Large Scale Machine Learning

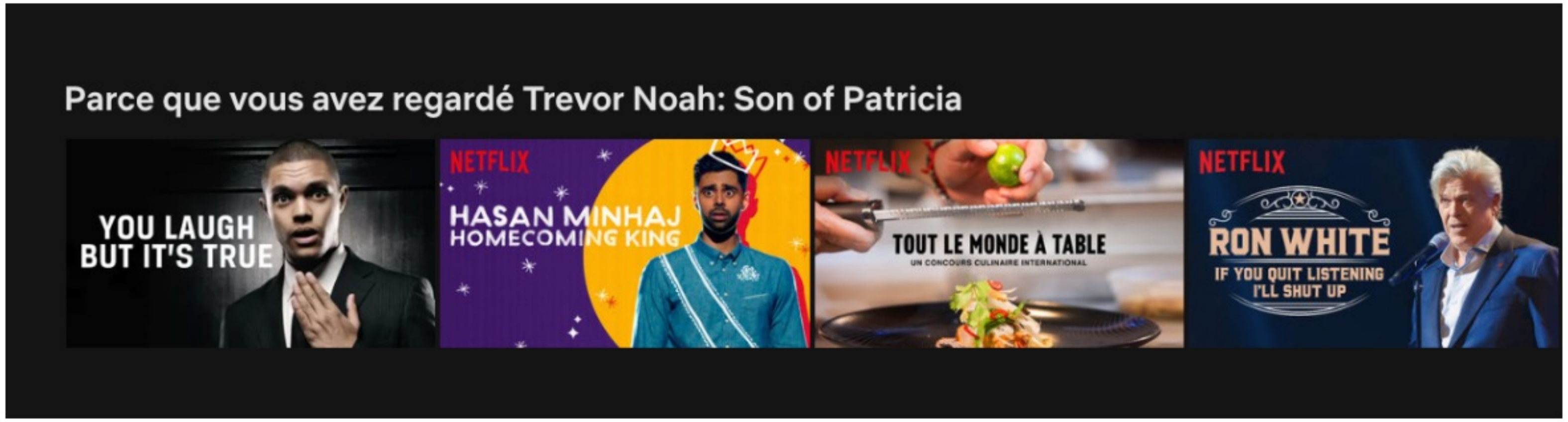
Matrix factorization

Recommender Systems

- Information overload
  - Many choices available
  - « The paradox of choice » (jam experiment, choice overload)

- Recommender system
  - Provide aid
  - Given a user and his « context » and a set of itesms => selection of items predicted to be « good » for the user

Recommender Systems



Collaborative filtering

- *« tell me what's popular among my peers »*
- One of the most often and successfully used techniques
- Widely applicable, does not need a lot of domain knowledge
- **Hypothesis**: Users who shared similar tastes in the past will continue to do so in the future

