Recombee

The Frontier of **Autoencoder-Based Recommender Systems**

Martin Spišák, Vojtěch Vančura

11 December 2024

We Move Quickly and Adjust to the Market Needs

- **10+ years** of continuous research and development
- **8 years** on the market
- **Exponential revenue growth**

9GAG audiomack zumper Tripadvisor

CASCADE PBS CREXI slickdeals showmax

STINGRAY eneba TEMPLE & WEBSTER BenefitHub

goldbelly konga.com MEDIAVINE PetMediaGroup

1000+ sites and apps

Recombee

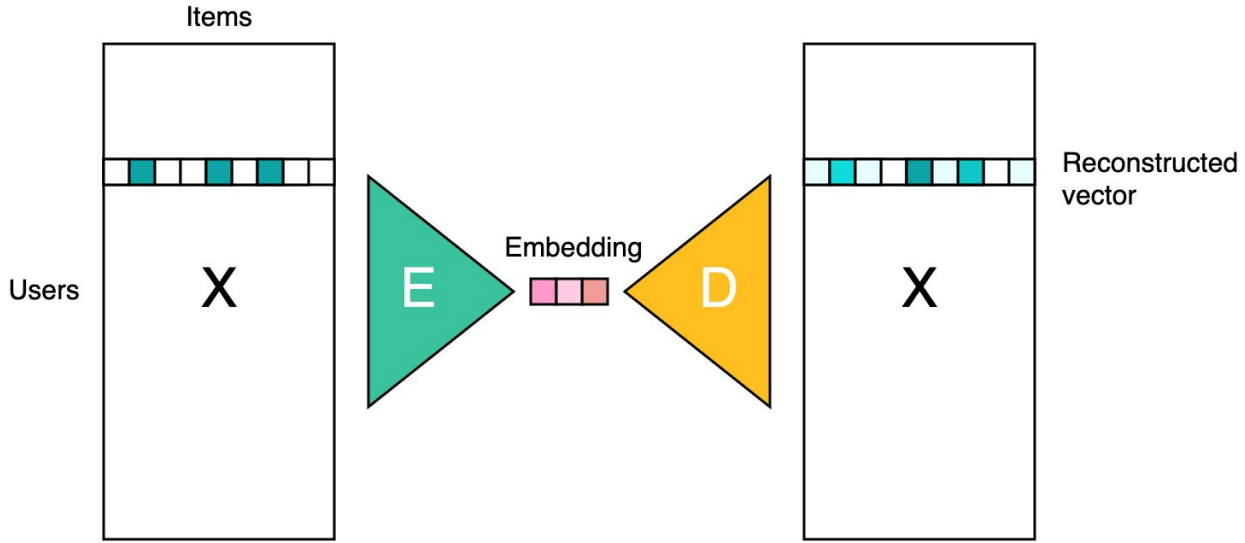
Outline

1. **How autoencoders work** in collaborative filtering tasks
2. **How to scale them** to huge datasets
3. How to incorporate **item information**
4. How we **teach autoencoders to generalize** to new items

1. **How autoencoders work** in collaborative filtering tasks

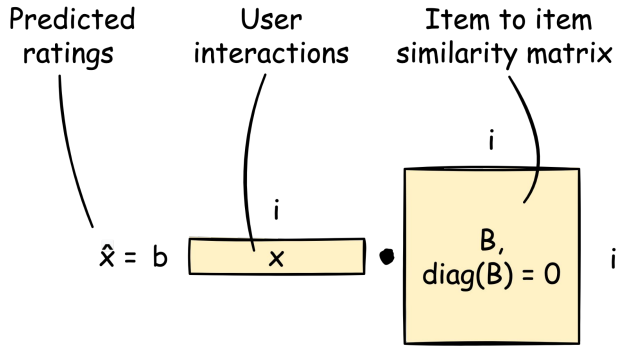
Interaction Autoencoders

- Autoencoder = self-supervised representation learning technique
- In RecSys context: Train a ML model to reconstruct rows of (sparse) interaction matrix X
- Denoising AutoEncoders (DAE), Variational AutoEncoder (VAE) etc.



