Apache SystemML - Declarative Large-Scale Machine Learning

Romeo Kienzler (IBM Waston IoT)
Berthold Reinwald (IBM Almaden Research Center)
Frederick R. Reiss (IBM Almaden Research Center)
Matthias Rieke (IBM Analytics)

Swiss Data Science Conference 16 - ZHAW - Winterthur
“High-level programming”

–Assembler vs. Python?
Why another lib?

- Custom machine learning algorithms
- Declarative ML
- Transparent distribution on data-parallel framework
  - Scale-up
  - Scale-out
- Cost-based optimiser generates low level execution plans
Why on Spark?

- Unification of SQL, Graph, Stream, ML
- Common RDD structure
- General DAG execution engine
  - lazy evaluation
  - distributed in-memory caching
2007-2008: Multiple projects at IBM Research – Almaden involving machine learning on Hadoop.

2009: We form a dedicated team for scalable ML.

2009-2010: Through engagements with customers, we observe how data scientists create ML solutions.
Research
June 2015: IBM announces open-source SystemML

September 2015: Code available on Github

November 2015: SystemML enters Apache incubation

February 2016: First release (0.9) of Apache SystemML

June 2016: Second Apache release (0.10)
Moved from Hadoop MapReduce to Spark
SystemML supports both frameworks
**Exact same code**
**300X faster** on 1/40th as many nodes
Alternating Least Squares

Customer $i$ bought product $j$. 

Diagram showing customer-product interactions.
Alternating Least Squares

Customer $i$ bought product $j$. 

$r_{ui}$
Alternating Least Squares

Customer $i$ bought product $j$. 

$r_{ui}$
Customer $i$ bought product $j$. 

$P_u$ 

Customers Factor
Customer $i$ bought product $j$. 
Customer $i$ bought product $j$.

$$\min_{q,p} \sum_{u,i} (r_{ui} - p_u^T q_i)^2$$
Multiply these two factors to produce a less-sparse matrix.

Customer $i$ bought product $j$.

$$r'_{ui} = p_u^T q_i$$

$$\min_{q,p} \sum_{u,i} (r_{ui} - p_u^T q_i)^2$$
Multiply these two factors to produce a less-sparse matrix.

Customer $i$ bought product $j$.

New nonzero values become product suggestions.

$$r'_{ui} = p_u^T q_i$$
val model = ALS.train(ratings, rank, numIterations, 0.01)
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
    i = i + 1; ii = 1;
    if (is_U) { 
        G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
    } else { 
        G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
    }
    norm_G2 = sum(G ^ 2); norm_R2 = norm_G2;
    R = -G; S = R;
    while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) { 
        if (is_U) { 
            HS = (W * (S %*% V)) %*% t(V) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            U = U + alpha * S;
        } else { 
            HS = t(U) %*% (W * (U %*% S)) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            V = V + alpha * S;
        }
        R = R - alpha * HS;
        old_norm_R2 = norm_R2; norm_R2 = sum(R ^ 2);
        S = R + (norm_R2 / old_norm_R2) * S;
        ii = ii + 1;
    }
    is_U = ! is_U;
}
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
    i = i + 1; ii = 1;
    if (is_U)
        G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
    else
        G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
    norm_G2 = sum(G^2); norm_R2 = norm_G2;
    R = -G; S = R;
    while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
        if (is_U) {
            HS = (W * (S %*% V)) %*% t(V) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            U = U + alpha * S;
        } else {
            HS = t(U) %*% (W * (U %*% S)) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            V = V + alpha * S;
        }
        R = R - alpha * HS;
        old_norm_R2 = norm_R2; norm_R2 = sum(R^2);
        S = R + (norm_R2 / old_norm_R2) * S;
        ii = ii + 1;
    }
    is_U = ! is_U;
}
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);

