Imperial College London, London, England, United Kingdom

Abstract— *This paper introduces the TensorDB system, a framework that fuses database infrastructure and application software to streamline the development, training, evaluation and analysing machine learning models. The design principle is to track the whole model building process with database and connected different components by database query mechanism. This design produces a highly flexible framework enable each component to be updated independently. The theoretical value is that it enables continuous machine learning. TensorDB is motivated by production application of machine learning model as consolidation of many engineering practice, and is could be served as the foundation for high level tools for machine learning application.*

Keywords: NoSQL Database, Machine Learning Framework, Continuous Learning

1. Introduction

In nature, learning is adaptive and progressive, which enables animals change their behaviour as the environment changes [1]. Most machine learning algorithms like deep learning, however, are not designed for progressive training. With the dominant batch learning approach, there is a clear separation between training and inference. Machine learning models are trained off-line with large amount of training data, and later when models are deployed for inference, the parameters are fixed. In application, such methodology faces many limitations. Most business is dynamic and the machine learning model must be updated continuously as business operates. For e-commerce business, the recommendation system must be updated as user preference changes. Continuous machine learning is also of great importance in research. Recent machine learning models such as reinforcement learning [2] and generative adversarial network [3] are continuous in nature. We argue that the continuous learning, adaptive and progressive features of the biologically system are indispensable for machine learning application.

In this work, we introduce the TensorDB, a framework solution that enables continuous training, evaluation, analysis, and deployment machine learning model. TensorDB is not an algorithm for continuous machine learning, but an autonomous life-cycle management system of machine learning models. It is based on the idea that continuous learning is achieved by continuous updating the training sets, continuous training, and continuous evaluation. Provided with unlimited storage and computing power, we can build thousands of models with different architecture, the evaluation system will keep on mining the model repositories and always select the one with the best performance for deployment. When we implement such a system, the key challenge lies in the data management of all the models, training set, parameters, and logs. The data management system of TensorDB is based on NoSQL database [4] and map reduce search engine [5]. It is implemented as distributed system that fuse database infrastructure and machine learning application software. With TensorDB, all the components of machine learning development are connected by the database query mechanism, which produces a flexible system that enables each component to be updated continuously.

Even continuous learning is not required, machine learning developers can benefit from TensorDB for streamlining the developing process. It could be used as a data warehouse for managing the training set, a logging system for recording training process, a version control system for machine learning models, an intelligent system for model selections, and a load balancing system for distributing machine learning jobs. TensorDB system combines all above functions and is a fully autonomous pipeline for sampling, training, evaluating, analyzing and deploying machine learning models.

The TensorDB is made of three components

1) Cohort recruitment for management of training set and models
2) A building/job system for training models and a logging system for recording the training process
3) Model mining system for evaluation and model selection

2. Related Work

Even there are many libraries for building and training deep learning models, little work has been done for the models management so far. Recently ModelHub [6] is proposed as the version control system for exploring and storing all the models. However, TensorDB is more ambitious and focuses on the working flow. From the technology, perceptive, a critical difference is that TensorDB is based on the NoSQL database and using map reduce as the searching method while ModelHub is a SQL based. The current implementation of TensorDB is based on MongoDB database engine.

NoSQL is scheme less, and it is more flexible and scales better. User can alter the database scheme even after the system starts running. The key technique in the TensorDB is a flexible binding system. With the support of a database, flexible binding is implemented as data search query. This is inspired by the katana lighting system [7] developed by Sony Image Works. The logging system resembles the EHR system for healthcare record management . It follows the NoSQL solution for Pharmacovigilance [8].

3. TensorDB Architecture

TensorDB is mainly made of several components. The cohort recruitment system is a data warehouse for managing training set and models. In NoSQL database, data is stored as key-value pairs. Our MongoDB [9] based implementation stores the data as documents, which is a dictionary of many key-value pairs. MongoDB replace table by collection which is a set of documents. With NoSQL system, searching is performed by map reduce methods. In MongoDB, this is mapped to the filtering, projection, and sorting. TensorDB maintains 5 collections.

1) The Model Collection which contains all the model architecture and parameters.
2) The Dataset Collection which contains all the training and validation set.
3) The Training Log which records the performance of each step during training
4) Evaluation Log which records all metrics for models on different validation set.
5) The Job System which maintains all the building jobs and is also served as queue for load balancing.

To be more precise, there are also two collections for models and dataset which are used as file system.

There is also corresponding software to operating on the data to run the TensorDB system.

1) Data Importer which imports training dataset into the database.
2) Recruiter is implemented as a query program based MongoDB query language. Each cohort is not physically stored in the database, the searching program is executed during run time when the cohort is assembled.
3) Builder will read the job queue and execute the training and evaluation job. All the model parameters of each epoch are stored in the database. The log of each training step is stored in the logging system.
4) Evaluation and Model Mining system will also rely on the job queue and execute all the validating jobs. The evaluation results of each validation sets and model metrics are stored in the database.
5) The analysis and visualization system explore the training and evaluation log.

