CS 598: Communication Cost Analysis of Algorithms

Lecture 23: Matrix and tensor completion (ALS, SGD, CCD)

Edgar Solomonik

University of Illinois at Urbana-Champaign

November 9, 2016

The matrix completion problem

Given a matrix $A \in \mathbb{R}^{m \times n}$ and a set of observations $\Omega \subseteq [1, m] \times [1, n]$ find

$$\underset{W \in \mathbb{R}^{m \times k}, H \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \sum_{(i,j) \in \Omega} \left(A(i,j) - \sum_k W(i,k) H(j,k) \right)^2 + \lambda (||W||_F + ||H||_F)$$

- WH^T is a low-rank model for A
- regularization prevents overfit to observed data
- this type of problem is typical in machine learning
  - regularization can be of a different type
  - data can be simply sparse rather than unobserved
  - similar numerical methods used in many cases
- Netflix prize: given m users and n films, build a recommender system
- today we will mostly follow
  - Yu, Hsieh, Si, Dhillon, "Parallel matrix factorization for recommender systems", 2013

Alternating least squares (ALS)

Repeat: fix W and solve for H then fix H and solve for W

- each step improves approximation, convergence to a minimum expected given satisfactory starting guess
- we have a quadratic optimization problem

$$\underset{W \in \mathbb{R}^{m \times k}}{\operatorname{argmin}} \sum_{(i,j) \in \Omega} \left(A(i,j) - \sum_{l} W(i,l) H(j,l) \right)^2 + \lambda ||W||_F$$

- the optimization problem is independent for rows of W
- $\forall i$ letting $a_i = A(i,:), w_i = W(i,:), h_i = H(i,:), \Omega_i = \{j : (i,j) \in \Omega\}$

$$\underset{w_i \in \mathbb{R}^k}{\operatorname{argmin}} \sum_{j \in \Omega_i} \left(a_i(j) - w_i h_j^{\mathsf{T}} \right)^2 + \lambda ||w_i||_2$$

ALS: quadratic optimization

Seek minimizer for quadratic vector equation

$$f(w_i) = \sum_{j \in \Omega_i} \left(a_i(j) - w_i h_j^{\mathsf{T}} \right)^2 + \lambda ||w_i||_F$$

- differentiating with respect to w_i gives

$$\frac{\partial f(w_i)}{\partial w_i} = 2 \sum_{j \in \Omega_i} h_j^{\mathsf{T}} \left(w_i h_j^{\mathsf{T}} - a_i(j) \right) + 2\lambda w_i = 0$$

- rotating $w_i h_j^{\mathsf{T}} = h_j w_i^{\mathsf{T}}$ and defining $\mathcal{H}_i = \sum_{j \in \Omega_i} h_j^{\mathsf{T}} h_j$ we get

$$(\mathcal{H}_i + \lambda I) w_i^T = \sum_{j \in \Omega_i} h_j^{\mathsf{T}} a_i(j)$$

- which is just a $k \times k$ dense symmetric linear system of equations

ALS: iteration cost

For updating each w_i, ALS is dominated in cost by two steps

1. $\mathcal{H}_i = \sum_{j \in \Omega_i} h_j^T h_j$
2. $\mathrm{QR}(\mathcal{H}_i + \lambda I)$ or another dense solver

These steps have the following costs

- the computation complexity is $O(|\Omega_i| k^2)$ for (1) and $O(k^3)$ for (2)
- to update the full matrix W the total cost is $O(|\Omega| k^2 + m k^3)$
- an interesting challenge is the parallelization of the total computation

$$\forall i \in \mathbb{R}^m,\ a, b \in \mathbb{R}^k, \mathcal{H}_i(a, b) = \sum_{j \in \Omega_i} h_j(a) h_j(b)$$