Collaborative filtering

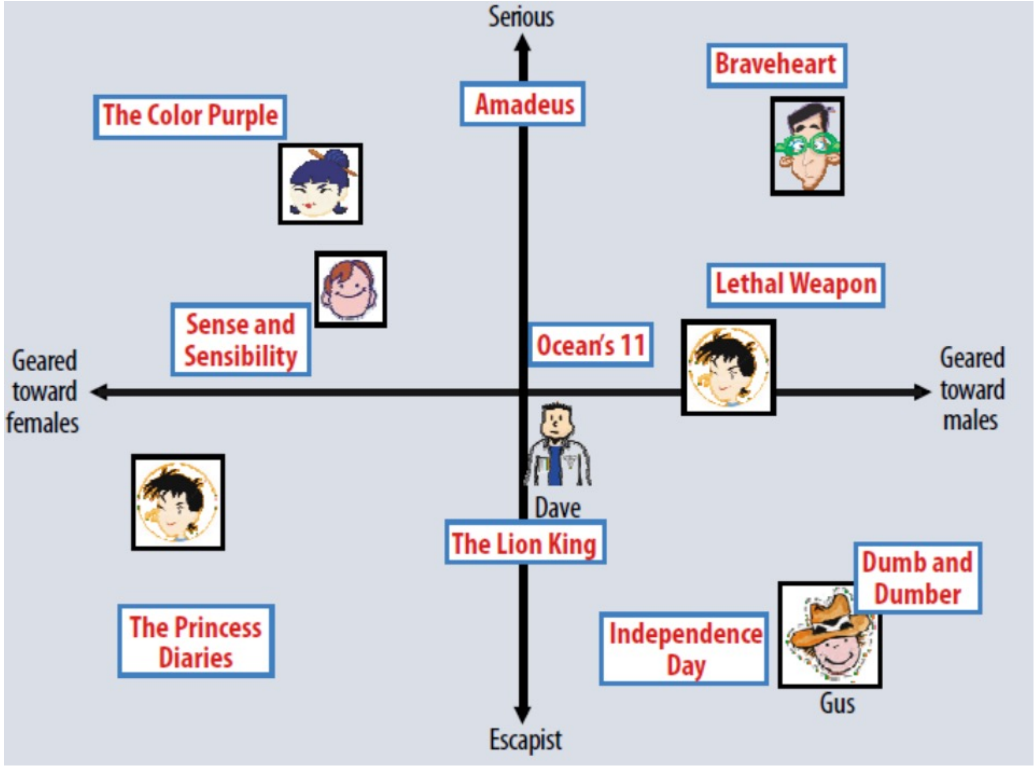
- **Most common setting:**
  - **Input**: matrix of user-item feedback or ratings (with missing values of course, this matrix very sparse)
  - **Output**: Predictions for missing values

- No need of item features (eg. movie genre, length or actors)

Collaborative filtering

- Explicit
  - The user rated explicitely the item (like/dislike, star rating …)
  - It requires effort from the user (friction)
  - Somewhat clear signal of what the user feels about the item
- Implicit
  - Click/Non Click, buying an item, visiting a page, viewing a video
  - Easier to collect, minimal friction
  - More « honest » (Netflix example: highly rated vs watched)
  - Recommended reading: https://www.wired.com/2013/08/qq-netflix-algorithm/

