Mining and Processing Biomedical Data

Dr. rer. nat. Krisztian Buza
adiunkt naukowy
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw, Poland
chrisbuza@yahoo.com

Matrix Completion

Matrix completion

- Given a sparse matrix
- We want to fill-in the unknown values
- The values of the matrix are dependent on each other
- Approaches
  - Search for similar rows/columns
  - Matrix factorization
  - Restricted Boltzmann Machines (RBM)
  - …

5	?	1	?	?	…
?	?	5	?	4	…
5	4	2	?	?	…
?	3	?	2	5	…
1	?	5	?	4	…
5	4	?	?	2	…
…	…	…	…	…	…

Example: Movie-rating prediction (recommender systems)

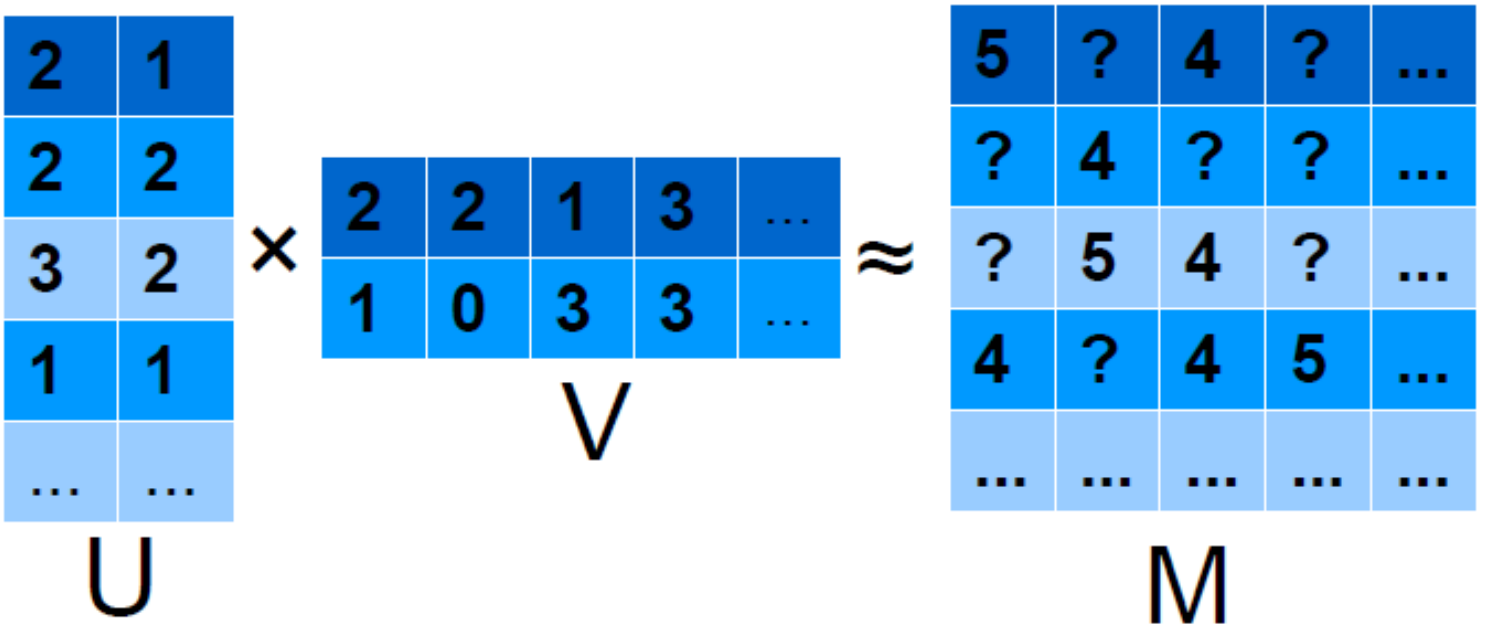
	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	
User 1	5	?	1	?	?	...
User 2	?	?	5	?	4	...
User 3	5	4	2	?	?	...
User 4	?	3	?	2	5	...
User 5	1	?	5	?	4	...
User 6	5	4	?	?	2	...
	...	...	...	...	...	...

