In-Database Machine Learning: Using Gradient Descent and Tensor Algebra

Maximilian E. Schüle, Frédéric Simonis, Thomas Heyenbrock, Alfons Kemper, Stephan Günnemann, Thomas Neumann
Rostock, 04. März 2019
What Need Database Systems for ML?

Database Systems  Machine Learning  Why don‘t use HyPer?
What Need Database Systems for ML?

**Machine Learning**: data in tensors and a parametrised loss function

**HyPer**

**Tensors**

**Gradient Descent**

**Advantages**: Optimisation problems are solvable in the core of database servers

**Goal**: Make database systems more attractive

**What it is**: Architectural blueprint for the integration of optimisation models in DBMS

**What it is not**: Study about the quality of different optimisation problems
What is Gradient Descent?

**Linear Regression**

$$m_{a,b}(rm) = a \cdot rm + b \approx medv$$

Optimal weights?
Gradient Descent!

$$l_{rm,medv}(a,b) = (m_{a,b}(rm) - medv)^2$$

**Training Data**

<table>
<thead>
<tr>
<th>RM</th>
<th>MEDV</th>
</tr>
</thead>
</table>

**Test Data**

<table>
<thead>
<tr>
<th>RM</th>
<th>MEDV</th>
</tr>
</thead>
</table>

How to optimise weights?
How to label data?
Approach

**HyPer**
- Integration as *operators* in relational algebra
- Representation of *mathematical functions* on relations
- Concept of *pipelines*

**Gradient Descent**
- Gradient needed
- Automatic differentiation

**Tensors**
- Representation of tensors
- Either: one relation represents one tensor
- Or: own tensor data type
Integration in Relational Algebra

**Operator Tree**
Operators for labelling and gradient descent: Pipelines (Weights/Data)

**Model / Loss Function**
Representation of a loss- as well as of a model function

Model function $m$

$$m_{w}(x) = \sum_{i \in m} x_i w_i \approx y$$

Loss function $l$

$$l_{x,y}(w) = (m_{w}(x) - y)^2$$

**Pipelining**
Integration as a pipeline breaker
Integration in Relational Algebra: Operator Tree

Two Operators needed
- Gradient descent to optimise weights of a parametrised loss function
- Labelling operator to label predicted values

Gradient Descent
- Initial weights and training data as input and optimised weights as output
- Lambda expression as loss function to be optimised

Labelling
- Input: test dataset and optimal weights
- Label: evaluated lambda expression for each tuple
Integration in Rel. Algebra: Lambda Functions

**Lambda Expression**
To inject user-defined code

```
select * from kmeans((table points), \( \lambda(a,b) \sqrt{(a.x-b.x)^2+(a.y-b.y)^2} \), 2)
```

\( \lambda(a,b) \sqrt{(a.x-b.x)^2+(a.y-b.y)^2} \)  
Euclidean Distance
Integration in Rel. Algebra: Lambda Functions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Relations/Lambda Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
<td>( w = { w_1, w_2, ..., w_m } )</td>
</tr>
<tr>
<td>( n ) tuple with ( m ) attributes</td>
<td>( x = { x_1, x_2, ..., x_m, y } )</td>
</tr>
<tr>
<td>Model function</td>
<td>( m_w(x) = \sum_{i \in m} x_i \cdot w_i \approx y )</td>
</tr>
<tr>
<td>Loss function</td>
<td>( l_{x,y}(w) = (m_w(x) - y)^2 )</td>
</tr>
</tbody>
</table>