EASE

- EASE is a **linear autoencoder** model with closed-form solution
  - linear regression but with huge model capacity
  - Encoder and decoder fused together
- EASE trains item-to-item weight matrix
- Diagonal of weights constrained to zero to prevent trivial solutions



Steck, Harald. "Embarrassingly shallow autoencoders for sparse data." In The World Wide Web Conference, pp. 3251-3257. 2019.

EASE - results

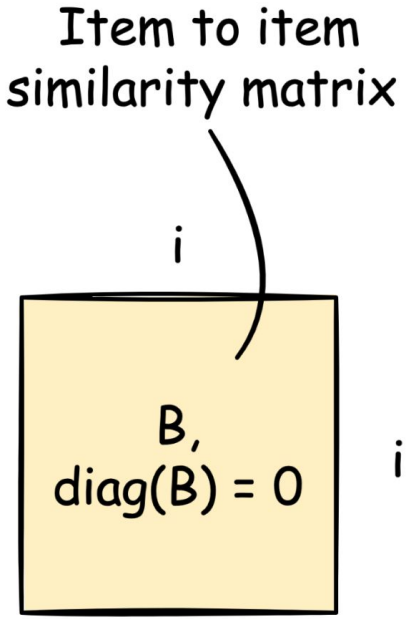
- Closed-form solution => very fast training
- (more-or-less) SotA recommendation quality

Table 1: The closed-form dense solution $\hat{\mathbf{B}}^{(\text{dense})}$ (see Eq. 4) obtains competitive ranking-accuracy while requiring only a small fraction of the training time, compared to the various models empirically evaluated in [33]. The standard errors of the ranking-metrics are about 0.002, 0.001, and 0.001 on *ML-20M*, *Netflix*, and *MSD* data [33], respectively.

	ML-20M			*Netflix*			*MSD*		
models	nDCG @100	Recall @20	Recall @50	nDCG @100	Recall @20	Recall @50	nDCG @100	Recall @20	Recall @50
$\hat{\mathbf{B}}^{(\text{dense})}$	0.423	0.392	0.522	0.397	0.364	0.448	0.391	0.334	0.430
	reproduced from [33]:								
MULT-VAE	0.426	0.395	0.537	0.386	0.351	0.444	0.316	0.266	0.364
MULT-DAE	0.419	0.387	0.524	0.380	0.344	0.438	0.313	0.266	0.363
CDAE	0.418	0.391	0.523	0.376	0.343	0.428	0.237	0.188	0.283
SLIM	0.401	0.370	0.495	0.379	0.347	0.428	–did not finish in [33]–		
WMF	0.386	0.360	0.498	0.351	0.316	0.404	0.257	0.211	0.312
training times									
$\hat{\mathbf{B}}^{(\text{dense})}$	2 min 0 sec			1 min 30 sec			15 min 45 sec		
MULT-VAE	28 min 10 sec			1 hour 26 min			4 hours 30 min		
data-set properties	136,677 users 20,108 movies 10 million interactions			463,435 users 17,769 movies 57 million interactions			571,355 users 41,140 songs 34 million interactions		

EASE - the catch

- EASE requires **O(n_items^2) memory** => <u>unfeasible</u> for large use-cases with millions of items



