Recommender systems with TensorFlow

Martin Jakomin
Jan Hartman

About us

Data scientists

Outbrain

Dr. Martin Jakomin

Jan Hartman

What do we do?

- We develop and maintain the “brains” behind the purchasing of online ad space
- Choosing the right ad <-> recommender system!
- We use TensorFlow for real-time ML predictions (1 billion predictions per second)
- Involved in many other interesting things (conversion, bid win prediction, bid price control…)

J. Hartman, D. Kopič - Scaling TensorFlow to 300 Million Predictions per Second, RecSys '21.
J. Hartman, D. Kopič - Exploration with Model Uncertainty at Extreme Scale in Real-Time Bidding, RecSys '22.

Agenda, part 1

- Recommender systems paradigms and algorithms
  - Content-based filtering:
    - Item feature extraction
    - Similarity measures and distances
  - Collaborative filtering:
    - Memory based (user-user, item-item, hybrid)
    - Basic algorithms (nearest neighbours, ...)
    - Model based - latent factors
    - Matrix Factorization
  - Deep learning for RS:
    - Deep Neural CF
    - Autoencoders
    - DeepFM
- Recommender systems evaluation and metrics
- A short case study - YouTube recommendations

Agenda, part 2

- **TensorFlow**: basic concepts, tensors, autodiff
- Keras high-level API: layers, models, fitting & evaluation
- Matrix factorization implementation
- Deep autoencoder implementation
- Bonus challenge

Repo link: github.com/janhartman/recsystf

Introduction

- We often make choices in life without sufficient personal experiences, thus we heavily rely on recommendations.
- Recommender systems (RS) are tools that assist and augment this natural decision making process.

Introduction

- RS are large-scale machine learning and knowledge discovery tools aimed at providing **personalized** recommendations.
- Examples: videos, movies and songs, web pages, articles, ads, books, social media posts, products, services ...
- In terms of **machine learning** problems:
  - estimation of user ratings (matrix completion),
  - picking topN items (ranking),
  - picking new unfamiliar items (novelty, diversity).

Paradigms And Algorithms

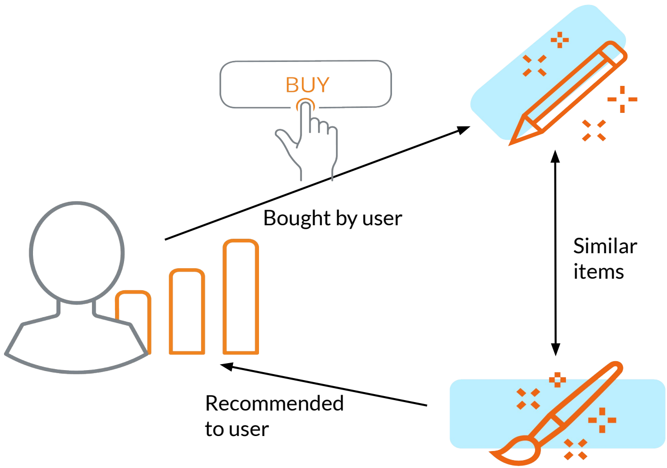
Typically we split RS into three categories:

- ***Content-based filtering*** (where the user is recommended items similar to the ones he preferred in the past),
- ***Collaborative filtering*** (where the user is recommended items that people with similar tastes and preferences liked in the past),
- ***Hybrid approaches*** (that combine collaborative and content-based methods).

Bobadilla, Jesús, et al. "Recommender systems survey." *Knowledge-based systems* 46 (2013): 109-132.

Content-based Filtering

- Recommendations are based on user choices made in the past (e.g. previous purchases), combined with rated items **descriptions**.
- The main idea: if you like a particular item in the past you will also like a **similar** item in the future.



Content-based Filtering

- **Similarity** between objects is therefore the basis for recommending new items.
- We can view this as a user-specific classification problem of user's likes and dislikes based on an item's features.
- CBF is best suited to situations where there is abundant data on items, but not for the users.

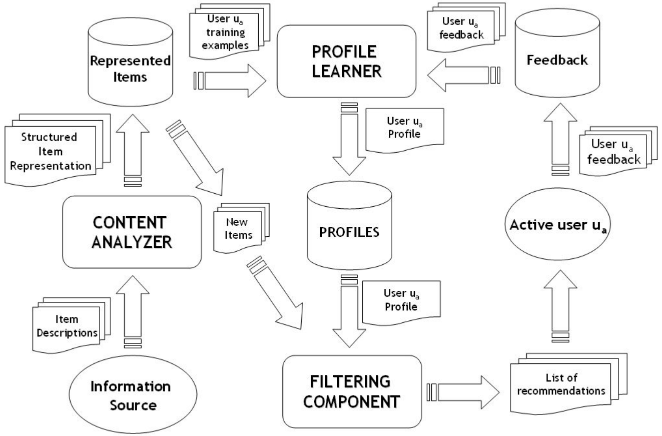
Content-based Filtering

★ Easy to implement
★ Transparent, Explainable
★ User independent (no user data needed)
★ Scalable
★ No item cold-start

⬇️ Difficult information extraction
⬇️ Limited content analysis
⬇️ Overspecialization (only recommending similar things)
⬇️ User cold-start

Content-based Filtering

- The recommendation process is usually performed in 3 steps:
  - Content Analysis (extracting features),
  - Profile learning (extracting user preferences),
  - Filtering (extract relevant items).



Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Springer, Boston, MA, 2011.

CBF - Item Similarity

- Item representation:
    - Structured,
    - Unstructured (vector space models, TF-IDF),
    - Mixed.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

CBF - Recommending

- Recommend simply by returning similar items to those which the user has rated in the past.
- Weighting can be applied in case of real-number ratings.
- More sophisticated approaches can use Bayesian Classifiers, cluster analysis, decision trees, and neural networks in order to estimate the probability that the user is going to like the item.

