УДК 681.3.

ПАРАЛЕЛІЗАЦІЯ НЕВІД'ЄМНОЇ ФАКТОРИЗАЦІЇ РОЗРІДЖЕНИХ МАТРИЦЬ НАДВЕЛИКОЇ РОЗМІРНОСТІ

Е.М. Насиров

Київський національний університет імені Тараса Шевченка,
03680, Київ, проспект Академіка Глушкова 4д.
E-mail: enasirov@gmail.com

У роботі описано побудову моделі паралелізації обчислення невід'ємної факторизації розріджених матриць надвеликої розмірності. Реалізації запропонованих моделей були порівняні в обробці надвеликої матриці.

This paper proposes parallel methods of non-negative huge sparse matrix factorization. The implementation of proposed methods was tested and compared on huge matrix.

Сьогодні невід'ємна факторизація матриць та тензорів є дуже популярною технологією в штучному інтелекті взагалі, та зокрема в комп'ютерній лінгвістиці. Використовуючи невід'ємну факторизацію в рамках парадигми латентно-семантичного аналізу, комп'ютерні лінгвісти застосовують даний підхід для розв'язання таких прикладних задач, як класифікація, кластеризація текстів та термінів [1, 2], побудова мір семантичної близькості [3], автоматичне виділення лінгвістичних структур та відношень (Selectional Preferences [4] and Verb Sub-Categorization Frames [5]) та багато інших.

Дана робота описує побудову моделі паралелізації обчислення невід'ємної факторизації розріджених матриць надвеликої розмірності, що представляє особливий інтерес та актуальність для її застосування у великих NLP системах загального тематичного напрямку, не обмежених використанням лише для вузьких предметних областей.

Задача невід'ємної факторизації розріджених матриць надвеликої розмірності постала в процесі розробки системи визначення міри семантичної близькості-зв'язності за технологією латентного семантичного аналізу [6]. Для побудови системи був оброблений великий текстовий корпус статей англомовної Вікіпедії. Обробка корпусу полягала у лексичному аналізі та лематизації лексем речень статей та в обчисленні частот вживання множини слів та словосполучень англійської мови в складі різних статей англомовної Вікіпедії. В результаті була побудована велика матриця Слова×Статті яка містила частотну оцінку вживання слів у текстах статей Вікіпедії. Розмірність матриці дорівнює 2,437,234 слів-словосполучень на 4,475,180 статей англомовної Вікіпедії. Після цього для усунення з матриці випадкових даних був встановлений пороговий рівень T=3, частотні оцінки в матриці, які менші за пороговий рівень T, були обнулені). В результаті була побудована розріджена матриця надвеликого розміру – 156,236,043 ненульових елементів при розмірі 2,437,234×4,475,180. Для невідємної факторизації розрідженої матриці такого розміру знадобилася розробка спеціальної моделі паралелізації матричних обчислень, яка була реалізована із застосуванням паралельних розподілених обчислень та обчислень на GPU. Низка реалізацій факторизації матриць була вже реалізова і представлена у [7 – 10]. Але жодна з розроблених моделей не є допустимим для поставленої задачі. Деякі з них [7–9] не задовольняють потреби розмірності матриці. Модель запропонована в [10] проводить факторизацію матриць запропонованого розміру за задовільний час, але потребує занадто великих розрахункових комп'ютерних потужностей, що є не завжди доступним.

Алгоритм невід'ємної матричної факторизації

Алгоритм невід'ємної матричної факторизації [11] виконує декомпозицію невід'ємної матриці V на невід'ємні матриці W та H таким чином, щоб:

$$V \approx W\,H.$$

Як функція оцінки μ може бути використана функція виміру відстані між двома невід'ємними матрицями. Однією з таких мір є квадрат Евклідової метрики:

$$\mu = \|A - B\|^2 = \sum_{i\,j} \left(A_{i\,j} - B_{i\,j}\right)^2.$$

Така цільова функція обмежена знизу. Нижня границя 0 досягається тоді і тільки тоді коли

$$A = B.$$

Отже, при використанні Евклідової метрики, факторизація матриці полягає у мінімізації $\|V - W H\|^2$ при умові невід'ємності W та H.