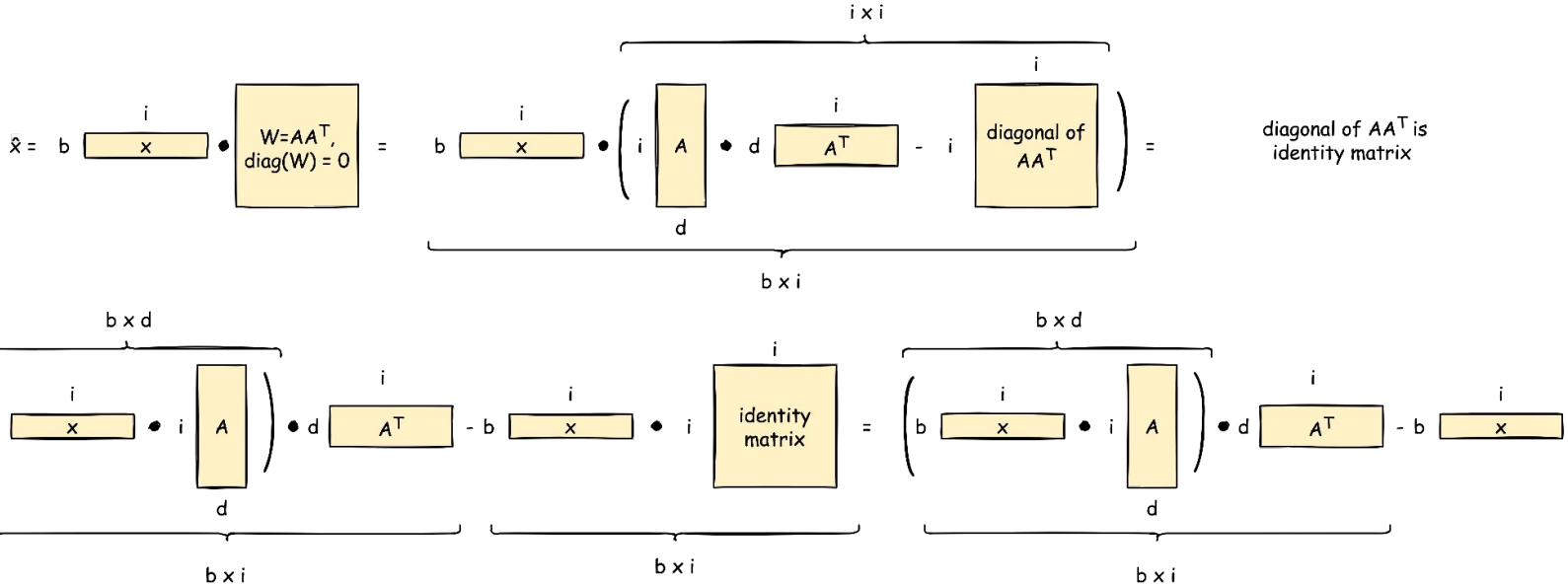
2. **How to scale them** to huge datasets

ELSA

- Assume that weight matrix B is a cosine similarity matrix
- B can be factorized $B=AA^T$ (encoder = decoder)
- A is latent representation for items
- On the diagonal of B are ones
- Rows of A are L2-normalized
- Can be trained using numerical optimisation with backprop