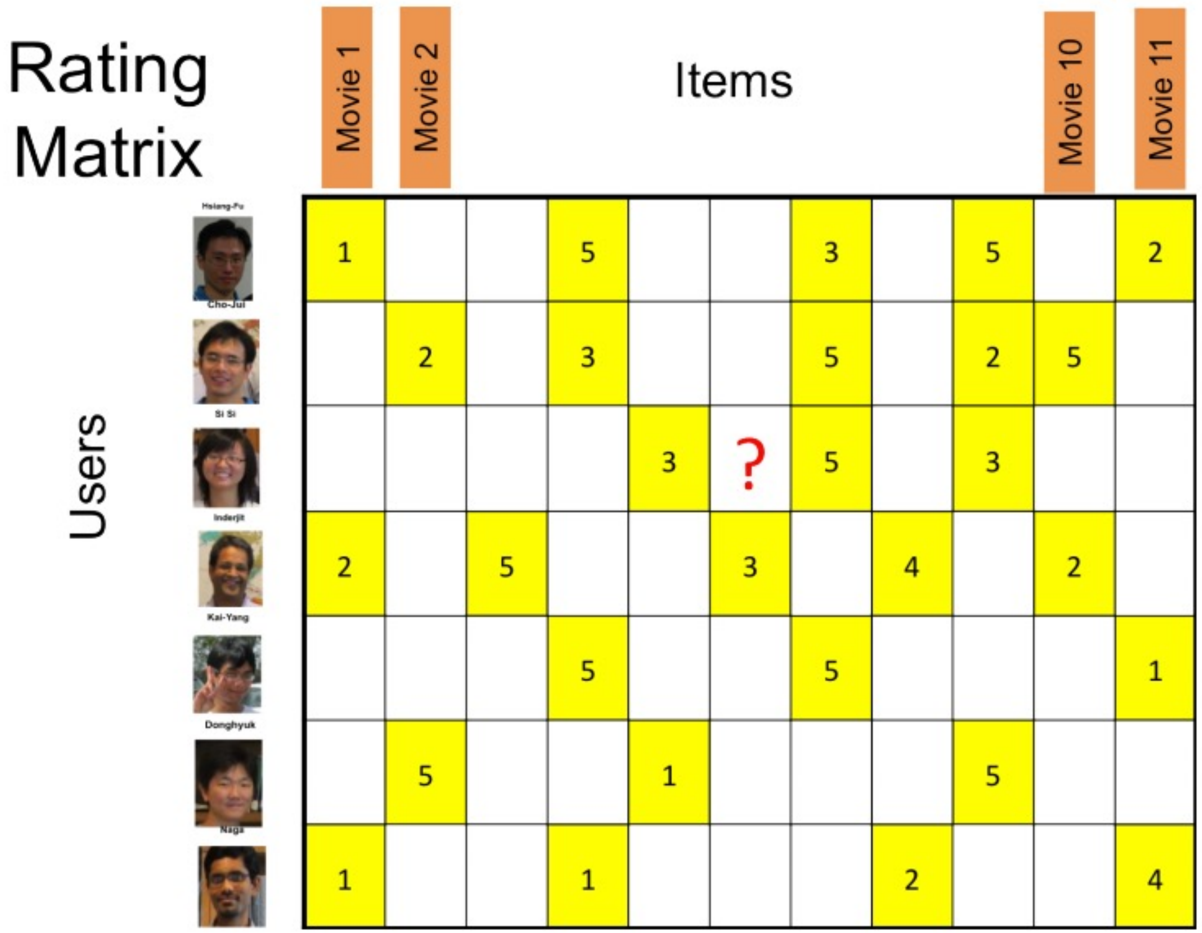
Matrix factorization



Matrix factorization

- Main idea: latent factors of users/items
  - Some users like action movies, romance, … at different proportions
  - Same can be said about movies
- Use these latent factors to
  - Predict interactions/ratings
  - Compute similarities between users and items

User-Item Matrix Factorization



User-Item Matrix Factorization

H^T

-0.07	-0.11	-0.53	-0.46	-0.06	-0.05	-0.53	-0.07	-0.35	-0.19	-0.14
0.13	-0.42	0.45	0.17	-0.25	-0.17	-0.18	0.27	-0.59	0.05	0.14
-0.21	-0.43	-0.23	0.16	0.08	0.17	0.57	-0.39	-0.37	-0.08	-0.15

W

-8.72	0.03	-1.03
-7.56	-0.79	0.62
-4.07	-3.95	2.55
-3.52	3.73	-3.32
-7.78	2.34	2.33
-2.44	-5.29	-3.92
-1.78	1.90	-1.68

1			5			3		5		2
	2		3			5		2	5	
				3		5		3		
2		5			3		4		2	
			5			5				1
	5			1				5		
1			1				2			4

User-Item Matrix Factorization

H^T

-0.07	-0.11	-0.53	-0.46	-0.06	-0.05	-0.53	-0.07	-0.35	-0.19	-0.14
0.13	-0.42	0.45	0.17	-0.25	-0.17	-0.18	0.27	-0.59	0.05	0.14
-0.21	-0.43	-0.23	0.16	0.08	0.17	0.57	-0.39	-0.37	-0.08	-0.15

W

-8.72	0.03	-1.03
-7.56	-0.79	0.62
-4.07	-3.95	2.55
-3.52	3.73	-3.32
-7.78	2.34	2.33
-2.44	-5.29	-3.92
-1.78	1.90	-1.68