- when the full matrix is observed, so $\Omega = [1, m] \times [1, n]$, we have

$$\forall a, b \in \mathbb{R}^k, \mathcal{H}(a, b) = \sum_{j=1}^{n} h_j(a) h_j(b)$$

or simply $\mathcal{H} = H^T H$

Parallel ALS for dense matrices

Lets first consider parallelizing ALS when $\Omega = [1, m] \times [1, n]$

- this case is easier and is relevant for tensor factorizations
- we need to compute $\mathcal{H} = H^T H$ and then QR of $\mathcal{H}$
- the first matrix multiplication has complexity

$$O(nk^2/P \cdot \gamma + (nk^2/P)^{2/3} \cdot \beta + \alpha)$$

- to solve the linear systems in parallel, each processor can do the QR factorization redundantly

$$O(k^3 \cdot \gamma + k^2 \cdot \beta + \alpha)$$

- the overall complexity is then

$$O(k^2(k + n/P) \cdot \gamma + (k^2 + (nk^2/P)^{2/3}) \cdot \beta + \alpha)$$

- it is also possible to do the QR in parallel, which makes sense for sufficiently large k

ALS for dense tensors

Given an order d tensor T with $N = m^d$ elements

- we want to express T based on d matrices with dimensions $m \times k$
- generally, we will contract $d - 1$ matrices and optimize with respect to one matrix W
- the contracted tensor H is of dimension $N/m \times k$
- forming H by the last matrix multiplication so long as $m > 2k$

$$O\left(\frac{Nk}{P} \cdot \gamma + \left(\frac{Nk}{P}\right)^{2/3} \cdot \beta + \alpha\right)$$

- optimizing W via ALS costs the same as for dense matrices with $n = N/m$

$$O\left(k^2(k + N/(mP)) \cdot \gamma + (k^2 + (Nk^2/(Pm))^{2/3}) \cdot \beta + \alpha\right)$$

Parallel ALS for matrix completion

The simplest parallelization approach is to replicate H on all processors

- each processor updates m/P rows of W, by computing appropriate $\mathcal{H}_i$ to update each w_i
- each processor must also compute m/P QR factorizations of size $k \times k$
- the communication cost is $O(nk \cdot \beta)$ for updating W
- the computation cost assuming load balance is

$$O((mk^2/P + |\Omega|k^2/P) \cdot \gamma)$$

Memory-limited parallel ALS

What if we do not have enough memory to store all of H on each processor?

- we are faced with a challenging communication pattern to parallelize
- we could rotate rows of H along a ring of processors
- each processor computes contributions to the $\mathcal{H}_i$ it owns
- may need multiple ring passes if not enough memory to store m/P $\mathcal{H}_i$ matrices
- communication complexity is at least

$$O(nk \cdot \beta + P \cdot \alpha)$$

Updating a single variable

Rather than solving optimization problems for rows w_i, we can try to solve for elements of w_i, recall that we have

$$\underset{W \in \mathbb{R}^{m \times k}}{\operatorname{argmin}} \sum_{(i,j) \in \Omega} \left(A(i,j) - \sum_{l} W(i,l) H(j,l) \right)^2 + \lambda ||W||_F$$

- lets find the best z to replace $W(i,t)$
- $\operatorname{argmin}_z \sum_{j \in \Omega_i} \left(A(i,j) - zH(j,t) - \sum_{l \neq t} W(i,l) H(j,l) \right)^2 + \lambda z^2$
- the solution is

$$z = \frac{\sum_{j \in \Omega_i} H(j,t) \left(A(i,j) - \sum_{l \neq t} W(i,l) H(j,l) \right)}{\lambda + \sum_{j \in \Omega_i} H(j,t)^2}$$

Coordinate descent

If $\forall (i,j) \in \Omega$ we define $R(i,j) = A(i,j) - \sum_{l=1}^{k} W(i,l)H(j,l)$ then

$$z = \frac{\sum_{j \in \Omega_i} H(j,t)\Big(A(i,j) - \sum_{l \neq t} W(i,l)H(j,l)\Big)}{\lambda + \sum_{j \in \Omega_i} H(j,t)^2}$$

can be computed as

$$z = \frac{\sum_{j \in \Omega_i} H(j,t)\Big(R(i,j) + W(i,t)H(j,t)\Big)}{\lambda + \sum_{j \in \Omega_i} H(j,t)^2}$$

and $R(i,j)$ can be updated as

$$R(i,j) \leftarrow R(i,j) - (z - W(i,t))H(j,t) \quad \forall j \in \Omega_i$$

both using $O(|\Omega_i|)$ operations

Cyclic coordinate descent (CCD)

