Factorization Methods in Machine Learning

Cho-Jui Hsieh

UCLA & Google

Outline

- Matrix Factorization and Recommender Systems
  - Matrix factorization
  - Extreme multi-label classification
  - Two-tower models
- Low-rankness for Efficient Deep Learning
  - Low-rank Model Compression
  - Low-rank approaches for efficient training

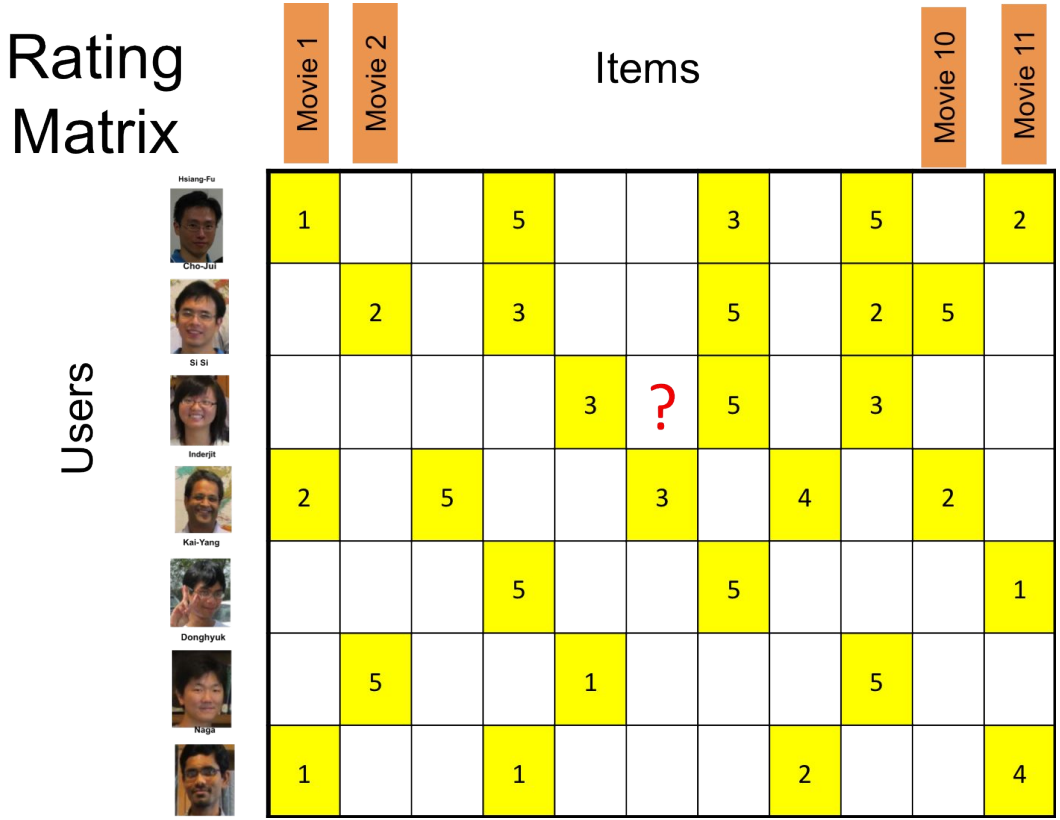
Outline

- Matrix Factorization and Recommender Systems
  - **Matrix factorization**
  - Extreme multi-label classification
  - Two-tower models
- Low-rankness for Efficient Deep Learning
  - Low-rank Model Compression
  - Low-rank approaches for efficient training

Netflix Prize (2009)

How to predict whether a user likes a movie based on rating history?

Recommender Systems



Matrix Factorization Approach

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

$$A \approx W H^T$$

Matrix Factorization Approach

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

$$A \approx W H^T$$

Matrix Factorization Approach

$$\min_{\substack{W \in R^{m \times k} \\ H \in R^{n \times k}}} \sum_{(i,j) \in \Omega} (A_{ij} - w_i^T h_j)^2 + \lambda \left(\|W\|_F^2 + \|H\|_F^2 \right),$$

- A: m-by-n matrix
- Ω: the set of observed entries
- Regularized terms to avoid over-fitting

Matrix Factorization Approach

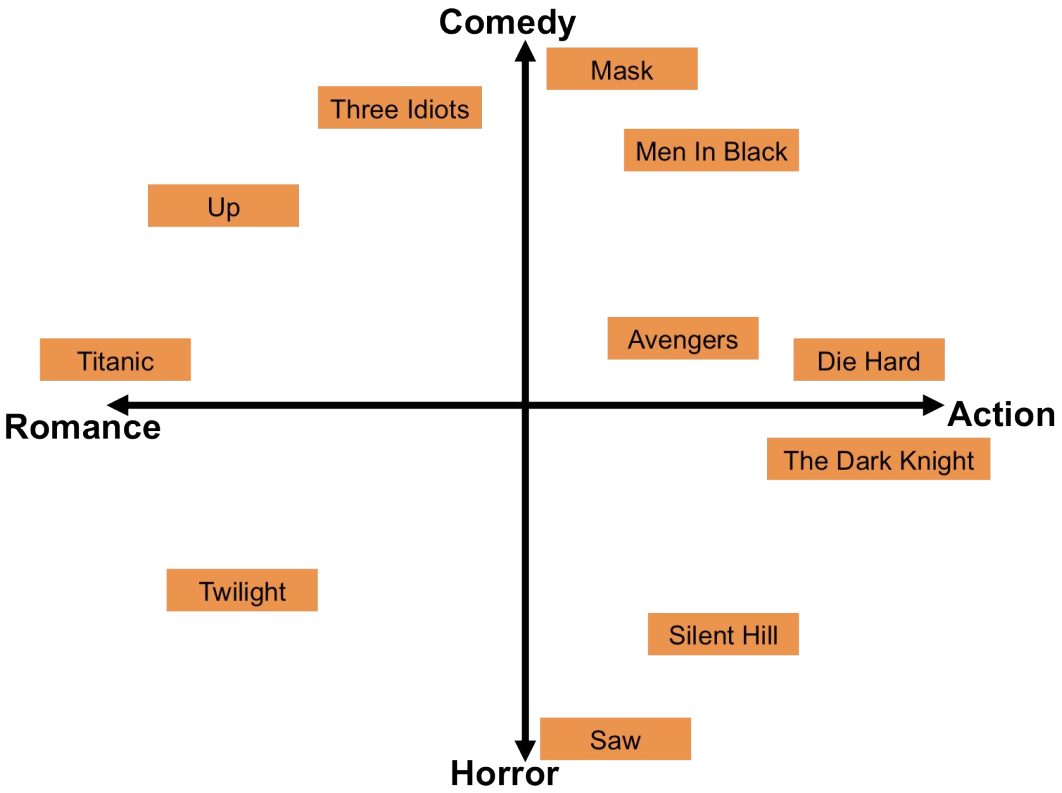
$$\min_{\substack{W \in R^{m \times k} \\ H \in R^{n \times k}}} \sum_{(i,j) \in \Omega} (A_{ij} - w_i^T h_j)^2 + \lambda \left(\|W\|_F^2 + \|H\|_F^2 \right),$$

- A: m-by-n matrix
- Ω: the set of observed entries
- Regularized terms to avoid over-fitting

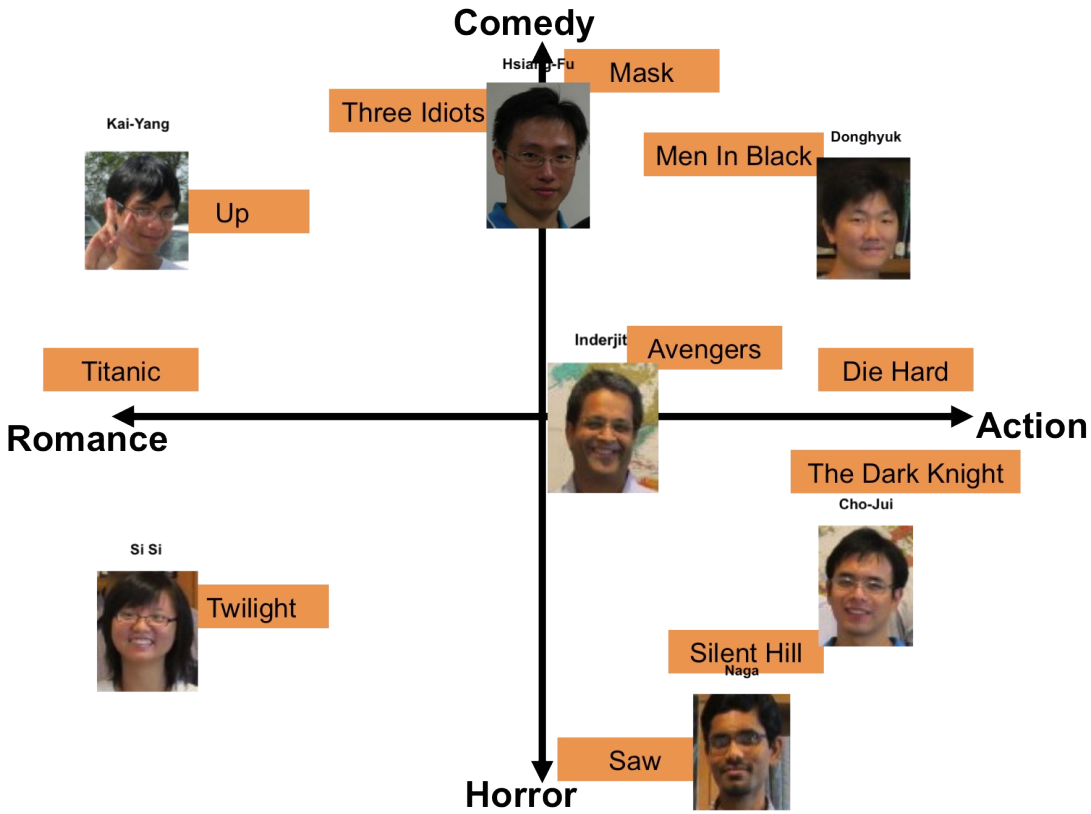
Matrix factorization maps users/items to latent feature space