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

Unconstrained MF

$$\min_{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}} \sum_{(i,j) \in \Omega} \left(A_{ij} - u_i^T v_j\right)^2 + \lambda \left(\|U\|_F^2 + \|V\|_F^2\right)$$

- A m-by-n rating matrix
  - m number of users
  - n number of items
- u_i embedding (latent factors vector) for user $i \in \mathbb{R}^k$
- v_j embedding (latent factors vector) for item $j \in \mathbb{R}^k$
- $\Omega = \left\{(i,j) \mid A_{ij} \; is \; observed\right\}$
- Regularization terms to avoid overfitting

Unconstrained MF: Iterative optimization

- Use SGD (or second order methods to find the parameters)

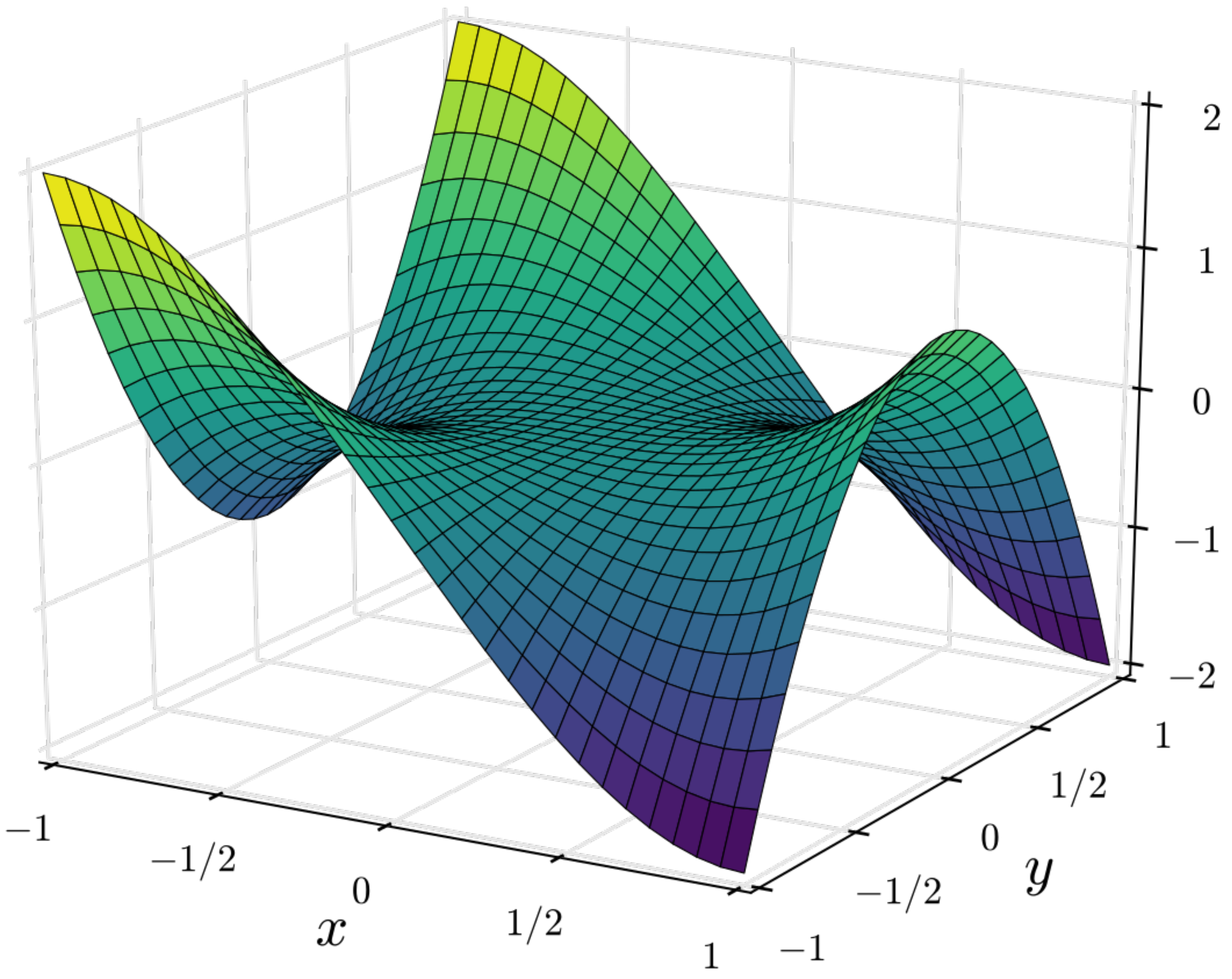
$$L(A_{ij}, u_i, v_j) = \left(A_{ij} - u_i^T v_j\right)^2 + \lambda\,(\|u_i\|^2 + \|v_j\|^2)$$