In application, most of the code and database scheme is fixed. The flexibility comes from the searching based recruiter. Search based recruiter never directly binds model to data. For example, the training set could be specified as the last 1000 imported, which will keep on changing if data are generated continuously, and the evaluation be specified to validates only five models that have the least training errors

3.1 The Cohort Recruitment System

The cohort management system is in principle a data warehouse that stores training datasets and models. There are three kinds of data 1) training sets, 2) model parameters, 3) model architecture.

The storage scheme separates meta data file and actual data. Query is performed on the meta data fields, the actual data are stored as blobs in a database based file system(GridFS). In our implementation, the raw data are stored as binary strings, which are result of serialization operation. Images are based on the Numpy serializer, models are based on the Tensorflow graph serializer and parameters are also based on Numpy serializer.

Software tools are required to extract the meta information, which is the role data of importer. The meta information is designed to be information rich, containing as much valuable information as possible. For medical images, besides the information in the dicom headers, it can also include semantic information such as the parts of the body scanned, volume of brain ventricle, and grade of tumour. This is only possible with NoSQL database, which enables a very flexible data structure. For training data, we put both data and labels in one document. For models, the architecture and parameters are stored separately. The meta information of parameters includes reference to training data sets, upload time, training errors, epoch number and step time of back propagation.

The recruitment system is implemented as search query based on the fields of meta information. They are used for finding the case of interests, which is very valuable for development of machine learning model, such as controlling the balance of positive and negative examples.

A daemon process is invoked regularly to check all machines are running their jobs properly. In case some machine is down, redeploy its jobs on an available one.

3.2 The Building System

The Building System treats a training machine learning model as a software building. By combining dataset and algorithms, the building system just run model training forever. In the implementation of TensorDB, we define a collection called jobs which serve as a queue and load balancer. The job document contains model architecture; the model initializes parameters, training datasets and time stamp which indicates when the job is generated. There is also a status flag to indicate whether the job is ready, being

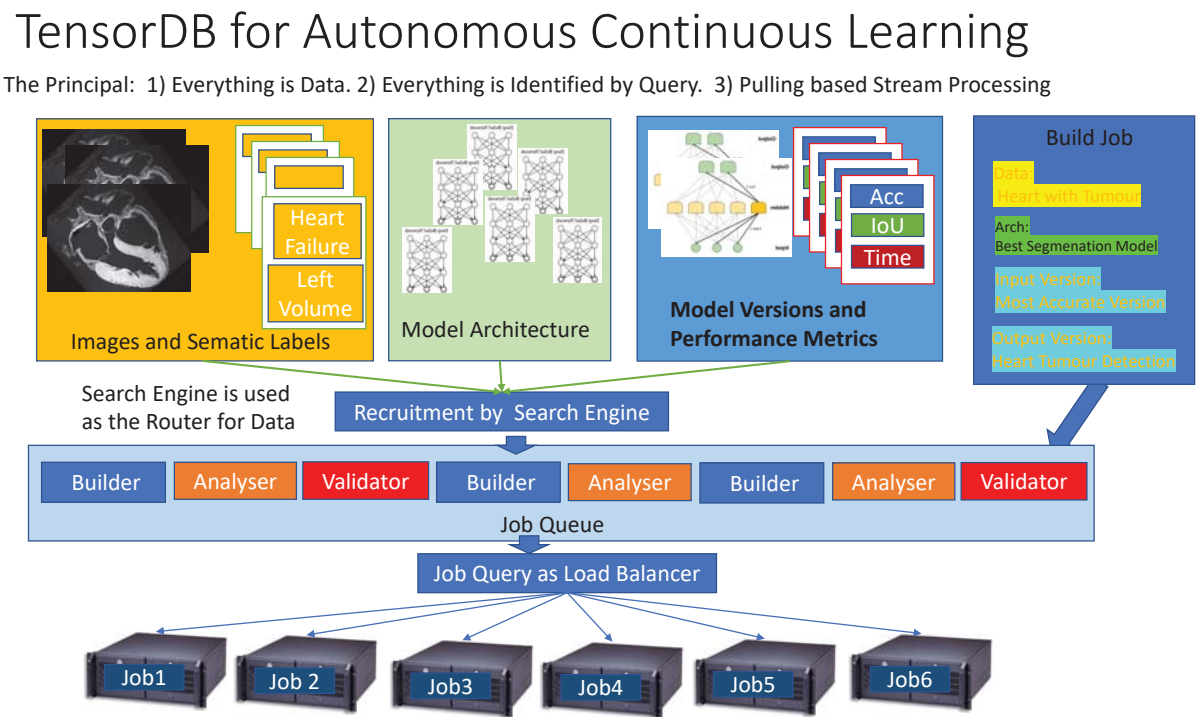

Fig. 1: Tensor DB workflow: the database query mechanism connects all the components. The work is distributed across multiple machine

processed or finished. The evaluation system shares the job queue with the building system, with different job types.

