Parallel computational paradigms for large-scale data processing
— Week #10
Today

A. Distributed computing for machine learning
   - Background
   - Short introduction to MapReduce/Hadoop & Spark

B. Summary of material seen so far

Note: Most lectures so far used stats concepts. Today we’ll turn to computer science.
Distributed Computation for Machine Learning
Data & Computation
Data & Computation

• We generate massive quantities of data
Data & Computation

- We generate massive quantities of data

1. Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s
   (source: internetlifestats.com)
Data & Computation

- We generate massive quantities of data

  1. Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s
     (source: internetlifestats.com)
  2. Banks, insurance companies, etc.
Data & Computation

- We generate massive quantities of data

  1. Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s
     (source: internetlifestats.com)
  2. Banks, insurance companies, etc.
  3. Modestly-sized websites
Data & Computation

- We generate massive quantities of data

  1. Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s
     (source: internetlifestats.com)
  2. Banks, insurance companies, etc.
  3. Modestly-sized websites

- Both large $n$ and large $p$
Data & Computation

- We generate massive quantities of data
  1. Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s (source: internetlifestats.com)
  2. Banks, insurance companies, etc.
  3. Modestly-sized websites
- Both large \( n \) and large \( p \)
- In general computation will scale up with the data
We generate massive quantities of data

1. Google 4K searches/s, Twitter: 6K tweets/s, Amazon: 100s sold products/s (source: internetlifestats.com)
2. Banks, insurance companies, etc.
3. Modestly-sized websites

- Both large \( n \) and large \( p \)
- In general computation will scale up with the data

- Often fitting an ML models requires one or multiple operations that looks at the whole dataset

  e.g., Linear regression \( w = (X^T X)^{-1}X^T Y \)
Issues with massive datasets

1. Storage

2. Computation
Moore’s Law: The number of transistors on microchips doubles every two years

Moore’s law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
Modern Computation paradigms
Modern Computation paradigms

- Floating point operations per second (Flop)
- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga
Modern Computation paradigms

1. “Single” computers

- Large Computers
  - 513, 855 TFlops

Floating point operations per second (Flop)

- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken
Modern Computation paradigms

1. “Single” computers
   - Large Computers
     - 513, 855 TFlops

2. Distributed computation
   - ~200, 000 TFlops
     (Folding@home)

Floating point operations per second (Flop)

- Smart phone ~ 0.6 TFlops
- 1 Tera: 1,000 Giga

https://www.top500.org/lists/top500/list/2020/06/
Modern Computation paradigms

1. "Single" computers
   - Large Computers
     - 513, 855 TFlops

2. Distributed computation
   - ~200, 000 TFlops
     (Folding@home)

3. Specialized hardware
   - Focusses on subset of operations
     - Graphical Processing Unit (GPU), Field Programmable Gated Array (FPGA)
     - ~100 TFlops

Floating point operations per second (Flop)

Smart phone ~ 0.6 TFlops

1 Tera: 1,000 Giga

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken
GPUs & other specialized hardware
Hardware

- Central Processing Unit (CPU)
  - Computer’s cognition
  - Executes all the instructions from software
  - Arithmetics, read/writes, logic, etc.

http://www.personal.psu.edu/users/d/dlm99/cpu.html
Neural Networks

- Linear Algebra:
  - Multiplication $\text{vector} \times \text{matrix}$
    - Neuron activation of multiple neurons
    - Neuron activation for multiple datum
  - Multiplication $\text{matrix} \times \text{matrix}$
    - Activation of multiple neurons for multiple datum
- The exact dimensions of these vector & matrix depend on the data size (mini batch) and the number of neurons
Parallelizing neural network computations

\[ C = AB \]
Parallelizing neural network computations

\[ C = AB \]
\[ C_{ij} = \sum_k A_{ik} B_{kj} \]
Parallelizing neural network computations

\[ C = AB \]

\[ C_{ij} = \sum_k A_{ik} B_{kj} \]
Parallelizing neural network computations

- The value of $C$ can be computed independently from one another.
- Matrix multiplication can be parallelized.