CBF - Similarity Measures

- Euclidean distance:

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}.$$

- Manhattan distance:

$$\sum_{i=1}^{n}|p_i - q_i|,$$

- Minkowski distance

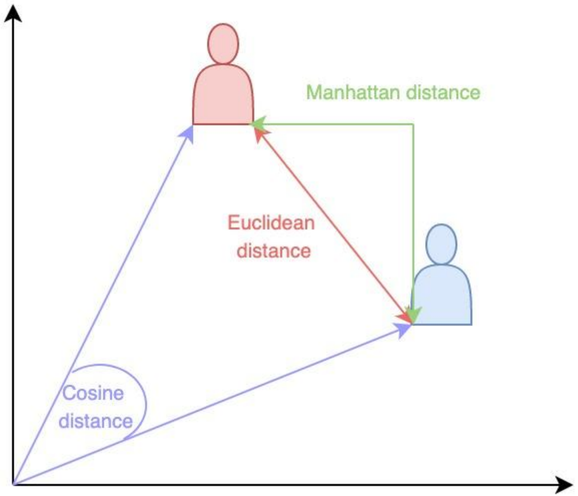
$$D(X,Y) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{\frac{1}{p}}$$

- Jaccard distance

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

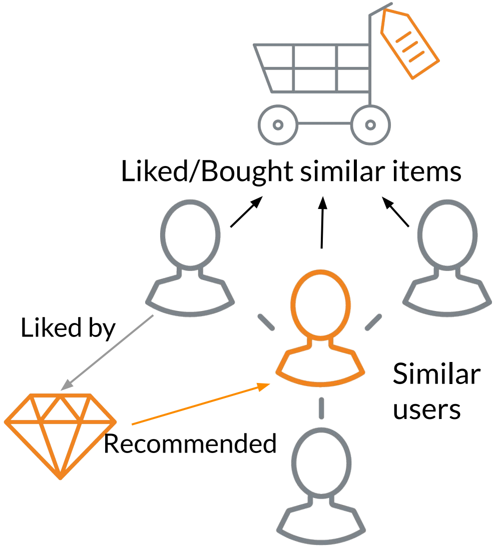
- Cosine distance

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$



Collaborative Filtering

- Recommendations are usually based on user’s **neighbours** choices made in the past (of similar items).
- The main idea: people who agreed in the past will **agree** in the future and will like similar kinds of items.



Collaborative Filtering

- Oppose to CBF, in collaborative filtering (CF) we generally only use past **rating** information (though, other information/context can be incorporated into the models).
- No “feature extraction” needed, models can learn abstract (latent) features on the fly.
- This allows for serendipitous recommendations (items can be completely different from what was seen and liked in the past).

Collaborative Filtering

★ Superior modelling and accuracy
★ Serendipity
★ Efficient and scalable
★ Justifiable and transparent*
★ Robust and stable
★ Doesn’t need item information

⬇️ Black box models*
⬇️ User and Item cold-start
⬇️ Can be biased

* Depends on the model

Collaborative Filtering

CF methods are usually grouped in two general classes:

- **Memory-based / Neighborhood-based / Heuristic-based**
  - Where user/item ratings stored in the system are directly used for prediction
- **Model-based / Latent factors**
  - Where user/item ratings are used to learn a predictive model

Memory-based CF

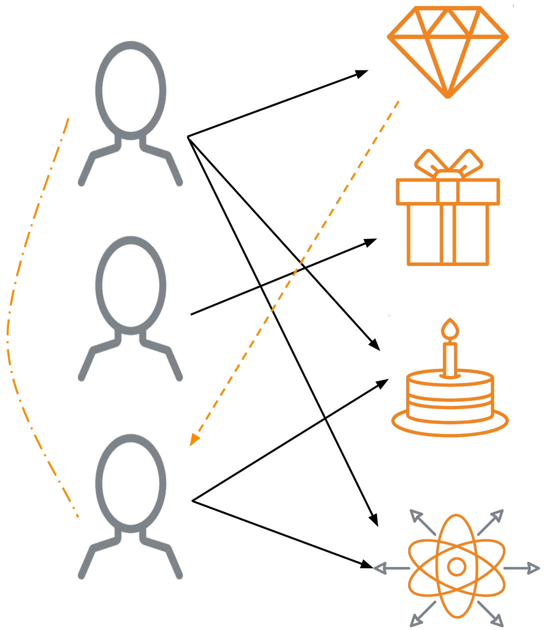
- Most basic approach where ratings are calculated using simple statistics and aggregations upon stored data.
- Very **simplistic**, but can often capture local associations in the data.
- Generally split into:
  - ***user-based*** (where users are recommended items from their neighbouring users with similar rating patterns),
  - ***item-based*** (where users are recommended items based on their ratings of the set of similar items).

Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Springer, Boston, MA, 2011.

(Memory) User-based CF

- Predict a rating for a user ***u*** for an item ***i*** using the ratings for the item ***i*** by users most similar to ***u***.
- Weighted kNN (regression or classification).

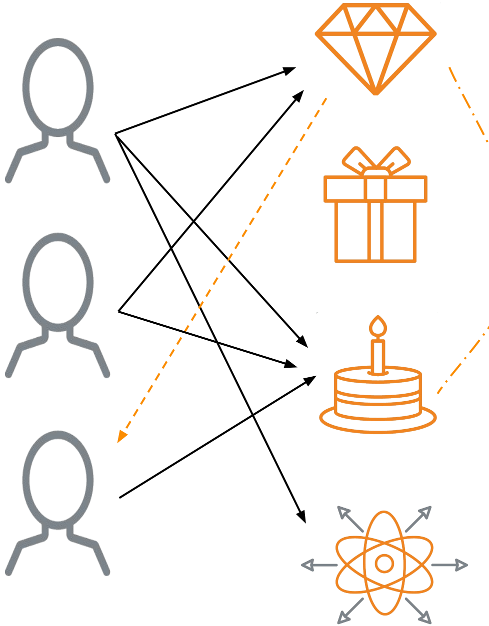
$$\hat{r}_{ui} = \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}$$



(Memory) Item-based CF

- Predict a rating for a user ***u*** for an item ***i*** using the ratings by the user ***u*** for items most similar to ***i***.
- Weighted kNN (regression or classification).

$$\hat{r}_{ui} = \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} r_{uj}}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}$$



Memory-based CF

- Memory-based CF methods are simplistic, transparent (explainable), efficient and allow for serendipitous predictions.
- We can further extend MCF with various normalization (centering and biases), choosing different similarity metrics (Euclidean, Cosine, ...) or by applying various pre-filtering and post-filtering stages (neighbourhood selection), speedups (locality-sensitive hashing), ...
- Finally, we can combine user-based and item-based approaches using Hybrid models.

Model-based CF

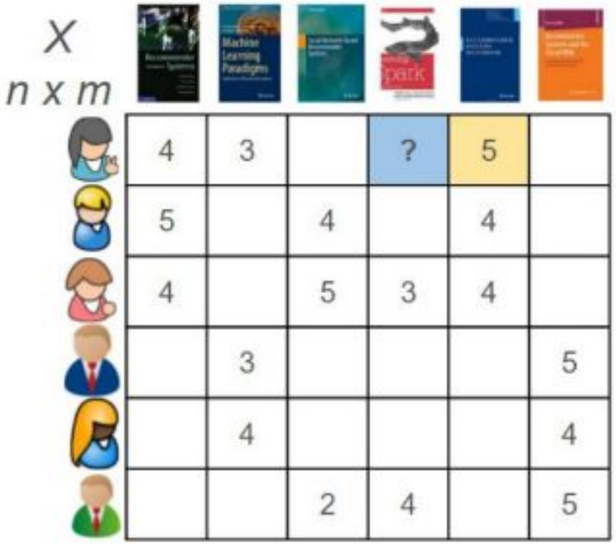
- Contrary, model-based approaches rather use ratings in order to learn a **predictive model** (which is then used to make new predictions).
- Goal of modelling user-item interactions with factors representing their latent **characteristics**.
- These factors are inferred solely from the ratings and might represent obvious dimensions (e.g. movie genres) or something completely uninterpretable to us.