while(i < mi) {
  i = i + 1; ii = 1;
  if (is_U)
    G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
  else
    G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
  norm_G2 = sum(G^2); norm_R2 = norm_G2;
  R = -G; S = R;
  while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
    if (is_U) {
      HS = (W * (S %*% V)) %*% t(V) + lambda * S;
      alpha = norm_R2 / sum (S * HS);
      U = U + alpha * S;
    } else {
      HS = t(U) %*% (W * (U %*% S)) + lambda * S;
      alpha = norm_R2 / sum (S * HS);
      V = V + alpha * S;
    }
    R = R - alpha * HS;
    old_norm_R2 = norm_R2; norm_R2 = sum(R^2);
    S = R + (norm_R2 / old_norm_R2) * S;
    ii = ii + 1;
  }
  is_U = ! is_U;
}
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
    i = i + 1; ii = 1;
    if (is_U)
        G = (W * (U ** V - X)) ** t(V) + lambda * U;
    else
        G = t(U) ** (W * (U ** V - X)) + lambda * V;
    norm_G2 = sum(G ** 2); norm_R2 = norm_G2;
    R = -G; S = R;
    while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
        if (is_U) {
            HS = (W * (S ** V)) ** t(V) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            U = U + alpha * S;
        } else {
            HS = t(U) ** (W * (U ** S)) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            V = V + alpha * S;
        }
        R = R - alpha * HS;
        old_norm_R2 = norm_R2; norm_R2 = sum(R ** 2);
        S = R + (norm_R2 / old_norm_R2) * S;
        ii = ii + 1;
    }
    is_U = ! is_U;
}
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
    i = i + 1; ii = 1;
    if (is_U)
        G = (W * (U **%** V - X)) **% t(V) + lambda * U;
    else
        G = t(U) **% (W * (U **% V - X)) + lambda * V;
    norm_G2 = sum(G ** 2); norm_R2 = norm_G2;
    R = -G; S = R;
    while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
        if (is_U) {
            HS = (W * (S **% V)) **% t(V) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            U = U + alpha * S;
        } else {
            HS = t(U) **% (W * (U **% S)) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            V = V + alpha * S;
        }
        R = R - alpha * HS;
        old_norm_R2 = norm_R2; norm_R2 = sum(R ** 2);
        S = R + (norm_R2 / old_norm_R2) * S;
        ii = ii + 1;
    }
    is_U = ! is_U;
}
Every line has a clear purpose!

25 lines’ worth of algorithm…

…mixed with 800 lines of performance code
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
    i = i + 1; ii = 1;
    if (is_U)
        G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
    else
        G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
    norm_G2 = sum(G^2); norm_R2 = norm_G2;
    R = -G; S = R;
    while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
        if (is_U) {
            HS = (W * (S %*% V)) %*% t(V) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            U = U + alpha * S;
        } else {
            HS = t(U) %*% (W * (U %*% S)) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            V = V + alpha * S;
        }
        R = R - alpha * HS;
        old_norm_R2 = norm_R2; norm_R2 = sum(R ^ 2);
        S = R + (norm_R2 / old_norm_R2) * S;
        ii = ii + 1;
    }
    is_U = ! is_U;
}
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
    i = i + 1; ii = 1;
    if (is_U)
        G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
    else
        G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
    norm_G2 = sum(G ^ 2); norm_R2 = norm_G2;
    R = -G; S = R;
    while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
        if (is_U) {
            HS = (W * (S %*% V)) %*% t(V) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            U = U + alpha * S;
        } else {
            HS = t(U) %*% (W * (U %*% S)) + lambda * S;
            alpha = norm_R2 / sum (S * HS);
            V = V + alpha * S;
        }
        R = R - alpha * HS;
        old_norm_R2 = norm_R2; norm_R2 = sum(R ^ 2);
        S = R + (norm_R2 / old_norm_R2) * S;
        ii = ii + 1;
    }
    is_U = ! is_U;
}
Architecture

High-Level Algorithm ➔ SystemML Optimizer ➔ Parallel Spark Program
Architecture

High-level language front-ends

Cost Based Optimizer

Multiple execution environments

High-Level Operations (HOPs)
General representation of statements in the data analysis language

Low-Level Operations (LOPs)
General representation of operations in the runtime framework
$t(U) \%\% (W \times (U \%\% S))$

Large dense intermediate

Can compute directly from $U$, $S$, and $W$!

(weighted divide matrix multiplication)
All operands fit into heap → use one node

**WDivMM**

(MapWDivMM)
Apache SystemML is a distributed and declarative machine learning platform.
Apache SystemML

Apache SystemML is a distributed and declarative machine learning platform.

Get SystemML
Apache SystemML

Apache SystemML is a distributed and declarative machine learning platform.

Get SystemML
Apache SystemML is a distributed and declarative machine learning platform.

Contribute to the project!

Browse the source!

Try out some tutorials!

Get SystemML
Demo