ALS for CP Decomposition

Yuxuan Long (307335)
December 18, 2020

1 Basic ALS

We use ALS to approximately solve the CP decomposition

$$\min_{\bar{A},\bar{B},\bar{C}\in\mathbb{R}^{n\times r}} \quad \left\|\mathcal{B}-[\![\bar{A},\bar{B},\bar{C}]\!]\right\|^2, \tag{1}$$

where $\mathcal{B}\in\mathbb{R}^{n\times n\times n}$, and r is the tensor rank.

In the implementation, we choose the tensor rank $r_1=4$ for $\mathcal{B}_1$, and $r_2=15$ for $\mathcal{B}_2$, so that the approximation error converges easily below $10^{-4}\left\|\mathcal{B}_i\right\|$ within a few number of iterations. The approximation error is simply measured as $\left\|\mathcal{B}-[\![\bar{A},\bar{B},\bar{C}]\!]\right\|$. When we set the tensor rank $r_1=3$ for $\mathcal{B}_1$, the algorithm converges in 8 iterations; when we set $r_1=4$, the algorithm converges in only one iteration. Therefore, we could formulate a conjecture (**i.e. Question 2**):

Conjecture 1.1. *The tensor rank of $\mathcal{B}_1$ is equal or smaller than 4.*

Proof. We could realise that:

$$\begin{aligned}\mathcal{B}_1(i_1,i_2,i_3) &= \sin\left(\zeta(i_1)+\zeta(i_2)+\zeta(i_3)\right)\\ &= \sin\left(\zeta(i_1)+\zeta(i_2)+\right)\cos\left(\zeta(i_3)\right)+\cos\left(\zeta(i_1)+\zeta(i_2)\right)\sin\left(\zeta(i_3)\right)\\ &= \sin\left(\zeta(i_1)\right)\cos\left(\zeta(i_2)\right)\cos\left(\zeta(i_3)\right)+\cos\left(\zeta(i_1)\right)\sin\left(\zeta(i_2)\right)\cos\left(\zeta(i_3)\right)\\ &\quad+\cos\left(\zeta(i_1)\right)\cos\left(\zeta(i_2)\right)\sin\left(\zeta(i_3)\right)-\sin\left(\zeta(i_1)\right)\sin\left(\zeta(i_2)\right)\sin\left(\zeta(i_3)\right),\end{aligned}$$

which means $\mathcal{B}_1$ can be written as the sum of four rank-1 tensors. □

2 Solving Linear System for Tensor

2.1 Direct Solver

We need to solve a linear system:

$$A\ \circ_1\ \mathcal{X}+A\ \circ_2\ \mathcal{X}+A\ \circ_3\ \mathcal{X}=\mathcal{B}, \tag{2}$$

where $A\in\mathbb{S}^n_{++}$ and $\mathcal{X},\mathcal{B}\in\mathbb{R}^{n\times n\times n}$. A direct way of solving it is by vectorization:

$$(I\otimes I\otimes A+I\otimes A\otimes I+A\otimes I\otimes I)\text{vec}(\mathcal{X})=\text{vec}(\mathcal{B}). \tag{3}$$

Solving this linear system can cost enormously. Specifically, the time complexity would be up to $\mathcal{O}(n^9)$, considering a solver for dense matrices.

2.2 LOW-RANK APPROXIMATION SOLVER

LOW-RANK FORMULATION We could notice that the solution of (2) could be equivalently solved by minimizing a convex quadratic:

$$\min_{\mathcal{X}\in\mathbb{R}^{n\times n\times n}} \quad Q := \frac{1}{2}\langle \mathcal{X},\ A\ \circ_1\ \mathcal{X} + A\ \circ_2\ \mathcal{X} + A\ \circ_3\ \mathcal{X}\rangle - \langle \mathcal{X},\ \mathcal{B}\rangle. \tag{4}$$