- the i-th user => i-th row of W; w_i
- the j-th item => j-th row of H; h_j
- $w_i^T h_j$ Measures the interaction between i-th user and j-th item

Latent Feature Space



Latent Feature Space

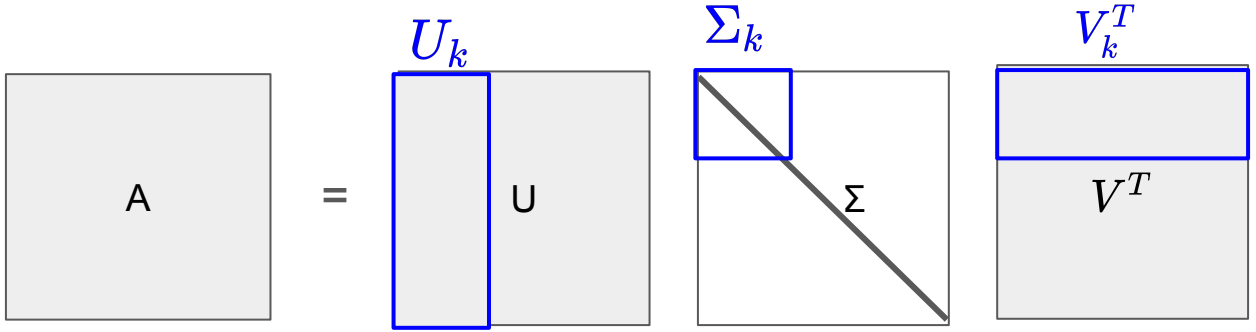


Fully observed case: connection to SVD

- Assume no regularization and A is fully observed, the problem becomes

$$\min_{W \in R^{m \times k}, H \in R^{n \times k}} \sum_{i,j} (A_{ij} - w_i^T h_j) = \|A - WH^T\|_F^2$$

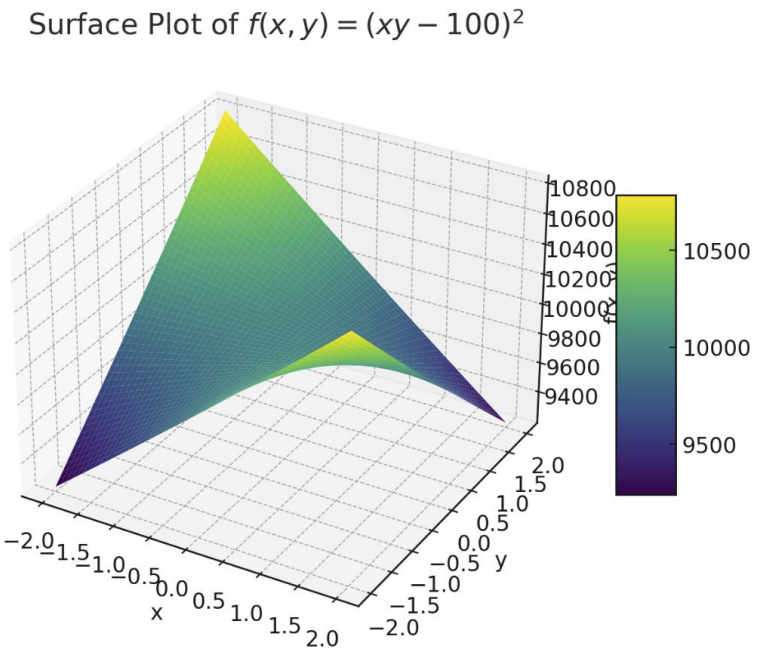
- Solution: Perform Singular Value Decomposition (SVD) on A and keeps top-k singular values/vectors



$$\approx (U_k \Sigma_k^{1/2})(V_k \Sigma_k^{1/2})^T$$

Properties of the Objective Function

- Unfortunately, no closed form solution in partially observed case
  - Need to solve the optimization problem using iterative methods
- The optimization problem is non-convex
- No guarantee to find the optimal solution without additional assumptions.



ALS: Alternating Least Squares

- Objective function:

$$\min_{W,H} \left\{ \frac{1}{2} \sum_{i,j \in \Omega} (A_{ij} - (WH^T)_{ij})^2 + \frac{\lambda}{2} \|W\|_F^2 + \frac{\lambda}{2} \|H\|_F^2 \right\} := f(W, H)$$

- Iteratively fix either H or W and optimize the other:

Input: partially observed matrix A, initial values of W, H
For $t = 1, 2, \ldots$
Fix W and update H: $H \leftarrow \text{argmin}_H f(W, H)$
Fix H and update W: $W \leftarrow \text{argmin}_W f(W, H)$

ALS: Alternating Least Squares

- Define $\Omega_j := \{(i,j) \in \Omega\}$
- The subproblem:

$$\underset{H}{\operatorname{argmin}} \frac{1}{2} \sum_{i,j \in \Omega} (A_{ij} - (WH^T)_{ij})^2 + \frac{\lambda}{2} \|H\|_F^2$$

$$= \sum_{j=1}^{n} \Bigg(\underbrace{\frac{1}{2} \sum_{i \in \Omega_j} (A_{ij} - \boldsymbol{w}_i^T \boldsymbol{h}_j)^2 + \frac{\lambda}{2} \|\boldsymbol{h}_j\|^2}_{\text{ridge regression problem}} \Bigg)$$

- n ridge regression problems, each with k variables
- Easy to parallel (n independent ridge regressions)

Stochastic Gradient Descent (SGD)

- Decompose the problem into Ω components:

$$f(W,H) = \frac{1}{2|\Omega|}\sum_{i,j\in\Omega}(A_{ij} - \boldsymbol{w}_i^T\boldsymbol{h}_j)^2$$
$$= \frac{1}{|\Omega|}\sum_{i,j\in\Omega}\Big(\underbrace{\frac{1}{2}(A_{ij} - \boldsymbol{w}_i^T\boldsymbol{h}_j)^2}_{f_{i,j}(W,H)}\Big)$$

- The gradient of each component:

$$\nabla_{\boldsymbol{w}_i} f_{i,j}(W,H) = (\boldsymbol{w}_i^T\boldsymbol{h}_j - A_{ij})\boldsymbol{h}_j$$
$$\nabla_{\boldsymbol{h}_j} f_{i,j}(W,H) = (\boldsymbol{w}_i^T\boldsymbol{h}_j - A_{ij})\boldsymbol{w}_i$$

- Each iteration: randomly pick an (i,j) pair and conduct SGD update

Stochastic Gradient Descent

$$\begin{pmatrix} h_1 & \boxed{h_2} & h_3 \end{pmatrix} \qquad \begin{pmatrix} h_1 & h_2; & \boxed{h_3} \end{pmatrix}$$

$$\begin{pmatrix} \boxed{w_1^T} \\ w_2^T \\ w_3^T \end{pmatrix} \begin{pmatrix} A_{11} & \boxed{A_{12}} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \qquad \begin{pmatrix} w_1^T \\ \boxed{w_2^T} \\ w_3^T \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & \boxed{A_{23}} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

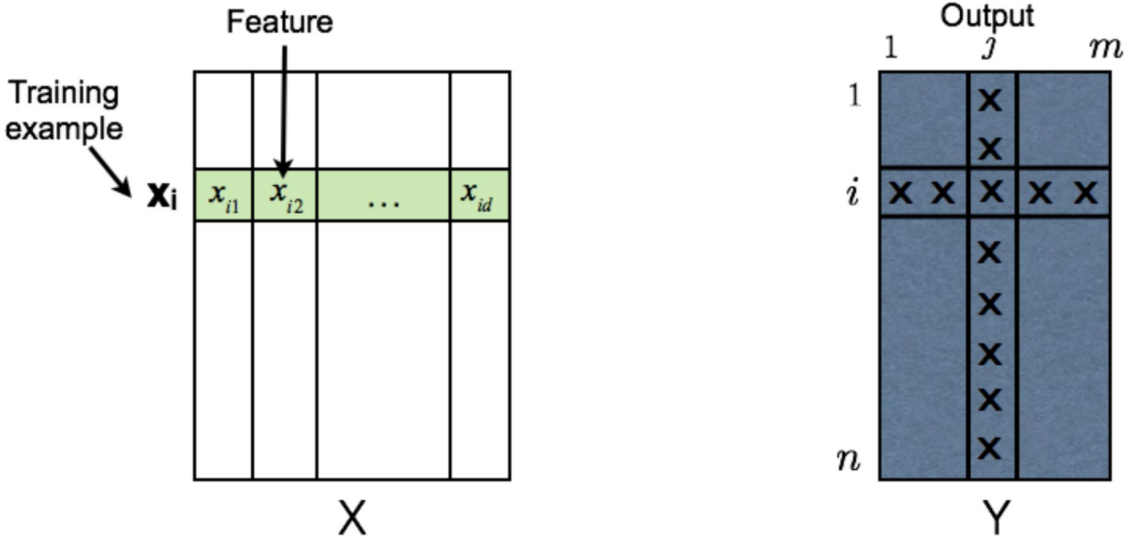
Non-trivial to parallelize ->
several work tries to parallelize this around 2012–2016