Така цільова функція не зростаюча при наступних правилах:

$$H_{ij} \leftarrow H_{ij} \frac{(W^T V)_{ij}}{(W^T W H)_{ij}}. \quad (1)$$

$$W_{ij} \leftarrow W_{ij} \frac{(V H^T)_{ij}}{(W H H^T)_{ij}}. \quad (2)$$

Виконання ітерацій алгоритму продовжується дот тих пір, поки не буде досягнута стаціонарна точка, або не буде виконана максимальна кількість ітерацій.

Аналіз моделі

В табл. 1 представлено необхідні обсяги пам'яті для зберігання матриць W та H при різних k при використанні типу даних float (32bit).

Таблиця 1. Необхідні обсяги пам'яті для зберігання матриць W та H при різних k

K	*100*	*200*	*300*
W	0.98Gb	1.95Gb	2.92Gb
H	1.79Gb	3.58Gb	5.37Gb
Всього	2.76Gb	5.53Gb	8.29Gb

Описаний алгоритм на кожній ітерації потребує в 2 рази більше пам'яті для збережень матриць (не враховуючи потреби на збереження початкової матриці V). Враховуючи такі потреби в пам'яті, для виконання алгоритму на одному комп'ютері необхідно проводити збереження даних на жорсткий диск. Далі буде розглянуто та порівняно 2 підходи до паралелізації алгоритму – це локальний та розподілений.

Паралелізація алгоритму з використанням GPU

Для спрощення, зробимо заміну в (1) та (2): $H' = H^T$. Отримаємо

$$(H'_t)_{ij} = (H'_{t-1})_{ij} \frac{(V^T W_{t-1})_{ij}}{(H'_{t-1} W_{t-1}^T W_{t-1})_{ij}}, \quad (4)$$

$$(W_t)_{ij} = (W_{t-1})_{ij} \frac{(V H'_t)_{ij}}{(W_{t-1} H_t'^T H'_t)_{ij}}. \quad (5)$$

Таким чином, обидві формули (4) та (5) ми можемо представити в одному вигляді, завдяки заміні H', W та V^T, або W, H' та V на *A*, *B* та *S* відповідно:

$$A_{ij} = A_{ij} \frac{(S B)_{ij}}{(A B^T B)_{ij}}. \quad (6)$$

Формула (6) може бути представлена як послідовність чотирьох кроків (7):

$$C = S B, \quad (7')$$

$$K = B^T B, \quad (7'')$$

$$D = A K, \quad (7''')$$

$$A_{ij} = A_{ij} \frac{C_{ij}}{D_{ij}}. \quad (7'''')$$

Такий порядок обчислень є оптимальним для виразу (6). Ці кроки маюсь обчислювальну складність $O(k*(nnz(S)+n))$, $O(k^2 m)$, $O(k^2 n)$ та $O(k n)$ відповідно, де $nnz(S)$ – кількість ненульових елементів матри-

ці *S*. Перші три кроки(7’–7’’’) підтримуються бібліотеками CUDA cuSPARSE [12] та cuBLAS [13]. Четвертий крок (7’’’’) для виконання потребує реалізації власного ядра GPU, але в той же час, це відносно швидка операція і тому може бути виконана на CPU.

Так як матриці занадто великі для збереження в пам’яті GPU, ці операції (7’–7’’’’) мають виконуватись над частинами, з врахуванням необхідності зменшення обсягів обміну даними з графічним адаптером.

Для виразу (7’) мають розкласти матриці $S = (S_1' | S_2' | \ldots | S_t')^T$ та $B = (B_1 | B_2 | \ldots | B_r)$, та підрахувати *C*:

$$C = \begin{pmatrix} S_1'B_1 & \cdots & S_1'B_r \\ \vdots & \ddots & \vdots \\ S_t'B_1 & \cdots & S_t'B_r \end{pmatrix}. \quad (8)$$

Для виразу (7’’) варто розглядати $B = (B_1' | B_2' | \ldots | B_t')$, а отже $K = (B_1'^T B_1' \ldots B_t'^T B_t')$. Підрахунок цього виразу не потребує додаткового завантаження будь-яких матриць у пам'ять графічного адаптера після виконання (7’).

Для підрахунку (7’’’) ми можемо залишити в пам'яті GPU матрицю *K* і помножити матрицю *A* як стовпчик блоків.