Matrix Factorization

- Essential tool in machine learning, broadly used for dimensionality reduction, compression, clustering, classification, ...
- Also, one of the most popular and most used CF model.
- Simple and efficient way to model explicit data and various other side information (implicit ratings, biases, temporal effects, etc.).

Matrix Factorization

- User-Item ratings can be simply represented with a matrix.
- Recommendations now becomes a **matrix completion** problem.
- We can solve this by factorizing this matrix into a product of (two) smaller matrices.

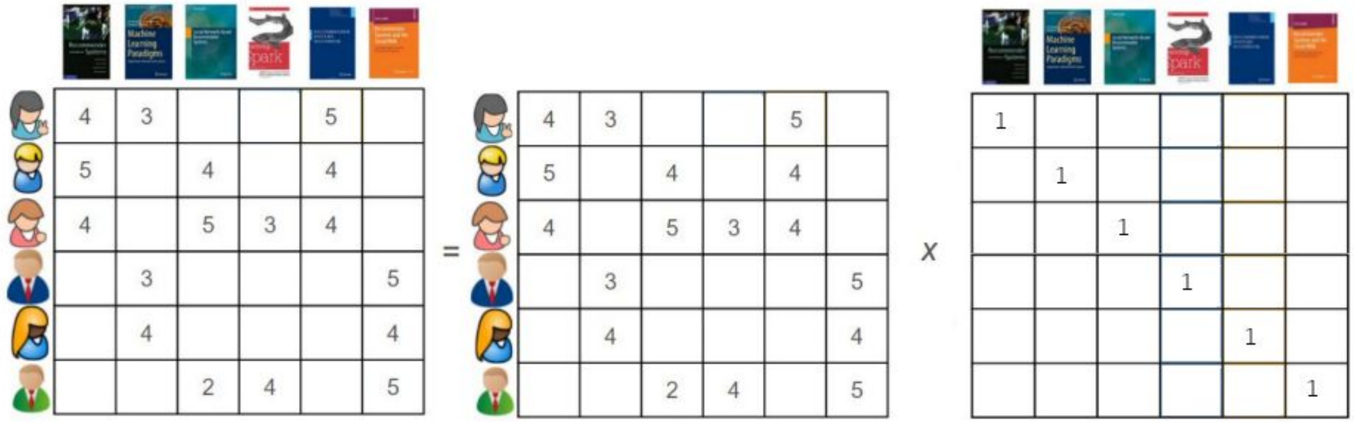


4	3		?	5	
5		4		4	
4		5	3	4	
	3				5
	4				4
		2	4		5

Nikita Sharma. *Recommender Systems with Python*. https://heartbeat.fritz.ai/recommender-systems-with-python-part-iii-collaborative-filtering-singular-value-decomposition-5b5dcb3f242b