Outline

- Matrix Factorization and Recommender Systems
  - Matrix factorization
  - **Extreme multi-label classification**
  - Two-tower models
- Low-rankness for Efficient Deep Learning
  - Low-rank Model Compression
  - Low-rank approaches for efficient training

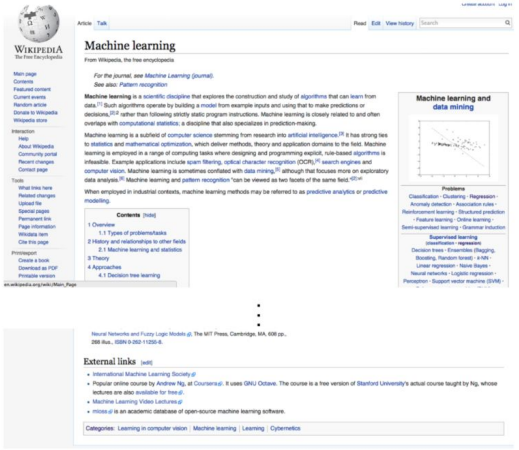
Multi-label classification

- Multi-label classification:
  Input data x_i is associated with m outputs, $y_i = (y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(m)})$
  each $y_i^{(j)} \in \{0, 1\}$

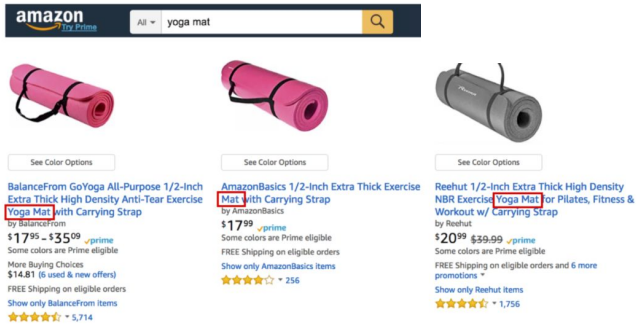


Extreme Multi-label Classification (XMC)

- Number of labels m is extremely large
- Each label could mean a token/item/product …
- Equivalent to recommendation system with query features (but not item features)



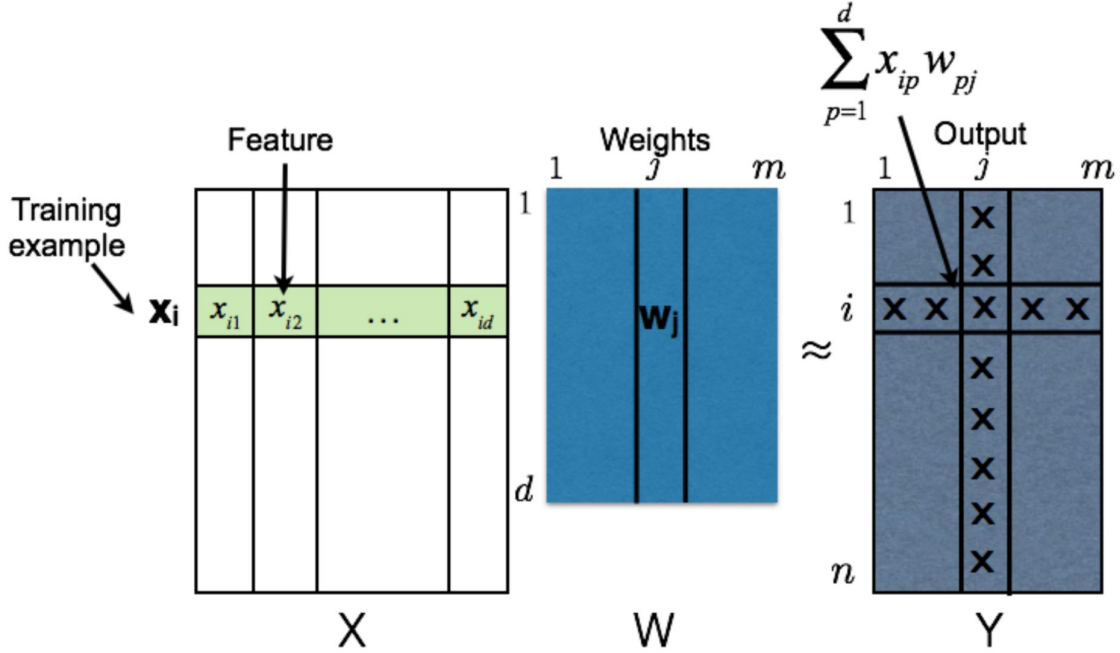
Wikipedia tagging:
page -> tags



Product search:
query -> product

Naive Linear Approach

- Simple linear predictive model
- Estimate model $W \in R^{d \times m}$, each column is a linear model to predict a particular label



Naive Linear Approach

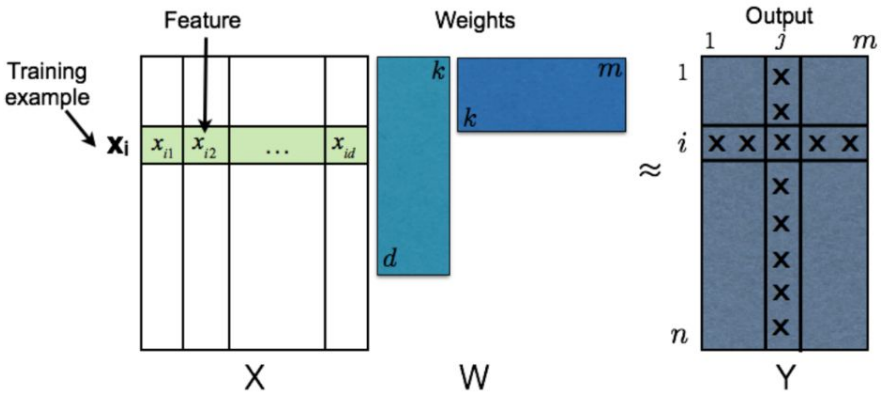
- Each w_j (model for j-th label) can be solved independently
  - Can be easily parallelized
- However, hard to scale to a large number of labels

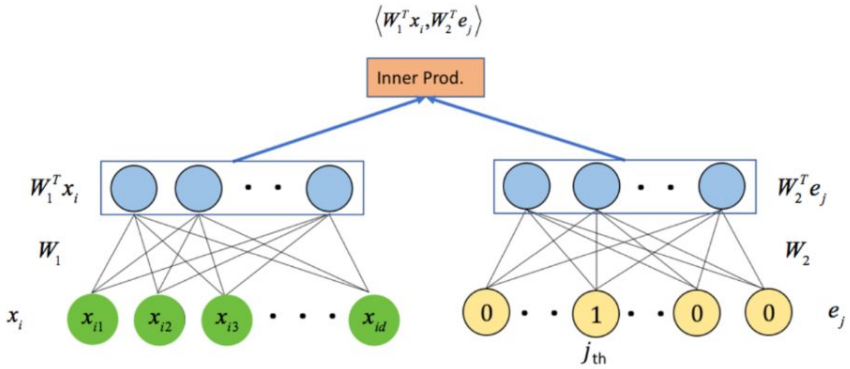
Naive Linear Approach

- Each w_j (model for j-th label) can be solved independently
  - Can be easily parallelized
- However, hard to scale to a large number of labels
- Consider Wikipedia data with 500K labels
  - 1.5M samples, 0.5M labels, 2.5M features (sparse TF-IDF)
- Time complexity:
  - Need to train 0.5M linear models
  - Take 1.8 months with 16-way parallelism
- Space complexity: O(dm)
  - 5TB (5000 billion) model size for this Wikipedia problem
  - Hard to serve in practice

Low-rank Approach

- Exploit correlations between outputs Y, where Y≅XW
- Assume W has a low-rank structure





Low-rank Approach

- Objective for low-rank multi-output regression

$$\min_{W:\text{rank}(W)\leq k} \|Y - XW\|_F^2$$

Low-rank Approach

- Objective for low-rank multi-output regression

$$\min_{W:\text{rank}(W)\leq k} \|Y - XW\|_F^2$$

- Closed form solution exists!

$$V_X \Sigma_X^{-1} M_k = \arg \min_{Z:\text{rank}(Z)\leq k} \|Y - XZ\|_F^2, \qquad (3)$$

where $X = U_X \Sigma_X V_X^T$ *is the thin SVD decomposition of* X, *and* M_k *is the rank-k truncated SVD of* $M \equiv U_X^T Y$.

