Factorbird: a Parameter Server Approach to Distributed Matrix Factorization

Sebastian Schelter, Venu Satuluri, Reza Zadeh
Distributed Machine Learning and Matrix Computations workshop in conjunction with NIPS 2014
Latent Factor Models

- Given $M$
  - sparse
  - $n \times m$
- Returns $U$ and $V$
  - rank $k$
- Applications
  - Dimensionality reduction
  - Recommendation
  - Inference
Seem familiar?

\[ \min_{U,V} \sum_{(i,j) \in M} (m_{ij} - u_i^T v_j)^2 + \lambda \left( \|u_i\|^2 + \|v_j\|^2 \right) \]

• So why not just use SVD?
Problems with SVD

(Feb 24, 2015 edition)

More detail....

- Initialize $W,H$ randomly
  - not at zero 😊
- Choose a random ordering (random sort) of the points in a stratum in each "sub-epoch"
- Pick strata sequence by permuting rows and columns of $M$, and using $M'[k,l]$ as column index of row $i$ in subepoch $k$
- Use "bold driver" to set step size:
  - increase step size when loss decreases (in an epoch)
  - decrease step size when loss increases
- Implemented in Hadoop and R/Snowfall

SVD: Drawbacks

- Optimal low-rank approximation in terms of Frobenius norm
- Interpretability problem:
  - A singular vector specifies a linear combination of all input columns or rows
- Lack of sparsity:
  - Singular vectors are dense!
Revamped loss function

- $g$ – global bias term
- $b^U_i$ – user-specific bias term for user $i$
- $b^V_j$ – item-specific bias term for item $j$
- prediction function
  \[ p(i, j) = g + b^U_i + b^V_j + u^T_i v_j \]
- $a(i, j)$ – analogous to SVD’s $m_{ij}$ (ground truth)

New loss function:
\[
\min_{g,b^U,b^V,U,V} \frac{1}{2} \left( \sum_{i,j \in M} w(i,j) (p(i, j) - a(i, j))^2 \right) + \frac{\lambda}{2} \left( g^2 + \|b^U\|^2 + \|b^V\|^2 + \|U\|^2_F + \|V\|^2_F \right)
\]
Algorithm 1: Matrix Factorization using SGD.

1. randomly initialize $U$ and $V$

2. while not converged do

3. randomly pick edge $(i, j)$

4. // compute weighted prediction error

5. $e_{ij} \leftarrow w(i, j)(a(i, j) - p(i, j))$

6. // update biases

7. $g \leftarrow g - \eta (e_{ij} + \lambda g)$

8. $b^U_i \leftarrow b^U_i - \eta \left( e_{ij} + \frac{\lambda}{n_i} b^U_i \right)$

9. $b^V_j \leftarrow b^V_j - \eta \left( e_{ij} + \frac{\lambda}{n_j} b^V_j \right)$

10. // update factors

11. $u_i \leftarrow u_i - \eta \left( e_{ij} v_j + \frac{\lambda}{n_i} u_i \right)$

12. $v_j \leftarrow v_j - \eta \left( e_{ij} u_i + \frac{\lambda}{n_j} v_j \right)$
Problems

1. Resulting $U$ and $V$, for graphs with millions of vertices, still equate to hundreds of gigabytes of floating point values.

2. SGD is inherently sequential; either locking or multiple passes are required to synchronize.
Problem 1: size of parameters

- Solution: Parameter Server architecture
Problem 2: simultaneous writes

• Solution: ...so what?

**HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent**

Feng Niu  
leon@cs.wisc.edu

Benjamin Recht  
brecht@cs.wisc.edu

Christopher Ré  
chrisre@cs.wisc.edu

Stephen J. Wright  
swright@cs.wisc.edu
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706

---

**Algorithm 1** HOGWILD! update for individual processors

1: loop
2: Sample $e$ uniformly at random from $E$
3: Read current state $x_e$ and evaluate $G_e(x_e)$
4: for $v \in e$ do $x_v \leftarrow x_v - \gamma G_{ev}(x_e)$
5: end loop
Lock-free concurrent updates?