**Lambda Functions in SQL**

```sql
create table trainingdata (x float, y float);
create table weights(a float, b float);
insert into trainingdata...
insert into weights...

select * from gradientdescent(  
  -- loss function as \( \lambda \)-expression
  \( \lambda(data, weights)(weights.a * d.x + weights.b - d.y)^2 \),  
  -- training set and initial weights
  (select x,y from trainingdata d),  
  (select a,b from weights),  
  -- learning rate and max. number of iteration
  0.05, 100
);
```

```sql
create table testdata (x float);
create table weights(a float, b float);
insert into trainingdata...
insert into weights...

select * from labeling(  
  -- model function as \( \lambda \)-expression
  \( \lambda(data, weights)(weights.a * d.x + weights.b) \),  
  -- training set and initial weights
  (select x,y from testdata d),  
  (select a,b from weights)  
);
```
Integration in Relational Algebra: Pipelining
Integration in Relational Algebra: Pipelining

Materialising
- Materialisation of all tuples (parallel/serial)
- Any optimisation method possible
- Parallelism: `parallel_for`

Pipelined
- No materialisation
- Stochastic gradient descent only
- Distribution to pipelines
- Downside: multiple copies of the operator tree

Combined
- First iteration in pipelines
- Remaining ones in the main thread
Automatic Differentiation for Gradient Descent

Need of a gradient for gradient descent: Automatic differentiation necessary

HyPer compiles SQL before execution
→ precompilation of the gradient, evaluation for each tuple using placeholders

```cpp
auto status = model_optimizer->trainable.train(
    ValuedNodes{model_gradient->model.placeholders, tensors});
```
Tensor Data Type

Extension of the PostgreSQL array data type

- transpose
- addition/subtraction/scalar product
- multiplication (inner tensor product)

Linear Regression

\[
(t^T)_{i_1i_2...i_n} = t_{i_2...i_n i_1}
\]

\[
(t+s)_{i_1i_2...i_n} = t_{i_1i_2...i_n} + s_{i_1i_2...i_n}
\]

\[
T \in \mathbb{R}^{I_1 \times \cdots \times I_m}, U \in \mathbb{R}^{J_1 \times \cdots \times J_n},
\]

\[
s_{i_1i_2...i_n} = \sum_{k \in [0]} t_{i_1i_2...i_{n-1}k} u_{kj}.
\]

Linear Regression in SQL with Tensors

```sql
select (array_inverse (array_transpose (x) * x)) * (array_transpose (x) * y)
from (select array_agg (x) x
from (select array [1, x_1, x_2] as x
from datapoints) sx)
) tx, (select array_agg (y) y
from datapoints)
yx
) ty;
```
Evaluation
Evaluation

Tools
HyPer, MariaDB 10.1.30, PostgreSQL 9.6.8 with MADlib v1.13, TensorFlow 1.3.0, R 3.4.2

Machine
Intel Xeon E5-2660 v2 CPU (20x 2.20 GHz)
256 GB DDR4 RAM
Nvidia GeForce GTX 1050 Ti

Data
Chicago Taxi Rides Dataset (10^6 Tupel)

Tests
Linear regression (2-3 attributes)
Logistic regression (2 attributes)
k-Means clustering
Evaluation – Runtimes of GD

Runs
5000 iterations

Database systems: no time for data loading needed
HyPer faster, PSQL and MariaDB (using procedures) slower
Evaluation – Ratio Computation/Loading Time

**Runs**
parameters: 10 iterations, $10^6/10^7$ tuples

**Most of the time: data loading**
Not necessary, when computation is done inside of the database system
Evaluation – Architectures

**Evaluation of the architectures:** materialising, pipelined, combined
Standard parameters: 10 iterations, $10^6$ tuples, one thread

**Observation**
Pipelined faster, but only allows stochastic GD and needs fixed number of iterations
All implementations scale
Combined plan: low
Conclusion

Database systems: more computations (tensors + gd)

Aim of the work
Saving time by moving ML operations into the core of DBMS
Gradient descent and labelling in SQL + Lambda
Different architectures for gradient descent

Future Work
Support of tensor datatypes
  Second view on relations: combining SQL and ArrayQL
Generic language for machine learning
  Dedicated language that compiles to SQL
Embedding of Python or R in SQL