**ML2SQL**

Compiling a Declarative Machine Learning Language to SQL and Python

Maximilian E. Schüle
schuele@in.tum.de

Matthias Bungeroth
bungeroth@in.tum.de

Dimitri Vorona
vorona@in.tum.de

Alfons Kemper
kemper@in.tum.de

Stephan Günnemann
guennemann@in.tum.de

Thomas Neumann
neumann@in.tum.de

Technical University of Munich

**ABSTRACT**

This demonstration presents a machine learning language MLearn that allows declarative programming of machine learning tasks similarly to SQL. Our demonstrated machine learning language is independent of the underlying platform and can be translated into SQL and Python as target platforms. As modern hardware allows database systems to perform more computational intensive tasks than just retrieving data, we introduce the ML2SQL compiler to translate machine learning tasks into stored procedures intended to run inside database servers running PostgreSQL or HyPer. We therefore extend both database systems by a gradient descent optimiser and tensor algebra.

In our evaluation section, we illustrate the claim of running machine learning tasks independently of the target platform by comparing the run-time of three in MLearn specified tasks on two different database systems as well as in Python. We infer potentials for database systems on optimising tensor data types, whereas database systems showed competitive performance when performing gradient descent.

**1 INTRODUCTION**

Database systems provide with SQL a declarative language that allows data manipulation and data retrieving without caring about optimisation details. With increasing hardware performance, database systems will not fully exploit the servers’ hardware potentials as long as they are used for data retrieval only. To shift computation to the data stored in database systems, algorithms can be specified in SQL—as it has been Turing complete since providing recursive tables—as user-defined functions. The latter allow injecting code as stored procedures to be executed inside the database system and make an additional data manipulation layer on top obsolete. Even though the run-time would decrease, user-defined functions are not fully established as they form a mixture of declarative and procedural language and are inconvenient to express for data scientists.

When dealing with data and minimisation problems, dedicated tools as TensorFlow [1] or Pytorch form the status quo for performing machine learning tasks with tensors and gradient descent. Another approach of formulating machine learning tasks is using a declarative language as MLog [7], that compiles code using TensorFlow, but, so far, it lacks support for use together with database systems. For computations inside of database systems, the support of linear algebra together with matrices or tensors is essential. Different studies focus on the integration of linear algebra [8] and matrices inside of database systems [5]. Going one step further, so-called array database systems replace relations by arrays as the native way of storing attributes. To support machine learning, TensorFlow [6] aims at providing tensor calculus on top of array database systems. One study even provide an own declarative language (BUDS) [2] on top of a prototyped database system that supports matrix data types. Comparable domain specific languages are Weld1 for data driven workloads and IBM SystemML2 for creating flexible algorithms, but both cannot be used inside of database systems. The intermediate language Ferry [3] allows to translate from various code (i.e. Ruby or Haskell) into SQL but is not designed for use with array datatypes.

However, while linear algebra in database systems have been integrated and declarative language concepts have been proposed, there is no successful study on bringing a declarative language, tensor calculus and gradient descent in database systems together. We therefore develop a declarative machine learning language aimed at optimising models for supervised machine learning and data analysis. Our MLearn language allows specifying tasks independently of the target language, changing the underlying engine and makes it easy to compare run-times and results of different underlying frameworks. This demonstration presents the ML2SQL compiler in particular, which compiles code written in MLearn to SQL (for PostgreSQL or HyPer [4]) or to Python using the frameworks NumPy and TensorFlow (see Fig. 1).

This demonstration paper is structured as follows: First, we introduce the MLearn language specifications and the details of the corresponding compiler. We evaluate the run-time of the generated code on the target platforms using the Chicaco taxi

---

1https://github.com/weld-project/weld

2https://systemml.apache.org
dataset as input data and linear regression as optimisation model (optimised by gradient descent or as closed form solution). At the end, we introduce the demonstration concept including our web interface for online testing and conclude by improvements that might increase database systems’ computation performance.

2 MLEARN AND THE ML2SQL COMPILER

The MLearn language is designed to define machine learning tasks in a declarative manner to be compiled to SQL or Python. We begin by introducing the language specification needed for enjoying the demonstration as a visitor. We proceed by listing the prerequisites of the target platforms in order to run the introduced tasks. Finally, we will give examples on how to use the MLearn language during the demonstration later on.

2.1 Language Specification

All operations work on integers, floating point numbers, Boolean values or strings as basic types, which can be composed to tensors. On these types, MLearn provides the following features (s. Lst. 5):

- **Reading CSV files** as the fundamental operation to store the data in variables or relations of the database system.
- **Mathematical expressions** as provided by NumPy or SQL (as part of the projection operator).
- **Tensors** form the main part in our computations. Beside mathematical operations we support accessing, slicing, concatenation and transposition.
- **Functions** allow to structure the code and to reduce code duplication. Also external functions imported from other files or of a target language are allowed.
- **Control blocks.** In addition to our declarative statements, we allow conditional expressions and loops.
- **Distributions** are used for data sampling when initializing tensors with random values.
- **Preprocessor statements** as known from C can be used to include files and to allow the abstraction of different functions to different files.
- **k-fold cross validation** as a predefined building block splits up a data set into training and test sets to find the best—so-called—hyper parameters.
- **Gradient descent** as a separate building block optimises the weights for a given loss function on input data.