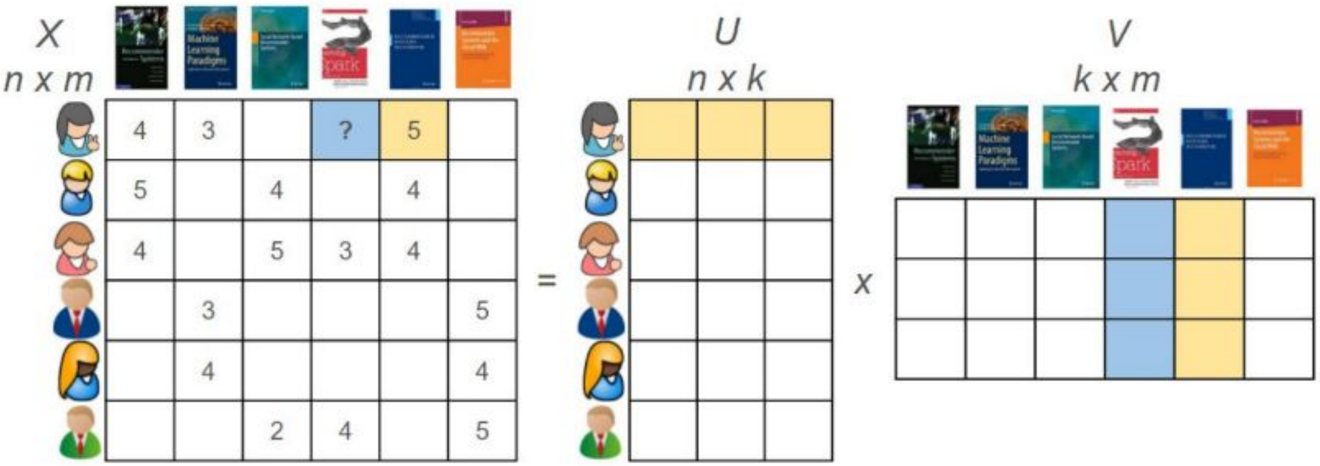
Matrix Factorization - Intuition

- A trivial factorization: ***X = X * I***

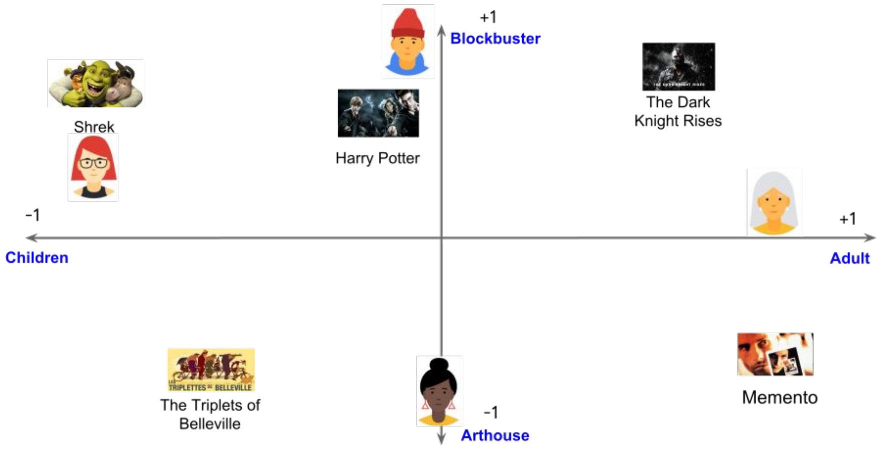


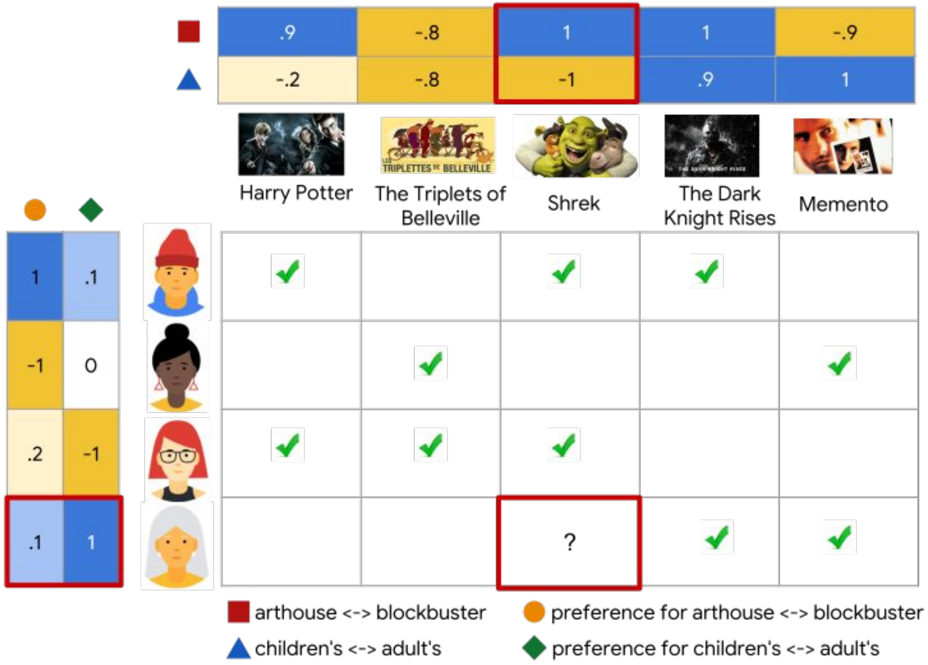
Matrix Factorization

- Low-Rank factorization: ***X = U * V***



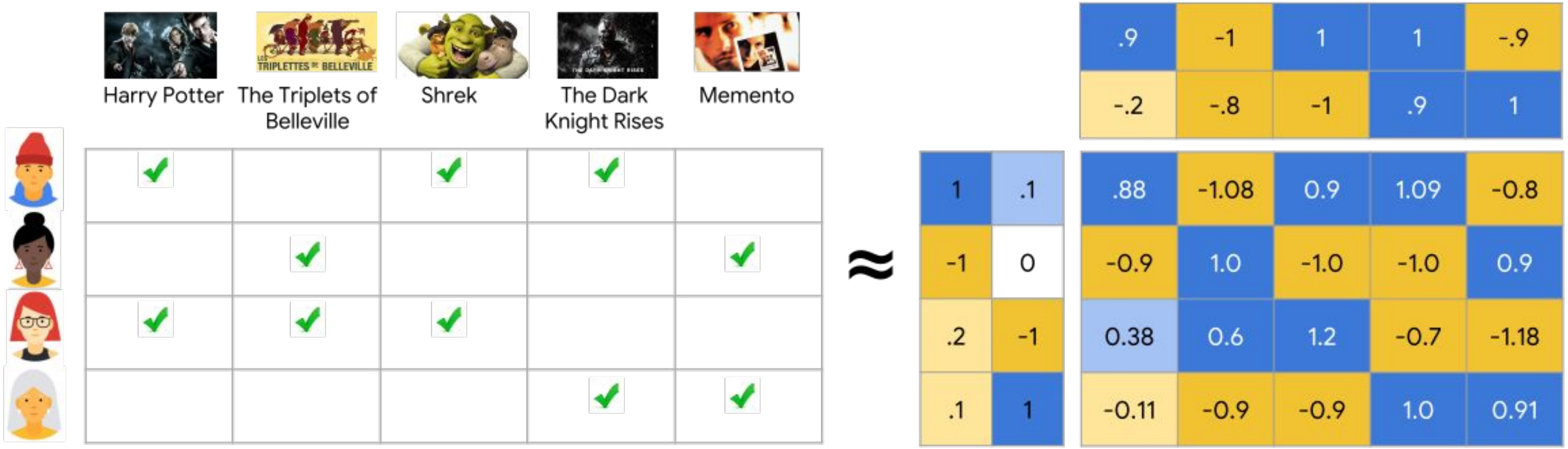
Matrix Factorization





https://developers.google.com/machine-learning/recommendation/

Matrix Factorization



Matrix Factorization

- **Solve** matrix factorization problem: $X \approx \hat{X} = UV^T$
- **Predict** with dot product (linear combinations) between latent user matrix and latent item matrix.
- There are (infinitely) many possible factorizations - how to find the best one?
- Define the matrix completion as an **optimization problem**.

Matrix Factorization

- Define the objective (loss) function, based on your needs.
- One of the most common is the squared loss (Frobenius norm): $$||X - \hat{X}||_F^2 = ||X - UV^T||_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij} - u_i v_j^T)^2$$

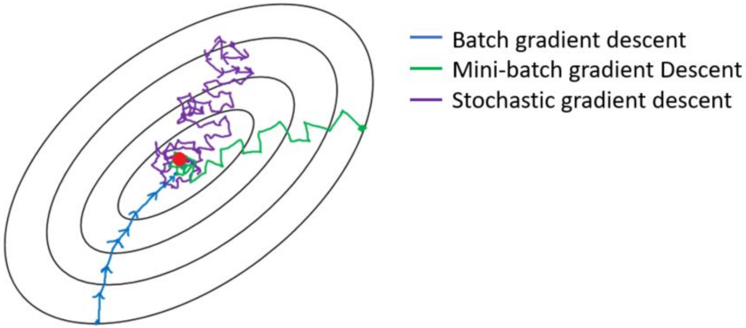
- Problem of huge **sparsity** (typically over 99%).
- We can use weighted (masked) matrix factorization:

$$\mathbb{W} \circ X \approx \mathbb{W} \circ \hat{X} = \mathbb{W} \circ (UV^T)$$

Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Springer, Boston, MA, 2011.

Matrix Factorization

- This kind of optimization problems are usually solved using various gradient descent methods: GD, SGD, MUR, PGD, CD ...
- Alternatively, we can use ALS (convex optimization).
- With SGD we can omit the weighted part and optimize only on the known (non-zero) ratings.



Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009): 30-37.

