A variation on SVD based image compression

Постановка задачі

Завдання алгоритму - стиснути зображення з мінімальними втратами інформації та з максимальною економією пам'яті. Автори використовують модифіковану версію алгоритму *SVD*, який, за їхніми словами, ефективніший за оригінальний алгоритм, оскільки потребує на 30% менше пам'яті для зберігання значень та векторів. Ця варіація алгоритму передбачає - попередня обробку зображення перестановкою - після чого воно обробляється оригінальним *SVD* алгоритмом - післяобробку - інвертованою перестановкою із попередньої обробки. Після опису алгоритму автори беруть різні зображення та перевіряють ефективність даного алгоритму порівнюючи з найпоширенішими. Після цього добавляють декілька покращень пов'язаних з пам'яттю.

Опис алгоритму власними словами

SVD розклад матриці

Означення

На вхід у нас є зображення - **A**. Розглядаємо його як матрицю. Наша задача розкласти нашу матрицю на 3 компоненти **A = UΣV*** [* - тут і надалі означатиме, що матриця є транспонованою (*transposed*)] , де: - **A** - вхідна матриця - **Σ** - діагональна матриця, де елементи є невід'ємними та посортованими звеху-вниз у незростаючому порядку - **U**,**V*** - ліва та права сингулярні матриці Також: - **r - ранг**, розмірність матриці **Σ**

Кроки

1. Знайти добуток AA*, A*A
2. Знайти власні значення (*eigenvalues*) матриці **AA*** та **A*A** (позначимо їх як λ_n)
3. власні значення матриці AA*: такі λ, де det(AA* - λI) = 0
4. власні значення матриці A*A: такі λ, де det(A*A - λI) = 0
5. Знайти сингулярні значення (*singular values*) матриці **AA*** або **A*A** з власних значень позначимо їх як ∂_n
6. Для SVD сингулярні значення є квадратні корені власних значень матриць **AA*** або **A*A**
7. Сформувати **Σ**
8. створити список $(\partial_1, ...\partial_n)$, де $\partial_i \geq \partial_j \geq 0, i < j$
9. **Σ** = $diag(\partial_1, ...\partial_n)$
10. Знайти **V*** з власних значень та матриці **A*A**
11. Підставляємо кожне λ_n в рівняння $A^*A - \lambda I = 0$ та знаходимо розв'язки рівнянь, що у нас вийшли (іншими словами знаходимо власні вектори)
12. Вектори-розв'язки сформують матрицю **V**
13. Дістанемо **V*** з **V**
14. Знайти **U** виразивши її з рівняння **A = UΣV***

Визначення розмірності

В подальших кроках ми шукаємо оптимальне значення **r**, таке щоб інформація, що залишилася максимально реперезентативно описувала зображення. Чим менше значення **r** тим менше пам'яті займатиме отримане зображення, але тим більші будуть втрати даних (зображення виглядатиме менш чітко) і навпаки. Суть в тому що значення **r** не впливає на розмірність вихідної матриці **A** і зображення буде тих самих розмірів після того як ми це все перемножимо.

Кінцеві розмірності матриць:

- $\mathbf{A}_r[n \times m]$
- $\mathbf{U}_r[n \times r]$, $\mathbf{U}_r$ - це перші r колонок **U**
- $\mathbf{\Sigma}_r[r \times r]$, $\mathbf{\Sigma}_r$ - це верхня-ліва підматриця **Σ**
- $\mathbf{V*}_r[r \times m]$, $\mathbf{V*}_r$ - це перші r рядків **V***

*S*SVD модифікація

Перш за все, слід зразу зауважити, що *S*SVD це лише модифікація звичайного SVD, яка передбачає додаткову попередню обробку матриці перед SVD-декомпозицією і кінцеву обробку опісля інвертовано. Попередня обробка складається із перетворення матриці **A** в Матрицю **X=S(A).** Автори наводять наступну формулу для визначення якому індексу в матриці **X** належатиме елемент $\mathbf{A}[i,j]$:

$$X[\lfloor i/n \rfloor n + \lfloor j/n \rfloor, (i \bmod n)n + j \bmod n] = A[i,j]$$

Кінцева обробка відбуваєтсья **аналогічною** інвертованою формулою, але вже для визначення індексів для **A.**

У багатьох випадках, при роботі з матрицею **X** (*S*SVD алгоритм) можна отримати до *30%* економії пам’яті у порівнянні з робтою з матрицею **A** (оригінальний SVD). Не у всіх випадках можна отримати значного підвищення ефективності. Насправді коли ми працюємо з чіткими зображеннями ліній чи геометричних фігур, *S*SVD буде менш ефективним. Тому його в рази краще використовувати при обробці фотографій.

Оцінка складності алгоритму

Розглянемо складність SVD декомпозиції:

Для простоти будемо вважати що ми завжди маємо справу з матрицями розмірності $[n \times n]$, де n - більше значення кількості колонок і рядків матриці. В реальності різциця між шириною і висотою зображення незначна, а розмірності матриць на які ми розкладаємо зображення менші ніж розмірність самого зображеня, а ми беремо верхнє значення. *Оцінки складності алгоритмів (множення матриць, пошук визначника тощо)*

- Транспоновану матрицю ми можемо взяти за `O(1)`
- Brute-force множення матриць займає `O(n^3)` , хоча алгоритм Копперсміта-Вінограда справляється за `O(n^2,3728..)`
- 1 крок: складається з транспонування і множення матриць, тому складність `O(n^2,3728..)`
- 2 крок: пошук детермінанта (і власних значень (*eigenvalues*), звичайно) займає `O(n^2,3728..)`
- 3 крок: пошук кореня `O(1)`
- 4 крок: посортувати **Σ** можна за `O(nlogn)`
- 5 крок:
  1. Підставляємо кожне λ_n в рівняння $A^*A - \lambda I = 0$ та знаходимо розв’язки рівнянь, що у нас вийшли (іншими словами знаходимо власні вектори)
  2. `O(n^2,3728..)`
  3. Нормалізувати вектори `O(n^2)`
  4. Дістанемо **V*** з **V**- це `O(1)`

- 6 крок: Знайти **U** = *A(ΣV)* = AVΣ* : `O(n^2,3728..)` + `O(1)` + `O(n^2,3728..)` = `O(n^2,3728..)`

Результат

Як бачимо - максимальна складність алгоритму на кожному із етапів не більше ніж `O(n^2,3728..)`, отже загальна складість нашого алгоритму дорівнює:

O(n^2,3728…) .

Посилання на оригінальну статтю

Використані посилання

http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm
http://www.d.umn.edu/~mhampton/m4326svd_example.pdf
https://youtu.be/P5mlg91as1c

Виконали *Забульський Володимир* та *Вей Роман*