Quiz question: How would you fill in this question mark?

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	
User 1	5	?	1	?	?	...
User 2	?	?	5	?	4	...
User 3	5	4	2	?	?	...
User 4	?	3	?	2	5	...
User 5	1	?	5	?	4	...
User 6	5	4	?	?	2	...
	...	...	...	...	...	...

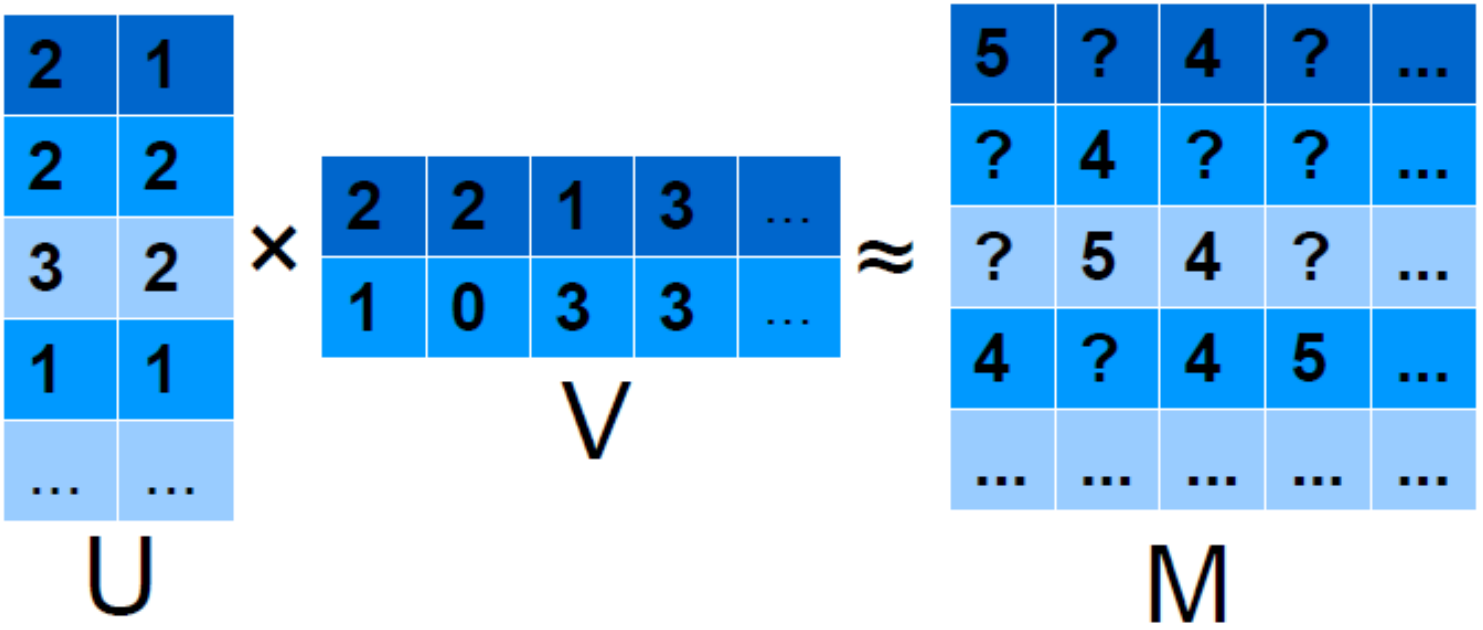
Matrix factorization

- We estimate matrix *M* as the product of two matrices *U* and *V*.
- Based on the known values of *M*, we search for *U* and *V* so that their product best estimates the (known) values of *M*