- For each non-missing entry A_{ij}
  - Read ith row of U and jth row of V
  - Update u_i and v_j

$$u_i \leftarrow u_i - \alpha\, \nabla_{u_i} L(A_{ij}, u_i, v_j)$$

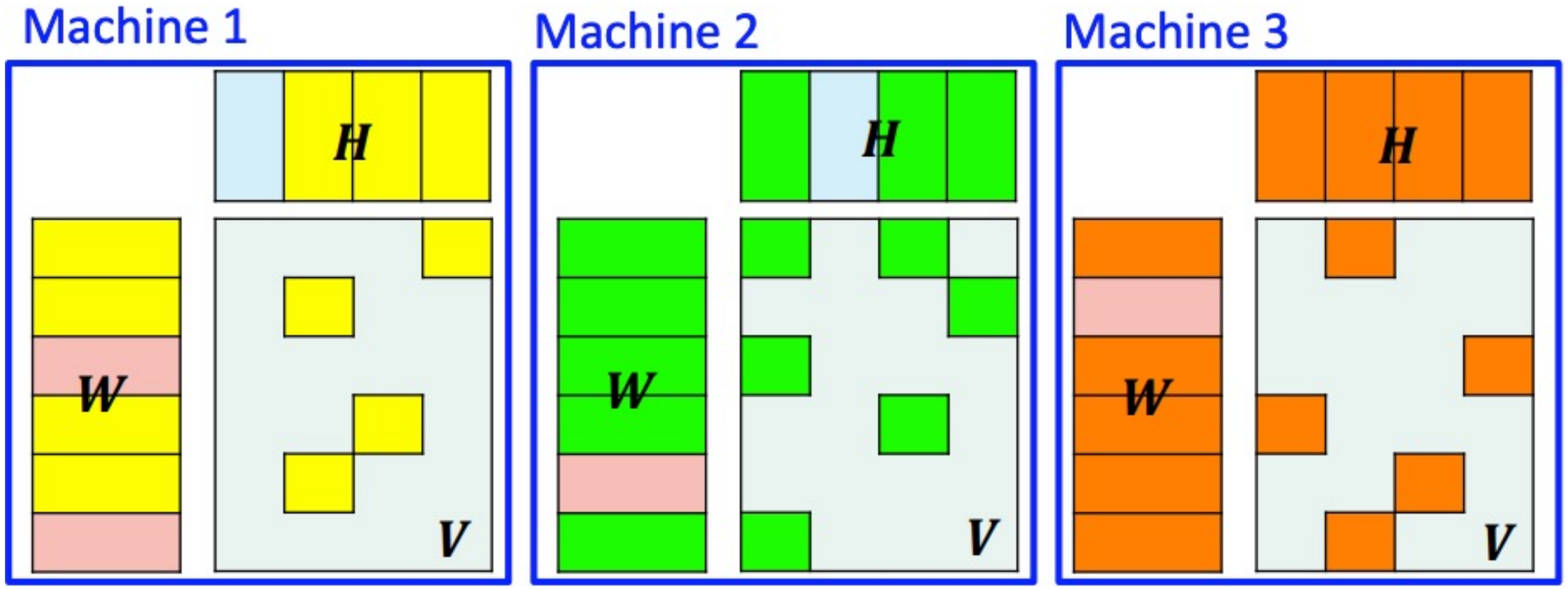
$$v_j \leftarrow v_j - \alpha\, \nabla_{v_j} L(A_{ij}, u_i, v_j)$$