For the sake of computational efficiency, we assume $\mathcal{B}$ is in the form of CP decomposition, i.e. $\mathcal{B} = [\![\bar{A}, \bar{B}, \bar{C}]\!]$. We further approximate the solution in the low-rank subspace, such that $\mathcal{X}$ is in the CP decomposition. We hence let $\mathcal{X} = [\![U, V, W]\!]$, with $U, V, W \in \mathbb{R}^{n\times p}$. In this way, we could have $\text{vec}(\mathcal{X}) = \sum_{i=1}^{p} w_i \otimes v_i \otimes u_i$, which gives $\langle \mathcal{X},\ \mathcal{B}\rangle = \sum_{i=1}^{p}\sum_{j=1}^{r}(w_i^T \bar{c}_j)(v_i^T \bar{b}_j)(u_i^T \bar{a}_j)$. With the property of Kronecker product, one could derive that

$$\begin{aligned}
&\langle \mathcal{X},\ A\ \circ_1\ \mathcal{X} + A\ \circ_2\ \mathcal{X} + A\ \circ_3\ \mathcal{X}\rangle \\
&= \frac{1}{2}\text{vec}(\mathcal{X})^T(I\otimes I\otimes A + I\otimes A\otimes I + A\otimes I\otimes I)\text{vec}(\mathcal{X}) \\
&= \frac{1}{2}\text{vec}(\mathcal{X})^T \sum_{j=1}^{p} w_j\otimes v_j\otimes Au_j + w_j\otimes Av_j\otimes u_j + Aw_j\otimes v_j\otimes u_j \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}(w_i^T w_j)(v_i^T v_j)(u_i^T A u_j) + (w_i^T w_j)(v_i^T A v_j)(u_i^T u_j) + (w_i^T A w_j)(v_i^T v_j)(u_i^T u_j).
\end{aligned}$$

Our goal is then to minimize Q with respect to U, V, W. The gradients of Q could be computed as:

$$\begin{aligned}
\frac{\partial Q}{\partial U} &= AU(W^TW * V^TV) + U(W^TW * V^TAV + V^TV * W^TAW) - \bar{A}(\bar{C}^TW * \bar{B}^TV) \\
\frac{\partial Q}{\partial V} &= AV(W^TW * U^TU) + V(W^TW * U^TAU + U^TU * W^TAW) - \bar{B}(\bar{C}^TW * \bar{A}^TU) \\
\frac{\partial Q}{\partial W} &= AW(V^TV * U^TU) + W(V^TV * U^TAU + U^TU * V^TAV) - \bar{C}(\bar{B}^TV * \bar{A}^TU).
\end{aligned}$$

In each substep of ALS, we set one gradient to zero which could optimally solve one variable while keeping other variables fixed. We could observe that, for instance, solving $\frac{\partial Q}{\partial U} = 0$ needs to solve a particular form of matrix equation: $AUE + UF = G$, with symmetric matrices $E, F \in \mathbb{S}^r$. Note that this matrix equation is equivalent to $(E\otimes A + F\otimes I)\text{vec}(U) = \text{vec}(G)$. In this way, solving $\frac{\partial Q}{\partial U} = 0$ gives:

$$\begin{aligned}
\text{vec}(U) = &\big((W^TW * V^TV)\otimes A + (W^TW * V^TAV + V^TV * W^TAW)\otimes I\big)^{-1} \\
&\cdot \text{vec}(\bar{A}(\bar{C}^TW * \bar{B}^TV)).
\end{aligned} \tag{5}$$

Similarly, solving $\frac{\partial Q}{\partial V} = 0$ and $\frac{\partial Q}{\partial W} = 0$ respectively gives:

$$\begin{aligned}
\text{vec}(V) = &\big((W^TW * U^TU)\otimes A + (W^TW * U^TAU + U^TU * W^TAW)\otimes I\big)^{-1} \\
&\cdot \text{vec}(\bar{B}(\bar{C}^TW * \bar{A}^TU)),
\end{aligned} \tag{6}$$

$$\begin{aligned}\text{vec}(W) = &\left((V^TV * U^TU) \otimes A + (V^TV * U^TAU + U^TU * V^TAV)) \otimes I\right)^{-1} \\ &\cdot \text{vec}(\bar{C}(\bar{B}^TV * \bar{A}^TU)).\end{aligned} \tag{7}$$

Solving one of those only involves inverting a $np \times np$ matrix, hence its cost is up to $\mathcal{O}(p^3n^3)$.

ALGORITHM DESIGN We could initialise a tensor rank for $\mathcal{X}$ and increase the tensor rank once the error gets stuck. The error here is defined as the relative residual error:

$$e_r = \|A \circ_1 \mathcal{X} + A \circ_2 \mathcal{X} + A \circ_3 \mathcal{X} - \mathcal{B}\| / \|\mathcal{B}\|,$$

where $\mathcal{B}$ here is in its original form. If the error e_r is smaller than the tolerance e_{tol}, then we stop the algorithm. In this project, we set $e_{tol} = 10^{-3}$ as $\mathcal{B}$ is already approximated with the relative error 10^{-4}.