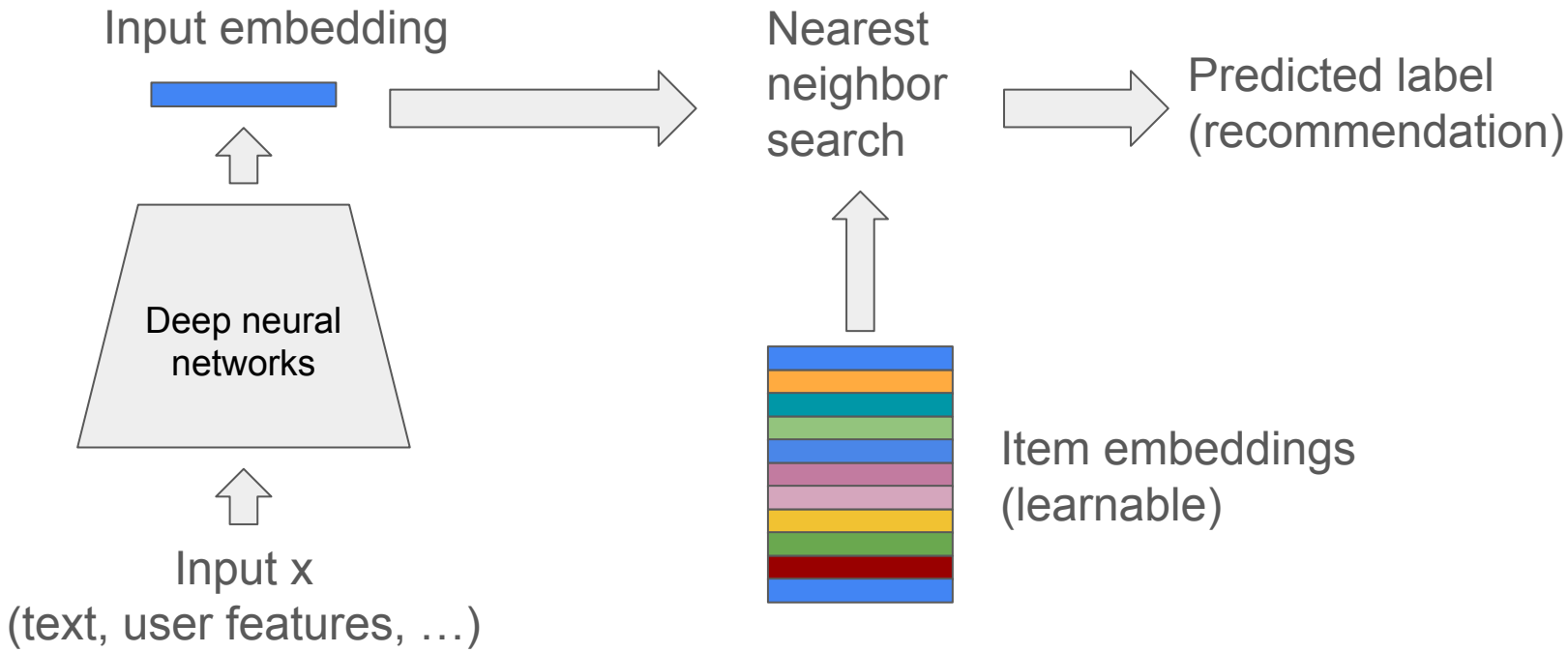
- Can also be solved efficiently by alternating minimization

[Yu, Jain, Kar, Dhillon] Large-scale Multi-label Learning with Missing Labels. ICML 2014.

Low-rank Approach

- Pros:
  - Reduced time and memory complexity by a factor of L/k
    (L: number of labels; k: rank)
  - Able to exploit correlations between labels
- Cons:
  - Restricted representation power
    (linear low-rank model may be insufficient)
  - Observation:
    - k=L achieves the best result (but requires lot of computation)

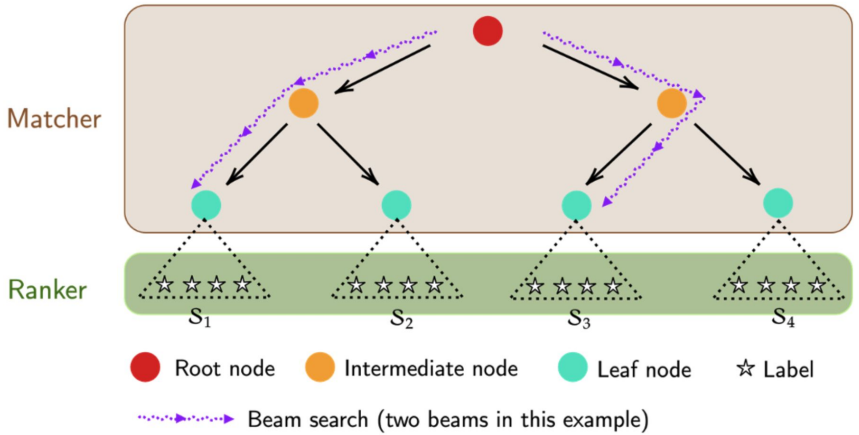
Replace linear projection by neural networks



Commonly used in large-scale advertising systems

Hierarchical label partition

- Construct a label partition tree
  - Each label's feature can be obtained by the mean of corresponding x
  - Conduct kmeans clustering
- Train a matcher to route x to one or few clusters
- Train a ranker to rank labels within clusters



Hierarchical Linear Model

- A linear classifier on each node
  - Inference cost: O(log(m) + S) linear predictions (S: number of labels on leaf node)
- Final ranker is linear models trained with positive and selected negative samples
  - Reduced training cost
- Can be extended to neural network models (NN on each node)

[Prabhu, Kag, Harsola, Agrawal, Varma] Parabel: Partitioned Label Trees for Extreme Classification with Application to Dynamic Search Advertising. WWW 2018.
[Yu, Zhong, Zhang, Chang, Dhillon] PECOS: Prediction for Enormous and Correlated Output Spaces. JMLR 2023.

Some Comparisons on XMC Benchmarks

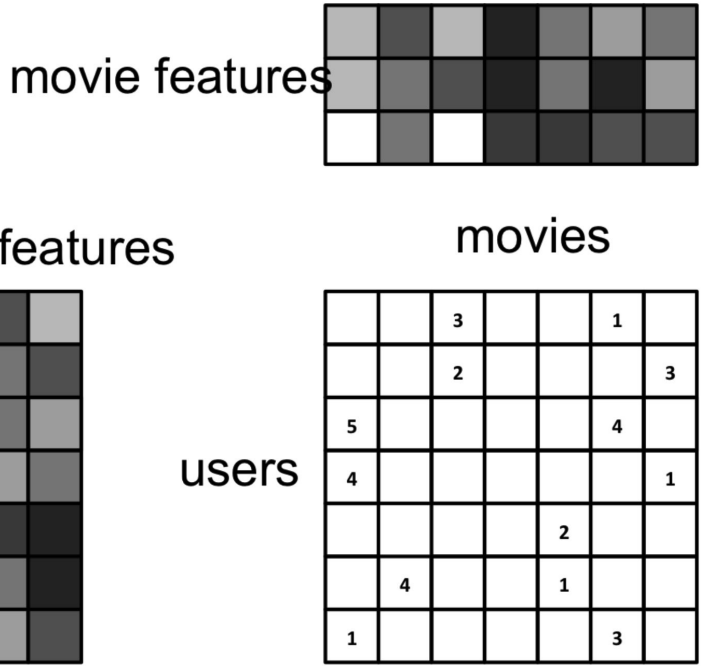
Method	P@1	P@3	P@5	PSP@1	PSP@3	PSP@5	P@1	P@3	P@5	PSP@1	PSP@3	PSP@5
	Amazon-670K						LF-AmazonTitles-131K					
DiSMEC$^{(s)}$	44.70	39.70	36.10	27.80	30.60	34.20	35.14	23.88	17.24	25.86	32.11	36.97
Parabel$^{(s)}$	44.89	39.80	36.00	25.43	29.43	32.85	32.60	21.80	15.61	23.27	28.21	32.14
XR-Linear$^{(s)}$	45.36	40.35	36.71	-	-	-	-	-	-	-	-	-
Bonsai$^{(s)}$	45.58	40.39	36.60	27.08	30.79	34.11	34.11	23.06	16.63	24.75	30.35	34.86
Slice$^{(d)}$	33.15	29.76	26.93	20.20	22.69	24.70	30.43	20.50	14.84	23.08	27.74	31.89
Astec$^{(d)}$	47.77	42.79	39.10	32.13	35.14	37.82	37.12	25.20	18.24	29.22	34.64	39.49
GLaS$^{(d)}$	46.38	42.09	38.56	**38.94**	**39.72**	41.24	-	-	-	-	-	-
AttentionXML$^{(d)}$	47.58	42.61	38.92	30.29	33.85	37.13	32.55	21.70	15.64	23.97	28.60	32.57
LightXML$^{(d)}$	49.10	43.83	39.85	-	-	-	38.49	26.02	18.77	28.09	34.65	39.82
XR-Transformer$^{(s+d)}$	50.11	44.56	40.64	36.16	38.39	40.99	38.42	25.66	18.34	29.14	34.98	39.66
Overlap-XMC$^{(s+d)}$	50.70	45.40	41.55	36.39	39.15	**41.96**	-	-	-	-	-	-
ELIAS $^{(d)}$	50.63	45.49	41.60	32.59	36.44	39.97	39.14	26.40	19.08	30.01	36.09	41.07
ELIAS ++$^{(s+d)}$	**53.02**	**47.18**	**42.97**	34.32	38.12	41.93	**40.13**	**27.11**	**19.54**	**31.05**	**37.57**	**42.88**
	Wikipedia-500K						Amazon-3M					
DiSMEC$^{(s)}$	70.21	50.57	39.68	31.20	33.40	37.00	47.34	44.96	42.80	-	-	-
Parabel$^{(s)}$	68.70	49.57	38.64	26.88	31.96	35.26	47.48	44.65	42.53	12.82	15.61	17.73
XR-Linear$^{(s)}$	68.12	49.07	38.39	-	-	-	47.96	45.09	42.96	-	-	-
Bonsai$^{(s)}$	69.20	49.80	38.80	-	-	-	48.45	45.65	43.49	13.79	16.71	18.87
Slice$^{(d)}$	62.62	41.79	31.57	24.48	27.01	29.07	-	-	-	-	-	-
Astec$^{(d)}$	73.02	52.02	40.53	30.69	36.48	40.38	-	-	-	-	-	-
GLaS$^{(d)}$	69.91	49.08	38.35	-	-	-	-	-	-	-	-	-
AttentionXML$^{(d)}$	76.95	58.42	46.14	30.85	39.23	44.34	50.86	48.04	45.83	15.52	18.45	20.60
LightXML$^{(d)}$	77.78	58.85	45.57	-	-	-	-	-	-	-	-	-
XR-Transformer$^{(s+d)}$	79.40	59.02	46.25	**35.76**	42.22	46.36	54.20	50.81	48.26	**20.52**	**23.64**	**25.79**
Overlap-XMC$^{(s+d)}$	-	-	-	-	-	-	52.70	49.92	47.71	18.79	21.90	24.10
ELIAS $^{(d)}$	79.00	60.37	46.87	33.86	42.99	47.29	51.72	48.99	46.89	16.05	19.39	21.81
ELIAS ++$^{(s+d)}$	**81.26**	**62.51**	**48.82**	35.02	**45.94**	**51.13**	**54.28**	**51.40**	**49.09**	15.85	19.07	21.52