Unconstrained MF: Iterative optimization



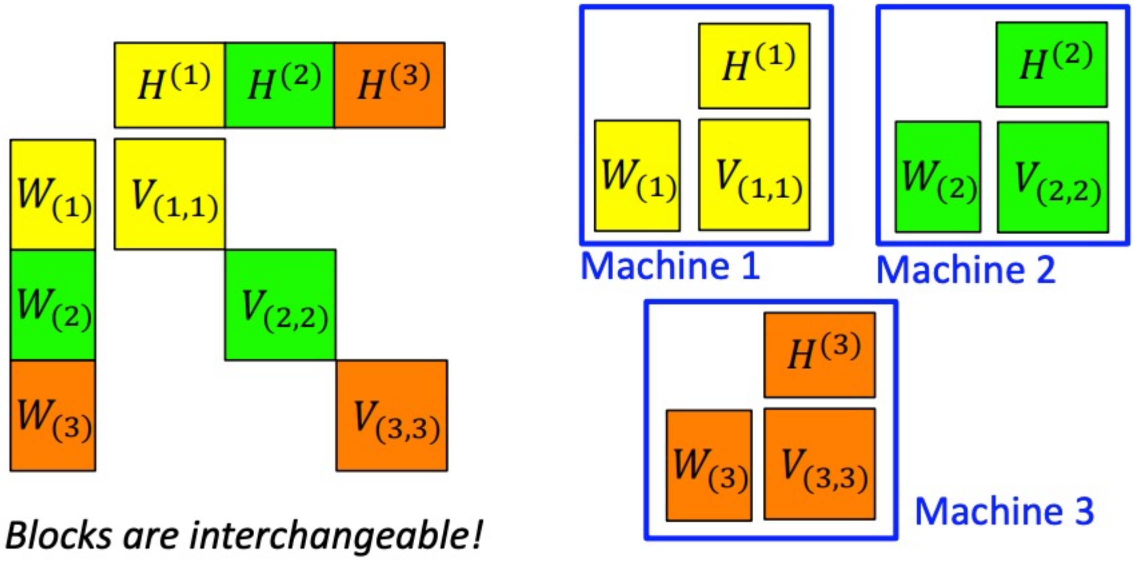
Unconstrained MF: Simple Parallel SGD

- Like in the previous lecture, we can distribute the work across multiple machines and average the gradient updates



Unconstrained MF: Simple Parallel SGD

- Even better, find set of independant or almost independant blocks



ALS: Alternative Least Squares

$$\min_{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}} L(A, U, V), \qquad L(A, U, V) = \sum_{(i,j) \in \Omega} \left(A_{ij} - u_i^T v_j\right)^2 + \lambda \left(\|U\|_F^2 + \|V\|_F^2\right)$$

- $L(A, U, V)$ is a non-convex function of U and V
  - Many local optima, saddle points …

- BUT $L(A, U, V)$ is a non-convex function of U (if we fix V), same thing for V (if we fix U)
  - One local optimum, that is also global
  - There is a closed form solution

