Machine Learning In A Snap

—

Thomas Parnell
Research Staff Member
IBM Research - Zurich
What are GLMs?

- Ridge Regression
- Lasso Regression
- Support Vector Machines
- Logistic Regression

Regression

Generalized Linear Models

Classification
Why are GLMs useful?

Fast Training
Can scale to datasets with billions of examples and/or features.

Less tuning
State-of-the-art algorithms for training linear models do not involve a step-size parameter.

Interpretability
New data protection regulations in Europe (GDPR) give E.U. citizens the right to “obtain an explanation of a decision reached” by an algorithm.

Widely used in industry
The Kaggle ”State of Data Science” survey asked 16,000 data scientists and ML practitioners what tools and algorithms they use on a daily basis.

37.6% of respondents use Neural Networks
63.5% of respondents use Logistic Regression
What is Snap Machine Learning?

Snap ML: A new framework for fast training of GLMs

<table>
<thead>
<tr>
<th>Framework</th>
<th>Models</th>
<th>GPU Acceleration</th>
<th>Distributed Training</th>
<th>Sparse Data Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>scikit-learn</td>
<td>ML/{DL}</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Apache Spark* MLlib</td>
<td>ML/{DL}</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TensorFlow**</td>
<td>ML</td>
<td>Yes</td>
<td>Yes</td>
<td>Limited</td>
</tr>
<tr>
<td>Snap ML</td>
<td>GLMs</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* The Apache Software Foundation (ASF) owns all Apache-related trademarks, service marks, and graphic logos on behalf of our Apache project communities, and the names of all Apache projects are trademarks of the ASF.
**TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.
Multi-level Parallelism

Level 1
Parallelism across nodes connected via a network interface.

Level 2
Parallelism across GPUs within the same node connected via an interconnect (e.g. NVLINK).

Level 3
Parallelism across the streaming multiprocessors of the GPU hardware.
GLMs can be effectively trained using stochastic coordinate descent (SCD).

[Shalev-Schwartz 2013]

Recently, asynchronous variants of SCD have been proposed to run on multi-core CPUs.

[Liu 2013]
[Tran 2015]
[Hsiesh 2015]

Twice parallel asynchronous SCD (TPA-SCD) is another recent variant designed to run on GPUs.

It assigns each coordinate update to a different block of threads that executed asynchronously.

Within each thread block, the coordinate update is computed using many tightly coupled threads.
When the dataset is too big to fit into aggregate GPU memory, we need to copy the training data onto the GPU in batches. The time required to copy the next batch of data on the GPU can become a bottleneck.

In Snap ML we use CUDA streams to implement a streaming pipeline. We copy the data for the next batch while the GPU is training on the current batch. With high-speed interconnects such as NVLINK, this allows us to completely hide the copy time behind the training time.
<table>
<thead>
<tr>
<th>libglm</th>
<th>snap-ml-local</th>
<th>snap-ml-mpi</th>
<th>snap-ml-spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underlying C++/CUDA template library.</td>
<td>Small to medium-scale data.</td>
<td>Large-scale data.</td>
<td>Large-scale data</td>
</tr>
<tr>
<td>Multi-GPU support.</td>
<td>Many-GPU support.</td>
<td>Many GPU support</td>
<td></td>
</tr>
<tr>
<td>scikit-learn compatible Python API</td>
<td>Python API.</td>
<td>Python/Scala /Java* API.</td>
<td></td>
</tr>
</tbody>
</table>

*Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.*
Example: snap-ml-local

```python
# Load data
from sklearn.datasets import load_from_svmlight_format
X, y = load_from_svmlight_format(filename_train)

# Train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Create the logistic regression
if use_snap_ml:
    from snap_ml import LogisticRegression
    lr = LogisticRegression(device_ids=[0, 1])
else:
    from sklearn.linear_model import LogisticRegression
    lr = LogisticRegression()

# Training
lr.fit(X_train, y_train)

# Inference
proba_test = lr.predict_proba(X_test)

# Evaluate logarithmic loss on test set
from sklearn.metrics import log_loss
test_loss = log_loss(y_test, proba_test)
```

Acceleration existing scikit-learn applications by changing only 2 lines of code.
Example: snap-ml-mpi

Describe application using high-level Python code.

```python
# Load data
from snap_ml_mpi.Loaders import load_from_snap_format
train_data = load_from_libsvm_format(train_filename)
test_data = load_from_libsvm_format(test_filename)

# Create the logistic regression
from snap_ml_mpi import LogisticRegression
lr = LogisticRegression(max_iter=200, dual=True, device_ids=[0,1,2,3], num_threads=128)

# Training
lr.fit(train_data)

# Inference
proba_test = lr.predict_proba(test_data)

# Evaluate logarithmic loss on test set
from snap_ml_mpi.Metrics import log_loss
test_loss = log_loss(y_test, proba_test)
```

Launch application on 4 nodes using mpirun (4 GPUs per node):

```
$ mpirun -n 4 -rf myrankfile python my_app.py
```
Click-through Rate (CTR) Prediction

Can use train ML models to predict whether or not a user will click on an advert?