The building software automatically accesses the database, finds an idle job from the job queue, and updates its status in atomic manner. When a job is running, training results of each step are recorded in the logging system. Each record document contains the model architecture, epoch number, time, accuracy, performance metrics, study id, and information of hosting computer. After each epoch, the building system uploads the parameters into the cohort system, which updates the job fields with the latest update time, current epoch, and model accuracy.

3.3 The Analysis and Visualization System

The analysis and visualization explores system the log data for insights of model development, its applications include visualization of the model learning speed, comparing convergence speed of different architectures or execution speed. The requirement is very flexible and opens many new opportunities such as meta learning that needs to select a best architecture. Rather than providing tools, TensorDB makes such analysis easier. As a popular database, there are many packages support data analysis and visualization with MongoDB and we can even export the data to other software.

3.4 Evaluation and Model Mining

Such a system is straight forward to be implemented with building blocks from the cohort system, building system and analysis system. In principle, the model mining system selects representative test data and evaluate all the models. After the system finds the best model, the system can also continue to improve it by recruiting new training set and launch another training job.

4. Application Evaluation

We test the TensorDB with several challenging deep learning research projects. For our recent work on brain tumor segmentation, several modes are required to segment high /low grade tumor from multiple imaging modalities [10]. To test our modes, we performance 5-fold cross validation. The training set is divided into 5 groups and five models are trained with 4 group data and valid on the left one. The combination of modality, training sets and model architecture produces more than 100 models. Manual management such as moving the data, model and parameters across machines are error prone and waste lots of programming and computing time. It is also very hard to monitor all the training process in one view.

The introduction of TensorDB solves all the problems by centralize the data in a database, tools are easily built and can help the research with a global view of differ-

ent model and architecture performance. The building and testing of the 100 models share the same application code with only different data set specified by query language. For distributed training, TensorDB is deployed as independent remote server. The drawbacks of TensorDB is mainly from the performance perspective. The logging into database is more time consuming than the printing on the screen. For our image segmentation, the overhead is insignificant. Each step will consume about 1.4 seconds and logging cost about 1.5 ms. The epoch consumes about 600 seconds with uploading model parameters to the database cost about 3 seconds. while downloading, parameters consumes about 2 seconds.

The bottleneck of TensorDB solution is loading training data. Compared with loading the training data locally, loading the training dataset with the recruiting system is usually about 10 times slower. Loading about 6000 high resolution images from local system cost about 30 seconds while from TensorDB cost about 300 seconds. If we constrain the loading time to be 5% for model update and model will finish in 20 epochs, each job will have time resolution of about two hours.

In terms of the storage burden, after running for a week, the system generates about 500 million log data which each only about 20 bytes. The training data size is about 30GB which is fixed. The real storage pressure comes from saving all the model parameters of each epoch, which consumes about 300 GB storage. For analysis, the speed depends on the counts of query result. when the count number is less than 100,000 will usually consumes about 4 seconds while 10,000 records will consume about 0.5 seconds. The system is not real-time but can performance analysis interactively.

5. Conclusion and Further work

TensorDB is a flexible framework for machine learning, the database based solution and the search based binding making the learning development more flexible and easy to management. It allows the system to continuous update the training set, the learning mode and update the deployment in a fully autonomous manner. The currently system has some performance issues, it is not real-time and the loading is time consuming. However, it is good enough for the machine learning application when models do not have to be updated more than 10 times a day. We will improve the training data loading time and investigate the possibility using this model from more agile tasks.

In future, we will test the system for reinforcement learning and unsupervised learning such as GAN. The analysis and evaluation from work provide a foundation for high level task, like meta learning (finding the right model architecture) or model composition, which is also a direction we want to explore in future.

References

[1] D. L. Silver, Q. Yang, and L. Li, "Lifelong machine learning systems: Beyond learning algorithms." in *AAAI Spring Symposium: Lifelong Machine Learning*, vol. 13. Citeseer, 2013, p. 05.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[4] A. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *arXiv preprint arXiv:1307.0191*, 2013.

[5] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 1029–1040.

[6] H. Miao, A. Li, L. S. Davis, and A. Deshpande, "Modelhub: Towards unified data and lifecycle management for deep learning," *arXiv preprint arXiv:1611.06224*, 2016.

[7] B. Hall, J. Selan, and S. LaVietes, "Katana's geolib: behind the scenes," in *ACM SIGGRAPH 2014 Talks*. ACM, 2014, p. 60.

[8] Y. May, Y. Li, J. Raffel, M. Craner, C. Hemingway, G. Giovannoni, J. Overell, R. Hyde, J. Van Beek, F. Thomas, *et al.*, "Optimise-a web-based solution for recording and analyzing longitudinal multiple sclerosis data (p2. 116)," *Neurology*, vol. 86, no. 16 Supplement, pp. P2–116, 2016.

[9] K. Banker, *MongoDB in action*. Manning Publications Co., 2011.

[10] B. Menze, E. Geremia, N. Ayache, and G. Szekely, "Segmenting glioma in multi-modal images using a generative-discriminative model for brain lesion segmentation," *MICCAI BraTS Challenge*, 2012.