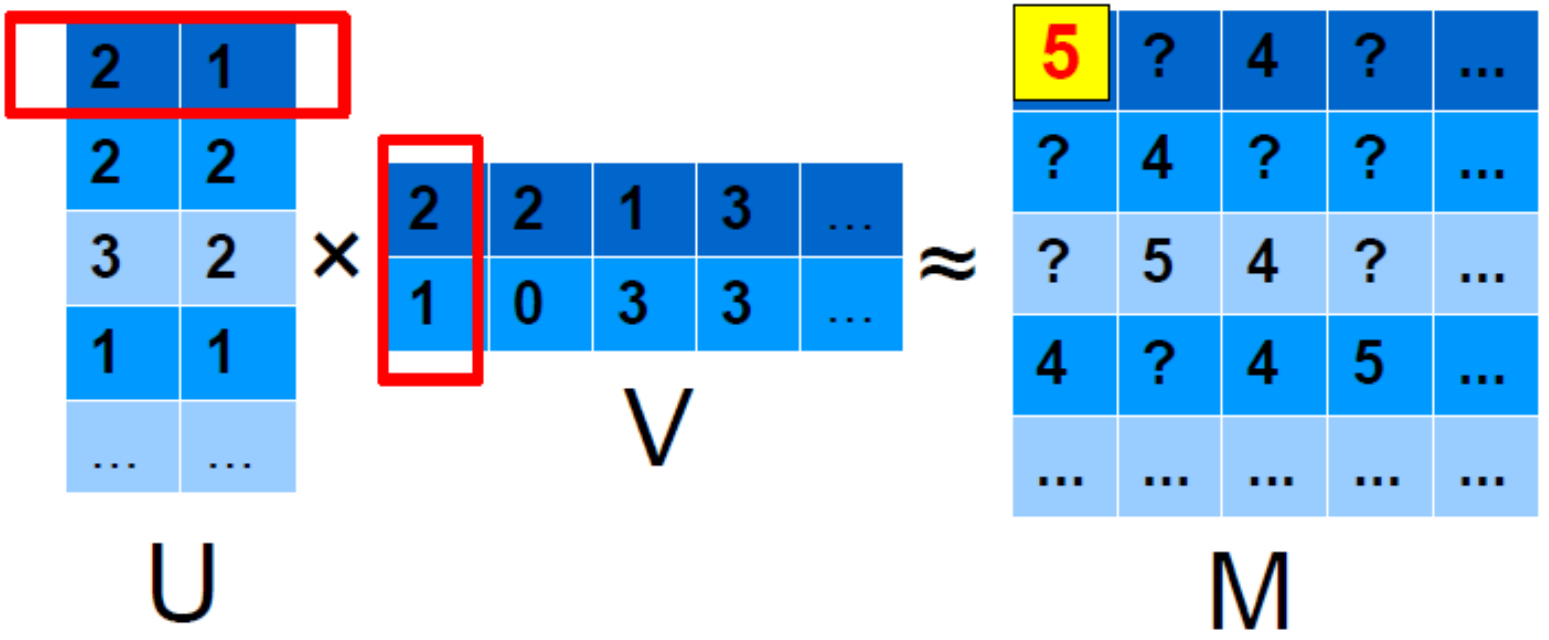


Matrix factorization algorithm

- Random initialization of *U* and *V*
- While *U* x *V* does not approximate the known values of *M* well enough
  - Choose a known value of *M,* we denote it by *x*
  - Adjust the values of the corresponding row and column of U and V respectively, so that the approximation becomes better