Outline

- Matrix Factorization and Recommender Systems
  - Matrix factorization
  - Extreme multi-label classification
  - **Two-tower models**
- Low-rankness for Efficient Deep Learning
  - Low-rank Model Compression
  - Low-rank approaches for efficient training

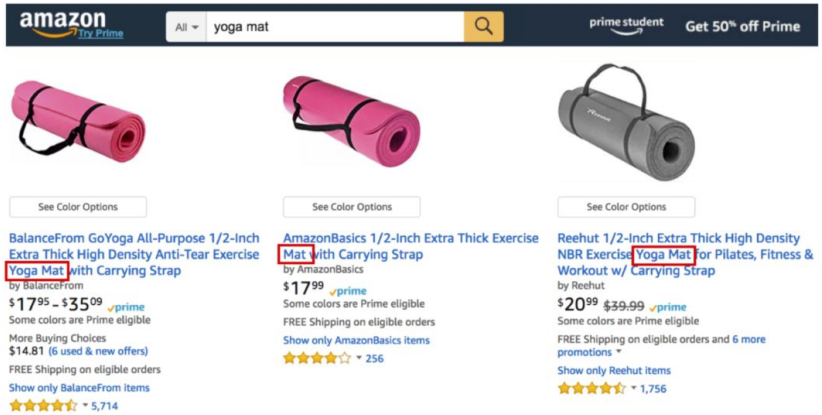
Matrix Completion with Two-sided Features

- Given: rating matrix $A \in R^{n_1 \times n_2}$, user features $X \in R^{n_1 \times d_1}$, item features $Y \in R^{n_2 \times d_2}$
- Goal: predict unknown ratings

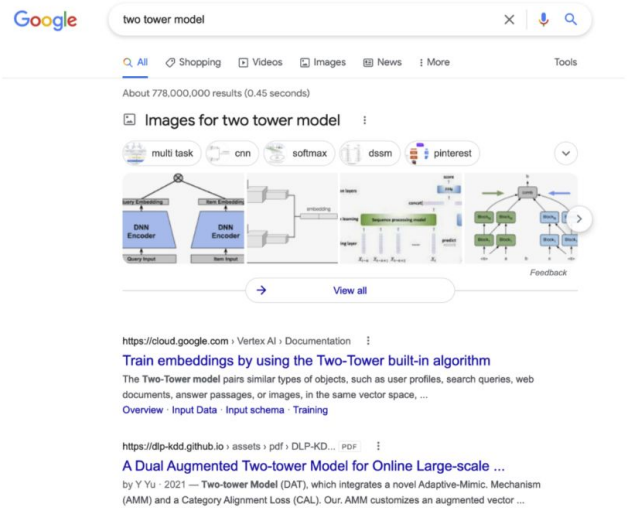


Applications in Search

- x: query
- y: documents/products/items



Amazon search



Google search

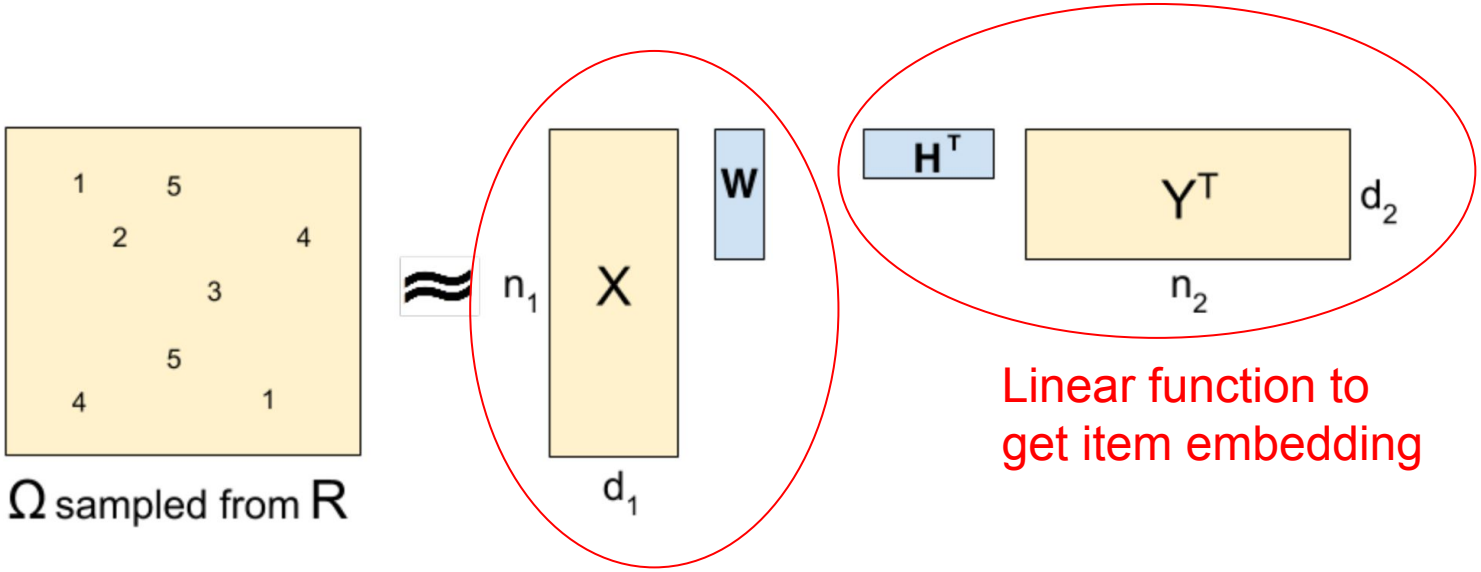
Inductive Matrix Completion (IMC)

- A popular approach to incorporate features in matrix completion.
- Given row features $X \in R^{n_1 \times d_1}$, column features $Y \in R^{n_2 \times d_2}$, the IMC objective is:

$$\min_{\substack{W \in \mathbb{R}^{d_1 \times k} \\ H \in \mathbb{R}^{d_2 \times k}}} \sum_{(i,j) \in \Omega} (\boldsymbol{x}_i^T W H^T \boldsymbol{y}_j - R_{ij})^2 + \frac{\lambda}{2} \|W\|_F^2 + \frac{\lambda}{2} \|H\|_F^2$$

  - Wx_i : k-dimensional embedding for user i
  - Hy_j : k-dimensional embedding for user j
  - Inner product in the k-dimensional embedding space => prediction value

Inductive Matrix Completion

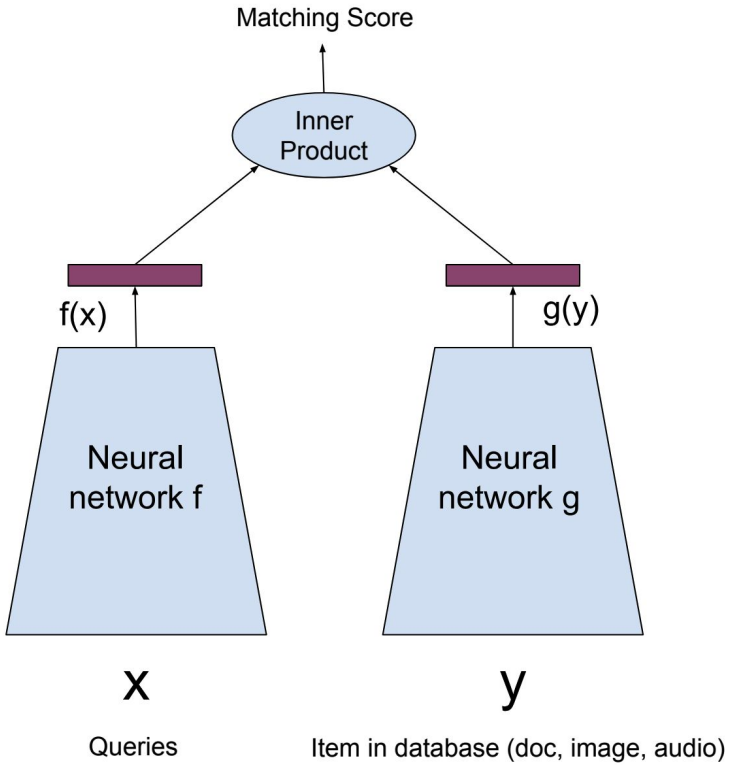


Two tower model with linear encoders

Dual encoder model (Two tower model)