$$C = AB$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$
Parallelizing neural network computations

- The value of $C$ can be computed independently from one another.
- Matrix multiplication can be parallelized.
- When parallelizing we can then obtain very significative gains.
- Depend on the size of $C$.

\[
C = AB
\]

\[
C_{ij} = \sum_k A_{ik}B_{kj}
\]
Parallelizing neural network computations

- The value of $C$ can be computed independently from one another.
- Matrix multiplication can be parallelized.
- When parallelizing we can then obtain very significative gains.
- Depend on the size of $C$.
- En practice, divide in tiles.

$$C = AB$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$
Parallelizing neural network computations

- The value of $C$ can be computed independently from one another.
- Matrix multiplication can be parallelized.
- When parallelizing we can then obtain very significative gains.
- Depend on the size of $C$.
- En practice, divide in tiles.

Note: not all linear algebra operations can be as easily parallelized. Notably, the matrix inverse.
Specialized hardware

- Graphical Processing Unit (GPU)
  - Initially designed for 3D games
  - Specialized for linear algebra operations
  - Thousands of cores (vs. a few tens for CPUs)
  - Fast access to memory
  - Multi-GPUs for a single computer
Specialized hardware

- Graphical Processing Unit (GPU)
- Initially designed for 3D games
- Specialized for linear algebra operations
- Thousands of cores (vs. a few tens for CPUs)
- Fast access to memory
- Multi-GPUs for a single computer

https://cryptomining-blog.com/tag/multi-gpu-mining-rig/
Even more specialized hardware

- Tensorflow Processing Unit (TPU)
- Developed by Google for neural networks (ASIC)
- Supports matrix multiplication operations
- Training & Test
- Precision of operations is lower compared to GPUs
- Multiple TPUs per “machine”

https://en.wikipedia.org/wiki/Tensor_Processing_Unit
Distributed Computing
Distributed Computing

• Faster computers can help
Distributed Computing

- Faster computers can help
- What about a large of “slow” computers working together?
  - Divide the computation into small problems
    1. All (slow) computers solve a small problem at the same time
    2. Combine the solution of small problems into initial solution
Distributed Computing

• Faster computers can help

• What about a large of “slow” computers working together?

• Divide the computation into small problems
  1. All (slow) computers solve a small problem at the same time
  2. Combine the solution of small problems into initial solution
Building our intuition with a simple example

• You are tasked with counting the number of houses in Montreal

1. Centralized (single computer):
   • Ask a marathon runner to jog around the city and count
   • Build a system to count houses from satellite imagery
Building our intuition with a simple example

- You are tasked with counting the number of houses in Montreal

1. Centralized (single computer):
   - Ask a marathon runner to jog around the city and count
   - Build a system to count houses from satellite imagery

2. Distributed (many computers):
   - Ask 1,000 people to each count houses from a small geographical area
   - Once they are done they report their result at your HQ
Tool for distributed computing (for machine learning)

  - Builds on MapReduce ideas
  - More flexible computation graphs
- High-level APIs
  - MLlib
Distributed Computing using MapReduce
MapReduce

- From Google engineers

  “MapReduce: Simplified Data Processing on Large Clusters”, Jeffrey Dean and Sanjay Ghemawat, 2004

- Now also known as (Apache) Hadoop

- Google built large-scale computation from commodity hardware

- Specific distributed interface

- Useful for algorithms that can be expressed using this interface
MapReduce

• Two types of tasks:

A. Map: Solve a subproblem (filtering operation)

B. Reduce: Combine the results of map workers (summary operation)
TASK: Create a document's bag-of-word representation

A. Map

- The black dog
- A black cat
- The blue cat

B. Reduce
TASK: Create a document's bag-of-word representation

A. Map

The black dog
A black cat
The blue cat

B. Reduce
A. Map

The black dog
A black cat
The blue cat

B. Reduce

The, 1
black, 1
dog, 1

A, 1
black, 1
cat, 1

The, 1
blue, 1
cat, 1
TASK: Create a document's bag-of-word representation

A. Map

- The, 1
- black, 1
- dog, 1

- A, 1
- black, 1
- cat, 1

- The, 1
- blue, 1
- cat, 1

B. Reduce

Partition by key

- The, 1
- black, 1
- dog, 1
- cat, 1
- blue, 1
- cat, 1
- The, 1
- black, 1
- dog, 1
- blue, 1
- cat, 1