Matrix Factorization - SGD

$$L = \sum_{u,i}(r_{ui} - (\mu + b_u + b_i + \mathbf{x}_u^\intercal \cdot \mathbf{y}_i))^2 + \lambda_{xb} \sum_u \|b_u\|^2 + \lambda_{yb} \sum_i \|b_i\|^2 + \lambda_{xf} \sum_u \|\mathbf{x}_u\|^2 + \lambda_{yf} \sum_u \|\mathbf{y}_i\|^2$$

$$\frac{\partial L}{\partial b_u} = 2(r_{ui} - (\mu + b_u + b_i + \mathbf{x}_u^\intercal \cdot \mathbf{y}_i))(-1) + 2\lambda_{xb} b_u$$

$$\frac{\partial L}{\partial b_u} = 2(e_{ui})(-1) + 2\lambda_{xb} b_u$$

$$\frac{\partial L}{\partial b_u} = -e_{ui} + \lambda_{xb} b_u$$

$$b_u \leftarrow b_u + \eta\left(e_{ui} - \lambda_{xb} b_u\right)$$

$$b_i \leftarrow b_i + \eta\left(e_{ui} - \lambda_{yb} b_i\right)$$

$$\mathbf{x}_u \leftarrow \mathbf{x}_u + \eta\left(e_{ui}\mathbf{y}_i - \lambda_{xf}\mathbf{x}_u\right)$$

$$\mathbf{y}_i \leftarrow \mathbf{y}_i + \eta\left(e_{ui}\mathbf{x}_u - \lambda_{yf}\mathbf{y}_i\right)$$

Matrix Factorization

- MF allows for a simple inclusion of various **side information** (like implicit ratings, tags), **constraints** (penalties and rewards), **regularization**, **biases** and **temporal dynamics**.
- Contextual information about the rating process (circumstances) can also be incorporated - CARS:
  - Contextual pre-filtering, post-filtering and modelling
  - $R : User \times Item \times Context \rightarrow Rating$
  - Tensor factorization

Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009): 30-37.
Adomavicius, Gediminas, and Alexander Tuzhilin. "Context-aware recommender systems." *Recommender systems handbook*. Springer, Boston, MA, 2011.

Matrix Factorization

- Constrained Matrix Factorization:
  - NMF (only additive linear combinations induce parts-based representation)
  - Sparse MF (constrained number of combinations)
  - Orthogonal MF (uniqueness, clustering)
- Data Fusion

$$\overset{\mathrm{R}}{\begin{bmatrix} * & \mathrm{R}_{12} & \ldots\mathrm{R}_{1r} \\ \mathrm{R}_{21} & * & \ldots\mathrm{R}_{2r} \\ \vdots & \vdots & \ddots \; \vdots \\ \mathrm{R}_{r1} & \mathrm{R}_{r2} & \ldots \; * \end{bmatrix}} \approx \overset{\mathrm{G}}{\begin{bmatrix} \mathrm{G}_1 & & & \\ & \mathrm{G}_2 & & \\ & & \ddots & \\ & & & \mathrm{G}_r \end{bmatrix}} \overset{\mathrm{S}}{\begin{bmatrix} * & \mathrm{S}_{12} & \ldots\mathrm{S}_{1r} \\ \mathrm{S}_{21} & * & \ldots\mathrm{S}_{2r} \\ \vdots & \vdots & \ddots \; \vdots \\ \mathrm{S}_{r1} & \mathrm{S}_{r2} & \ldots \; * \end{bmatrix}} \overset{\mathrm{G}^{\mathrm{T}}}{\begin{bmatrix} \mathrm{G}_1^{\mathrm{T}} & & & \\ & \mathrm{G}_2^{\mathrm{T}} & & \\ & & \ddots & \\ & & & \mathrm{G}_r^{\mathrm{T}} \end{bmatrix}}$$

Jakomin, Martin. *Incremental matrix factorization for simultaneous learning from parallel data streams*. Diss. University of Ljubljana 2019.

Deep Learning for RS

- Nowadays, deep learning is everywhere - RS are not an exception.
- Data is becoming more abundant and available.
- Main Idea - the user/item interaction may be **non-linear**.
- Deep models and Hybrids are SOTA.

Zhang, Shuai, et al. "Deep learning based recommender system: A survey and new perspectives." *ACM Computing Surveys (CSUR)* 52.1 (2019)

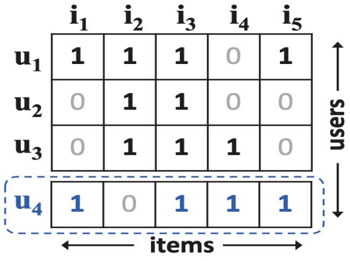
Deep RS

- ★ Non-linear transformations
- ★ Representation learning and better generalization
- ★ Sequence modelling (RNN)
- ★ Flexibility
- ★ Superior accuracy
- ★ Unified representation and data fusion

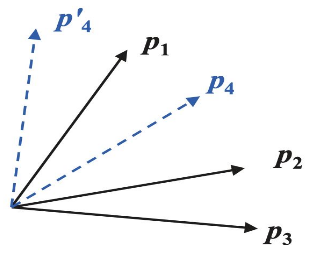
- ⬇️ Black box models - bad interpretability
- ⬇️ Data requirement and sparseness
- ⬇️ Extensive Hyperparameter tuning

Deep Neural CF

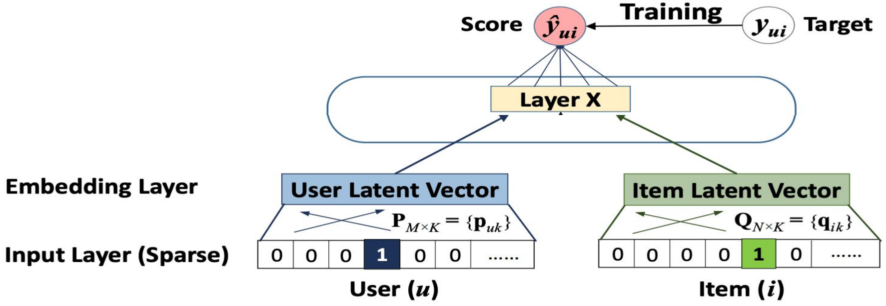
- User and Item embeddings (with one-hot input)
- Dense layers and single output



(a) user–item matrix



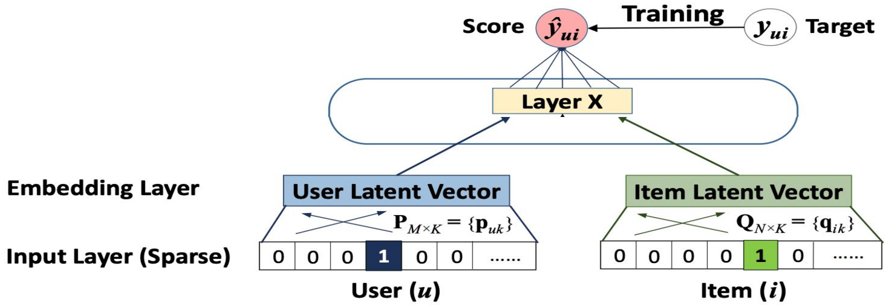
(b) user latent space



He, Xiangnan, et al. "Neural collaborative filtering." *Proceedings of the 26th international conference on world wide web*. 2017.

