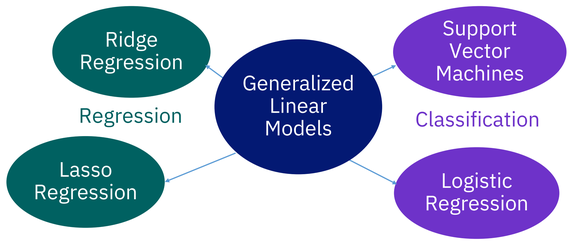
Snap Machine Learning:
A Hierarchical Software Framework for Machine Learning on Heterogeneous Systems

C. Dünner, A. Anghel, T. Parnell, D. Sarigiannis, N. Ioannou, H. Pozidis

Abstract

- Snap Machine Learning [1] is a new framework for **fast training** of **generalized linear models**.
- It benefits from **GPU acceleration** and it can be deployed in **single-node** and **multi-node systems**.
- Its **hierarchical structure** makes it suitable for **cloud-based environments**.
- It can train a logistic regression classifier on the **Criteo Terabyte Click Logs** data in **1.5 minutes**.
- **It will be available to try in June 2018 as part of IBM PowerAI (Tech Preview).**

Generalized Linear Models



$$\min_{\mathbf{w}} f(A^\top \mathbf{w}) + \sum_{i\in[n]} g_i(w_i) \quad A = [\mathbf{x}_1, ..., \mathbf{x}_n] \quad \mathbf{w} \in \mathbb{R}^n \quad \mathbf{x}_i \in \mathbb{R}^d$$

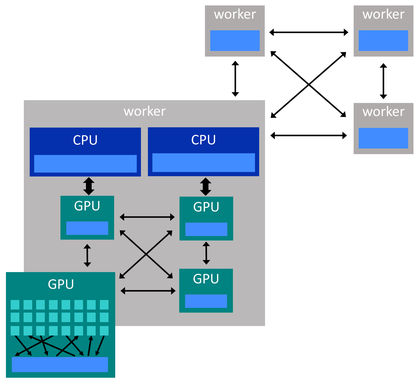
63.5% of the respondents of the Kaggle "State of Data Science" survey use Logistic Regression. Only 37.6% of the respondents answered Neural Networks.

What is Snap Machine Learning?

Framework	Models	GPU Acceleration	Distributed Training	Sparse Data Support
scikit-learn	ML\{DL}	No	No	Yes
Apache Spark* MLlib	ML\{DL}	No	Yes	Yes
TensorFlow**	ML	Yes	Yes	Limited
Snap ML	GLMs	Yes	Yes	Yes

* 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.

Snap Machine Learning Architecture



Level 1

Parallelism across nodes connected via a network interface.

Level 2

Parallelism across GPUs within the same node connected via an interconnect.

Level 3

Parallelism across the streaming multiprocessors of the GPU.

GPU Solver

Each local sub-task can be solved effectively using stochastic coordinate descent (SCD).[Shalev-Schwartz'13]

Recently, asynchronous variants of SCD have been proposed to run on multi-core CPUs. [Liu'13, Tran'15, Hsiesh'15]

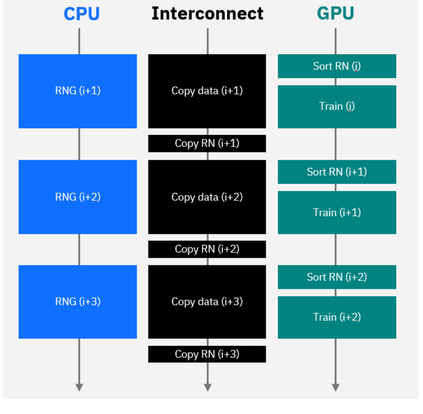
Twice-Parallel Asynchronous SCD [2] is a recent variant of SCD designed to run on GPUs. It assigns each coordinate update to a different block of threads executed asynchronously.

Primal-Dual Coordinate Descent

1: Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
2: Initialize $\mathbf{v}^{(0)} = \mathbf{0}$ $(\mathbf{v} := A^\top \mathbf{w})$
3: **for** $t = 0, 1, 2, ...$ **do**
4: randomly select coordinate $j \in [n]$
5: find coordinate update Δw_j
6: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta w_j \mathbf{e}_j$
7: $\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \Delta w_j \mathbf{x}_j^\top$
8: **end for**

Parallelized over streaming multiprocessors

CPU-GPU Streaming Pipeline



The CPU-GPU link can become a bottleneck during training.

Using CUDA streams, we copy the next set of data onto the GPU while the current set is being trained.

We introduce a 3rd pipeline stage where the CPU generates a set of random numbers for sampling in the GPU solver.

User API Overview

LibGLM

Underlying C++/CUDA template library.

*Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

snap-ml-local

- Small to medium-scale data.
- Single-node deployment.
- Multi-GPU support.
- scikit-learn compatible Python API.

snap-ml-mpi

- Large-scale data.
- Multi-node deployment in HPC environments.
- Many-GPU support.
- Python API.

snap-ml-spark

- Large-scale data.
- Multi-node deployment in Apache Spark environments.
- Many-GPU support.
- Python/Scala/ JAVA* API.

Local and MPI Examples

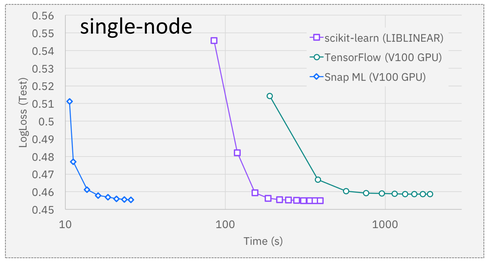
```python
# Load data
from sklearn.datasets import load_from_svmlight_format
X, y_ = load_from_svmlight_format(filename_train)

# 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 LogisticRegressi
    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)
```

```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)
```

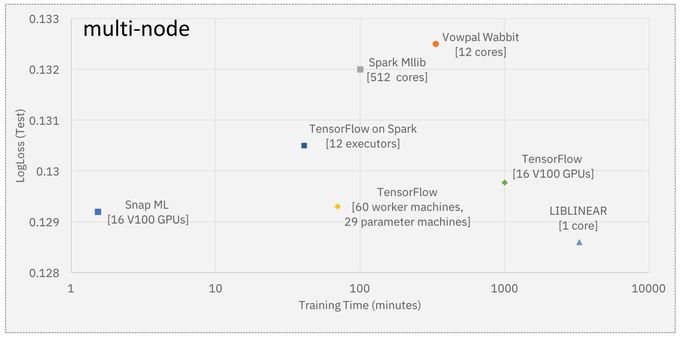
Launch application on 4 nodes using mpirun (4 GPUs per node):

```
$ mpirun -n 4 -rf myrankfile python my_app.py
```

Experimental Results



	Single-node	Multi-node
Dataset	criteo kaggle	criteo
# examples	45 million	4.2 billion
# features	1 million	1 million
Size	40 GB	3 TB
Nodes	1x Power AC922	4x Power AC922
GPUs	1x V100	16x V100
Network	n/a	InfiniBand
CPU/GPU Interface	NVLINK 2.0	NVLINK 2.0



[1] Snap Machine Learning, *C. Dünner et al.*, arXiv:1803.06333 (2018)
[2] Tera-Scale Coordinate Descent on GPUs, *T. Parnell et al.*, Future Generation Computer Systems (2018)
[2] Efficient Use of Limited-Memory Accelerators for Linear Learning on Heterogeneous Systems, *Dünner et al.*, NIPS (2017)