Guoyu Chen
The Ohio State University
chen.9605@osu.edu
Outline

● cuML Overview

● cuML vs. Scikit Learn

● RAPIDS

● Scalable Distributed ML
What is cuML?

- Python API matches scikit-learn API;
- Including time series algorithms (ARIMA);
- Traditional tabular ML tasks on GPUs;
- CUDA Machine Learning;
- 10-50× faster than CPU equivalents with large datasets (≥ 100,000 samples);
- Part of RAPIDS.
Deep learning vs. Classical ML

- Deep learning
  - Large and unstructured datasets (text, images, ...);
  - Convolution layers, activation layers, dense layers, ... 
  - Extract features automatically.

- Classical ML
  - Structured datasets;
  - Feature extraction is a separate step.
  - Samples as rows and features as columns;
  - Decision trees, random forests, support vector machines;
Code Example
```python
import cudf
from cuml.datasets import make_regression
from cuml.model_selection import train_test_split
from cuml.linear_model import LinearRegression
from cuml.metrics.regression import r2_score

n_samples = 2**10
n_features = 200
random_state = 23

# Lets generate some random regression data
X, y = make_regression(n_samples=n_samples, n_features=n_features,
                        random_state=random_state)
X = cudf.DataFrame(X)
y = cudf.DataFrame(y)
X_cudf, X_cudf_test, y_cudf, y_cudf_test = train_test_split(X, y,
                                                             test_size=0.2, random_state=random_state)

# Fit the model
cls_cuml = LinearRegression(fit_intercept=True, normalize=True, algorithm='eig')
cls_cuml.fit(X_cudf, y_cudf)

# Predict with the trained model
predict_cuml = cls_cuml.predict(X_cudf_test)

# Calculate R^2
r2_score_cuml = r2_score(y_cudf_test, predict_cuml)
print("R^2 score (cuml): %s" % r2_score_cuml)

import pandas as pd
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics.regression import r2_score

n_samples = 2**10
n_features = 200
random_state = 23

# Lets generate some random regression data
X, y = make_regression(n_samples=n_samples, n_features=n_features,
                        random_state=random_state)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size=0.2, random_state=random_state)

# Fit the model
cls_sk = LinearRegression(fit_intercept=True, normalize=True, n_jobs=-1)
cls_sk.fit(X_train, y_train)

# Predict with the trained model
predict_sk = cls_sk.predict(X_test)

# Calculate R^2
r2_score_sk = r2_score(y_test, predict_sk)
print("R^2 score (sklearn): %s" % r2_score_sk)
```
Run time comparison of cuML vs. sklearn models

![Graph showing run time comparison of cuML vs. sklearn models.](image)

Courtesy: [2012.04201] GPU Accelerated Exhaustive Search for Optimal Ensemble of Black-Box Optimization Algorithms (arxiv.org)
• A suite of open source software libraries and Python APIs that execute end-to-end data science and analytics pipelines entirely on GPUs;
• Including cuML, cuDF (a pandas-like dataframe manipulation library), cuGraph (a NetworkX-like accelerated graph analytics library);
• Incubated by NVIDIA and utilizes CUDA primitives for low-level compute optimization;
• Started from Apache Arrow and GoAi (2017) projects;
• Apache Arrow: Based on a columnar, in-memory data structure that delivers efficient and fast data interchange with flexibility to support complex data models.
Scalable Distributed Machine Learning
Distributed cuML with DASK: Training Stage

CUDA IPC: CUDA Inter-Process Communication (Intra node).
GPU Direct RDMA: Enabled network cards to bypass the CPU and access memory directly on the GPU (Inter node).

Courtesy: Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence (arxiv.org)
Distributed cuML with DASK: Trained Model

Courtesy: Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence (arxiv.org)
Distributed cuML with DASK: Inference Stage

Courtesy: Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence (arxiv.org)
Final Remark: Communication Middleware

- **CUDA-Aware MPI**
  - Mpi4py library exposes it to Python;
  - Allowing CUDA pointers to be passed around across multiple GPU devices;
  - Reductions and broadcasts are on the Host.

- **NVIDIA Collective Communications Library (NCCL)**
  - MPI-like API;
  - Reductions are entirely on GPUs;
  - Popular in libraries for distributed deep learning, like PyTorch, TensorFlow, Chainer, Horovod;
  - Used in classical ML libraries with distributed algorithms, such as XGBoost, H2O GPU, and cuML.
Final Remark: Communication Middleware (Cont’d)

- *MPI and NCCL assumes that workers communicate synchronously in real-time.*
  
  - Dask and Apache Spark in the application level:
    
    - Asynchronous task-scheduled systems;
    
    - Building directed acyclic computation graphs (DAGs) that represent the dependencies between arbitrary tasks.
Key Takeaways

● cuML is similar to scikit-learn, but running on GPU;

● RAPIDS project builds an ecosystem of GPU-accelerated Python tools, including cuML, cuDF, cuGraph …;

● Dask-CUDA enables both single-node multi-GPU and multi-node multi-GPU environments;

● CUDA IPC provides the possibility for intra-node memory scaling up;

● GPU Direct RDMA provides the possibility for inter-node memory scaling up;

● NCCL enables reductions entirely on GPUs;

● Dask and Apache Spark are asynchronous task-scheduled systems for general-purpose scalable distributed computing.
References

1. RAPIDS: https://rapids.ai/about.html.
Q & A