In-database Distributed Machine Learning: Demonstration using Teradata SQL Engine

Sandeep Singh Sandha
UCLA
sandha
@cs.ucla.edu

Wellington Cabrera
Teradata Labs
wellington.cabrera
@teradata.com

Mohammed Al-Kateb
Teradata Labs
mohammed.al-kateb
@teradata.com

Sanjay Nair
Teradata Labs
sanjay.nair
@teradata.com

Mani Srivastava
UCLA
mbs
@ucla.edu

ABSTRACT

Machine learning has enabled many interesting applications and is extensively being used in big data systems. The popular approach - training machine learning models in frameworks like Tensorflow, Pytorch and Keras - requires movement of data from database engines to analytical engines, which adds an excessive overhead on data scientists and becomes a performance bottleneck for model training. In this demonstration, we give a practical exhibition of a solution for the enablement of distributed machine learning natively inside database engines. During the demo, the audience will interactively use Python APIs in Jupyter Notebooks to train *multiple linear regression* models on synthetic regression datasets and *neural network models* on vision and sensory datasets directly inside Teradata SQL Engine.

PVLDB Reference Format:
Sandeep Singh Sandha, Wellington Cabrera, Mohammed Al-Kateb, Sanjay Nair, and Mani Srivastava. In-Database Distributed Machine Learning: Demonstration in Teradata. *PVLDB*, 12(12): 1854-1857, 2019.
DOI: https://doi.org/10.14778/3352063.3352083

1. INTRODUCTION

Relational databases systems remain as the principal technology to implement repositories for transactional data in enterprises. This makes data stored in relational tables the main source for business intelligence, executive reporting as well as many other data analytics tasks. The conventional approach to train machine learning models is to use external, cross-platform engines such as TensorFlow, Pytorch or Keras. With this approach, however, model computation tasks become more complex since a data analyst needs to handle data ingestion, manage data transformation, address performance regressions, etc.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352083

By and large, enabling machine learning natively inside database engines brings intelligence directly to where data lives. Native database support of machine learning helps data scientists steer clear of the complexity of handling data ingestion, cleansing, preparation, etc. It also relieves data scientists from many other challenges such as addressing sophisticated performance issues and dealing with tedious security and privacy protocols.

In this demonstration, we present our approach of enabling distributed machine learning using data parallel training natively to the Teradata SQL engine. Teradata SQL Engine runs on parallel clusters of a shared-nothing architecture [4], which offers high scalability, workload balancing and fault tolerance. Our approach is to 1) train local models in parallel on local data, 2) aggregate local models to obtain a global model, 3) replaces local models with the global model and 4) iterate until a max iteration count or a desired accuracy threshold is reached. Our goal is not to replace the machine learning frameworks but to enable widely-used machine learning algorithms organically as first-class citizens within database engines.

Throughout the demonstration, audience will interactively train and validate multiple linear regression models on synthetic regression datasets and neural networks models on MNIST [9], Fashion-MNIST [12] and SHL [6] datasets using Python API of Teradata in Jupyter Notebooks. Audience will be able to select the training data, decide model architecture and choose Teradata cluster size (small, medium or large) to train model interactively. Results of model training will be presented to audience as a final outcome of the demonstration.

2. SYSTEM OVERVIEW

In this section, we first give an overview of Teradata shared-nothing architecture. Then we explain our approach for parallel and distributed model training. And finally, we discuss implementation details.

2.1 Teradata Architecture

The SQL Engine of Teradata (see Figure 1) is a highly parallel engine with a shared-nothing architecture [10]. It can be deployed to Symmetric Multi Processing (SMP) and

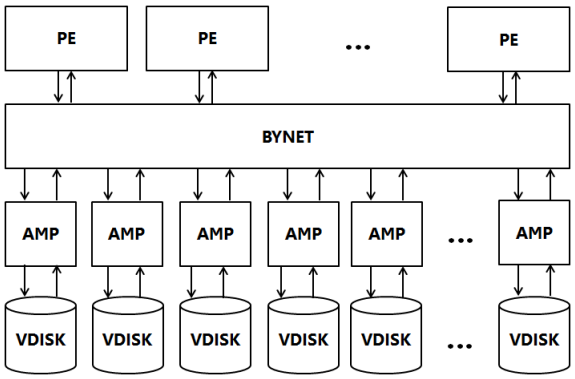

Figure 1: Teradata SQL Engine Shared-Nothing Architecture.