Також можна зменшити використання пам’яті завдяки комбінуванню (7’’’’) та (7’’’), так як (7’’’’) – по елементне правило і єдине інше правило, в якому використовується матриця A – (7’’’). Не буде потреби зберігати в пам’яті матрицю *D*, якщо виконувати (7’’’’) на блоці з матриці *A*, який необхідний для обчислення певного блоку матриці *D*.

Складність підрахунку Евклідової метрики між початковою та факторизовано моделлю складає $O(nm)$ $\mathrm{O}(nm)$. Такий підрахунок просто реалізовується з використанням графічних адаптерів.

Розподілений алгоритм

Наступним кроком для покращення швидкодії є використання мережі ПК для проведення обчислень. Розглянемо можливі розподіленні моделі. Припустимо, що мережа складається з двох вузлів. Можливі наступні моделі розподілення обчислень.

1. Підраховувати *W* та H' окремо на різних вузлах. Таким чином обидва вузли будуть перебувати в одному з двох альтернативних режимів. Або будуть вузлом підтримки іншого вузла (передача даних на інший вузол) або будуть проводити підрахунки (обчислювати матрицю за яку відповідають, за допомогою даних, отриманих з вузла підтримки). В такій моделі розподілення на кожній ітерації ми повинні будемо передати по мережі дві матриці такого ж розміру, як і *W* та *H*. Також вузол розрахунків буде в основному простоювати, тому що (7a) є найбільш витратним кроком з усіх чотирьох.

2. Розділити матриці *W* та H' на блоки та розподілити їх між вузлами. $H' = (H_1' | H_2')$ та $W = (W_1 | W_2)$. Робимо перший вузол відповідальним за H_1', W_1, другий – за H_2', W_2. При такому підході кожен вузол виступає як у ролі вузла підтримки, так і у ролі вузла розрахунків одночасно. В такій моделі вузли мають передавати по мережі обсяг даних рівний $1{,}5 \cdot (sizeof(W) + sizeof(H))$, де $sizeof(X)$ – обсяг пам’яті, необхідний для зберігання матриці *X*

3. Розділити матриці W та H' на блоки та розподілити їх між вузлами. $H' = (H_1' | H_2')^T$ та $W = (W_1' | W_2')^T$. Робимо перший вузол відповідальним за H_1, W_1', другий – за H_2, W_2'. В цій моделі, аналогічно до попередньої, кожен вузол мережі виступає одночасно як у ролі вузла підтримки, так і у ролі вузла розрахунків. Обсяг передачі даних вузлом рівний $sizeof(W) + sizeof(H)$.

В кожній з представлених моделей також є необхідність передачі одної або декількох матриць $[k; k]$, але їх розміром можна знехтувати в порівнянні з матрицями W та *H*. Для підрахунку функції оцінки μ в першій та третій моделі вузлам необхідно передати дані в обсязі

$$\frac{(sizeof(W) + sizeof(H)) * K}{2},$$

де K – кількість вузлів в мережі, у другій моделі необхідно передати в $(K-1)$ раз більше даних.

Варто також зазначити, що ріст мережі призведе до експоненціального росту сумарного обсягу переданих даних, хоча обсяг передачі даних одного вузла буде не більш ніж $2 \cdot (sizeof(W) + sizeof(H))$.

Для кращого розподілення розрахунків між вузлами мережі, враховуючи розрідженість початкової матриці *V*, необхідно виконати перестановки рядків та стовпчиків для нормалізації кількості ненульових елементів у кожному блоці.

Очевидно, що третя модель найкраще підходить з точки зору мережевого обміну даними та використання GPU. Тому саме ця модель була використана в нашому дослідженні, реалізована з використання мережі з чотирьох вузлів і описана далі.

Згідно з моделлю 3 необхідно розділити матриці W, H' і V наступним чином:

$$W = (W_1' | W_2' | W_3' | W_4')^T,$$

$$H' = (H_1 | H_2 | H_3 | H_4)^T, \quad V = \begin{pmatrix} V_{11} & \cdots & V_{14} \\ \vdots & \ddots & \vdots \\ V_{41} & \cdots & V_{44} \end{pmatrix}.$$

Алгоритм складається з трьох основних фаз: ініціалізація, ітерації, підрахунок метрики. Під час фази ініціалізації матриці W, H' і V розподіляються між чотирма вузлами так, щоб *i*-ий вузол отримав матриці W_i', H_i та V_{ki}, V_{ik}, де $k = \overline{1,..,4}$. Дана фаза показана на рис. 1, a.