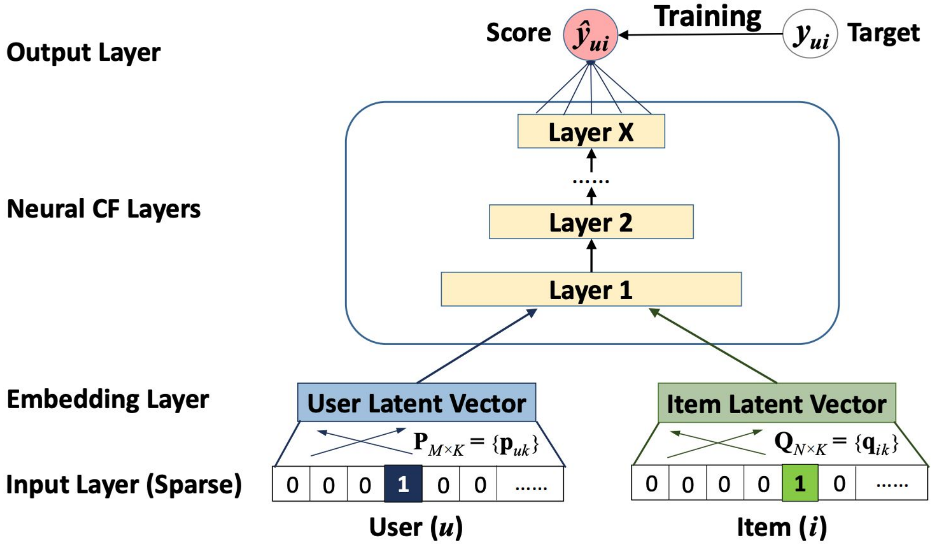
Deep Neural CF

- This is a generalization of the matrix factorization



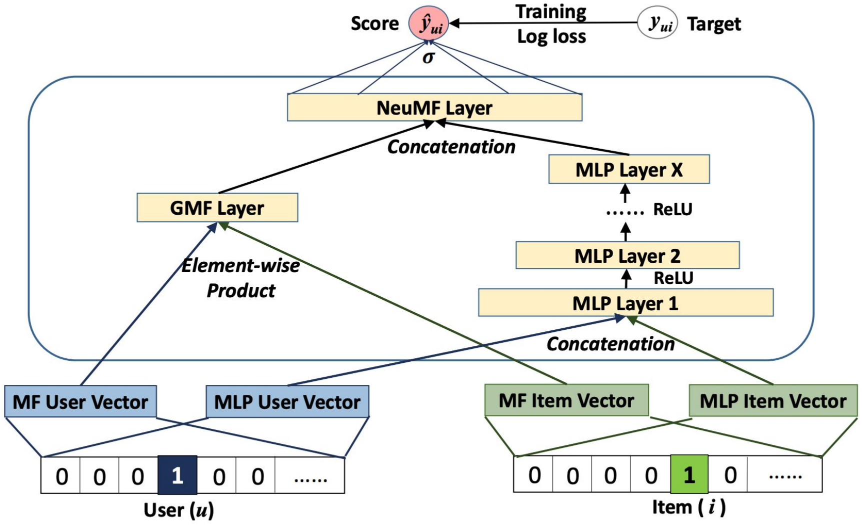
Deep Neural CF

- We can add multiple dense layers



Deep Neural CF

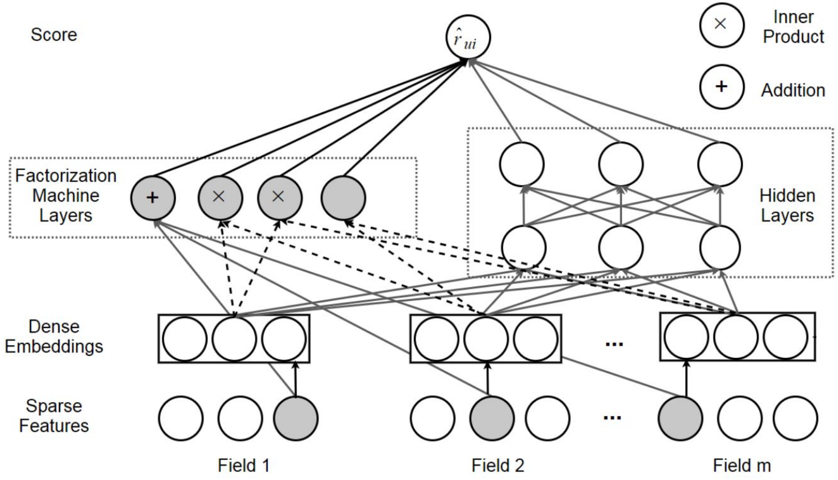
- Or combine both models



Deep Factorization Machines

- Main idea: Learn both, low-order (FM) and high-order (DNN) feature interactions.

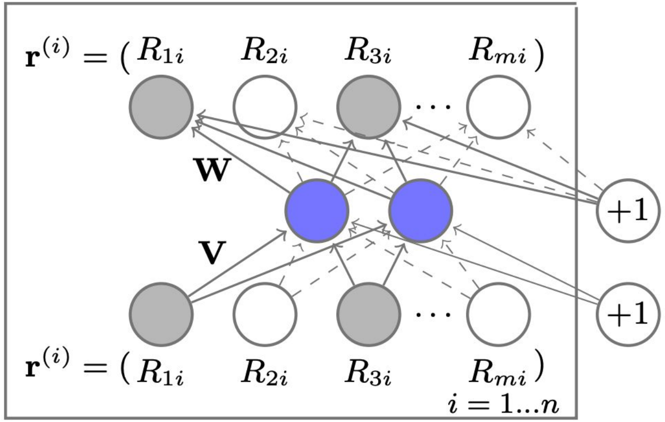
	User				Item				Categories				History				Quantity	
x^1	1	0	0	...	1	0	0	...	1	0	1	...	1	0	1	...	2	y^1
x^2	1	0	0	...	0	1	0	...	0	2	1	...	0	0	1	...	4	y^2
x^3	0	1	0	...	1	0	0	...	3	0	14	...	1	0	0	...	5	y^3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x^n	0	0	0	...	0	0	1	...	0	1	5	...	0	0	1	...	1	y^n



Guo, Huifeng, et al. "DeepFM: a factorization-machine based neural network for CTR prediction." *arXiv preprint arXiv:1703.04247* (2017).
Rendle, Steffen. "Factorization machines." *2010 IEEE International Conference on Data Mining*. IEEE, 2010.
Zhang, Shuai, et al. "Deep learning based recommender system: A survey and new perspectives." *ACM Computing Surveys (CSUR)* 52.1 (2019).