Laurent Charlin — 60629
TASK: Create a document's bag-of-word representation

A. Map

The, 1
black, 1
dog, 1

The black dog

A black cat

The, 1
blue, 1
cat, 1

The blue cat

B. Reduce

The, 1
black, 1
dog, 1

The, 2
black, 2
dog, 1
cat, 2
Some details

- Typically the number of subproblems is higher than the number of available machines
  - ~linear speed-up wrt to the number of machines
- If a node crashes, need to recompute its subproblem
- Input/Output
  - Data is read from disk when beginning
  - Data is written to disk at the end
MapReduce is quite versatile

- When I was at Google the saying was (roughly):

  “If your problem cannot be framed as MapReduce you haven’t thought hard enough about your problem.”

- A few examples of “map-reduceable” problems:

  - Intuition: Your problem needs to be decomposable into map functions and reduce functions

  - Sorting, filtering, distinct values, basic statistics

  - Finding common friends, sql-like queries, sentiment analysis
MapReduce for machine learning

1. Training linear regression
   - Reminder: there is a closed-form solution

\[ w = (X^T X)^{-1} X^T Y \]
MapReduce for machine learning

1. Training linear regression

• Reminder: there is a closed-form solution

\[ w = (X^T X)^{-1} X^T Y \]

\[ w = (\sum_{ij} X_i^T X_j)^{-1} (\sum_{i} X_i^T Y_i) \]
MapReduce for machine learning

1. Training linear regression

   • Reminder: there is a closed-form solution

\[
\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}
\]

\[
\mathbf{w} = (\sum_{ij} \mathbf{X}_i^\top \mathbf{X}_j)^{-1} (\sum_{i} \mathbf{X}_i^\top \mathbf{Y}_i)
\]

   • Each term in the sums can be computed independently
MapReduce for machine learning

1. Training linear regression
   - Reminder: there is a closed-form solution

\[ w = (X^T X)^{-1} X^T Y \]

\[ w = \left( \sum_{ij} X_i^T X_j \right)^{-1} \left( \sum_i X_i^T Y_i \right) \]

- Each term in the sums can be computed independently
MapReduce for machine learning

1. Training linear regression
   - Reminder: there is a closed-form solution

\[ w = (X^T X)^{-1} X^T Y \]

2. Other models we studied have a closed form solution (e.g., Naive Bayes and LDA)

Each term in the sums can be computed independently.

\[ w = \left( \sum_{ij} X_i^T X_j \right)^{-1} \left( \sum_i X_i^T Y_i \right) \]
MapReduce for machine learning

1. Training linear regression
   - Reminder: there is a closed-form solution

   \[
   w = (X^T X)^{-1} X^T Y
   \]

   \[
   w = \left( \sum_{ij} X_i^T X_j \right)^{-1} \left( \sum_{i} X_i^T Y_i \right)
   \]
   • Each term in the sums can be computed independently

2. Other models we studied have a closed form solution (e.g., Naive Bayes and LDA)

3. Hyper-parameter search
   - A neural network with 2 hidden layers and 5 hidden units per layer and another with 3 hidden layers and 10 hidden units
Shortcomings of MapReduce

- Many models are fitted with iterative algorithms
  - Gradient descent:
    1. Find the gradient for the current set parameters
    2. Update the parameters with the gradient
- Not ideal for MapReduce
  - Would require several iterations of MapReduce
  - Each time the data is read/written from/to the disk
Distributed computing using Apache Spark
(Apache) Spark