In our design, we set the initial tensor rank as $p_0 = \text{round}\left(\frac{n}{\max\{0,\log(n/25)\}+1}\right)$. In this setting, we have $p_0 = n$ if $n \leq 25$; otherwise, the greater the n, the smaller the ratio p_0/n. This allows a high compression of $\mathcal{X}$ when large n is given, hence avoiding issue of over-fitting. For instance, if $n = 30$, the tensor rank of $\mathcal{X}$ may have to be 28 to solve the linear system; if $n = 200$, one may only require the tensor rank to be 59, note that $\frac{59}{200} << \frac{28}{30}$.

Computing the relative residual error can cost $\mathcal{O}(pn^3)$, and the memory complexity is at least $\mathcal{O}(n^3)$. For the sake of efficiency, we check the relative residual error in every two iterations. If the relative residual error is found to be only changed in some small magnitude $s \cdot e_{tol}$, then we increment the tensor rank p by some positive integer proportional to the current error. At this case, we append some random columns to U, V, W, then halve the value of s as to enable a smooth convergence. Note that small s allows more iterations on a certain tensor rank, this may bring more computation but could result in a good tensor compression. However, if s is too small, the algorithm may get stuck as the relative error could always stay above e_{tol}. So, we set a lower bound for s, i.e. $(\frac{1}{2})^5 = 3.125 \cdot 10^{-2}$. The details of this algorithm design could be seen below.

NUMERICAL STABILITY Ideally, U, V, W should have full column rank, due to a good compression of the tensor. So, by Schur product theorem (Hadamard product of two positive definite matrices is also positive definite) and the basic property of Kronecker product, inverting a positive definite matrix is involved in (5), (6) and (7). For instance, when solving (6), W^TW and U^TU should ideally have full rank, hence $W^TW * U^TU \succ 0$ by Schur product theorem, then $(W^TW * U^TU) \otimes A \succ 0$. In practice, over-fitting may be encountered if the tensor rank is sometimes set too large, so that U, V, W have some singular values close to zero. This means that U^TU, V^TV, W^TW have very large condition number. So, we may have to invert a badly conditioned matrix when solving U, V, W. This normally happens when n is large or a bad initialization is given. We hence propose to initialise orthogonal columns for U, V, W. When bad conditioning is detected, we add small positive element to the matrix diagonal. This may temporarily lead to inaccurate results, but does not impede the algorithm convergence.

Orthogonalised ALS (i.e. add QR decomposition after every sub-iteration) is also implemented in order to completely solve the numerical issues, nevertheless the convergence of orthogonalised ALS is a problem since its relative error could even hardly drop below 0.1. In the future, the convergence of orthogonalised ALS could be investigated.

Algorithm 1: ALS for solving the linear system

Initialise: $p \leftarrow p_0 = \text{round}\left(\frac{n}{\max\{0,\log(n/25)\}+1}\right)$; $i \leftarrow 0$; $e_r \leftarrow +\infty$; $s \leftarrow 0.5$;
Initialise $U, V, W \in \mathbb{R}^{n \times p_0}$ with random orthogonal columns;
while $e_r \geq e_{tol}$ **do**
  Normalize columns of W; Solve U by (5);
  Normalize columns of U; Solve V by (6);
  Normalize columns of V; Solve W by (7);
  $i \leftarrow i + 1$;
  if i *is even* **then**
    Compute relative residual error $\bar{e}_r$;
    if $|e_r - \bar{e}_r| < s \cdot e_{tol}$ & $p < n$ & $\bar{e}_r \geq e_{tol}$ **then**
      Increment p by $\min\{n - p, \text{round}(\bar{e}_r / e_{tol})\}$;
      Append random columns to U, V, W to satisfy the tensor rank p;
      $s \leftarrow 0.5s$ if $s \geq 4e{-}2$;
    end
    $e_r \leftarrow \bar{e}_r$;
  end
end

2.3 EXPERIMENTS

We run all the experiments in a octa-core 16GB PC. We would focus on the discussion of timing performance, regarding the tensor rank and size.

For the CP decomposition using ALS, it takes very little time for small n. When $n = 200$, it takes only 4 seconds for decomposing $\mathcal{B}_1$ and $\mathcal{B}_2$; when $n = 400$, it takes about 63 seconds in total. So, we could empirically estimate the total time complexity as $\mathcal{O}(n^4)$. Theoretically, each iteration costs $\mathcal{O}(rn^3)$ where r is the tensor rank set for $\mathcal{B}$.