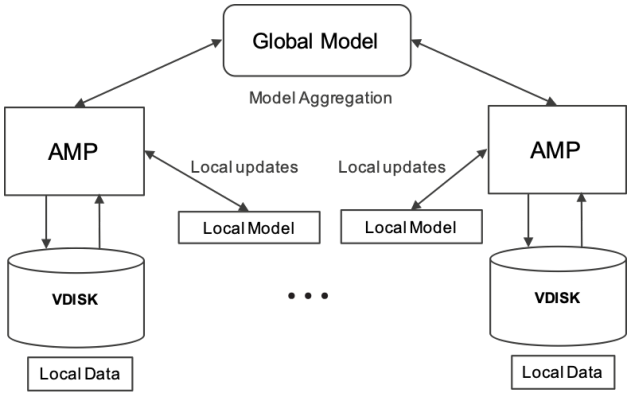

Figure 2: Parallel and Distributed Model Training in Teradata SQL Engine.

Massively Parallel Processing (MPP) systems [5]. The architecture (Figure 1) has four main software components: PE, AMP, VDisk and BYNET.

A PE (Parsing Engine) is a multithreaded virtual processor responsible for controlling a set of sessions, validating security rules, parsing requests, optimizing queries and dispatching processing steps to AMPs. An AMP (Access Module Processor) consists of a collection of worker tasks, which are threads that perform operations on a portion of the data, e.g., sorting, aggregation and join. A VDISK (Virtual Disk) is a physical space of the storage allocated and assigned by the virtual storage subsystem to each AMP. The BYNET is a communication layer used by PEs and AMPs to exchange messages through bidirectional, multicast and point-to-point communications between processes.

2.2 Distributed Machine Learning

Nearly all machine learning algorithms have an objective/loss function defined over model parameters and input data. Model parameters are updated iteratively towards optimal value so as to minimize (or alternatively maximize) the objective function. From the iid (independent and identically distributed) assumption on input data, updates to the model parameters can be aggregated in data parallel fashion [11]. We enable distributed machine learning using *Data Parallel Training* natively to database.

Data Parallel Training: In Teradata shared-nothing architecture, data parallel training (see Figure 2) is motivated by the parameter server [11] and federated learning [8] approaches. Algorithm 1 explains the pseudo code of our approach. We consider each AMP as a compute unit which trains its local model using the local data stored on VDISK. Training of local models is done using batch stochastic gradient descent [7]. After the local training on each AMP, local models are aggregated to get a global model. Various aggregation strategies (like weighted averaging, etc.) can be considered. We assume data is uniformly distributed among the AMPS, and we directly average the local model parameters. During each iteration, the global model replaces the local model on each AMP.

Algorithm 1 is a direct extension of federated learning algorithm modified for the shared-nothing architecture of Teradata SQL Engine. During training process, data movement is completely avoided. Local models are aggregated to share the learned knowledge across AMPs. In current setting, during training updates to the global model are delayed until each AMP processes its local data. Several other extensions like delaying updates only for a few data points or sharing only the model updates are also possible. We leave these extension for future exploration.

2.3 Implementation

The concept of Teradata table operator is used to implement Algorithm 1. Similar to database physical operators, a table operator 1) receives an input stream as a set of rows 2) performs an arbitrary computation that transforms or summarizes the input and 3) produces an output stream of rows that may be used by another operator. Table operators are compiled code integrated to the query processing that runs in parallel on each AMP and performs a computation based on the local data. We created an in-house implementation in C-language for the batch stochastic gradient descent, which was scheduled on each AMP using table operator.

3. ML MODELS AND WORKFLOW

The machine learning models which are currently supported within Teradata SQL Engine are multiple linear regression and fully connected neural networks. We update the model parameters iteratively. The hyperparameters such as batch size and learning rate can be controlled for each iteration. The iterations can be terminated after the desired validation matrix (accuracy, loss) is achieved or after predefined iteration maximum count.

3.1 Multiple Linear Regression and Neural Network

Multiple Linear Regression: Multiple linear regression is very widely used simple model. It assumes a linear relationship between the explanatory variables and the dependent variable.

Neural Network Neural network are the state of art machine learning models for applications like vision, speech, natural language processing, human activity recognition, etc. In our prototype implementation, Teradata SQL engine supports fully connected 3-layer (input, output and one hidden layer) neural network for multi-class classification.

Data: X: features Y: labels or dependent variable
Function *trainModel(maxIteration)*
  $i = 0$
  while $i <$*maxIteration* **do**
    Copy *GlobalModel* to each AMP's *LocalModel*
    for *each AMP* **do in Parallel**
      LocalModel=LocalTraining(*LocalModel*)
    end
    GlobalModel=Aggregate all *LocalModels*
    $i = i + 1$
  end
  return *GlobalModel*

Function *LocalTraining(LocalModel)*
  for *each batch(X,Y) of local training data* **do**
    Compute the output O of the *LocalModel* using X
    Compute the error E between O and Y
    Update *LocalModel* to minimize the error E.
  end
  return *LocalModel*

Function *Validation()*
  $Val\_error = 0$
  $Val\_accuracy = 0$
  for *each AMP* **do in Parallel**
    for *each batch(X,Y) of local validation data* **do**
      Compute the output O of the *LocalModel* using X
      Compute the error data points *err* using O and Y
      $Val\_error = Val\_error + err$
    end
  end
  $Val\_error = Val\_error / Size\_of\_validation\_data$
  $Val\_accuracy$ = 1.0 - Val_error
  return $Val\_accuracy$

Algorithm 1: Data parallel training within Teradata

3.2 ML Workflow

In this section, we present the machine learning workflow, with focus on neural network models.