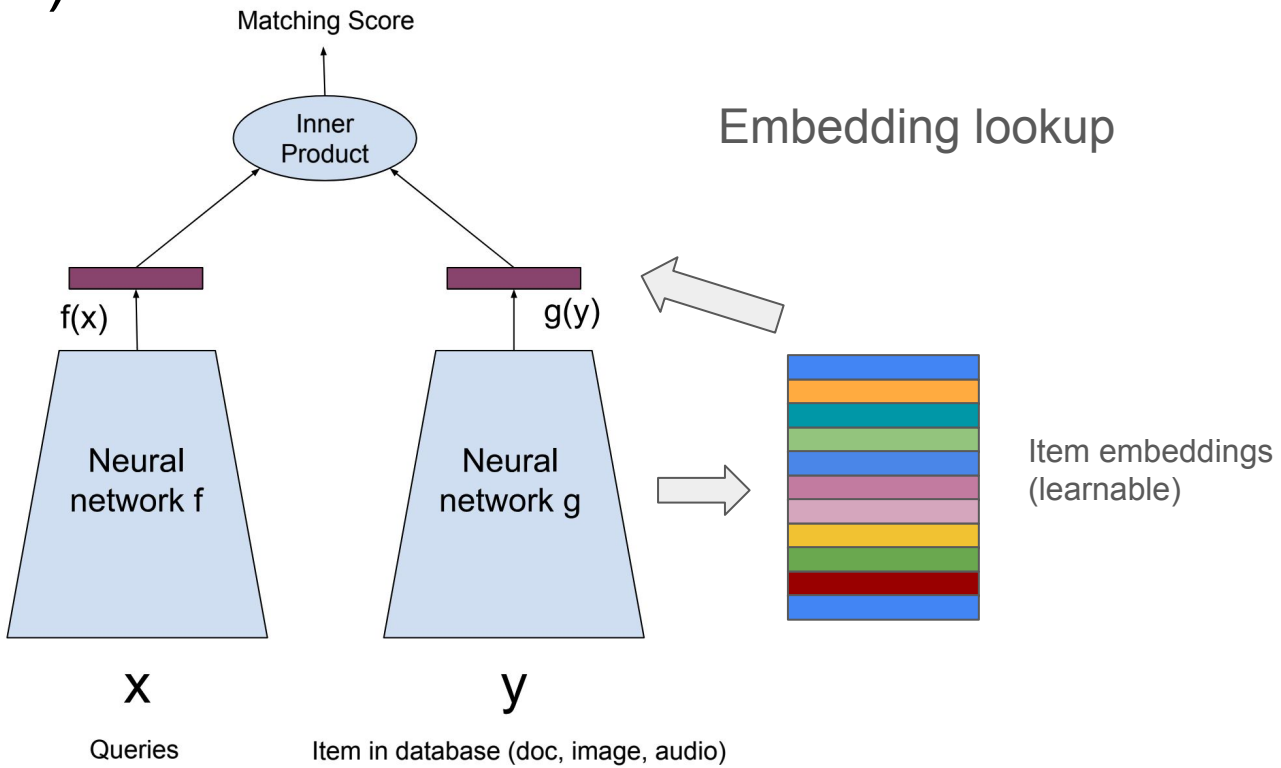
- In inductive matrix completion
  - Given x_i (user i) and y_j (item j)
  - Project to a k-dimensional latent space (by linear functions)
  - Predict rating by inner product $(W\boldsymbol{x}_i)^T H\mathbf{y}_j$
- In general, we can replace the linear function by any nonlinear function
  - $\boldsymbol{x}_i \to f(\boldsymbol{x}_i),\ \mathbf{y}_j \to g(\mathbf{y}_j)$
  - Predict rating by $f(\boldsymbol{x}_i)^T g(\mathbf{y}_j)$

Deep two tower model



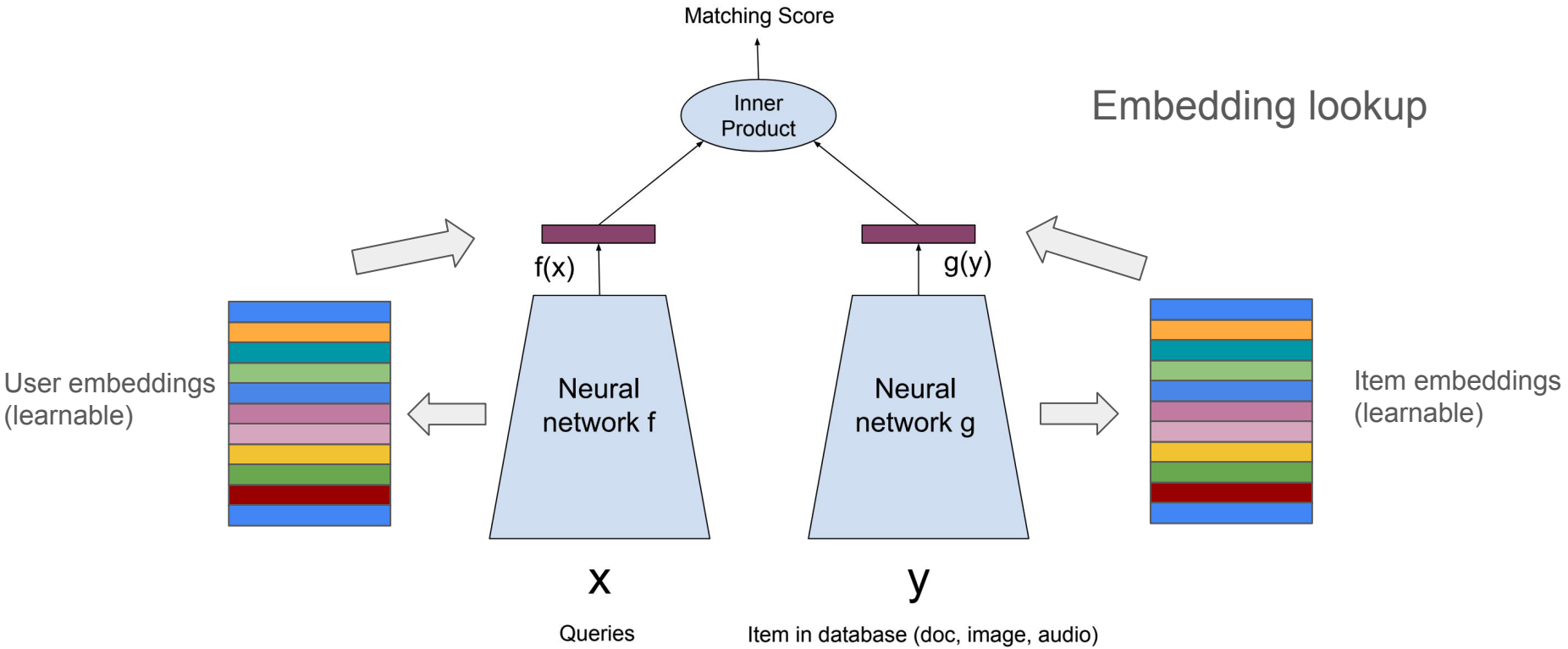
Extreme Multilabel Classification (XMC)

If y_j is item id (one-hot vector) and g is linear
=> equivalent to XMC (with one-sided feature)



Matrix Completion

If both x_i and y_j are id (one-hot vector) and f, g are linear
=> equivalent to original matrix completion (no feature)



A Unified View

- Most recommender systems follow the “encoder”-based structure
  - Map both query and items to a latent space
    - By embedding lookup (no feature)
    - Or by linear mapping (matrix factorization), DNN (deep recommender systems)
  - Find the nearest neighbor in the latent space (efficient ANNS algorithms required)
- Embedding lookup v.s. DNN encoders
  - Embedding lookup: better “memorization”
  - DNN encoder: better “generalization”
  - **How to choose or can we combine both?**
- Can we get rid of the “encoder”-based structure?
  - **LLM-based recommendation?**

Challenges in Two Tower Models

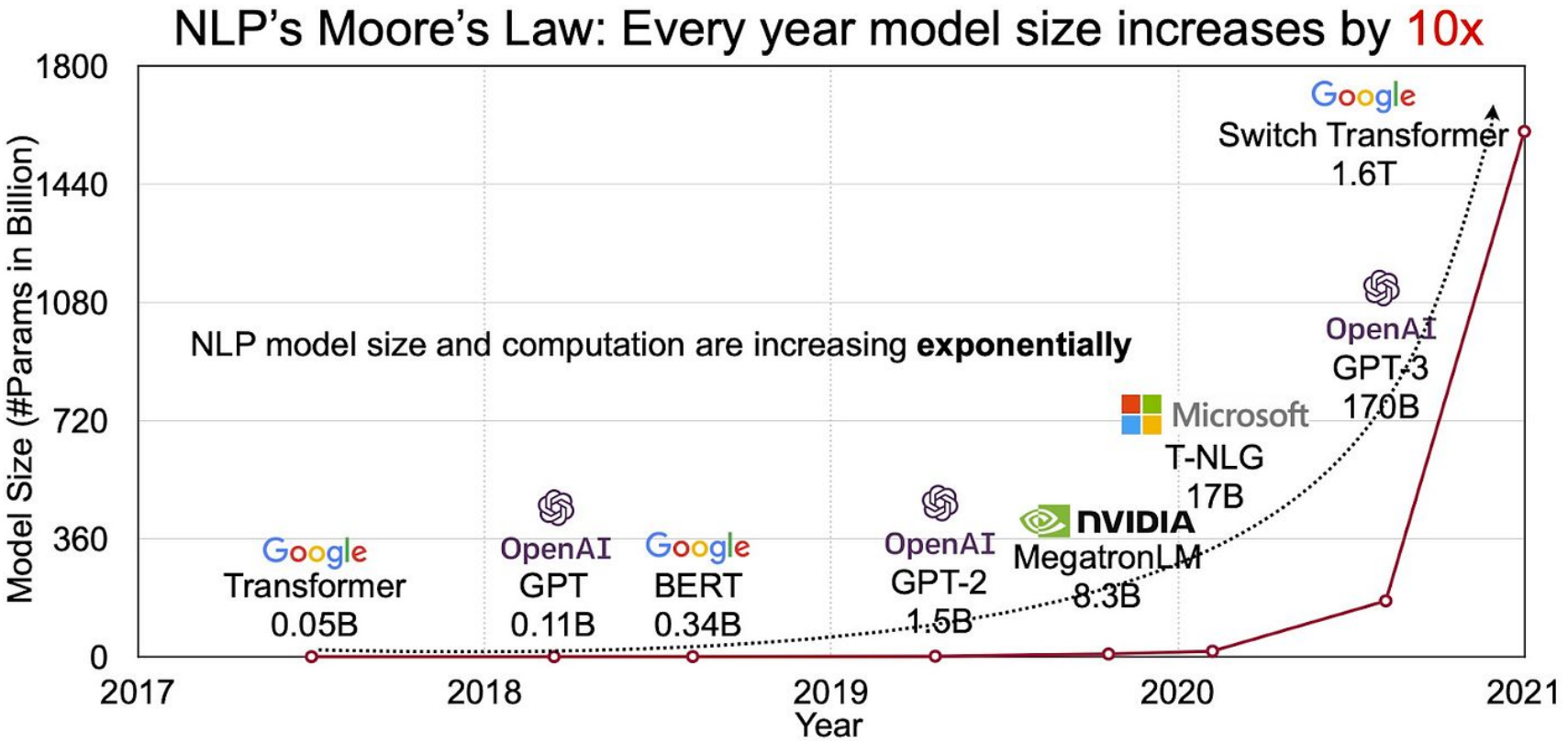
- Training:
  - Negative sampling: how to choose negative samples for an input
  - Large embedding table: large memory for training
- Inference:
  - How to find the nearest embedding in the latent space?
- Maximum Inner Product Search (MIPS) or Approximate Nearest Neighbor Search (ANNS) problem:
  - Given a database $v_1, \ldots, v_n$
  - For a query u, find v_i that maximizes $u^T v_i$ (or minimize $\|u - v_i\|$)
  - Brute Force approach: O(nd) time, n can be trillions
  - Two popular approaches: Quantization methods and graph-based traversal