ALS: Alternative Least Squares

- Repeat until convergence
    - Assume item factors V are fixed, solve for u_i

$$u_i = (V^T V + \lambda I)^{-1} V^T r_{i*}$$

    - Then assume user factors U are fixed, solve for V

$$v_j = (U^T U + \lambda I)^{-1} U^T r_{*j}$$

    - Alternate both steps for a few iterations

Distributed ALS: Block ALS

- This is a high level distribution strategy for ALS
  - Relies on the fact that U and V must fit in memory (not the ratings R)
  - **Partition** Ratings by user to create R_1 and by item to create R_2 (so we have two different copies)
  - **Broadcast** the two matrices U and V
  - Using R_1 and V we can **update** U
  - Using R_2 and U we can **update** V

Item-Item matrix

- Let’s shift our attention away from the user-item matrix

- Can we build a pairwise item interaction matrix that makes sense ?
  - Co-counts: does not seem like a great idea, popular products will have a high interaction number with everything

  - Is there a quantity that can characterize the discrepancy between the observed cooccurence and the one they would have had if they were unrelated

Pointwise Mutual Information PMI

- The PMI of a pair of outcomes x and y belonging to discrete random variables X and Y

$$pmi(x, y) = \log \frac{p(x,y)}{p(x)\, p(y)}$$

  - 0 means the two variables are independent
  - Positive values means it's more "likely" to observe y if x if present (and vice versa) : $\frac{p(y|x)}{p(y)} > 1$
  - Negative values means it's less "likely" to observe y if x if present (and vice versa) : $\frac{p(y|x)}{p(y)} < 1$

Pointwise Mutual Information PMI

- For two products x and y, we can estimate empirically the PMI in a given dataset

$$pmi(x, y) = \log \frac{\#(x,y)\ \#E}{\#x\ \#y}$$

  - $\#(x, y)$ number of times x and y were interacted with by the same user (cooccurrence)
  - $\#x$ $(\#y)$ number of times x appears in the dataset
  - $\#E$ total number of all interactions in the dataset

Singular Value Decomposition

- The Singular Value Decomposition of matrix A is the triplet (U, V, Σ) that verify

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

- Regular SVD: if $\boldsymbol{A}$ is $\boldsymbol{m} \times \boldsymbol{m}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ and $\boldsymbol{\Sigma}$ are $\boldsymbol{m} \times \boldsymbol{m}$
- Truncated SVD: $\boldsymbol{A}$ is $\boldsymbol{m} \times \boldsymbol{m}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are $\boldsymbol{m} \times \boldsymbol{k}$, $\boldsymbol{\Sigma}$ is $\boldsymbol{k} \times \boldsymbol{k}$ $\quad (k \ll m)$

Singular Value Decomposition

- With Truncated SVD, in our case, each row of Matrix U will correspond to an embedding of size k of a product

- The embeddings will capture relevant characteristics of the products, and similar products will have similar embeddings

Singular Value Decomposition

- Singular Value Decomposition is very **expensive** for large matrices, both **memory** and **time** wise
- A growing area of research, *randomized numerical linear algebra,* combines probability theory with numerical linear algebra to develop fast, randomized algorithms with theoretical guarantees
- The main insight is that *randomness* is an algorithmic resource creating **efficient**, unbiased approximations of nonrandom operations.

Randomized SVD

- Consider the general problem of low-rank matrix approximation.
  - Given an $m \times n$ matrix $\boldsymbol{A}$, we want $m \times k$ and $k \times n$ matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ such that $k \ll n$ and $\boldsymbol{A} \approx \boldsymbol{BC}$.
  - To approximate this computation using randomized algorithms, Halko et al propose a two-stage computation
    - Step 1: Compute an approximate basis for the range of $\boldsymbol{A}$. We want a matrix $\boldsymbol{Q}$ with ℓ orthonormal columns ($k \leq \ell \leq n$) that captures the action of $\boldsymbol{A}$. Formally, $\boldsymbol{A} \approx \boldsymbol{QQ}^*\boldsymbol{A}$
    - Step 2: Given such a matrix $\boldsymbol{Q}$—which is much smaller than $\boldsymbol{A}$—use it to compute our desired matrix factorization.