Core business of many internet giants.

Labelled training examples are being generated in real-time.

Dataset: criteo-kaggle
Number of features: 1 million
Number of examples: 45 million
Size: 40 Gigabytes (SVM Light)

Dataset: criteo-tb
Number of features: 1 million
Number of examples: 4.2 billion
Size: 3 Terabytes (SVM Light)
Single-node Performance

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Examples</th>
<th>Nodes</th>
<th>Total GPUs</th>
<th>Network</th>
<th>CPU-GPU interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>criteo-kaggle</td>
<td>45 million</td>
<td>1x Power AC922 server</td>
<td>1x V100</td>
<td>n/a</td>
<td>NVLINK 2.0</td>
</tr>
</tbody>
</table>

![Graph showing performance](image-url)

- scikit-learn (LIBLINEAR)
- TensorFlow (V100 GPU)
- Snap ML (V100 GPU)
Terabyte-scale Benchmark

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Examples</th>
<th>Nodes</th>
<th>Total GPUs</th>
<th>Network</th>
<th>CPU-GPU interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>criteo-tb</td>
<td>4.2 billion</td>
<td>4x Power AC922 server</td>
<td>16x V100</td>
<td>InfiniBand</td>
<td>NVLINK 2.0</td>
</tr>
</tbody>
</table>

![Graph showing performance data](image)
Conclusions

Snap ML is a new framework for fast training of GLMs.

Snap ML benefits from GPU acceleration.

It can be deployed in single-node and multi-node environments.

The hierarchical structure of the framework makes it suitable for cloud-based deployments.

It can leverage fast interconnects like NVLINK to achieve streaming, out-of-core performance.

Snap ML significantly outperforms other software frameworks for training GLMS in both single-node and multi-node benchmarks.

Snap ML can train a logistic regression classifier on the Criteo Terabyte Click Logs data in 1.5 minutes.

Snap ML is available as part of IBM PowerAI (v5.2 onwards)

The Snap ML team:

Celestine Dünner
Dimitrios Sarigiannis
Andreea Anghel
Nikolas Ioannou
Haris Pozidis
Thomas Parnell

https://www.zurich.ibm.com/snapml/
Example: snap-ml-spark

Describe application using high-level Python code:

```python
# Load data
from snap_ml_spark import DatasetReader
train_data = DatasetReader().setFormat("libsvm").load(train_filename)
test_data = DatasetReader().setFormat("libsvm").load(test_filename)

# Create the logistic regression
from snap_ml_spark import LogisticRegression
lr = LogisticRegression(max_iter=40, regularizer=1.0, use_gpu=True)

# Training
lr.fit(train_data)

# Inference
test_data_with_pred = lr.predict_proba(test_data)

# Evaluate accuracy on the test set
from snap_ml_spark.Metrics import accuracy
accuracy = accuracy(test_data_with_pred)
```

Launch application on Spark cluster (1 GPU per Spark executor):

```
$ spark-submit --jars snap-ml-spark-v1-ppc64le.jar my_app.py
```
**Out-of-core performance (PCIe Gen3)**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Examples</th>
<th>Nodes</th>
<th>Total GPUs</th>
<th>Network</th>
<th>CPU-GPU interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>criteo-tb</td>
<td>200 million</td>
<td>1x Intel Xeon* Gold 6150</td>
<td>1x V100</td>
<td>n/a</td>
<td>PCIe Gen3</td>
</tr>
</tbody>
</table>

**Diagram**

- **S1**
  - Init
  - Train chunk (i) on GPU
  - Copy chunk (i+1) onto GPU
  - Time: 12ms, 90ms, 318ms, 330ms

- **S2**
  - Init
  - Train chunk (i+1) on GPU
  - Copy chunk (i+2) onto GPU
  - Time: 12ms, 90ms, 318ms, 330ms

*Intel Xeon is a trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IBM Research - Zurich / Machine Learning In A Snap / August 2018 / © 2018 IBM Corporation
Out-of-core performance (NVLINK 2.0)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Examples</th>
<th>Nodes</th>
<th>Total GPUs</th>
<th>Network</th>
<th>CPU-GPU interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>criteo-tb</td>
<td>200 million</td>
<td>1x Power AC922 server</td>
<td>1x V100</td>
<td>n/a</td>
<td>NVLINK 2.0</td>
</tr>
</tbody>
</table>

IBM Research - Zurich / Machine Learning In A Snap / August 2018 / © 2018 IBM Corporation