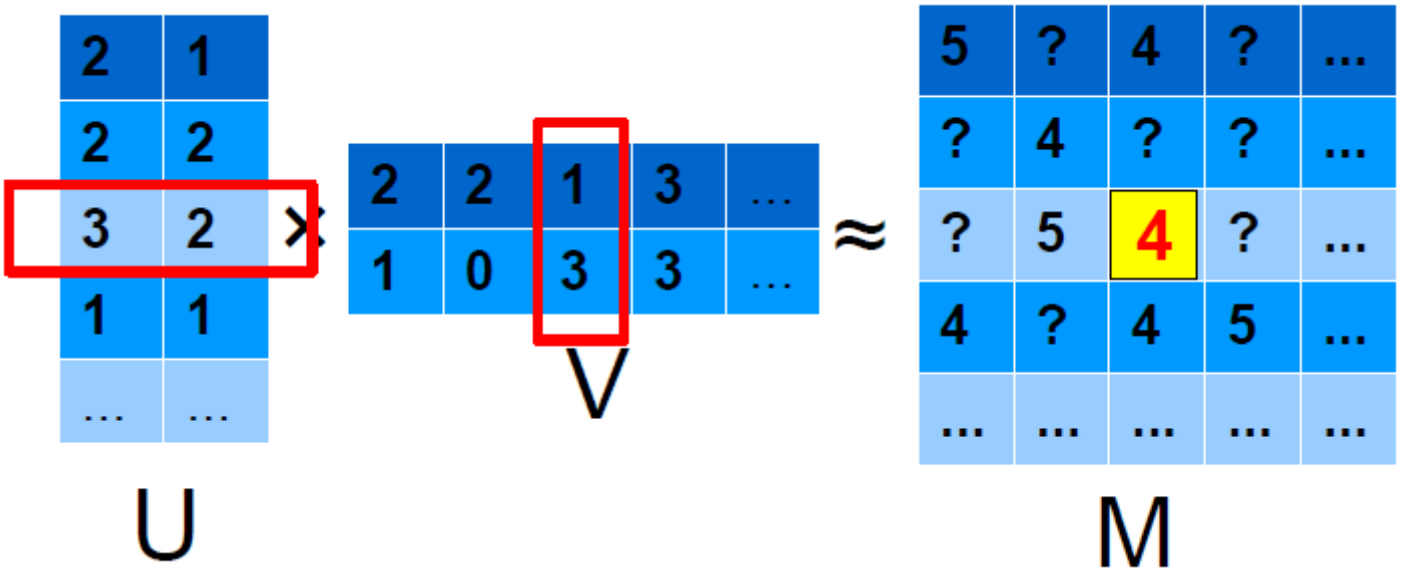


Example for an adjustment step



(2*2)+(1*1) = 5 which equals to the selected value → we do not do anything

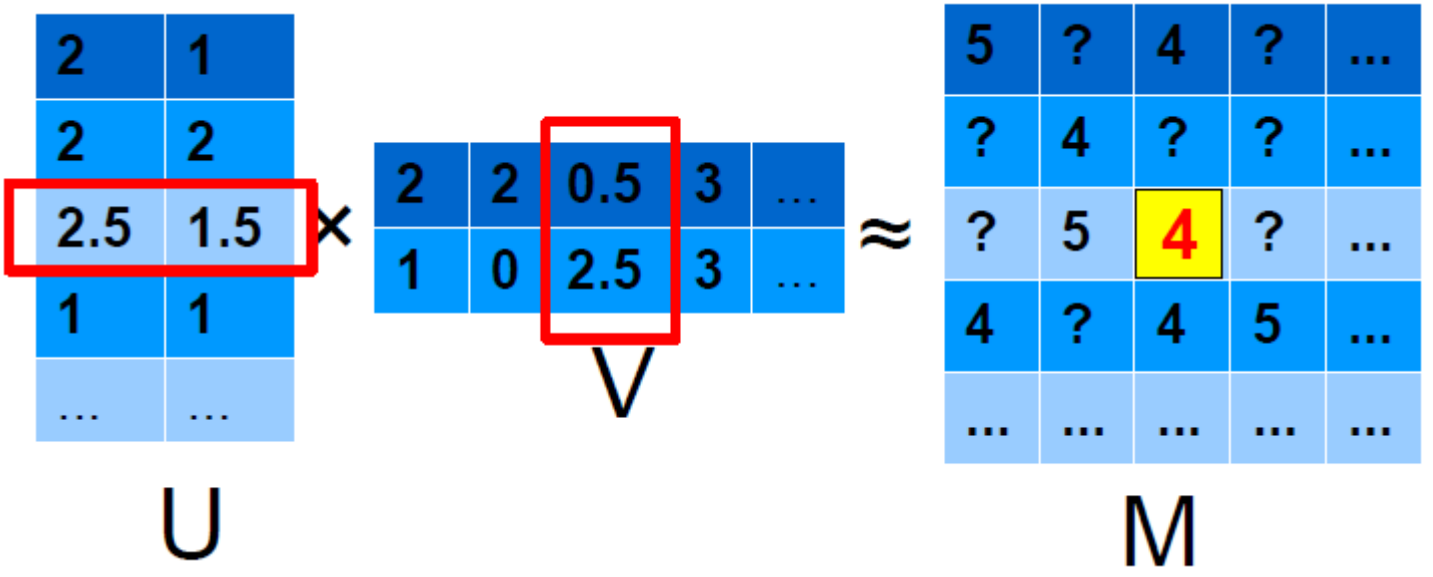
Example for an adjustment step



(3*1)+(2*3) = 9

9 > 4 → we decrease the values of the corresponding rows so that their products will be closer to 4