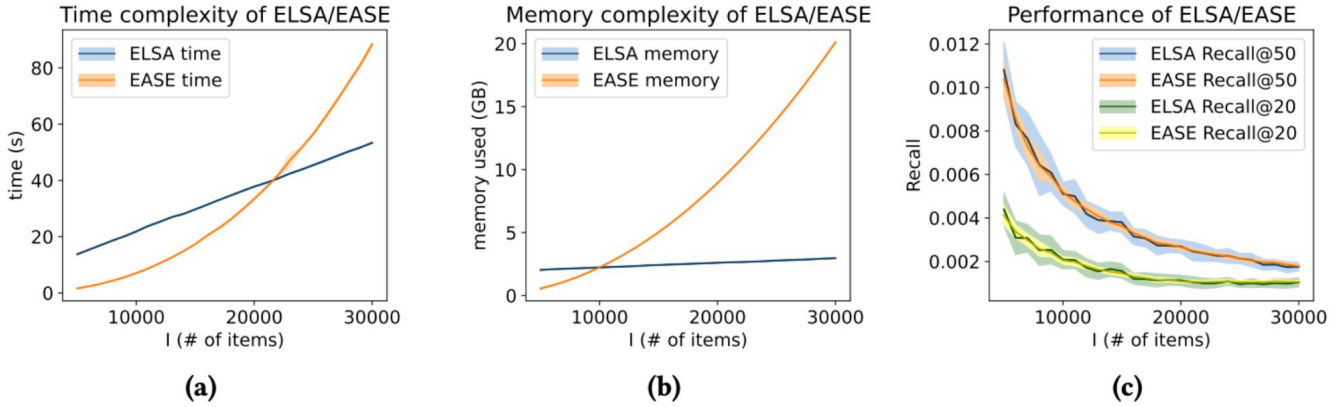
ELSA

- Weight matrix is factorized into low-rank structure
- Neither for optimizing the model nor predicting the recommendations, it will be necessary to store a dense matrix with dimensions superior to i×d



ELSA - results

- Same performance as EASE
- Linear complexity with respect to the number of items



Comparison of EASE and ELSA: a) in terms of time complexity, b) memory complexity, c) performance on synthetic data

ELSA

https://github.com/recombee/ELSA

```
pip install elsarec
```

SANSA

ELSA: memory requirements `n_items x latent_dim` (still large)

Dense weight matrix of* EASE *can be accurately approximated by a product of 2 sparse matrices

- tool: **factorized sparse approximate inversion**
  - sophisticated numerical algorithm
  - can extract *global information* from the item–item network
  - factorization is the key to compression
- user chooses final model density

SANSA

Architecture and training procedure

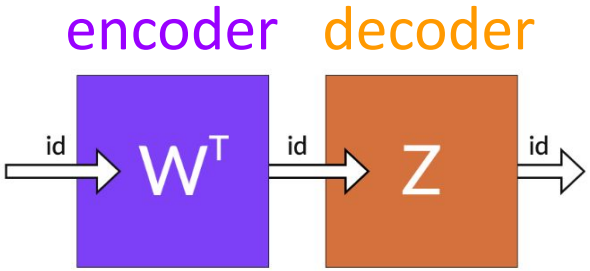


Fig. 1. SANSA is a sparse nonsymmetric encoder-decoder model. To disallow recommending input items, we mask the prediction vector, or add an input-output residual connection.

input user-item interaction matrix X, L2 regularization λ

1: compute sparse $LDL^T \approx P(X^TX + \lambda I)P^T$ (for some permutation P)
2: compute sparse $K \approx L^{-1}$
3: $W \leftarrow KP$
4: $Z_0 \leftarrow D^{-1}W$
5: $\vec{r} \leftarrow \text{diag}(W^TZ_0)$
6: $Z \leftarrow$ scale the columns of Z_0 by $-\vec{r}$

return W^T, Z

Algorithm 1. The training procedure of SANSA is based on factorized sparse approximate inversion. The final scaling is applied to the decoder only.

SANSA - results

Dense dataset

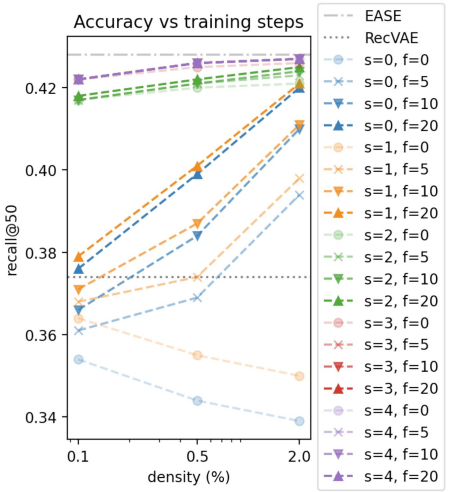
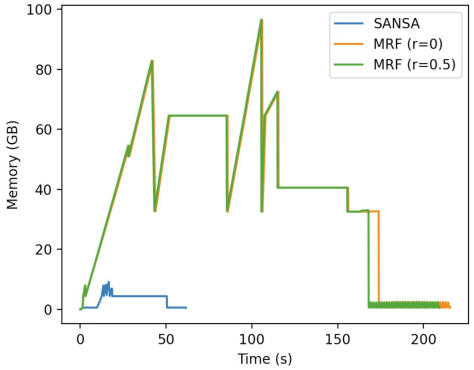


Figure 2: Accuracy of SANSA (CHOLMOD) on MSD after various number of training scans s and finetune steps f.