3.2.1 Data Ingestion and Preprocessing

The input data is assumed to be present in the relational tables. Rich SQL functionalities supported by Teradata SQL Engine can be used to preprocess the data. In Figure 3, views (NNTrain, NNTest) are created for train data and test data in Sub Figure A. The input table schema for multiple linear regression is considered in format with first column as primary index followed by the dependent variables and last column is considered as the explanatory variable. For neural network, the first column is considered as primary index followed by dependent variables and explanatory variables. The explanatory variables are expected in one-hot encoded format for multi-class classification.

3.2.2 Model Architecture and Initialization

The model parameters are stored in the SQL table. For multiple linear regression the model parameters are stored as a single row in the model table. In case of neural network the model is stored as a table with rows and columns representing weights. The initial model table is created by the user with the desired initialization of weights. We create initial model weights in Python and then insert them into the model table. In Figure 3, Sub Figure B shows the weight initialization for a 3-layer neural network having 784 input, 16 hidden and 10 output nodes respectively for MNIST and Fashion-MNIST datasets.

3.2.3 Model Training

In Figure 3, Sub Figure C, model training is done by calling Function TrainModel(maxIteration) of Algorithm 1. The test accuracy output per iteration is shown on the MNIST for neural network defined in Sub Figure B.

3.2.4 Model Evaluation

After every training iteration, the global model can be validated on the test (or validation) data to monitor the training progress. The training iteration can be terminated after the desired results on test (or validation) data are achieved. In Figure 3, Sub Figure D illustrates the visualization created by monitoring the training process for MNIST and Fashion-MNIST on the test data.

4. DEMONSTRATION DESCRIPTION

Our demo will present to audience the ease of training models within Teradata SQL engine. The SQL queries to the machine learning table operator will be demonstrated for both multiple linear regression and neural network models. We will use Python API of Teradata SQL Engine to show the entire workflow of input data creation, training and validation. The demo will be shown using interactive Jupyter notebooks. Audience will experience the entire machine learning workflow presented in Figure 3. The audience will be free to vary the number of iterations or the number of nodes in neural network model. To check scalability, three Teradata clusters of different size would be available for computation: small, medium and large. While the model is being computed iteratively, the user can observe the convergence of the algorithm, presented by visualization generated by the application. Likewise, we will demonstrate computation of Linear Regression models on clusters of different sizes.

Multiple Linear Regression: We will generate a random regression dataset and show model training. The learned model parameters will be compared by solving the problem with a well-known Python library to validate the accuracy of our implementation in Teradata SQL Engine.

Neural Network: We will demonstrate the training of a neural network model for multi-class classification problems. We will use the classical MNIST and Fashion-MNIST vision datasets to test our in-database implementation. The MNIST dataset consists of 70,000 images of 10-class handwritten digits. Fashion-MNIST consists of images of 70,000 fashion products from 10 categories. The test accuracy results obtained using 4 AMP Teradata system on MNIST and Fashion-MNIST are shown in Figure 3 (Sub Figure D).

We will also demonstrate the use case of predicting mode of transportation using SHL dataset [6]. SHL dataset consists of sensor data recorded by a smartphone.

5. RELATED WORK

In this section, we will give a brief overview of the machine learning capabilities in existing SQL database or closely related engines. Google BigQuery supports training of linear regression and logistic regression models [1]. Microsoft SQL