Example for an adjustment step

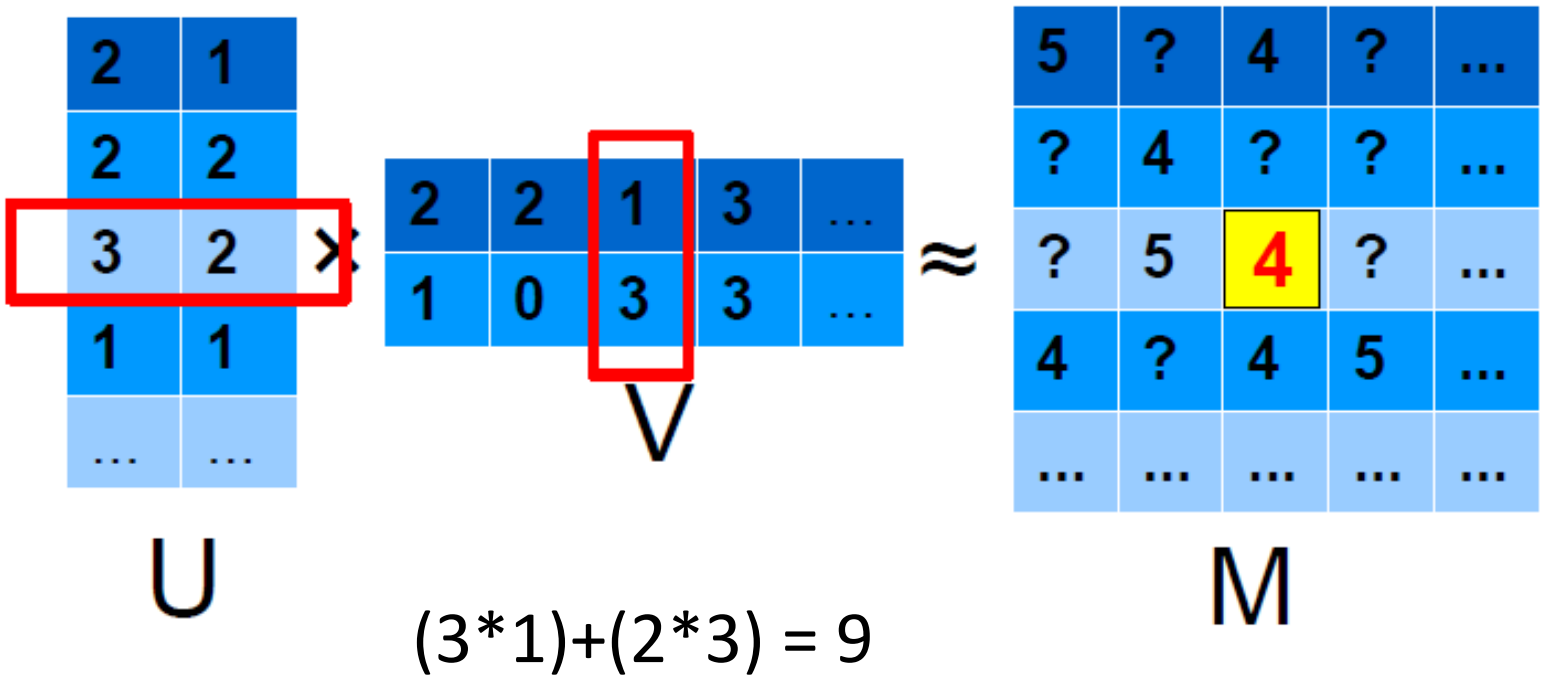


(3*1)+(2*3) = 9

9 > 4 → we decrease the values of the corresponding rows so that their products will be closer to 4

What is a good adjustment step?

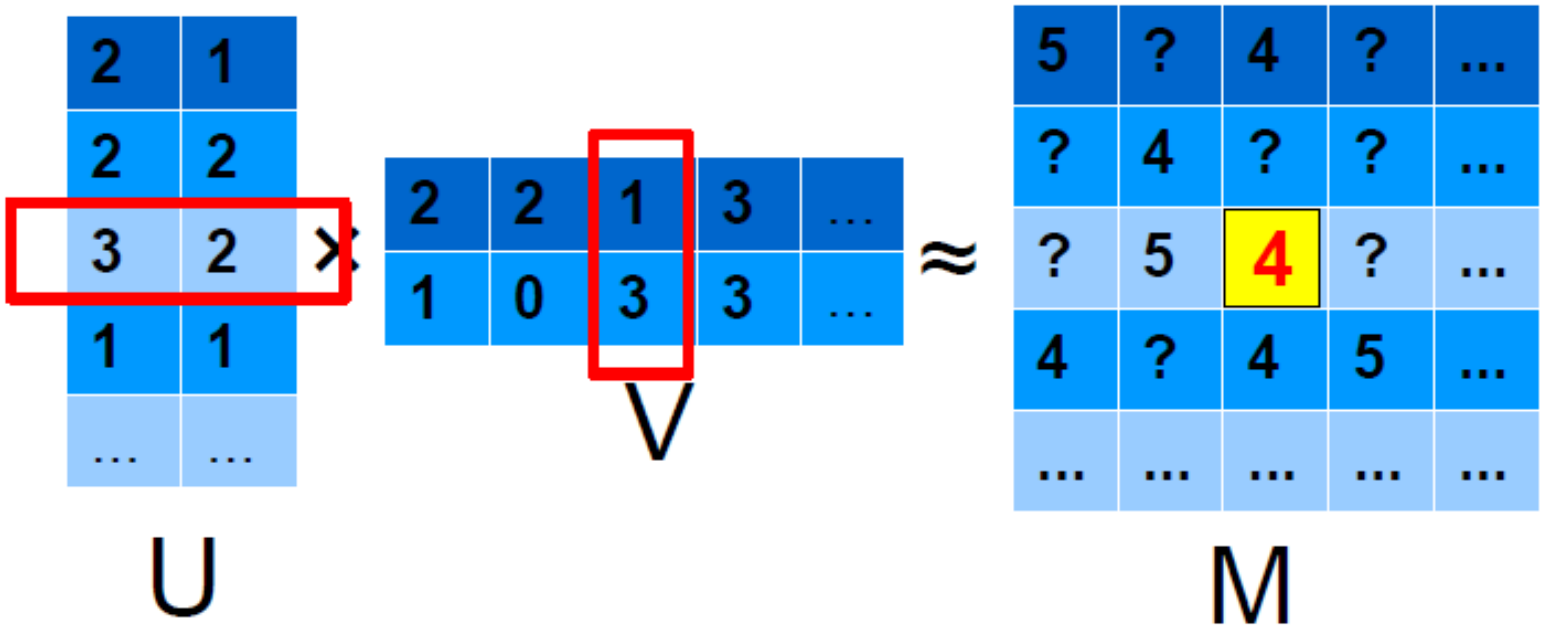
- 1. The adjustment should be proportional to the error → let it be ε-times the error
  - In the current example: error = 9 – 4 = 5
    with ε=0.1 we will decrease all the values in the corresponding rows and columns by 0.1*5=0.5