Sparse dataset

Amazon Books	SANSA (ICF)	MRF ($r = 0$)	MRF ($r = 0.5$)	EASE^R	SLIM	ITEM-CF	ULTRA-GCN
			results reprinted from [13]:				
r@20	**0.077**	0.071	0.069	0.071	0.075	0.074	0.068
n@20	**0.064**	0.058	0.055	0.057	0.060	0.061	0.056
TRAINING RESOURCES							
vCPU	**2**	16	16	28	28	28	20*
memory usage (GB):							
peak	**9.18**	96.45	96.58	—— not measured; costly ——			
avg.	**3.87**	49.12	49.75	—— not measured; costly ——			
time	**49 s**	172 s	167 s	222 m	316 m	57 m	45 m

*and a GPU (RTX 2080)



Scaling hypothesis: larger dataset => greater compression

*At O(1M) items we get **100 000x compression** vs. full EASE*

SANSA

https://github.com/glami/sansa

```
pip install sansa
```

3. How to incorporate **item information**

ELSA with item information

ELSA assumes weight matrix B is a cosine similarity matrix

Given images/descriptions/metadata, we can compute embeddings for items (SVD, deep learning…)

Idea:

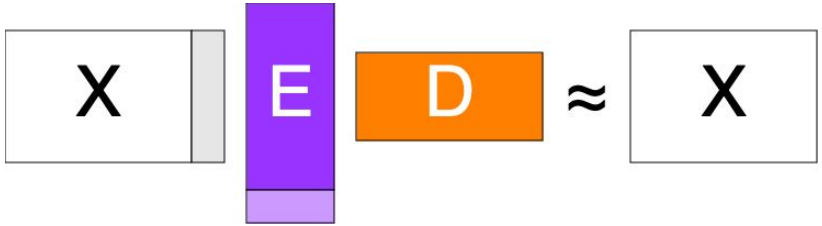
- Instead of $B=AA^T$ (encoder = decoder), let's represent B=ED
- Use the embedding matrix as either E or D
- Learn the other one using gradient descent

ELSA with item information

Implications:

Without retraining, we can expand E or D by generating embeddings for items not seen during training (cold start), and

- [expanded encoder] use new interactions with unseen items to recommend seen items



- [expanded decoder] use interactions with seen items to recommend unseen items

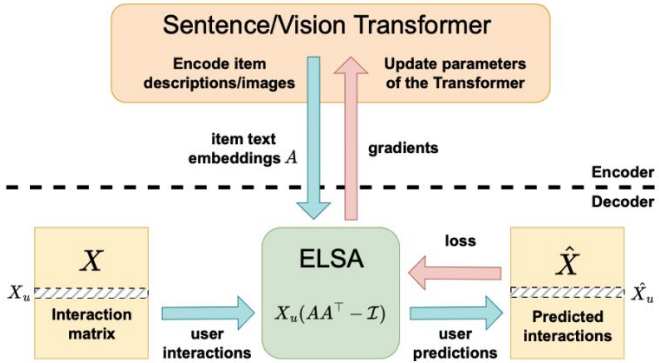
4. How we **teach autoencoders to generalize** to new items

beeFormer

Assume, that A is not initialized randomly, but computed from attributes (descriptions, images) with neural network (NN). Then, we can **propagate gradients from ELSA to NN that generated A** and update weights of that NN. However we face the following challenge:

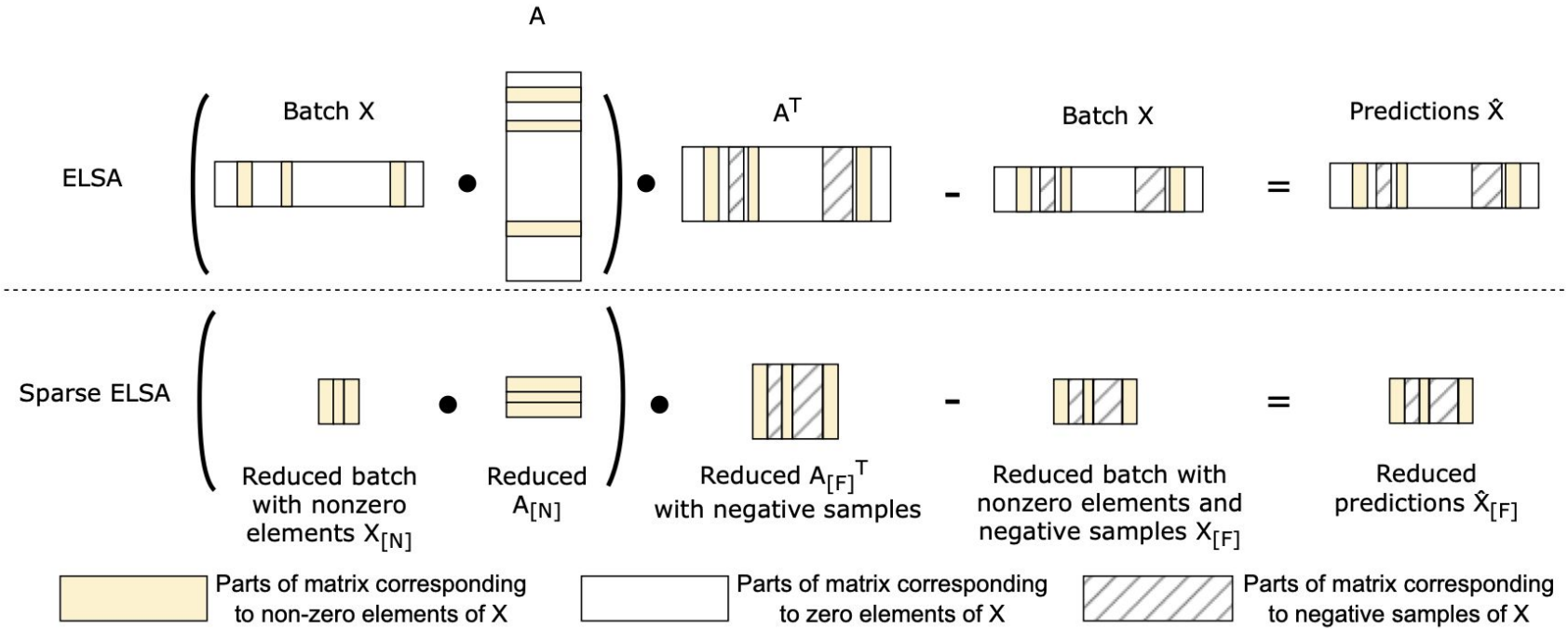
In every training step, we need to encode all items in the catalogue (possibly millions of items) with NN. This is unfeasible for both time and GPU memory reason.

To make training feasible we employ three following tricks.



beeFormer - trick #1 - leveraging sparsity

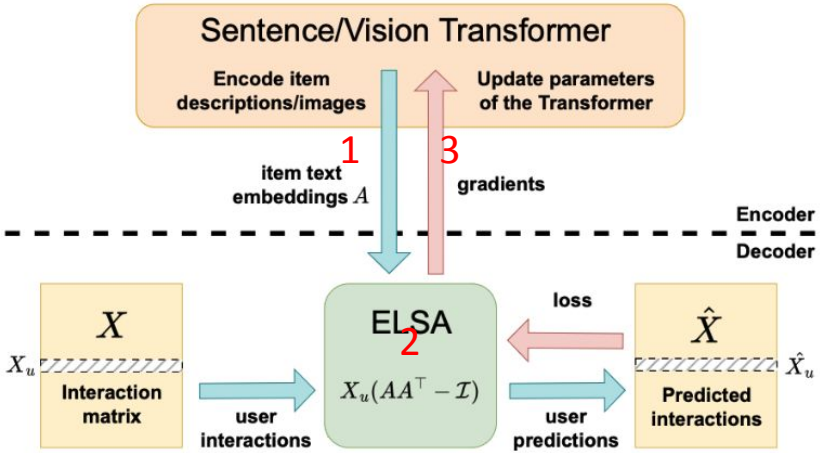
Interaction data are typically (very) sparse. Therefore, we can omit columns with only zeros (or limit them with negative sampling):



beeFormer - trick #2 - gradient checkpointing

Storing the whole graph for backpropagation would be costly, therefore we split training into three steps:

1. Compute A without tracking gradients for NN
2. Perform ELSA step and track gradients for A (gradient checkpoint)
3. Compute A while tracking gradients to optimize NN.



beeFormer - trick #3 - gradient accumulation

We don’t need to compute gradients for all items at once during optimisation of NN.

We can do it in batches and accumulate gradients during the loop.

Finally we can update NN once with accumulated gradients.

Algorithm 1 beeFormer training step procedure in Python using PyTorch

Input:
batch of interactions `X`
Transformer model `transformer`
descriptions of the selected items as sequences of tokens `tokenized_texts`
PyTorch optimizer `optimizer` optimizing the parameters of a Transformer model `transformer`

Output:
Transformer model `transformer` with updated weights
predictions `X_pred`
computed loss `loss`

```
with torch.no_grad():
    A_list = []
    for t in tokenized_texts:
        A_list.append(torch.nn.functional.normalize(transformer(t)))
    A = torch.vstack(A_list)

A.requires_grad = True
X_pred = X @ A @ A.T - X

loss = torch.nn.MSELoss(
    torch.nn.functional.normalize(X),
    torch.nn.functional.normalize(X_pred),
)
loss.backward()

checkpoint = A.grad

optimizer.zero_grad()
for i, t in enumerate(tokenized_texts):
    A_i = transformer(t)
    A_i.backward(gradient=checkpoint[i])
optimizer.step()
```

beeFormer - results

Dataset Method	Sentence Transformer	R@20	R@50	N@100
GB10K CBF	*Llama-goodbooks-mpnet*	0.2646	0.3990	0.3869
	Llama-goodbooks-nomic	**0.3070**	**0.4248**	**0.4161**
GB10K Heater	all-mpnet-base-v2	0.2078	0.3221	0.3195
	nomic-embed-text-v1.5	0.2154	0.3317	0.3193
	bge-m3	0.2052	0.3113	0.3099
	Llama-goodbooks-mpnet	0.2570	0.3800	0.3749
	Llama-goodbooks-nomic	0.2520	0.3577	0.3644
ML20M CBF	*Llama-movielens-mpnet*	0.4193	**0.6147**	**0.4066**
	Llama-movielens-nomic	**0.4330**	0.5404	0.3874
ML20M Heater	all-mpnet-base-v2	0.3114	0.4331	0.3407
	nomic-embed-text-v1.5	0.3049	0.4285	0.3270
	bge-m3	0.2847	0.3932	0.3161
	Llama-movielens-mpnet	0.4320	0.5979	0.3937
	Llama-movielens-nomic	0.4263	0.5346	0.3734

Data Method	Transformer	R@20	R@50	N@100
Images CBF	*Img-amazfashion-clip-vit-b32*	0.4216	0.4676	0.4071
	Img-amazfashion-clip-vit-l14	**0.4490**	**0.4947**	**0.4262**
	Img-amazfashion-dinov2-small	0.4169	0.4612	0.4024
	Img-amazfashion-dinov2-base	0.4356	0.4739	0.4174
	Img-amazfashion-nomic	0.4403	0.4927	0.4148
Images Heater	clip-vit-base-patch32	0.3919	0.4247	0.3745
	clip-vit-large-patch14	0.4076	0.4407	0.3903
	dinov2-small	0.3914	0.4264	0.3830
	dinov2-base	0.3916	0.4289	0.3828
	nomic-embed-vision-v1.5	0.4132	0.4489	0.3994