Фаза ітерації складається з двох аналогічних етапів, один для підрахунку H', а другий для W. Кожен етап включає 3 кроки.

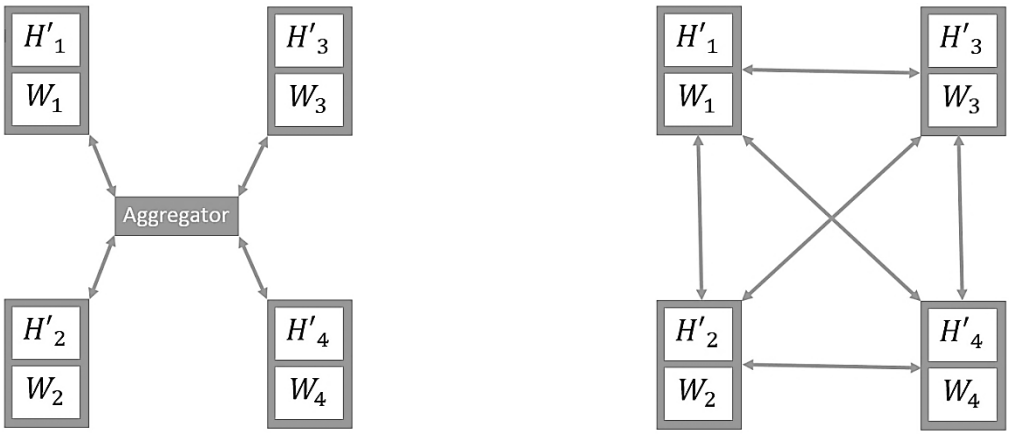
На першому кроці кожний вузол підраховує матрицю $W_i' \times W_i'^T$ розміру $k \times k$ та надсилає її до агрегатора. Агрегатор об'єднує всі отримані блоки в одну матрицю K_w та надсилає отриманий результат всім вузлам. Цей крок показано на рис. 1, a.

На другому кроці кожен вузол підраховує власне $(V_{1i}^T | V_{2i}^T | V_{3i}^T | V_{4i}^T)^T \times W_i'$. Отримана матриця збігається за розмірами з H'. Далі кожний вузол ділить цю матрицю згідно з початковим розбиттям матриці H та передає отримані блоки відповідним вузлам. Цей крок показано на рис. 1, b.

На третьому кроці вузли підраховують матрицю $H_i \times K_w$ та виконує оновлення матриці H'. Цей крок не потребує передачі даних по мережі.

Ці три кроки необхідні для підрахування матриці H'. Після поновлення H', аналогічні кроки необхідно виконати для W. А саме необхідно підрахувати наступні матриці: $H_i \times H_i^T$, $(V_{1i}^T | V_{2i}^T | V_{3i}^T | V_{4i}^T)^T \times H_i$ та $W_i' \times K_H$.

У фазі підрахунку метрики кожен вузол передає відповідний блок H' всім іншим блокам. Після отримання всіх блоків, кожний вузол підраховує відповідну части метрики. Ця фаза також показана на рис. 1, b.



a b

Рис. 1. Розподілена модель з чотирма вузлами

Результати аналізу

Був реалізований розподілений алгоритм з використанням GPU, який був описаний вище, та проведена факторзація розрідженої матриці оцінок части вживання слів англійської мови в різних статтях Вікіпедії. Для порівняння швидкодії була також реалізована локальна модель з використанням GPU та жорсткого диску для запису даних.

Обидві моделі працювали з однаковими вхідними даними. Локальна версія була запущена на одному з вузлів з тими ж обмеженнями пам'яті. Для проведення тестів були використані наступні апаратні засоби: Intel Core i7 CPU, NVIDIA GeForce GTX560 1Gb, 8Gb RAM, 1Gbit LAN, SATA III HD.

В табл. 2 порівняно час та ресурси необхідні для виконання однієї ітерації у локальній та розподіленій версії. В табл. 3 порівняно час та ресурси, необхідні для підрахунку функції оцінки μ. Дані таблиць отриманні при $k = 300$.