(3*1)+(2*3) = 9

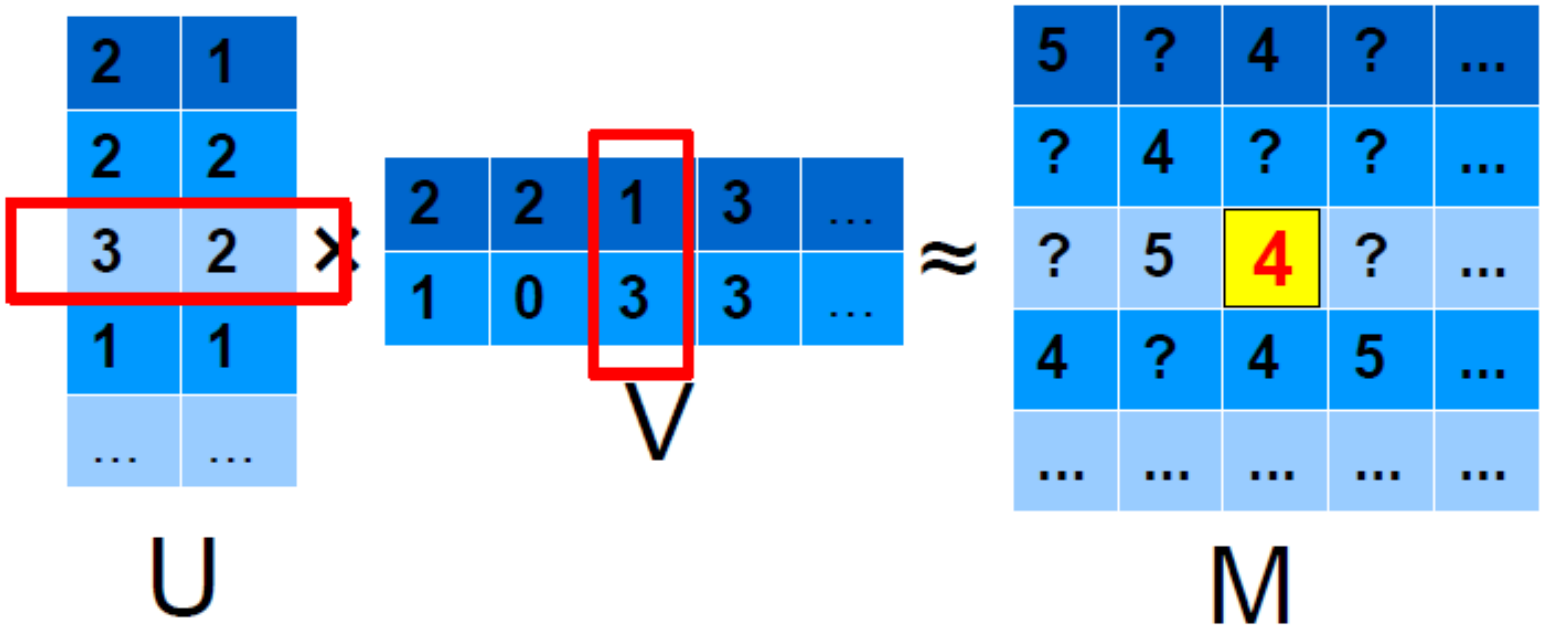
What is a good adjustment step?

