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 intense 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 show 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—or 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 to 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

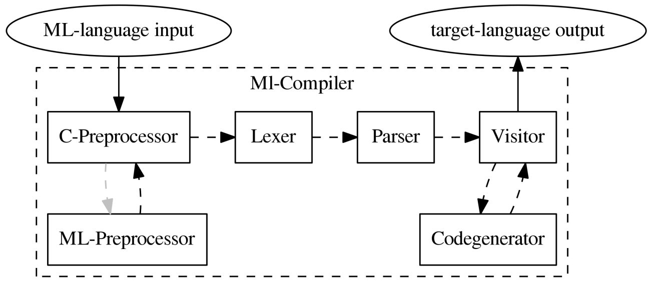


Figure 1: The compilation process: the *MLearn* language first gets preprocessed twice for handling includes, then the language gets tokenised and parsed. For each target platform, a generator allows to translate the abstract syntax tree into the target language.

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, TensorDB [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 Weld[1] for data driven workloads and IBM SystemML[2] 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

[1] https://github.com/weld-project/weld
[2] https://systemml.apache.org

Series ISSN: 2367-2005
10.5441/002/edbt.2019.56

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 precede 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 one. Also, we wrap a PostgreSQL library extension around our already presented gradient descent [9] library to allow in-database gradient descent in PostgreSQL.

2.3 Example

Fig. 2 shows the exemplary usage of the *MLearn* language where we specify linear regression as closed form solution:

$$\vec{w} = (X'^T X')^{-1} X'^T \vec{y}.$$

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 (`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 Chicaco taxi rides dataset[3].

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,

[3] https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew/data

```
A = [[1.1,0.98,87.3,3],[0.1,3.15,42.05,3.3],
     [100.5,26.8,10.1,225.1],
     [1097.5,23000,10.1,24850.1]]
X = A[: , 0:2]
y = A[: , 3]
bias[1,len(X,0)] : 1
X = (bias::X.T).T
Xt = X.T
w = (Xt*X)^(-1) * Xt * y
print '%' , w
```

Listing 1: The specification in *MLearn*.

```
import numpy as np
def main():
    A = np.array([np.array([1.1,0.98,87.3,3]),np.array([0.1,3.15,42.05,3.3]),np..
        array([100.5,26.8,10.1,225.1]),np.array([1097.5,23000,10.1,24850.1])])
    X = DATA[ :,0:2 +1]
    y = DATA[ :,3:3 +1]
    bias = np.full( ( 1,np.size(X ,0 )), 1)
    X = (np.append(bias,X.T, axis=0)).T
    Xt = X.T
    w = np.dot( np.dot( np.linalg.matrix_power(( np.dot(Xt, X)), (-1)), Xt), y)
    print( '{}'.format(  w))
if __name__ == "__main__": main()
```

Listing 2: The translated code for Python using NumPy.

```
CREATE OR REPLACE FUNCTION ML_main() AS $$
  var A = array[array[1.1::float,0.98::float,87.3::float,3],array[
         0.1::float,3.15::float,42.05::float,3.3::float],array
        [100.5::float,26.8::float,10.1::float,225.1::float],array
        [1097.5::float,23000,10.1::float,24850.1::float]];
  var X = array_resetlower(array_slice(A,1,array_length(A,1),(0+1)
        ::int,(2+1)::int));
  var y = array_resetlower(array_slice(A,1,array_length(A,1),(3+1)
        ::int,(3+1)::int));
  var bias = array_fill(1::float, 1,array_length(X,0+1));
  X = array_transpose((array_cat(bias,array_transpose(X))));
  var Xt = array_transpose(X);
  var w = power((Xt*X), (-1)::int)*Xt*y; debug_print( '%',w);
$$ LANGUAGE 'hyperscript' strict;
select ML_main(); DROP FUNCTION ML_main();
```