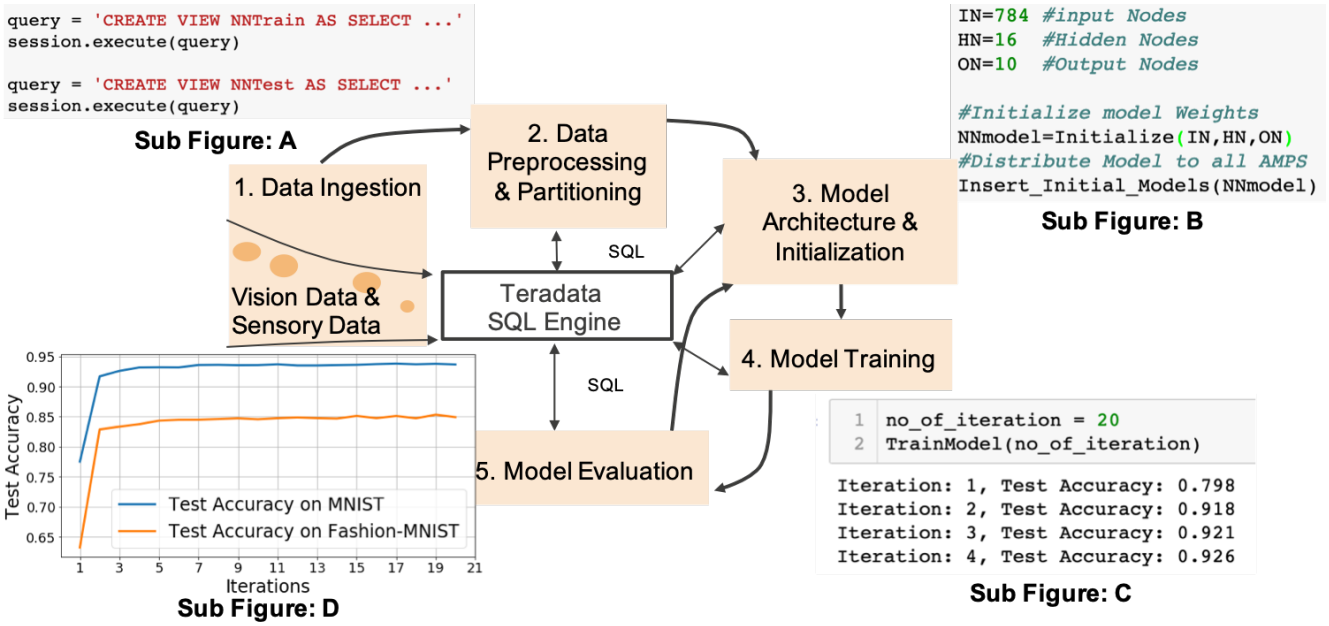

Figure 3: Demo Interactive Workflow: Sub Figure A shows creation of NNTrain (train) view and NNTest (test) view from data table. Sub Figure B illustrates definition of 3-layer fully connected neural network along with initialization. Sub Figure C demonstrates iterative training process along with test accuracy per iteration. Sub Figure D shows the variation in test accuracy with iterations for MNIST and Fashion-MNIST.

server [2] and Oracle [3] supports training of neural network models. However, in this work, we enabled distributed machine learning natively within SQL engines using data parallel training. Our approach considers machine learning as a first-class citizen within database. The algorithms and approach presented can be adopted to enable distributed machine learning capabilities within other database engines.

6. CONCLUSION AND DISCUSSION

The work we present in this demonstration provides a first step to realize the distributed machine learning within a shared-nothing architecture of the database. We are working on providing capabilities to define widely-used neural network architectures with convolutional and LSTM layers. The existing training process can be made communication-efficient by aggregating only the updates from local models rather than the entire local models during iterations. We implemented batch stochastic gradient descent. Different optimization algorithms can be made available to the user. We leave these extensions for future development.

7. REFERENCES

[1] Google BigQuery ML. `https://cloud.google.com/bigquery/docs/bigquery`. Accessed: 2019-03-14.

[2] Neural Networks in Microsoft SQL Server. `https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/microsoft-neural-network-algorithm`. Accessed: 2019-03-14.

[3] Neural Networks in Oracle. `https://docs.oracle.com/en/database/oracle/oracle-database/18/dmcon/neural-network.html`. Accessed: 2019-03-14.

[4] C. Ballinger and R. Fryer. Born To Be Parallel: Why Parallel Origins Give Teradata an Enduring Performance Edge. *IEEE Data Eng. Bull.*, 20(2):3–12, 1997.

[5] J. Catozzi and S. Rabinovici. Operating System Extensions for The Teradata Parallel VLDB. In *VLDB*, 2001.

[6] H. Gjoreski, M. Ciliberto, L. Wang, F. J. O. Morales, S. Mekki, S. Valentin, and D. Roggen. The university of sussex-huawei locomotion and transportation dataset for multimodal analytics with mobile devices. *IEEE Access*, 2018.

[7] G. Hinton, N. Srivastava, and K. Swersky. Neural Networks for Machine Learning Lecture 6a Overview of Mini-batch Gradient Descent. *Coursera Lecture slides*, 2012.

[8] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[9] Y. LeCun. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[10] Michael Stonebraker. The Case for Shared Nothing. *IEEE Database Eng. Bull.*, 9(1):4–9, 1986.

[11] A. Qiao, A. Aghayev, W. Yu, H. Chen, Q. Ho, G. A. Gibson, and E. P. Xing. Litz: Elastic framework for high-performance distributed machine learning. In *USENIX Annual Technical Conference*, 2018.

[12] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.