Dataset	Sentence Transformer	R@20	R@50	N@100
GB10K	all-mpnet-base-v2	0.1017	0.1886	0.1739
	nomic-embed-text-v1.5	0.1146	0.2069	0.1896
	bge-m3	0.1134	0.1953	0.1838
	Llama-movielens-mpnet	0.1781	0.2856	0.2737
	Llama-amazbooks-mpnet	0.2648	0.3955	0.3792
	Llama-movielens-nomic	0.1964	0.2832	0.2793
	Llama-amazbooks-nomic	**0.2908**	**0.4263**	**0.3998**
ML20M	all-mpnet-base-v2	0.0788	0.1550	0.1042
	nomic-embed-text-v1.5	0.1113	0.2143	0.1511
	bge-m3	0.1409	0.2125	0.1578
	Llama-goodbooks-mpnet	0.1641	0.2771	0.2100
	Llama-goodbooks-nomic	**0.2710**	**0.4033**	**0.2695**
Amazon fashion	all-mpnet-base-v2	0.4386	0.4804	0.3497
	nomic-embed-text-v1.5	0.4234	0.4754	0.3480
	bge-m3	0.4465	0.4882	0.3526
	Llama-goodbooks-nomic	**0.4800**	**0.5272**	**0.3995**
	Llama-movielens-nomic	0.4658	0.5111	0.3887

Cold start scenario - beeFormer trained models **generalize towards unobserved items**

Zero shot scenario - beeFormer trained models **transfer knowledge from one dataset to another**

beeFormer - results

Scenario Method	Model	R@20	R@50	N@100
zero-shot CBF	all-mpnet-base-v2	0.0218	0.0336	0.0193
	nomic-embed-text-v1.5	0.0387	0.0560	0.0320
	bge-m3	0.0398	0.0546	0.0313
	Llama-goodbooks-mpnet	0.0650	0.0928	0.0513
	Llama-goodbooks-nomic	**0.0706**	**0.1005**	**0.0551**
	Llama-movielens-mpnet	0.0266	0.0372	0.0221
	Llama-movielens-nomic	0.0555	0.0734	0.0419
supervised CF	KNN	0.0370	0.0562	0.0303
	ALS MF	0.0344	0.0580	0.0313
	ELSA	0.0367	0.0628	0.0346
	SANSA	0.0421	0.0678	0.0362
supervised CBF	*Llama-amazbooks-mpnet*	0.0706	0.1045	0.0571
	Llama-amazbooks-nomic	**0.0765**	**0.1150**	**0.0623**

Comparison with supervised CF models. Note that model trained on movies beats supervised CF models on books data.

Dataset	Scenario	Model	R@20	R@50	N@100
GB10K	cold-start	Llama-goodbooks-mpnet	0.2646	0.3990	0.3869
	zero-shot	Llama-movielens-mpnet	0.1781	0.2856	0.2737
	cold-start	*Llama-goodlens-mpnet*	0.2600	0.4109	0.3984
	cold-start	Llama-goodbooks-nomic	**0.3070**	**0.4248**	**0.4161**
	zero-shot	Llama-movielens-nomic	0.1964	0.2832	0.2793
	cold-start	*Llama-goodlens-nomic*	0.2891	0.4113	0.4032
ML20M	zero-shot	Llama-goodbooks-mpnet	0.1641	0.2771	0.2100
	cold-start	Llama-movielens-mpnet	0.4193	0.6147	0.4066
	cold-start	*Llama-goodlens-mpnet*	**0.4519**	**0.6153**	**0.4106**
	zero-shot	Llama-goodbooks-nomic	0.2710	0.4033	0.2695
	cold-start	Llama-movielens-nomic	0.4330	0.5404	0.3874
	cold-start	*Llama-goodlens-nomic*	0.3899	0.4965	0.3557

beeFormer can accumulate knowledge from multiple datasets (domains)

beeFormer

https://github.com/recombee/beeformer

https://huggingface.co/beeformer

Recombee

Thank You For Your Attention

Q & A