Low-rankness for Efficient Deep Learning

Outline

- Matrix Factorization and Recommender Systems
  - Matrix factorization
  - Extreme multi-label classification
  - Two-tower models
- Low-rankness for Efficient Deep Learning
  - **Low-rank Model Compression**
  - Low-rank approaches for efficient training

Model Size is Exponentially Growing in DNN

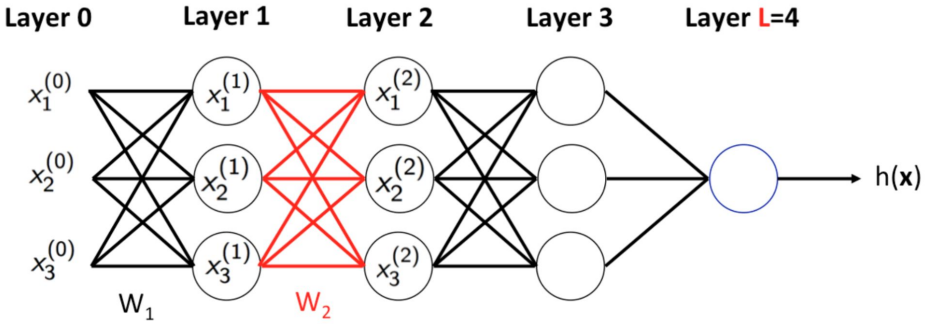


Model Size is Important for Deploying ML Models

- Need model compression for
  - Run models on device
  - Reduce the serving cost

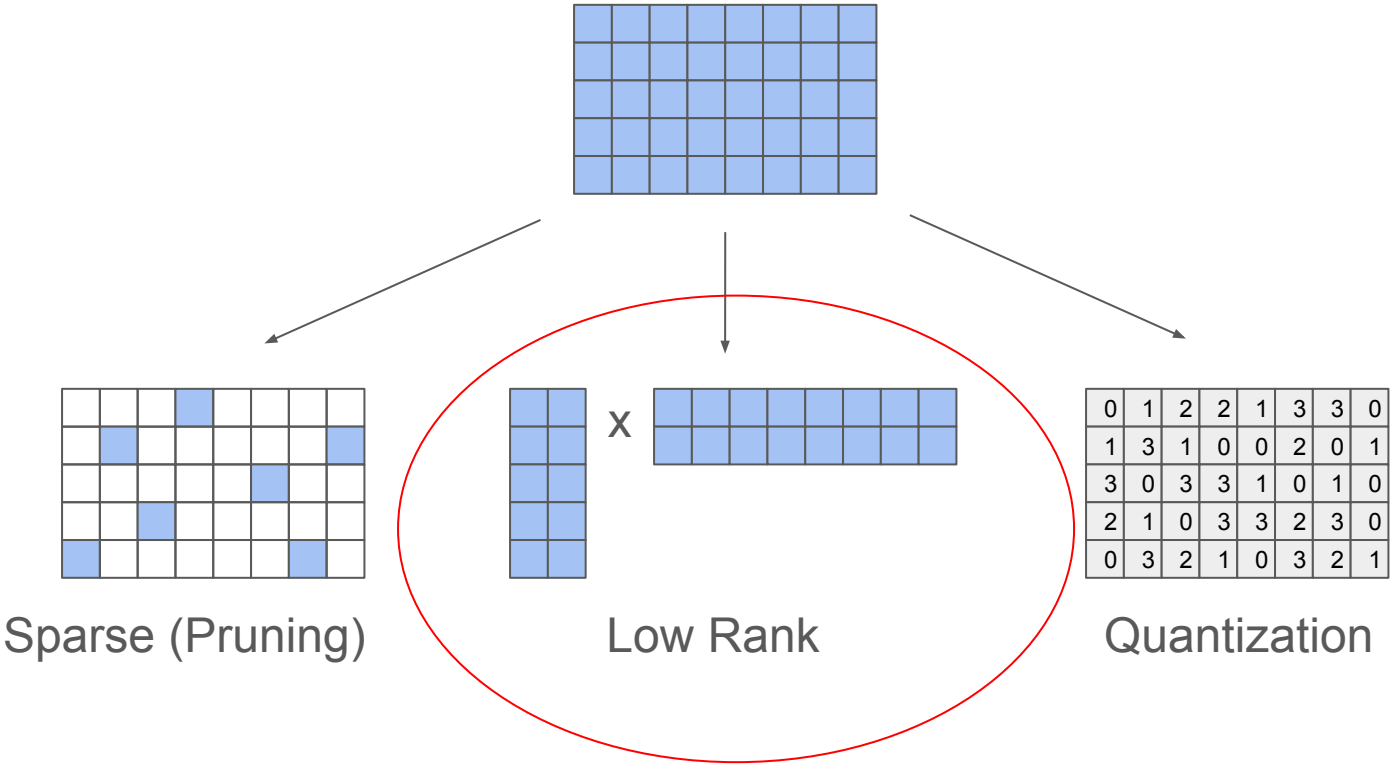
Model Compression for Neural Networks

- Neural network weights: matrices and tensors



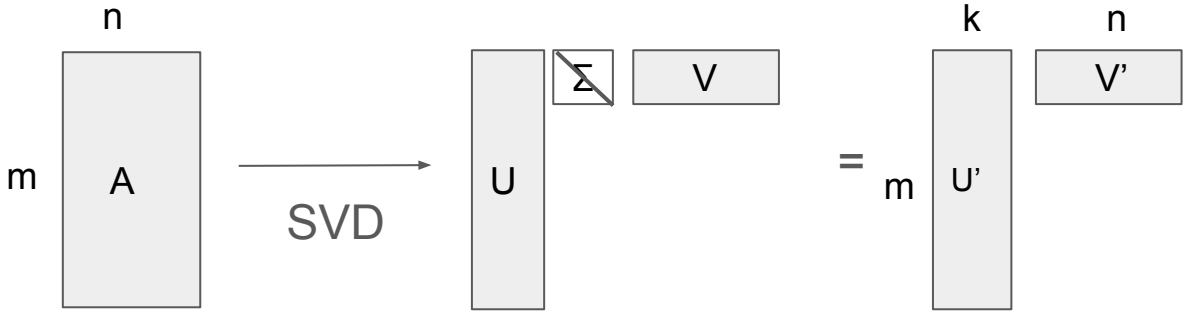
- In LLM, each weight matrix can be very large (e.g., 8192 x 32768 = 268 million)

Algorithms for Neural Network Compression



Low Rank Model Compression

- Perform Singular Value Decomposition (SVD) on the weight matrix

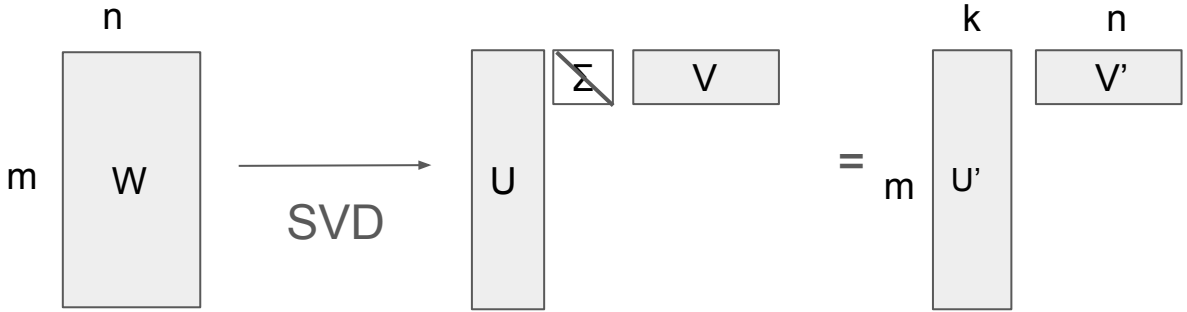


$$U' = U\Sigma^{1/2}$$
$$V' = \Sigma^{1/2}V$$

- Model size: mn -> mk + kn
- Memory reduced by a factor of ~min(m, n)/k
- m=1000, n=1000, k=10 -> model size reduced from 1 million to 20K

Low Rank Model Compression

- Perform Singular Value Decomposition (SVD) on the weight matrix



$$U' = U\Sigma^{1/2}$$
$$V' = \Sigma^{1/2}V$$

- Theoretical guarantee: SVD guarantees that

$$U', V' = \arg\min_{U \in R^{n \times k}, V \in R^{k \times n}} \|W - UV\|_F$$

- **Can we do better?**

Is SVD Optimal?