Listing 3: As HyPerScript code for HyPer.

```
DO $$ declare
  A float[][]; X float[][]; Xt float[][];
  bias float[][]; w float[][]; y float[][];
begin
  A := array[array[1.1::float,0.98::float,87.3::float,3],array
        [0.1::float,3.15::float,42.05::float,3.3::float],array
        [100.5::float,26.8::float,10.1::float,225.1::float],array
        [1097.5::float,23000,10.1::float,24850.1::float]]
  X := A[:][0+1:2+1]; y := A[:][3+1:3+1];
  bias := array_fill(1::float, ARRAY[1,array_length(X,0+1)]);
  X := matrix_transpose((array_cat(bias,matrix_transpose(X))));
  Xt := matrix_transpose(X);
  w := matrix_power((Xt * X), (-1)::int ) * Xt * y;
  RAISE NOTICE '%',w;
END$$;
```

Listing 4: For PostgreSQL as PL/pgSQL procedure.

Figure 2: Closed form linear regression specified in *MLearn* and translated to Python and SQL: Lst. 1 shows the initial specification, a matrix of fixed values is created, then the optimal weights are computed by solving an equation system; Lst. 2 shows the translated code to Python using the NumPy matrix library calls. The other listings show the stored procedures in HyPerScript for HyPer (Lst. 3) and in PL/pgSQL for PostgreSQL (Lst. 4).

```
expression: INJECT | 'print' mathexplist | 'if' '(' mathexp ')' '{' explist '}' ['else' '{' explist '}'] | ('continue' | 'break')
 | 'while' '(' mathexp ')' '{' explist '}' | 'for' VARNAME ('from' mathexp 'to' mathexp | 'in' interval) '{' explist '}' | functions
 | 'create' 'tensor' VARNAME 'from' VARNAME '(' varlist ')'
 | 'save' 'tensor' VARNAME 'to' (VARNAME|STRING) [':' STRING] '(' varlist ')'
 | VARNAME '[' accessor (',' accessor)* ']' ('=' | ':') mathexp | VARNAME (',' VARNAME)* '=' VARNAME '(' [mathexp (',' mathexp)*] ')'
 | VARNAME '[' accessor (',' accessor)* ']' '~' VARNAME '(' [mathexp (',' mathexp)*] ')' | 'import' VARNAME
 | VARNAME '=' (mathexp | 'distribution' '(' VARNAME ',' VARNAME ')') | VARNAME '~' VARNAME '(' [mathexp (',' mathexp)*] ')'
 | returnType* 'function' VARNAME '(' [VARNAME (',' VARNAME)*] ')' '{' explist '}' | 'return' mathexp (',' mathexp)*
 | ('readcsv'|'writecsv') '{' (('name:' VARNAME) | ('file:' STRING) | ('columns:' varlist) | ('replace␣empty␣entries:' mathexp)
                              |('delimiter:' STRING) | ('replace:' '{' (STRING ':' STRING)+ '}') | ('delete␣empty␣entries'))+ '}'
 | 'gradientdescent' '{' (('function:' STRING) | ('data:' varlist) | ('optimize:' [nameshape (',' nameshape)*])
                        | ('learningrate:' mathexp) | ('maxsteps:' mathexp) | ('batchsize:' mathexp) | ('threshold:' mathexp))+ '}'
 | 'plot''{'(('xData:'mathexp) | ('yData:'mathexp) | ('xLabel:'STRING) | ('yLabel:'STRING) | ('type:'STRING) | ('filename:'STRING))+'}'
 | 'crossvalidate' '{' (('minfun' ':' VARNAME '=' fun) | ('kernel' ':' VARNAME ':' VARNAME ':' mathexp)
                      | ('data' ':' VARNAME (',' VARNAME)*) | ('n' ':' mathexp) | ('lossfun' ':' fun)
                      | ('folds' ':' mathexp) | ('test' '{' (VARNAME '=' interval)+ '}'))+ '}' ;
```

Listing 5: Grammar of the MLearn language: Predefined building blocks for gradient descent, cross validation, CSV file handling as well as procedural control blocks, function calls and the declaration of variables are allowed as expressions.

we run a PostgreSQL and an HyPer database server fed with an excerpt of the Chicago taxi dataset. The demonstration visitors are invited to try out the introduced types of tensor algebra as well as minimising arbitrary loss functions as, for example, linear or logistic regression.