Таблиця 2. Результати порівняння виконання ітерації локальної та розподіленої версії

	Local	Distributed
Прочитано	34.44Gb	6.22Gb
Записано	16.58Gb	6.22Gb
Час ітерації (розрахунки)	58s	62s
Час ітерації (I/O)	729s	287s

Таблиця 3. Результати порівняння обчислення функції оцінки μ

	Local	Distributed
Прочитано	13.66Gb	6.22Gb
Записано	0	6.22Gb
Час (розрахунки)	45865s	11371s
Час (I/O)	192s	280s

На рис. 2 показано графік отриманої залежність як результати факторизації від *k* та кількості ітерацій.

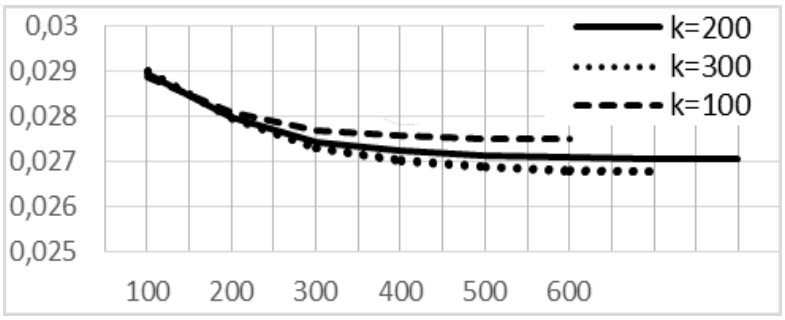


Рис. 2. Залежність збіжності алгоритму факторизації від *k* та кількості ітерацій

Застосування моделі до факторизації тензорів

Для того, щоб обчислити невід'ємні матриці-компоненти $\{A, B, C\}$ зазвичай застосовується обмежений оптимізаційний підхід для мінімізації підходящої функції оцінки. Зазвичай мінімізують наступну функцію:

$$D_F(\underline{Y} \| [A,B,C]) = \|\underline{Y} - [A,B,C]\|_F^2 + \alpha_A \|A\|_F^2 + \alpha_B \|B\|_F^2 + \alpha_C \|C\|_F^2,$$

де α_A, α_B, α_C – невід'ємні регуляційні параметри.

Існує щонайменше три різні підходи до розв'язання такої оптимізаційної задачі. Перший підхід полягає у використанні векторної форми функції оцінки: $J(X) = vec(\underline{Y} - [A,B,C]) = 0$. Для розв'язання використовується метод найменших квадратів. Цей метод для факторизації тензорів вперше використав Паатеро. Якобіан такої функції може мати розмір $ITQJ \times (I+T+Q)$, а отже такий підхід потребує значних розрахункових витрат.

У другому підході Акар, Колда та Дунлаві запропонували штучно оптимізувати функцію оцінки використовуючи техніку нелінійної зв'язної градієнтної оптимізації [1].

Третім, і найпоширенішим підходом є використання техніки ALS. В цьому підході підраховується градієнт функції оцінки для кожної матриці окремо.

$$\nabla_A D_F = -Y_{(1)}(C \odot B) + A[(C^T C) \circledast (B^T B) + \alpha_A I],$$
$$\nabla_B D_F = -Y_{(2)}(C \odot A) + B[(C^T C) \circledast (A^T A) + \alpha_B I],$$
$$\nabla_C D_F = -Y_3(B \odot A) + C[(B^T B) \circledast (A^T A) + \alpha_C I].$$

Прирівнюючи кожну компоненту градієнта до 0 та накладаючи умову невід'ємності можна отримати ефективні та прості правила оновлення матриць:

$$A \leftarrow [Y_{(1)}(C \odot B) + [(C^T C) \circledast (B^T B) + \alpha_A I]^{-1}]_+,$$
$$B \leftarrow [Y_{(2)}(C \odot A) + [(C^T C) \circledast (A^T A) + \alpha_B I]^{-1}]_+,$$
$$C \leftarrow [Y_{(3)}(B \odot A) + [(B^T B) \circledast (A^T A) + \alpha_C I]^{-1}]_+.$$

Основними перевагами такого підходу є висока швидкість збіжності та можливість розподілення для задач великої розмірності.

Запропонована вище розподілена модель може бути застосована для факторизації тензорів. Правила оновлення є однаковими. Тому буде показано підрахунок на прикладі однієї матриці:

$$A \leftarrow \left[Y_{(1)}(C \odot B) + [(C^T C) \circledast (B^T B) + \alpha_A I]^{-1}\right]_+ .$$

Для мережі з двох вузлів необхідно розділити матриці W та H' на блоки таким чином, що $C' = (C_1|C_2)^T$ та $B = (B_1'|B_2')^T$, та розподілити їх між вузлами так, щоб перший вузол був відповідальним за C_1, B_1', другий за C_2, B_2'.