2.2 Target Language

We designed our machine learning language to compile to Python code with the libraries that data scientists would use. To work with SQL we picked out the disk-based database system PostgreSQL and its main-memory counterpart, HyPer. We assume for both systems an underlying script language, PL/pgSQL in PostgreSQL and HyPerScript in HyPer, that combines declarative SQL statements with procedural control blocks. The tensor operations in Python are performed using NumPy library calls. HyPer has already implemented all basic tensor operations including addition, (scalar) multiplication, power (including inverse for matrices on negative exponents), transposition, initializing an identity matrix and filling a matrix by a predefined value. As those operations do not exist in PostgreSQL, we have implemented these operations as C library function calls, also supporting parallelism. Furthermore, we make use of predefined array operations as slicing and concatenation to divide the input dataset into training and testing sets. We specify linear regression as closed form solution:

$$\hat{w} = (X^T X)^{-1} X^T y.$$  

The example code (see Lst. 1) splits a tensor (A) into features (X, the first three attributes) and labels (y). Then a tensor out of the value 1 as bias is prepended in front of the features. Afterwards, the optimal weights ($\hat{w}$) are computed out of tensor algebra. The compiled code to Python can be seen in Lst. 2, the one for PostgreSQL in Lst. 4 and the one for HyPer in Lst. 3. We can see that the code written in our declarative language is much more compressed.

3 EVALUATION

For evaluation (s. Fig. 3), we specify linear regression as closed form or using gradient descent in our machine learning language and let the tasks run on the following target platforms: PostgreSQL version 10.5, Python 2.7.15 with NumPy 1.13.3 and TensorFlow 1.3.0 and the current HyPer system. We used an Ubuntu 18.04.01 LTS server with two sockets and twenty cores of Intel Xeon E5-2660 v2 processors in total (supporting hyper-threading). The server has 256 GiB of main memory and uses 1 TiB of SSD as background storage. As test data served 85 mio. tuples of the Chicago taxi rides dataset. We tested the run-time of loading data from CSV (s. Fig. 3a), the run-time of linear regression in closed form (s. Fig. 3b) and by using gradient descent (s. Fig. 3c). For gradient descent, we used a learning rate of 0.0000005 and varied the number of iterations from 1 to $10^4$, but we used a constant input size (the whole dataset). The time measurements consider only the run-time needed for the specific operations, the time for data loading and array creation is measured separately.

The results show that data loading took in all three systems about the same time, no system seems to dominate one another. For the matrix operations used for closed form linear regression, Python using NumPy still dominates the other systems (even PostgreSQL does not support two dimensional array creation for more than $10^7$ tuples). Hence, the integration of matrix calculus in database systems still has to be improved. Whereas using gradient descent, both database systems show competitive performance. Even some performance benefits originate from the used gradient descent optimiser, the results underline the possibility of analyzing data where it is stored.

In summary, the evaluation underlines the claim that the ML2SQL compiler makes it easy to compare different systems and that database systems show competitive performance on certain tasks.

4 DEMONSTRATION

For our demonstration scenario, we have created an interactive web interface (see Fig. 4) that allows formulating tasks in MLearn, compiling to the choosable target language and executing the tasks. Switching between the different target platforms (Python, HyPer, PostgreSQL) makes it possible to compare the results and the run-times of each target language. Behind the web interface,
The results showed that it was indeed feasible to run the tasks as stored procedures inside of database systems showing comparable run-time especially during matrix creation.

Overall we have shown the potential of a declarative machine learning language of expressing tasks compactly and being independent of the underlying engine. As future work—to boost the capabilities of database systems for array data—remains the development of efficient array data types and the standardised integration of optimisation methods such as gradient descent inside of database systems.

Acknowledgements

This work is part of the TUM Living Lab Connected Mobility (TUM LLCM) project and has been funded by the Bavarian Ministry of Economic Affairs, Energy and Technology (StMWi) through the Centre Digitisation.Bavaria, an initiative of the Bavarian State Government. Dimitri Vorona has been sponsored in part by the German Federal Ministry of Education and Research.
Figure 3: Run-time of (a) data loading from CSV, (b) solving linear regression using equation systems or (c) by gradient descent: For data loading and closed form linear regression, we varied the input size; for gradient descent we varied the number of iterations. PostgreSQL did not support the needed array operations for more than $10^7$ tuples.

(BMBF), grant TUM: 01IS12057. This research has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 725286).

REFERENCES