- SVD is optimal if we want to approximate the weight
- But what we care is to approximate the input -> output mapping (the prediction)
- Assume input of the layer is $x_1, x_2, \ldots, x_n$
- We want to approximate $Wx_1, Wx_2, \ldots, Wx_n$
- Even if W is not low-rank, it's possible to get perfect approximation!

A toy example

- W is full rank but x is in a low-dimensional space

$$W = \begin{bmatrix} 7 & 0 & 2 & 3 & 1 \\ 9 & 6 & 7 & 5 & 0 \\ 6 & 1 & 8 & 0 & 3 \\ 4 & 3 & 2 & 1 & 4 \\ 1 & 2 & 2 & 1 & 2 \end{bmatrix}, \quad x \in \text{span}\left(\begin{bmatrix} 2 \\ 2 \\ 5 \\ 5 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 6 \end{bmatrix} \right).$$

- There exists a low-rank approximate to preserve Wx exactly

$$\underbrace{\begin{bmatrix} 7 & 0 & 2 & 3 & 1 \\ 9 & 6 & 7 & 5 & 0 \\ 6 & 1 & 8 & 0 & 3 \\ 4 & 3 & 2 & 1 & 4 \\ 1 & 2 & 2 & 1 & 2 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 5 & 2 \\ 5 & 2 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}}_{x}$$

$$= \underbrace{\begin{bmatrix} 43 & 23 \\ 90 & 39 \\ 66 & 41 \\ 45 & 37 \\ 29 & 21 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} -1 & -1 & 0.5 & 0.5 & 0 \\ -0.5 & 0 & 0 & 0 & 0.25 \end{bmatrix}}_{V^T} \underbrace{\begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 5 & 2 \\ 5 & 2 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}}_{x},$$

Is SVD Optimal?

- Assume input of the layer is $x_1, x_2, \ldots, x_n$
- We want to approximate $Wx_1, Wx_2, \ldots, Wx_n$
- Objective function:

$$U', V' = \arg\min_{U \in R^{m \times k}, V \in R^{k \times n}} \sum_{i=1}^{n} \|Wx_i - UVx_i\|^2$$

In matrix form (assume $X = [x_1, x_2, \ldots, x_n]$)

$$U', V' = \arg\min_{U \in R^{m \times k}, V \in R^{k \times n}} \|WX - UVX\|_F^2$$

Closed Form Solution

- We can solve the problem in closed form

$$U', V' = \arg\min_{U \in R^{m \times k}, V \in R^{k \times n}} \|WX - UVX\|_F^2$$

Closed Form Solution

- We can solve the problem in closed form

$$U', V' = \arg\min_{U \in R^{m \times k}, V \in R^{k \times n}} \|WX - UVX\|_F^2$$

- We have seen the same formulation in inductive matrix completion in the first part of lecture (Yu et al., 2014)

$$V_X \Sigma_X^{-1} M_k = \arg \min_{Z:\text{rank}(Z) \leq k} \|Y - XZ\|_F^2, \qquad (3)$$

where $X = U_X \Sigma_X V_X^T$ is the thin SVD decomposition of X, and M_k is the rank-k truncated SVD of $M \equiv U_X^T Y$.

Data-aware Low-rank Compression (Drone)

- To compress a weight matrix W of layer l:
  - Forward propagate training samples through layer 1 to l-1:

$$X = [x_1, x_2, \ldots, x_n]$$

  - Solve for the optimal U', V'

$$U', V' = \arg\min_{U \in R^{n \times k}, V \in R^{k \times n}} \|WX - UVX\|_F^2$$

  - Approximate W by U'V'

- Repeat this for each layer to compress the whole network

[Chen, Yu, Dhillon, Hsieh] Data-aware Low-Rank Compression for Large NLP Models.

Performance on BERT-base compression

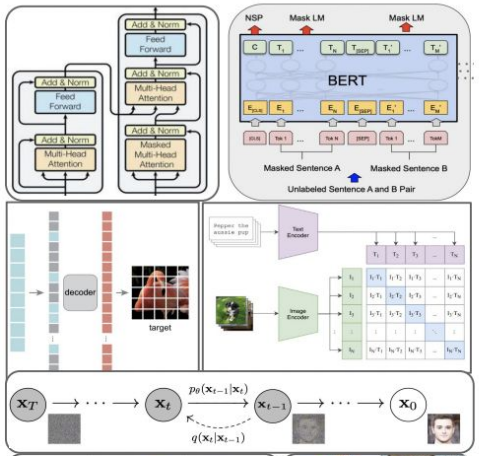
Methods	MNLI	QQP	SST-2	QNLI	MRPC	RTE	CoLA	STS-B
Original	84.3	90.9	92.3	91.4	89.5	72.6	53.4	87.8
SVD	74.4	50.8	73.1	52.2	63.8	47.3	12.4	33.6
DRONE	82.0	89.4	90.0	88.5	86.7	70.0	52.5	85.8
DRONE-Retrain	82.6	90.1	90.8	89.3	88.0	71.5	53.2	87.8
CPU Speedup Ratio	1.60x	1.25x	1.64x	1.20x	1.92x	1.31x	1.33x	1.52x
GPU Speedup Ratio	1.28x	1.38x	1.45x	1.28x	1.56x	1.33x	1.29x	1.57x

Outline

- Matrix Factorization and Recommender Systems
  - Matrix factorization
  - Extreme multi-label classification
  - Two-tower models
- Low-rankness for Efficient Deep Learning
  - Low-rank Model Compression
  - **Low-rank approaches for efficient training**

Parameter-efficient Fine-tuning

- Fine-tuning a large model (e.g., LLM) on some tasks
- (GPU) Memory cost proportional to model size
- Requires large number of GPUs for fine-tuning

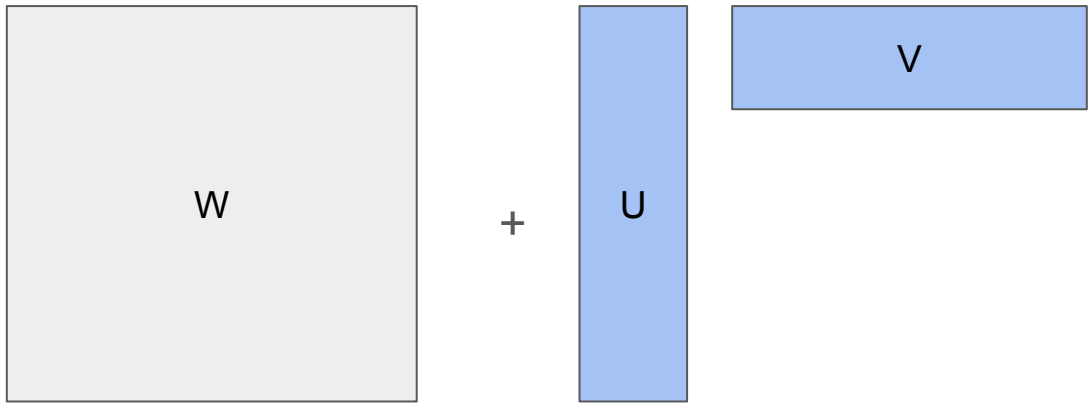


Task

finetune

LoRA finetuning

- Add a low-rank component to a weight matrix
- Only update the low-rank component during fine-tuning
- LoRA: Low-Rank Adaptation of Large Language Models



[Hu, Shen, Wallis, Allen-Zhu, Li, Wang, Wang, Chen] LoRA: Low-Rank Adaptation of Large Language Models. ICLR 2022.

Benefits

- Efficiency (memory and time):
  - Reduce the learnable parameter size significantly (optimizer states)
  - Improve the speed slightly due to reduced size of learnable parameters
- Sample efficiency:
  - Less learnable parameters -> can be trained with less samples
- Model sharing:
  - Can finetune multiple models based on the same LLM but different LoRA factors
  - Reduced memory cost when serving all those models together
    - LLM: 70B, LoRA1: 1B, LoRA2: 1B, …, LORA10: 1B
    - Overall serving size: 80B (instead of 70*10 B)

Results on Roberta/Deberta

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

LoRA for Pretraining

- Several recent work trying LoRA-based methods for memory efficient pretraining
- Cannot fix the same low-rank factor through the whole pretraining stage
  - Suboptimal performance due to reduced learnable parameters
- Idea: periodically add back the low-rank factor to the weights, and restart a new low-rank subspace

[Lialin, Shivagunde, Muckatira, Rumshisky] ReLoRA: Low-Rank Adaptation of Large Language Models. ICLR 2024.
[Zhao, Zhang, Chenm, Wang, Anandkumar, Tian] GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection, 2024.

Summary and other interesting directions

- Low-rank decomposition is a simple technique for DNN compression
  - Can achieve both speedup and memory efficiency
  - Can be combined with other methods
- Low-rank approaches can be useful in training
  - Fine-tuning efficiency
  - Pretraining efficiency (still under-development)
  - Efficient second order optimization
    - Second order optimizers often need to manipulate large precondition matrices
    - Low-rank can help!