При такому розподіленні на першому кроці кожний вузол підраховує матрицю $C_i^T \times C_i$ розміру $k \times k$ та надсилає її до агрегатора. Агрегатор об'єднує всі отримані блоки в одну матрицю K_C та надсилає відповідні частини до вузлів. Далі аналогічним чином виконуються розрахунки для $(B^r B)$. На наступному кроці вузлам необхідно підрахувати $S_i = [(C^T C)_i \circledast (B^T B)_i + \alpha_A I]$ та передати відповідні матриці на агрегаті на якому підрахується $S^{-1} S^{-1}$. Після чого вузлам необхідно підрахувати відповідні частини результуючої матирці у вигляді $(C \odot B)_i + S_i^{-1}$.

Таким чином можливе використання запропонованого розподіленої моделі до факторизації тензорів.

Висновки

В роботі описано локальний та розподілений підходи до невід'ємної факторизації надвеликих матиць. В результаті роботи було отримано високі показники швидкодії роботи розподіленого алгоритму в порівнянні з локальним. Експерименти показали, що процес підрахунку матриць збігається за 100 ітерацій, а час необхідний для фактризації опрацьованої матриці для розподіленої моделі – 9,6 годин та 21 година для локальної моделі з використанням GPU. Подальша робота полягає в аналізі та зменшенні обсягів передачі даних між вузлами, а отже, і часу що для цього необхідний.

1. *Xu, W., Liu, X., & Gong, Y.* (). Document-clustering based on 71n-negative matrix factorization // In Proceedings of SIGIR'03, July 28–August 1, Toronto – 2003. – P. 267–273.
2. *Farial Shahnaz, Michael W. Berry, V. Paul Pauca, Robert J.* Plemmons Document clustering using nonnegative matrix factorization // Information Processing and Management: Volume 42 Issue 2, March 2006. – P. 373 – 386
3. *Anatoly Anisimov, Oleksandr Marchenko, Andrey Nikonenko, Elena Porkhun, Volodymyr Taranukha:* Ukrainian WordNet: Creation and Filling. FQAS 2013. – P. 649–660
4. *Tim Van de Cruys.* Anon-negative tensor factor-ization model for selectional preference induction. Natural Language Engineering. – 2010. – 16(4). – P. 417–437.
5. *Tim Van de Cruys, Laura Rimell, Thierry Poibeau, and Anna Korhonen*. Multi-way Tensor Factorization for Unsupervised Lexical Acquisition. In International Conference on Computational Lin-guistics (COLING), Mumbai, India, 08/12/2012-15/12/2012, P. 2703–2720. The COLING 2012 Organizing Committee. 2012.
6. *Scott Deerwester, Susan T. Dumais, GeorgeW. Furnas, Thomas K. Landauer, and Richard Harshman.* Indexing by latent semantic analysis // J. OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE. – 1990. – 41(6). P. 391–407.
7. *Brett W. Bader, Tamara G. Kolda, et al*. Matlab tensor toolbox version 2.5. Available online, January. 2012.
8. *Khushboo Kanjani*. Parallel non negative matrix factorization for document clustering. May, 2007.
9. *Volodymyr Kysenko, Karl Rupp, Oleksandr Marchenko, Siegfried Selberherr, and Anatoly Anisimov*. Gpu-accelerated non-negative matrix factorization for text mining. of Lecture Notes in Computer Science, Springer. – 2012. – V. 7337 – P. 158–163.
10. *Chao Liu, Hung-chih Yang, Jinliang Fan, Li-Wei He, and Yi-Min Wang.* Distributed nonnegative matrix factorization for web-scale dyadic data anal-ysis on mapreduce. In Proceedings of the 19th In-ternational Conference on World Wide Web, WWW'10, New York, NY, USA. ACM. – 2010. – P. 681–690.
11. *Daniel D. Lee and H. Sebastian Seung.* Algorithms for non-negative matrix factorization. – 2000.
12. *nVidia*, 2013b. CUDA CUSPARSE Library. nVidia, August.
13. *nVidia*, 2013a. CUBLAS Library User Guide. nVidia, v 5.0 edition, October.