矩阵分析与应用

李保滨

2024秋季学期

讨论题一 矩阵转置的快速实现

1. 最简单的实现方法：遍历（并非快速实现）

对于矩阵$A_{m\times n}$
新建一个矩阵$B_{n\times m}$,随后遍历矩阵A,按照$b_{ji}=a_{ij}$赋值即可

```python
def transpose_matrix(matrix):
    rows = len(matrix)
    cols = len(matrix[0])

    # 创建一个空的转置矩阵
    transposed_matrix = [[0] * rows for _ in range(cols)]

    # 填充转置矩阵
    for i in range(rows):
        for j in range(cols):
            transposed_matrix[j][i] = matrix[i][j]

    return transposed_matrix
```

空间复杂度$O(mn)$
时间复杂度$O(mn)$

2. 就地转置：适用于方阵

对于方阵$A_{n\times n}$,我们可以直接交换a_{ij}和a_{ji}即可
不需要额外的矩阵存储空间

```python
def in_place_transpose(matrix):
    n = len(matrix)

    for i in range(n):
        for j in range(i + 1, n):
            # 交换元素 A[i][j] 和 A[j][i]
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

    return matrix
```

空间复杂度$O(1)$
时间复杂度$O(n^2)$,需要遍历矩阵的上三角/下三角

3. 分块转置

对于大矩阵转置的问题，我们的内存和硬件条件是有限的，因此可以对矩阵进行分块转置

将大矩阵分成若干个小块（block），逐块进行转置操作。这样可以避免一次性将整个矩阵加载到内存中，适用于内存有限的情况。通过使用块操作，矩阵的转置可以在小的内存片段上逐步完成。

具体实现步骤：

1. 将矩阵划分为多个小块，每个块的大小取决于内存容量和缓存大小。
2. 对每个小块进行转置，并存储在新的位置。
3. 依次处理所有块，直到完成整个矩阵的转置。

```python
import numpy as np

def blockwise_transpose(matrix, block_size):
    n, m = matrix.shape
    transposed_matrix = np.zeros((m, n), dtype=matrix.dtype)

    # 分块处理
    for i in range(0, n, block_size):
        for j in range(0, m, block_size):
            # 获取当前块的行和列范围
            i_end = min(i + block_size, n)
            j_end = min(j + block_size, m)

            # 转置并复制块到目标位置
            transposed_matrix[j:j_end, i:i_end] = matrix[i:i_end, j:j_end].T

    return transposed_matrix

# 示例矩阵
matrix = np.random.rand(10000, 10000)
block_size = 100  # 根据可用内存和缓存设置块大小

transposed_matrix = blockwise_transpose(matrix, block_size)
```

时间复杂度：$O(mn)$，与直接转置相同，但更好地利用了缓存和内存
空间复杂度：$O(mn)$ 存储转置后的矩阵，但在每次处理的块大小上优化了内存使用

4. 对于稀疏矩阵的更好的方法

对于稀疏矩阵$A_{m\times n}$,其中有k个非零元
我们可以使用稀疏矩阵的压缩格式来进行存储：即只存储非零元及其索引位置
转置操作可以通过更改索引来高效实现

> 常用的稀疏矩阵存储格式有CSR（Compressed Sparse Row，压缩行存储格式）和CSC（Compressed Sparse Column，压缩列存储格式）

```python
from scipy.sparse import csr_matrix

# 创建一个稀疏矩阵，m 行 n 列
m, n = 5, 4  # 示例大小，可以根据需要调整
row_indices = [0, 1, 2, 3, 4, 0, 2]  # 非零元素的行索引
col_indices = [0, 1, 2, 3, 0, 2, 3]  # 非零元素的列索引
data = [10, 20, 30, 40, 50, 60, 70]  # 非零元素的值

# 创建 CSR 格式的稀疏矩阵
sparse_matrix = csr_matrix((data, (row_indices, col_indices)), shape=(m, n))

print("原始矩阵（CSR格式）：")
print(sparse_matrix)

# 转置稀疏矩阵
transposed_matrix = sparse_matrix.transpose()

print("\n转置后的矩阵（CSR格式）：")
print(transposed_matrix)
```

稀疏矩阵$A_{m\times n}$,其中有k个非零元
时间复杂度：取决于非零元素数量,$O(k)$。
空间复杂度：$O(k)$，比稠密矩阵更节省空间。

5. 特殊的操作

使用NumPy库的内置函数`transpose` 或 `.T`属性来快速转置矩阵

```python
import numpy as np

# 示例矩阵
matrix = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

# NumPy 的转置方法
transposed_matrix = matrix.T
```

时间复杂度：$O(1)$，因为 NumPy 的`.T`只返回一个视图，不会重新分配内存。
空间复杂度：$O(1)$，不需要额外的存储空间（只是返回一个视图）。

6. 其他

矩阵转置还可以使用硬件进行加速，也还可以配合其他的库进行加速。
更多的方法同学们可以自行深入研究。

如果有所补充或纠正，请发邮件**liutongqing23@mails.ucas.ac.cn**