Large-Scale Machine Learning with the HeAT Framework

Martin Siggel, Kai Krajsek, Philipp Knechtges, Markus Götz, Claudia Comito, Björn Hagemeier

DLR-SC, High-Performance Computing

WaW 3 Machine Learning, 20.11.2018, DLR Cologne
Motivation: Helmholtz Analytics Framework (HAF)

- Joint project of all 6 Helmholtz centers to foster data analytics within Helmholtz.

- Scope: Systematic development of domain-specific data analysis techniques in a co-design approach between domain scientists and information experts.

People who have applications in mind.

People who provide the tools (us!).
Motivation: HAF Use Cases

Earth System Modelling

SEVIRI Satellite Images – Near Real Time

Research with Photons

Neuroscience

Aeronautics and Aerodynamics

Structural Biology
Motivation: HAF Use Case Methods

- **Clustering**
  - k-means, mean shift clustering

- **Uncertainty quantification**
  - Ensemble methods

- **Dimension reduction**
  - Autoencoder, reduced order models

- **Feature learning**
  - Image descriptors, autoencoder

- **Data assimilation**
  - Kalman filter, 4Dvar, particle filter/smooother

- **Classification/Regression**
  - Random forest, CNN, SVM

- **Modelling**
  - Fiber tractography, point processes

- **Optimization techniques**
  - L-BFGS, simulated annealing

- **Hyper-parameter optimization**
  - Evidence framework, grid search

- **Interpolation**
  - Radial basis function, Kriging

- **Data mining**
  - Frequent item set mining
Greatest Common Denominator?

Machine Learning

= Lots of Data

+ Numerical Linear Algebra

Sounds like something HPC has been doing anyway.

https://xkcd.com/1838/
What is HeAT?

- **HeAT** = Helmholtz Analytics Toolkit
- Developed within the Helmholtz Analytics Framework Project
- The hot new machine learning / data analytics framework to come.
- Developed in the open:
  - Available on [https://github.com/helmholtz-analytics](https://github.com/helmholtz-analytics) and [https://pypi.org/project/heat](https://pypi.org/project/heat)
- Liberally licensed: MIT
Big Data Landscape
Big Data/Deep Learning Libraries

Big Data

- hadoop
- DISCO
- mahout
- Apache Storm
- Apache Spark
- DASK

Deep Learning

- PyTorch
- TensorFlow
- dy/net
- PaddlePaddle
- CNTK
- mxnet
- ArrayFire
- Chainer
- Keras
Which technology stack to use?

- We do not want to reinvent the wheel and there are already plenty of machine learning frameworks available.

- Common denominator: all frameworks provide Python as a front end language

  ➔ We also choose

  ➔ Which framework to use as a basis?
    ➔ It is better to measure than to guess ➔ Benchmark!
Technology Benchmarks

• Testing several Machine Learning Frameworks

by implementing

• K-means
• Self-Organizing Maps (SOM)
• Artificial Neural Networks (ANN)
• Support Vector Machines (SVM)

• Evaluation criteria:

• Feature completeness
• Ease of implementation
• Performance (on-going effort)
Evaluation

- Feature completeness:

<table>
<thead>
<tr>
<th>Framework</th>
<th>GPU</th>
<th>MPI</th>
<th>AD</th>
<th>LA</th>
<th>nD Tensors</th>
<th>SVD</th>
<th>Dist. tens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pytorch</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TensorFlow</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MXNet</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Arrayfire</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- Ease of implementation and usage: Pytorch and MXNet
- Performance: Pytorch and MXNet

- Since we need MPI for distributed parallelism, we have chosen Pytorch as a basis for our Machine Learning toolkit: HeAT
Scope:

- Facilitating Use Cases of HAF in their work
- Bringing HPC and Machine Learning / Data Analytics closer together
- Ease of use

Design:

- PyTorch
  - Tensor Linear Algebra
  - Automatic Differentiation
  - Pythonic NumPy-like interface
  - GPU support
  - Distributed Parallelism (MPI)

Algorithms:

- k-means
- SVM
- NN
<table>
<thead>
<tr>
<th>Runs on:</th>
<th>Data structure</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumPy</td>
<td>ND-Tensor</td>
<td>- Elementwise operations</td>
</tr>
<tr>
<td></td>
<td>shape: (4, 3, 2)</td>
<td>- Slicing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Matrix operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Reduction</td>
</tr>
</tbody>
</table>
**Runs on:**
- CPU
- GPU