- 2. We should take into account how much each value of the row/column contributes to the error
  - For the selected row:
    3 is multiplied by 1 → 3 is adjusted by ε*5*1 = 0.5
    2 is multiplied by 3 → 2 is adjusted by ε*5*3 = 1.5
  - For the selected column respectively:
    ε*5*3=1.5 and ε*5*2=1.0



What is a good adjustment step?

- 3. We prefer simpler models (avoid overfitting).
- At each adjustment step: subtract additionally λ-times the value
  - For the selected row: subtract additionally λ*3 from 3, and λ*2 from 2 .
  - For the selected column respectively: λ*1 and λ*3



What is a good adjustment step?

- 3. We prefer simpler models (avoid overfitting).
- At each adjustment step: subtract additionally λ-times the value
  - For the selected row: subtract additionally λ*3 from 3, and λ*2 from 2 .
  - For the selected column respectively: λ*1 and λ*3

Optional Homework:
read Section 1.1. “Example: Polynomial Curve Fitting,” from Christopher M. Bishop: Pattern Recognition and Machine Learning, Springer, 2006, pages 4-11

U M

Matrix Factorization Algorithm

Input: matrix *M* with *n* rows and *m* columns, integer *K*,
real number *eps*, real number *lambda*

1. **Create *U* and *V* matrices and initialize their values randomly**
   (*U* has *n* rows, *K* columns; *V* has *K* rows, *m* columns)
2. **While *U* x *V* does not approximate *M* well enough**
   (or the maximal number of iterations is not reached)
3. **For each known element *x* of *M***
4. **Let *i* and *j* denote the row and column of *x***
5. **Let *x′* be the dot product of the corresponding row of *U* and column of *V***
6. ***err* = *x′* − *x***
7. **for (k=0; k < K; k++)**
8. $u_{i,k} \leftarrow u_{i,k} - eps*err*v_{k,j} - lambda*u_{i,k}$
9. $v_{k,j} \leftarrow v_{k,j} - eps*err*u_{i,k} - lambda*v_{k,j}$
   // simultaneous update!
10. **end for**
11. **end for**
12. **end while**

A few words about maths...

- Why is the previously shown algorithm a good one?
- Error function:
  sum of squared errors + regularization

$$\sum_{i,j} \left(m_{i,j} - \sum_{k=0}^{K} u_{i,k} v_{k,j} \right)^2 + c \left(\sum_{i,j} u_{i,j}^2 + \sum_{i,j} v_{i,j}^2 \right)$$

  where c is the weight of the regularization term
  (i. e., a constant giving the importance of the regularization term)
- Minimization of the sum of squared errors plus the regularization term by gradient descent leads to the previously shown algorithm

How to set the parameters ε, λ and *K* ?

1. Select a subset of the known values of *M*

2. Execute the previous matrix factorisation algorithm using the selected subset only

3. Evaluate the result of the factorisation using the non-selected known values of *M*, i.e., check how well the product *U* x *V* estimates the non-selected, but known values of *M*

- In order to measure how well *U* x *V* estimates the non-selected, but known values of *M,* one can use for example the mean absolute error (MAE) or mean squared error (MSE), see e.g. Wikipedia

4. Repeat steps 2 and 3 for various settings of the values of the parameters, and select the parameter values that give the best result

5. Execute the algorithm using the selected parameter values using ALL the known values of *M,* and finally estimate the missing values of *M* using the product of *U* and *V*

Additional issues

- Local optimum vs. global optimum
- Memory-efficient implementation
  - sparse representation of *M*