• Assumptions

1. $f$ is Lipschitz continuously differentiable
2. $f$ is strongly convex
3. $\Omega$ (size of hypergraph) is small
4. $\Delta$ (fraction of edges that intersect any variable) is small
5. $\rho$ (sparsity of hypergraph) is small
Factorbird Architecture

mathematical view

Co-partition M and V according to the number of learner machines

partitioning scheme

systems view

Co-locate partitions of M and V on learner machines
Parameter server architecture

• Open source!
  – http://parameterserver.org/
Factorbird Machinery

- memcached – Distributed memory object caching system
- finagle – Twitter’s RPC system
- HDFS – persistent filestore for data
- Scalding – Scala front-end for Hadoop MapReduce jobs
- Mesos – resource manager for learner machines
trait Learner {
  def initialize(factors: FactorVector): Unit
  def update(u_i: FactorVector, v_j: FactorVector,
              a_ij: Float, n_i: Int, n_j: Int, w_ij: Float): Float
}

trait Predictor {
  def predict(u_i: FactorVector, v_j: FactorVector): Float
}

trait LossEstimator {
  def estimateRegularizationComponent(
    numRowsOfU: Int, sampleOfU: Iterator[FactorVector],
    numColumnsOfV: Int, sampleOfV: Iterator[FactorVector]): Double
  
  def estimateErrorComponent(numEdges: Long,
    sampleOfEdges: Iterator[Edge], partitionOfU: FactorMatrix,
    partitionOfV: FactorMatrix): Double
}
Model assessment

- Matrix factorization using RMSE
  - Root-mean squared error
    \[ \text{RMSD}(\hat{\theta}) = \sqrt{\text{MSE}(\hat{\theta})} = \sqrt{\text{E}((\hat{\theta} - \theta)^2)}. \]

- SGD performance often a function of hyperparameters
  - \( \lambda \): regularization
  - \( \eta \): learning rate
  - \( k \): number of latent factors
**[Hyper]Parameter grid search**

- aka “parameter scans:” finding the optimal combination of hyperparameters
  - Parallelize!

\[
m \times (c \times k)
\]

\[
(\eta_1, \lambda_1) \quad (\eta_1, \lambda_2) \quad (\eta_2, \lambda_1) \quad (\eta_2, \lambda_2)
\]

\[
(c \times k) \times n
\]

Figure 4: Packing many models into one for hyperparameter search.
Experiments

• “RealGraph”
  – Not a dataset; a framework for creating graph of user-user interactions on Twitter

**Figure 1: Twitter RealGraph Framework.**

Experiments

• Data: binarized adjacency matrix of subset of Twitter follower graph
  \(- a(i, j) = 1 \text{ if user } i \text{ interacted with user } j, 0 \text{ otherwise} \)

• All prediction errors weighted equally \((w(i, j) = 1)\)

• 100 million interactions
• 440,000 [popular] users
Experiments

- 80% training, 10% validation, 10% testing

Figure 5: Prediction quality on held-out data with increasing model complexity.
Experiments

- \( k = 2 \)
- Homophily

Figure 6: Plot of a selection of twitter users as positioned by a factorization with \( k = 2 \) of a sample of RealGraph.
Experiments

• Scalability of Factorbird
  – large RealGraph subset
  – 229M x 195M (44.6 quadrillion)
  – 38.5 billion non-zero entries

• Single SGD pass through training set: ~2.5 hours

• ~ 40 billion parameters
Important to note

• As with most (if not all) distributed platforms:

@SpectralFilter cool! I'd emphasize that this architecture only makes sense if the model is larger than memory. Otherwise it's overkill.
Future work

• Support streaming (user follows)
• Simultaneous factorization
• Fault tolerance
• Reduce network traffic
• s/memcached/custom application/g
• Load balancing
Strengths

- Excellent extension of prior work
  - Hogwild, RealGraph
- Current and [mostly] open technology
  - Hadoop, Scalding, Mesos, memcached
- Clear problem, clear solution, clear validation
Weaknesses

• Lack of detail, lack of detail, lack of detail
  – How does number of machines affect runtime?
  – What were performance metrics of the large RealGraph subset?
  – What were some of the properties of the dataset (when was it collected, how were edges determined, what does “popular” mean, etc)?
  – How did other factorization methods perform by comparison?
Questions?

QUESTIONS?