Scalable Machine Learning

- or -

what to do with all that Big Data infrastructure

Mikio L. Braun
TU Berlin
blog.mikiobraun.de
@mikiobraun

Strata+Hadoop World
London, 2015
Complex Data Analysis at Scale

• Click-through prediction
• Personalized Spam Detection
• Image Classification
• Recommendation
• …
Click Through Prediction

$p(\text{click}|q,a) = 2$
Personalized Spam Detection

"bag of word"

Dear: 1  Sir: 1  I: 1 have: 1 a: 5 Lucrative: 1 ...
4711/dear: 1 4711/sir: 1 4711/I: 1 4711/have: 1...

personalized "bag of words"
Supervised Learning in a Nutshell

Data

Model

\[ \text{learning} = \frac{\text{adjusting parameters}}{\text{to minimize}} \]

True vs. Predicted (error, loss, cost)

Input \( X \)

Parameters

Output \( Y \)
Large Scale Learning

- 100 millions of examples
- Millions of (potential) features
- Linear models: Learn one weight per feature
- Deep learning: Complex models with several stages of processing
ML Pipeline

ML Pipeline:

1. Data extraction
2. Feature extraction & preprocessing
3. Learning Algorithm
4. Model Selection & Evaluation via Training/Test splits
5. Apply model to Future Data

DB → Logs → Data extraction

Raw Data → Features

Learning Algorithm

Model

MapReduce = mostly MR = ?
Easily parallelized

- Data preparation, extraction, normalization
- Parallel runs of ML pipelines for evaluation
- Mass application of data sets
What about the training step?

- Optimization problems
- Sometimes closed-form solutions (→ matrix computations)
- Most often not, or it is slow
Gradient Descent

- Very generic way to optimize functions
- Update parameters along line of steepest descent of cost function

e.g. logistic regression

\[ w \leftarrow w + \eta \sum_{i=1}^{n} (y_i - \sigma(w^T X_i))X_i \]
Gradient Descent in Parallel

Data

compute predictions

sum up gradient
Stochastic Gradient Descent

- #1 workhorse for large scale learning
- Gradient Descent = Error on whole data set
- Stochastic Gradient Descent =

  **consider one point at a time**

1. predict on one point
2. compute gradient for that one point
3. update parameters
Why can we do this?

- Learning is about prediction performance on future data.
- Lots of freedom to find alternative algorithms.

- Learning involves:
  - Training Data → Learned Model → Optimization Problem
  - or → Probabilistic Model → Future Data
  - predict with high accuracy
Ad Click Prediction at Google

Ad Click Prediction: a View from the Trenches

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, Jeremy Kubica
Google, Inc
mcmahan@google.com, gholt@google.com, dsculley@google.com

ABSTRACT
Predicting ad click-through rates (CTR) is a massive-scale learning problem that is central to the multi-billion dollar online advertising industry. We present a selection of case studies and topics drawn from recent experiments in the setting of a deployed CTR prediction system. These include improvements in the context of traditional supervised learning based on an FTRL-Proximal online learning algorithm (which has excellent sparsity and convergence properties) and the use of per-coordinate learning rates.

We also explore some of the challenges that arise in a wide range of online advertising contexts beyond traditional supervised learning. Sponsored search advertising, contextual advertising, display advertising, and real-time bidding auctions have all relied heavily on the ability of learned models to predict ad click-through rates accurately, quickly, and reliably [28, 15, 33, 1, 10]. This problem setting has also pushed the field to address issues of scale that even a decade ago would have been almost inconceivable. A typical industrial model may provide predictions on billions of events per day, using a correspondingly large feature space, and then learn from the resulting mass of data.

In this paper, we present a series of case studies drawn from recent experiments in the setting of the deployed sys-

• Variant of SGD
• With tons of hacks: 16-bit fixed point, sharing memory, etc.
• No parallelization!
SGD is hard to parallelize

Data ➔ stream by data

Parameters

only a few MB max!

update
Parameter Servers

Dean et al. “Large Scale Distributed Deep Networks”, NIPS 2012
Li et al. “Communication Efficient Distributed Machine Learning with the Parameter Server”, NIPS 2014
Parallelized vs. Distributed Search

- Faster optimization by parallel searches
- Randomized updates lead to overall better performance
Approximation is key

- Goal is good predictions on future data
- You can solve an easier problem instead

→ not just for the learning algorithms!
Approximating Features with Hashing

- Millions of features, but often very sparse
- Use random projections to “compress” feature space

Attenberg et al. “Collaborative Email-Spam Filtering with the Hashing-Trick” CEAS 2009
Counting: Count Min-Sketch

- update: hash value, increase counters
- query: hash value, take minimum

Clustering with Count Min-Sketches

Scalable ML is...

- NOT: learning algorithms + parallelization
- INSTEAD:
  - faster learning algorithms
  - approximative optimization
  - feature hashing
  - approximative counting
  - asynchronous distributed search
  - AND parallelization
So where are we with Spark and friends

- Original MR paper: Focus on simple stuff
- Distributed Collections & Dataflow the right abstraction?
- Distributed Matrices?
- Who tells us what the right approximation is?
- New mode of asynchronous, distributed, approximative computation?
- OR go with something which is slower but easier to parallelize?
Summary

- Large scale data sets: billions of examples, millions of features
- A lot of the ML pipeline (feature extraction, data preparation, evaluation) is parallelizable and should be!
- Large scale learning algorithms are harder to parallelize
- Approximation, asynchronous distributed computing, etc.
Thank you!