Randomized SVD

- In the case of SVD, imagine we had access to $\boldsymbol{Q}$. Then randomized SVD is the following:
  - Given an orthonormal matrix $\boldsymbol{Q}$ such that $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}$
    - Form $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}$ ($m \times k$ matrix)
    - Compute the SVD of $\boldsymbol{B}$, i.e. $\boldsymbol{B} = \widetilde{\boldsymbol{U}}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^*$
    - Set $\boldsymbol{U} = \boldsymbol{Q}\,\widetilde{\boldsymbol{U}}$
    - Return $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^*$

Randomized SVD

- The efficiency of this algorithm comes from $\boldsymbol{B}$ being small relative to $\boldsymbol{A}$.
    - Since $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^{*}\boldsymbol{A} = \boldsymbol{Q}(\widetilde{\boldsymbol{U}}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^{*})$, setting $\boldsymbol{U} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$ produces a low-rank approximation, $\boldsymbol{A} \approx \boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^{*}$
- Note that *randomness* only occurs in Step 1 and that Step 2 is *deterministic* given $\boldsymbol{Q}$
- Thus, the algorithmic challenge is to efficiently compute $\boldsymbol{Q}$ through randomized methods

Randomized SVD

- The goal of a randomized range finder is to produce an orthonormal matrix $\boldsymbol{Q}$ with as few columns as possible such that

$$\|(A - \boldsymbol{Q}\boldsymbol{Q} * \boldsymbol{A})\| \leq \varepsilon$$

- for some desired tolerance ε

Randomized SVD

- Let ℓ be a sampling parameter indicating the number of Gaussian random vectors to draw for $\boldsymbol{\Omega}$.
- Draw an $n \times \ell$ Gaussian random matrix $\boldsymbol{\Omega}$
- Generate an $m \times \ell$ matrix $\boldsymbol{Y} = \boldsymbol{A\Omega}$
- Generate an orthonormal matrix $\boldsymbol{Q}$, e.g. using QR factorization $\boldsymbol{Y} = \boldsymbol{QR}$
- Note that the Algorithm takes as input a sampling parameter ℓ where $\ell \geq k$ ideally. Then $p = \ell - k$ is the *oversampling parameter*.

Randomized SVD

- Intuition
  - A is a $m\ x\ n$ matrix of rank exactly k
  - Let's draw k random vectors $\{\ \omega^{(i)} : i = 1,2, \ldots, k\ \}$ and form a set of $y^{(i)}\ =\ A\omega^{(i)}$
  - Because the random vectors $\{\ \omega^{(i)}\ \}$ form a linearly independant set, it is very likely that $\{\ y^{(i)}\ \}$ are also linearly independant
  - So now we have k independant vectors in the range of a matrix of size k, they then form a basis of that range
  - To produce an orthonormal basis, we just need to orthonomalize the $\{\ y^{(i)}\ \}$
  - Which is just running QR decomposision of $\boldsymbol{Y} = \boldsymbol{A\Omega}$

(Y is formed from $\left\{y^{(i)}\right\}$. and Ω is formed from $\{\ \omega^{(i)}\ \}$)

Randomized SVD

- When the singular spectrum of A decays slowly, randomized SVD can have high reconstruction error, how can we increase the rate of spectrum decay ?

- Power iterations

  - Decompose $(\boldsymbol{A}\boldsymbol{A}^{\boldsymbol{T}})^q A$ instead for which the singular value matrix is $\boldsymbol{\Sigma}^{\boldsymbol{2q+1}}$

  - The main idea is that power iterations do not modify the singular vectors, but make the spectrum decay more rapidly

Randomized SVD

- This is what we use in production at Criteo !

- We open sourced Spark-RSVD
  - Spark-RSVD is a lib to compute approximate SVD decomposition of large sparse matrices (up to 100 million rows and columns) using an iterative algorithm for speed and efficiency.

Randomized SVD

Randomized SVD