We run direct solver and low-rank solver (ALS) to compare their timing performance, which is shown in Figure 1. We could see that the running time for direct solver grows dramatically as n increases. Note that the running time for direct solver is almost same for $\mathcal{B}_1$ and $\mathcal{B}_2$, so we test only $\mathcal{B}_1$ here. When $n \geq 85$, the direct solver would suffer from some memory error which breaks the program. Therefore, in my PC, the largest n that the direct solver could handle is 84 (**i.e. Question 3**), and this takes one hour for running. By rough estimation from the experiments, the time complexity of direct solver would be between $\mathcal{O}(n^7)$ and $\mathcal{O}(n^8)$, which is better than the theoretical estimate $\mathcal{O}(n^9)$ due to the use of sparsity structure.

For the low-rank solver using ALS, the timing performance is shown in Figure 1, where we test the algorithm on $\mathcal{B}_1$ and $\mathcal{B}_2$ respectively (timing for CP decomposition is not included). As seen from the green and yellow curves, the linear system that involves $\mathcal{B}_2$ seems to have fewer cost, due to fewer iterations at this case. By rough estimation, the time complexity of this low-rank solver is $\mathcal{O}(n^3)$, which is several order of magnitude smaller than the direct solver. The evidence for the complexity $\mathcal{O}(n^3)$ is shown in Figure 1, where the purple curve represents the cubic $\frac{n^3}{10000}$. This empirical estimation may not be true for very large n, as the linear system may become ill-conditioned, which could accidentally slow down the convergence.

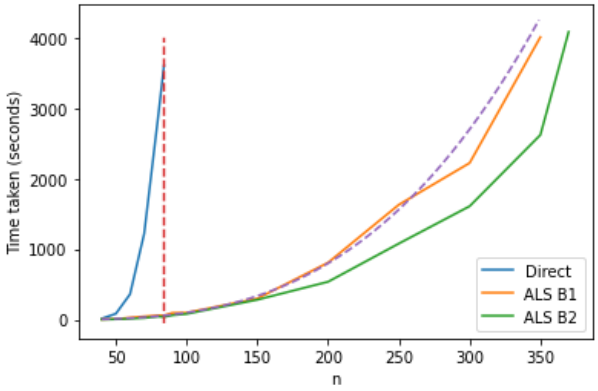

Figure 1: Timing performance of direct solver and low-rank solver (ALS)

In our current design, the low-rank solver would always return the solution within 50 iterations for $n \geq 30$. The number of iterations for large n is even less than that for the small n. When $n < 30$, especially for $\mathcal{B}_1$, the algorithm convergence seems to be hard. For a small $n < 30$, nevertheless, it is sufficient to use the direct solver to efficiently compute an accurate solution.

Since at each iteration, ALS only has to solve a $np \times np$ linear system, the low-rank solver could handle some n much larger than 84. Given the same amount of time— one hour of running time, the size n that low-rank solver could handle is around 350 (i.e. **Q4**), as shown in Figure 1.

Tensor ($n = 200$)	Initial rank	Final rank	Time (seconds)	Iteration number
$\mathcal{B}_1$	60	64	847	**24**
$\mathcal{B}_1$	50 (current)	59	**810**	30
$\mathcal{B}_1$	40	**54**	936	48
$\mathcal{B}_2$	60	63	**409**	**12**
$\mathcal{B}_2$	50 (current)	55	538	22
$\mathcal{B}_2$	40	**53**	890	46

Table 1: Influence of initial tensor rank to the algorithm convergence

For $\mathcal{B}$ with $n = 200$, we did some experiments on different initial ranks, i.e. rank 60, 50 (the default setup in our design) and 40, as seen in Table 1. A smaller initial rank could lead to a good compression of the solution, as the final rank could be smaller. However, decreasing the initial rank brings much more iterations for convergence, hence more time would be possibly consumed. So, there is a trade-off between the solution compression (i.e. the tensor rank) and the number of iterations. If the tensor rank p is set too large, the total time cost would be large as well even if the number of iterations is reduced, e.g. the first row in Table 1.

In summary, we have designed an ALS algorithm for approximately solving the large linear system. In the future, we could further improve the numerical stability and algorithm efficiency, e.g. use existing method for solving Sylvester equation. The design of adaptive tensor rank is never perfect but still needs improvement for both good compression and low time cost.