Autoencoders

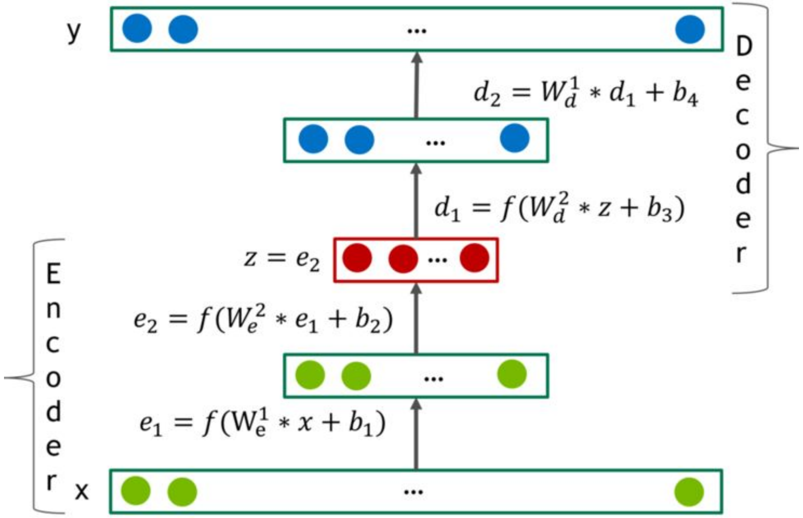
- Generalization of the PCA
- Aims to reconstruct the input on the output level



Sedhain, Suvash, et al. "Autorec: Autoencoders meet collaborative filtering." *Proceedings of the 24th international conference on World Wide Web*. 2015.

Deep Autoencoders

- Item-based
- User-based: denoising autoencoder



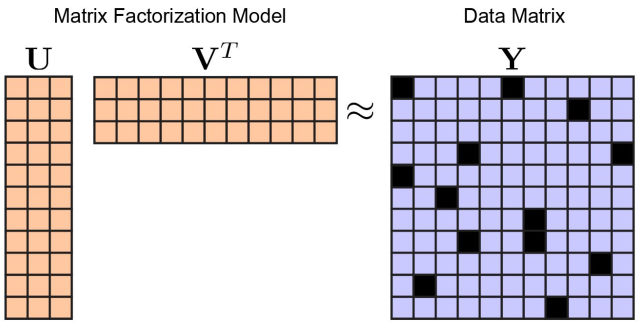
Kuchaiev, Oleksii, and Boris Ginsburg. "Training deep autoencoders for collaborative filtering." *arXiv preprint arXiv:1708.01715* (2017).

Evaluating Recommender Systems

- Inherently difficult, usually the same dataset in used for learning and evaluation, highy sparsity, bias and outliers.
- How to measure success? User **satisfaction** is hard to define!
- How to optimize for novelty and serendipity?
- In general, we try to optimize the pre-defined loss function (which does not guarantee best predictions or best user satisfactions).

Evaluating Recommender Systems

- We commonly use a holdout set (sub-matrix, random sample) on which we apply predictive accuracy metrics, such as RMSE, MAE, …
- For topN recommendations we typically use coverage measures, such as precision and recall (and F1).



Random Holdout

Alex Williams. *How to cross-validate PCA, clustering, and matrix decomposition models*. http://alexhwilliams.info/itsneuronalblog/2018/02/26/crossval/

YouTube - A short case study

- Largest platform for creating, sharing and discovering video content.
- And also one of the **largest** recommender system in industry!
- Main Goal: Maximize the **time** users spend on the platform (and thus maximize the number of served ads).
- Vast majority of views come from recommendations (more than 70 %).
- However, YouTube is not fully transparent how their algorithms work.

Spandana Singh. *Why Am I Seeing This?* https://www.newamerica.org/oti/reports/why-am-i-seeing-this/

History

- In the past YouTube recommended videos according to **click-rate**, but this incentivized users to create click-baity videos.
- YouTube then changed their strategy to recommend videos based on overall **watch time**.
- Emphasis on user satisfaction and quality watch time (hard to measure).
- Search results are also personalized.
- Offline metrics, such as precision, recall, ranking loss are widely used, however the most important metric is the live **A/B testing**.

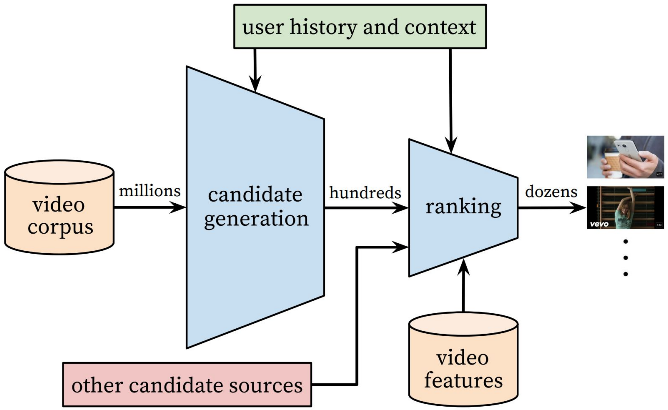
Deep Neural Networks for Youtube Recommendations

- Used matrix factorization in the past.
- One of first to use deep neural network for recommendations .
- Built upon TensorFlow.
- Two-stage design (with two deep neural networks), one for candidate generation and one for ranking.
- This two stage approach allows for recommendation from millions of different videos.

Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations." *Proceedings of the 10th ACM conference on recommender systems*. 2016.

The two-stage design

- The candidate generation uses collaborative filtering in order to provide broad personalization (using various features) with a high precision.
- Ranking network scores each video according to the desired objective function and thus provides fine-level representation among candidates with a high recall.