- Advantages over MapReduce

1. Less restrictive computations graph (DAG instead of Map then Reduce)
   - Doesn’t have to write to disk in-between operations
2. Richer set of transformations
   - map, filter, cartesian, union, intersection, distinct, etc.
3. In-memory processing
Spark History

• Started in Berkeley’s AMPLab (2009)

• Version 1.0 2014
  • Based on Resilient Distributed Datasets (RDDs)

• Version 2.0 June 2016
  • V2.3 February 2018, V2.4.4 September 2019, V3.3.1 October 2022

• Pyton + Spark: pySpark

• Good (current) documentation:
  2. Project docs: https://spark.apache.org/docs/latest/
Resilient Distributed Datasets (RDDs)

- A data abstraction
  - Collection of partitions. Partitions are the distribution unit.
  - Operations on RDDs are (automatically) distributed.
- RDDs support two types of operations:
  1. Transformations
     - Transform a dataset and return it
  2. Actions
     - Compute a result based on an RDD
     - These operations can then be “chained” into complex execution flows
DataFrames

- An extra abstraction on top of RDDs
  - Encodes rows as a set of columns
    - Each column has a defined type
  - Useful for (pre-processed) machine learning datasets
- Same name as `data.frame` (R) or `pandas.DataFrame`
  - Similar type of abstraction but for distributed datasets
- Two types of operations (for our needs): transformers, estimators.
Spark’s “Hello World”

data = spark.read.format("libsvm").load("hdfs://...")
model = LogisticRegression(regParam=0.01).fit(data)
Spark’s “Hello World”

```python
DataFrame → data = spark.read.format("libsvm").load("hdfs://...")
model = LogisticRegression(regParam=0.01).fit(data)
```
Spark’s “Hello World”

```python
DataFrame(data = spark.read.format("libsvm").load("hdfs://..."))

model = LogisticRegression(regParam=0.01).fit(data)
```

*DataFrame* → data = spark.read.format("libsvm").load("hdfs://...")

model = LogisticRegression(regParam=0.01).fit(data)

*Estimator*
Parallel gradient descent

- Logistic Regression

$$y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)}$$
Parallel gradient descent

• Logistic Regression

\[ y = \frac{1}{1 + \exp(-w_0 - w_1x_1 - w_2x_2 - \ldots - w_px_p)} \]

• No closed-form solution, can use gradients

\[ \frac{\partial \text{Loss}(Y, X, w)}{\partial w_i} \]
Parallel gradient descent

• Logistic Regression

\[ y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)} \]

• No closed-form solution, can use gradients

\[ \frac{\partial}{\partial w_i} \text{Loss}(Y, X, w) \]

• Loss functions are often decomposable

\[ \frac{\partial}{\partial w_i} \sum_j \text{Loss}(Y_j, X_j, w) \]
Parallel gradient descent

• Logistic Regression

\[ y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)} \]

• No closed-form solution, can use gradients

\[ \frac{\partial \text{Loss}(Y, X, w)}{\partial w_i} \]

• Loss functions are often decomposable

\[ \frac{\partial \sum_j \text{Loss}(Y_j, X_j, w)}{\partial w_i} \]
ML setup

1. Load your data as an RDD

Machine Learning Library (MLlib) Guide
MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Feature extraction: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and loading algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Classification and Regression - RDD-based API
The spark.mllib package supports various methods for binary classification, multiclass classification, and regression analysis. The table below outlines the supported algorithms for each type of problem.

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Supported Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Classification</td>
<td>linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes</td>
</tr>
<tr>
<td>Multiclass Classification</td>
<td>logistic regression, decision trees, random forests, naive Bayes</td>
</tr>
<tr>
<td>Regression</td>
<td>linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression</td>
</tr>
</tbody>
</table>

https://spark.apache.org/docs/latest/ml-guide.html
Takeaways

• Distributed computing is useful:
  • for large-scale data
  • for faster computing

• Current frameworks (e.g., spark) offer easy access to popular ML models + algorithms

• Useful speedups by decomposing the computation into a number of identical smaller pieces

• Still requires some engineering/coding