**Data structure:**
- ND-Tensor

**Shape:** (4, 3, 2)

**Operations:**
- Elementwise operations
- Slicing
- Matrix operations
- Reduction
- Automatic differentiation
Runs on: MPI or

Data structure
ND-Tensor
shape: (4, 3, 2)

Operations
- Elementwise operations
- Slicing
- Matrix operations
- Reduction
- Automatic differentiation
Data Distribution

Example:

```python
import heat as ht
# construct a range tensor
>>> range_data = ht.arange(6, split=1)

Server#1  [0, 1]
Server#2  [2, 3]
Server#3  [4, 5]

>>> range_data.mean()
2.5
>>> range_data.argmax()
5
```
What has been done so far?

• The core technology has been identified

• Implementation of a distributed parallel tensor class has begun

• Implemented standard vector and matrix operations

• Parallel data I/O via HDF 5 and NETCDF

• A first implementation of the k-means algorithm is available
Example: k-means

- Core idea: **k clusters** around centroids

- Minimization

\[
\arg\min_c \sum_{i=1}^{k} \sum_{x \in C_i} \left\| x - \hat{x} \right\|^2
\]

- Algorithm sketch
1. Choose k centroids
2. For each point calculate distance to centroids
3. Assign point to **closest centroid**
4. Estimate new centroid as **mean** of points
5. Go to 2. until **convergence**
Example: k-means

2. For each point calculate distance to centroids
3. Assign point to closest centroid

```
>>> data.shape
(18, 2, 1)

>>> centroids.shape
(1, 2, 2)
```

**NumPy**

```
>>> distances = ((data - centroids) ** 2).sum(axis=1, keepdims=True)
>>> matching_centroids = np.expand_dims(distances.argmin(axis=2), axis=2)
```

**HeAT**

```
>>> distances = ((data - centroids) ** 2).sum(axis=1)
>>> matching_centroids = distances.argmin(axis=2)
```

```
>>> matching_centroids.shape
(18, 1, 1)
```
Example: k-means

Numpy vs. HeAT

```python
>>> matching_centroids.shape
(18, 1, 1)
```

NumPy

```python
>>> for i in range(self.n_clusters):
>>>     selection = (matching_centroids == i).astype(np.int64)
```

HeAT

```python
>>> for i in range(self.n_clusters):
>>>     selection = (matching_centroids == i).astype(ht.int64)
```

```python
>>> selection.shape
(18, 1, 1)
```
Example: k-means

Numpy vs. HeAT

```python
>>> matching_centroids.shape
(18, 1, 1)

>>> data.shape
(18, 2, 1)
```

**NumPy**

```python
>>> for i in range(self.n_clusters):
...     new_centroids[:, :, i:i+1] = ((data*selection).sum(axis=0, keepdims=True) /
...     selection.sum(axis=0).clip(1.0, sys.maxsize))
```

**HeAT**

```python
>>> for i in range(self.n_clusters):
...     new_centroids[:, :, i:i+1] = ((data*selection).sum(axis=0) /
...     selection.sum(axis=0).clip(1.0, sys.maxsize))
```

```python
>>> new_centroids.shape
(1, 2, 2)
```
Transparent development process

Github for code review, issue tracking, sprint planning

Travis for continuous integration

Mattermost for discussions

Feel free to join us there!
What’s next?

• We said that HPC people and codes are trained in Numerical Linear Algebra ….
• ….. but this mostly holds only for sparse matrices!
• In, e.g. SVMs, one has to solve a linear system with the matrix

\[ K = \begin{pmatrix} k_{11} & k_{12} & \cdots \\ k_{21} & k_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \]

\[ k_{ij} = \exp (-\gamma \|x_i - x_j\|^2) \]

• The \( k_{ij} \) never become zero, but can be arbitrarily close to zero.
• Very unfortunate to have many almost zeros, but still \( O(n^2) \) complexity for matrix vector multiplication.
• Could one not partially approximate the matrix with low-rank matrices? → **Hierarchical Matrices**

Figure taken from Steffen Börm’s lecture notes „Numerical Methods for Non-Local Operators“
Thanks for listening.
Questions?