The single-variable update in coordinate is cheap with respect to ALS

- updating all of w_i costs $O(|\Omega_i|k)$ operations with coordinate descent rather than $O(|\Omega_i|k^2 + k^3)$ operations with ALS
- by solving for all of w_i at once, ALS obtains a more accurate solution than coordinate descent
- with coordinate descent there is also more flexibility in the update ordering
- cyclic coordinate descent (CCD) takes the same update ordering as ALS, but with more fine-grained and less accurate updates
- CCD++ is an alternative that updates a column of W then a column of H, which correspond to an outer product (affects all entries in A), before moving to a subsequent column

Parallel CCD++

Yu, Hsieh, Si, and Dhillon 2013 propose using a row-blocked layout of H and W

- they keep track of a corresponding block row and block column of A and R on each processor (using twice the minimal amount of memory)
- every column update in CCD++ is then fully parallelized, but an allgather of each column is required to update R
- the complexity of updating all of W and all of H is then

$$O(|\Omega|k/P \cdot \gamma + (m+n)k \cdot \beta + k \cdot \alpha)$$

Short pause

Gradient-based update

Rather than solving for $w_i = W(i,:)$ exactly, improve it iteratively

- improve by gradient descent with parameter η
- recall that we had

$$f(w_i) = \sum_{j \in \Omega_i} \left(a_i(j) - w_i h_j^T \right)^2 + \lambda ||w_i||_F$$

and

$$\frac{\partial f(w_i)}{\partial w_i} = 2 \sum_{j \in \Omega_i} h_j^T \left(w_i h_j^T - a_i(j) \right) + 2\lambda w_i$$

- we can use $R(i,j) = a_i(j) - h_j^T w_i$ to write this as

$$\frac{\partial f(w_i)}{\partial w_i} = -2 \sum_{j \in \Omega_i} R(i,j) h_j + 2\lambda w_i$$

- a full gradient descent method would update $w_i = w_i - \eta \frac{\partial f(w_i)}{\partial w_i}$

Stochastic gradient descent (SGD)

Stochastic gradient descent performs fine-grained updates based on samples of the gradient

- again the full gradient is

$$\frac{\partial f(w_i)}{\partial w_i} = -2 \sum_{j \in \Omega_i} R(i,j) h_j + 2\lambda w_i$$

- for a given (i,j) SGD computes updates of the form

$$w_i \leftarrow w_i - \eta(\lambda w_i / |\Omega_i| - R(i,j) h_j)$$

- SGD randomly selects pairs $(i,j) \in \Omega$ and updates w_i (and h_j in a dual fashion)
- it then updates $R(i,j) = A(i,j) - w_i^{\mathsf{T}} h_j$
- each update costs $O(k)$ operations
- $O(|\Omega|)$ yield the same total cost as CCD-based updates of W and H

Asynchronous SGD

Like other iterative methods, its attractive to execute SGD asynchronously

- especially when the sequence is fully-randomized and executed on a shared-memory threaded architecture
- this approach is examined in [Niu, Recht, Re, Wright 2011]
- the asynchronicity can slow down convergence

Blocked SGD

[Gemulla, Haas, Nijkamp, Sismanis 2011] propose a distributed blocking for SGD

- each processor updates a set of independent blocks
- loses true randomization of updates (which is usually used to prove convergence)
- can define $P \times P$ grid of blocks of dimension $m/P \times n/P$
- diagonal blocks are independent as well as appropriate combinations of subdiagonals and superdiagonals of blocks
- assuming $\Theta(|\Omega|/P^2)$ updates are performed on each block (changing every entry), the BSP complexity for $|\Omega|$ updates is

$$O(|\Omega|k/P \cdot \gamma + \min(m,n)k \cdot \beta + P \cdot \alpha)$$