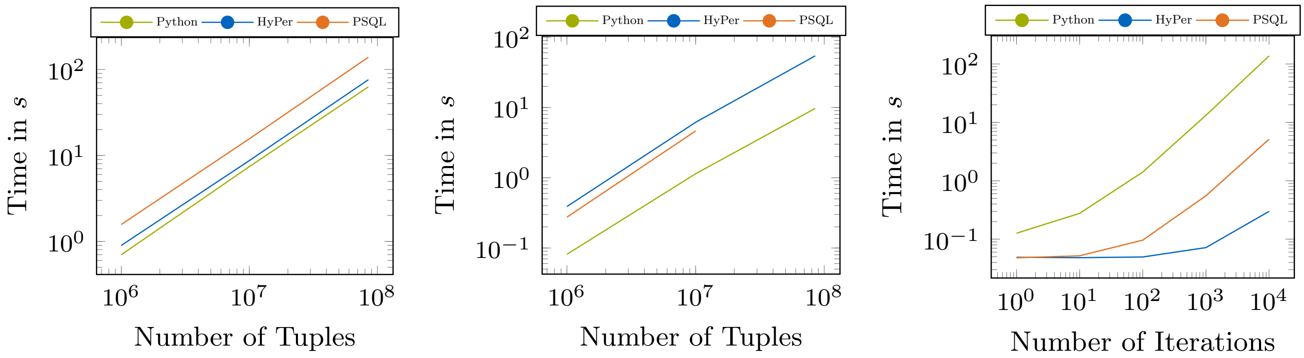
5 CONCLUSION

This demonstration presented the first declarative machine learning language *MLearn*, which allows describing machine learning tasks independently of the target engine and whose compiler allows running the code in the core of database systems. This paper first introduced comparable approaches before it presented the language specifications for performing linear regression and gradient descent using any possible loss function. Then, we evaluated the run-time of the tasks on the different target platforms PostgreSQL, HyPer and Python using NumPy and TensorFlow. 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



(a) Loading data from CSV. **(b) Closed form linear regression.** **(c) Linear regression with gradient descent.**

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.

Figure 4: Web interface for an interactive exploration of the *MLearn* language: Above left, the text editor allows to specify tasks, which are translated into the selected target language (above right). The code will be executed in the terminal.

(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

[1] M. Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[2] Z. J. Gao, S. Luo, L. L. Perez, and C. Jermaine. The BUDS language for distributed bayesian machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 961–976, 2017.

[3] T. Grust, M. Mayr, J. Rittinger, and T. Schreiber. FERRY: database-supported program execution. In *ACM SIGMOD, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 1063–1066, 2009.

[4] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 195–206, 2011.

[5] D. Kernert, F. Köhler, and W. Lehner. Bringing linear algebra objects to life in a column-oriented in-memory database. In *Proceedings of the 1st International Workshop on In Memory Data Management and Analytics, IMDM 2013, Riva Del Garda, Italy, August 26, 2013.*, pages 37–49, 2013.

[6] M. Kim and K. S. Candan. Tensordb: In-database tensor manipulation with tensor-relational query plans. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 2039–2041, 2014.

[7] X. Li, B. Cui, Y. Chen, W. Wu, and C. Zhang. Mlog: Towards declarative in-database machine learning. *PVLDB*, 10(12):1933–1936, 2017.

[8] S. Luo, Z. J. Gao, M. N. Gubanov, L. L. Perez, and C. M. Jermaine. Scalable linear algebra on a relational database system. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, pages 523–534, 2017.

[9] M. Schüle, F. Simonis, T. Heyenbrock, A. Kemper, S. Günneman, and T. Neumann. In-database machine learning: Gradient descent and tensor algebra for main memory database systems. In *18th symposium of "Database systems for Business, Technology and Web" (BTW), in Rostock, Germany. Proceedings*, 2019.