Accelerated Machine Learning Algorithms in Python

Patrick Reilly, Leiming Yu, David Kaeli
reilly.pa@husky.neu.edu

Northeastern University Computer Architecture Research Lab
Outline

• Motivation and Goals
• Background information
• Algorithm Descriptions and Results
• Areas to Improve
Motivation

• Machine Learning Algorithms are typically computationally expensive
  • Scale poorly with large datasets

• Focus on Python
  • Popular prototyping language
  • Use scikit-learn, a Python library
  • Optimized machine learning algorithms
Objectives

• Attempt to speed up Machine Learning Algorithms in Python by utilizing GPUs in Python
  • Test using the Cython and PyCUDA frameworks
  • Compare to scikit-learn
    • Single and multithreaded

• Focus on k-Nearest Neighbors, Logistic Regression and k-Means
  • k-Nearest Neighbors - classifier
  • Logistic Regression - supervised learning
  • k-Means - unsupervised learning
Why GPUs?

- Graphical Processing Units (GPUs) are made up of many processing cores
- Designed for massively parallel computations
- Machine Learning Algorithms can benefit from parallel computing

NVIDIA Maxwell SMM Architecture
Cython and PyCUDA

- Cython is a Python-based language that is extended to support C-types and functions
  - Can generate C code from Python code
  - Can be used to link C/C++ code to Python
    - Can be extended to allow for launching CUDA code
- PyCUDA extends Python to allow for easy access to the CUDA API
  - PyCUDA’s base layer is written in C++ for speed
  - Can call kernel without need for additional wrapper
Cython vs. PyCUDa

- Both enable Python to call CUDA code that will be run on a GPU
- PyCUDa allows for easier access to CUDA API
- Cython averaged slower times

k-Means runtime with Cython and PyCUDa
Test Conditions

- Run optimized algorithms that utilize PyCUDA against scikit-learn versions
- Run on a machine with an Intel Core i7-4790k and NVIDIA 950 GTX
k-Nearest Neighbors (kNN)

k-Nearest Neighbors Pseudocode:

**Training**: Points whose labels are known

for all points \( p \) to be classified:

- Compute distance to \( p \) from all members of **Training**
- Get the \( k \) shortest distances from \( p \)
- Classify \( p \) with the most common label of the \( k \) closest points in **Training**
k-Nearest Neighbors (kNN)

- Classifies by checking the labels of the k nearest points
- Requires computing distances between test point and all other points
- Requires a sort to get the k-minimum distances
kNN Speed up

- Finding distances and sorting is expensive
- Distances computed with kernel
- Sort handled using NumPy function `argpartition`
- Speed ups of 27.9x-106.9x for classifying one point
Logistic Regression Pseudocode:

**Training**: Points whose labels are known

**Betas**: Weights to predict labels

**While** not converged:

- Multiply **Training** points with corresponding **Betas**
- Sum the results of **Training** * **Betas**
- Run sum through Sigmoid function to compute **Expected Classes**
- Compute error based on actual labels and **Expected Classes**
Logistic Regression (LR)

- Logistic Regression takes a Bayesian approach to prediction
  - Computes probability that point lies in a certain class
  - Requires minimization of error function
Logistic Regression Speed Up

• Error was calculated based on differences with expected classes
  • Simplified using matrices
  • CUBLAS was used to speedup matrix operations
• Timed average time per iteration due to variety of halting methods
• Speed increase of 46.4x-197.9x
k-Means Pseudocode:

**Data:** Points to be split into k groups

Randomly initialize k points as means

While means have not converged:

Classify points in Data by closest mean

Update means to center of points in their class

Compute error to check convergence
k-Means

- Computes means within the data set that determine class

- A new point is classified by the closest mean

- Requires a large number of distance computations over many iterations to find the optimal arrangement
k-Means Speed Up

- Cached centroids in constant memory
- Used shuffle instruction to compute local histogram
- Used atomic adds to update memberships located in global memory
- Speed up of 41.7x to 129.8x
Takeaway

- GPUs can provide large speedups when training machine learning algorithms
- Can benefit from speed increase in Python
Future Improvements

- kNN loses speed increase when classifying many points
  - Bottlenecked by argpartition
  - Add non-brute force methods
- Logistic Regression currently runs for a max number of iterations
  - Implement a halting optimization mechanism
- Implement more Machine Learning algorithms to build a CUDA enhanced Machine Learning library for Python
Thank you for listening!
Questions?