SVD

Владимир Яшин

18 сентября 2016 г.

1 Задача SVD

Иногда неудобно хранить матрицу большой размерности в памяти компьютера, а два длинных вектора — проще.

Рассмотрим матрицу $A \in \mathbb{R}^{m\times n}$. Согласно методу SVD (Singular Value Decomposition) указанную матрицу A, всегда можно разложить на три матрицы, а именно

$$A = U\Sigma V^t,$$

где $A \in \mathbb{R}^{m\times n}$ — исходная матрица;
$U \in \mathbb{R}^{m\times m}$ — ортогональная матрица, в которой столбцы являются собственными векторами AA^t;
$\Sigma \in \mathbb{R}^{m\times n}$ — диагональная матрица, где на главной диагонали квадратный корень *положительных* собственных значений матриц AA^t и A^tA;
$V \in \mathbb{R}^{n\times n}$ — ортогональная матрица, в которой столбцы являются собственными векторами A^tA.

Так как нам не удаётся найти собственные числа матриц, не прибегая к итеративным подсчётам, воспользуемся методом градиентного спуска.

2 Задача градиентного спуска

Допустим, мы хотим локально оптимизировать функцию $\varepsilon(\theta) : \mathbb{R}^n \to \mathbb{R}$. Возьмём некоторый вектор вещественных решений $\theta = \theta_0$, от которых алгоритм начнёт свою работу. Итерации выглядят следующим образом

Первая итерация:

$$\theta_1 := \theta_0 \pm \alpha \cdot \nabla\varepsilon(\theta = \theta_0).$$

к-ая итерация:

$$\theta_k := \theta_{k-1} \pm \alpha \cdot \nabla\varepsilon(\theta = \theta_{k-1}),$$

где $\alpha \in \mathbb{R}$ — скорость обучения. От неё зависит скорость оптимизации;
$\varepsilon(\theta = \theta_i) \in \mathbb{R}^n$ — градиент в точке θ_i.

Повторять итерации пока изменения θ не станут настолько близкими к нулю, насколько это нужно исследователю.

3 SVD и градиентный спуск

Применим метод градиентного спуска, чтобы найти разложение матрицы $A \in \mathbb{R}^{m \times n}$, подразумевая, что

$$A \approx B \cdot C^t,$$

где $B \in \mathbb{R}^{m \times k}$, а $C \in \mathbb{R}^{n \times k}$.

Для начала необходимо выбрать целевую функцию для оптимизации. В данном случае разумно взять функцию ошибки $\varepsilon : \mathbb{R}^{m \times n} \to \mathbb{R}$ вида

$$\varepsilon = \frac{1}{2} \cdot ||\hat{A} - A|| = \frac{1}{2} \cdot ||B \cdot C^t - A||,$$

где $B \cdot C^t$ — произведение матриц $B \in \mathbb{R}^{m \times k}$ и транспонированной $C \in \mathbb{R}^{n \times k}$;
$\frac{1}{2}$ — множитель, который в дальнейшем сократиться при взятии производной от квадратов.

Указанное уравнение можно также переписать как сумму

$$\varepsilon = \frac{1}{2} \cdot ||\hat{A} - A|| = \frac{1}{2} \cdot \sum_j^n \sum_i^m (\hat{A}_{ij} - A_{ij})^2.$$

Изменяемыми параметрами функции ε являются матрицы B и C. Поэтому взяв производные для каждого $b_{ij}, c_{ij} \quad \forall i, j \in [1 \dots m], [1 \dots n]$ получим

$$B = B - \alpha \cdot \nabla\varepsilon_B,$$
$$C = C - \alpha \cdot \nabla\varepsilon_C,$$

где $\nabla\varepsilon_B$ и $\nabla\varepsilon_C$ — матрицы вида

$$\nabla\varepsilon_B = (\hat{A} - A) \cdot C,$$
$$\nabla\varepsilon_C = (\hat{A} - A)^t \cdot B.$$

Итерации метода градиентного спуска выглядят следующим образом

Первая итерация:

$$\begin{aligned}
\hat{A}_0 &:= B_0 \cdot C_0^t \\
\nabla\varepsilon_{B_0} &:= (\hat{A}_0 - A) \cdot C_0 \\
\nabla\varepsilon_{C_0} &:= (\hat{A}_0 - A)^t \cdot B_0 \\
B_1 &:= B_0 - \alpha \cdot \nabla\varepsilon_{B_0} \\
C_1 &:= C_0 - \alpha \cdot \nabla\varepsilon_{C_0} \\
\hat{A}_1 &:= B_1 \cdot C_1^t
\end{aligned}$$

Вторая итерация:

$$\begin{aligned}
\nabla\varepsilon_{B_1} &:= (\hat{A}_1 - A)\cdot C_1 \\
\nabla\varepsilon_{C_1} &:= (\hat{A}_1 - A)^t \cdot B_1 \\
B_2 &:= B_1 - \alpha\cdot\nabla\varepsilon_{B_1} \\
C_2 &:= C_1 - \alpha\cdot\nabla\varepsilon_{C_1} \\
\hat{A}_2 &:= B_1 \cdot C_1^t
\end{aligned}$$

к-ая итерация:

$$\begin{aligned}
\nabla\varepsilon_{B_{k-1}} &:= (\hat{A}_{k-1} - A)\cdot C_{k-1} \\
\nabla\varepsilon_{C_{k-1}} &:= (\hat{A}_{k-1} - A)^t \cdot B_{k-1} \\
B_k &:= B_{k-1} - \alpha\cdot\nabla\varepsilon_{B_{k-1}} \\
C_k &:= C_{k-1} - \alpha\cdot\nabla\varepsilon_{C_{k-1}} \\
\hat{A}_k &:= B_k \cdot C_k^t
\end{aligned}$$

Код на Питоне можно найти в Листинге 1 или на ГитХабе вместе с примером.

Листинг 1: SVD using gradient descent on Python.

```python
def SVD(mat, initial_mat1, initial_mat2, learn_rate, iterations):
    # the m by n matrix that we want to approximate
    A = mat
    # two matrices from which we will start: B is m by k
    B = initial_mat1
    # C is n by k
    C = initial_mat2
    # learning rate, or step of learning
    alpha = learn_rate
    # number of iterations
    N = iterations
    # A ~ B * C^t : the first approximation based on given initial matrices
    A_app = np.dot(B, C.T)
    # gradient descent
    for i in range(N):
        # partial derivatives for matrices
        dLdB = np.dot((A_app - A), C)
        dLdC = np.dot((A_app - A).T, B)
        # updating matrices
        C = C - alpha * dLdC
        B = B - alpha * dLdB
        # calculating approximated matrix
        A_app = np.dot(B, C.T)
    # returning two matrices that can be used for A approximation
    return B, C, A_app
```