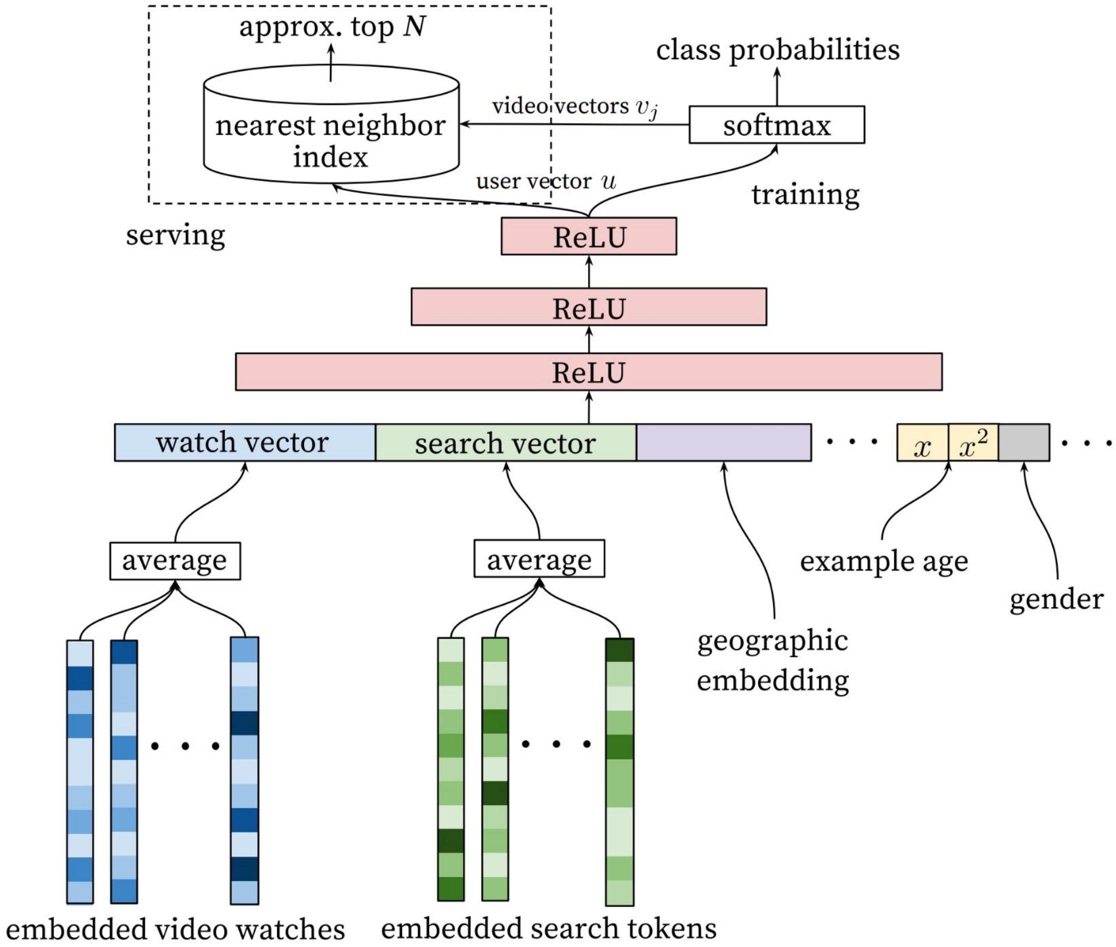


Candidate generation

- Neural networks as a generalization of matrix factorization (for embedding user's behaviours).
- Recommendation is viewed as an extreme multiclass (1M) classification.
- Explicit data is very sparse, but implicit (such as video views) is more abundant.
- User watch history is represented by a sequence of sparse video IDs (which is then embedded).
- Arbitrary continuous and categorical features can be easily added to the model.

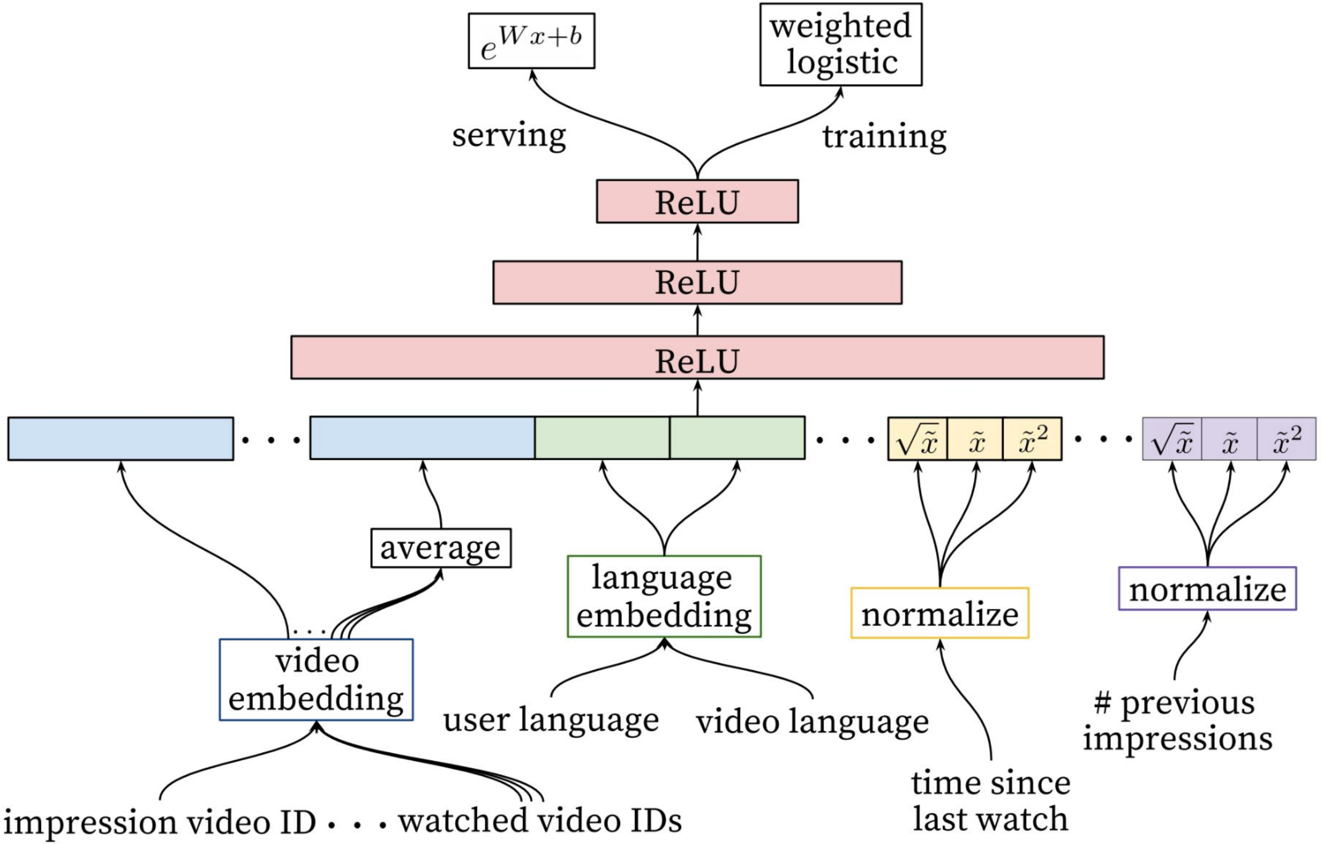
Candidate generation

- Age of the video is very important feature in order to recommend “fresh” examples.
- Candidates are found by nearest neighbour search from the last layer of the network.
- Limited number of training example per user (to avoid heavy raters).
- Adding features and layers significantly improves the accuracy.


approx. top N
nearest neighbor index
video vectors v_j
softmax
class probabilities
user vector u
training
serving
ReLU
ReLU
ReLU
watch vector
search vector
x
x^2
average
average
example age
gender
geographic embedding
embedded video watches
embedded search tokens

Ranking

- A similar deep neural network is used as for candidate selection.
- More video features (such as thumbnails, etc.) are used.
- Feature engineering is very important (although deep learning alleviates this problem).
- Shared embedding for similar features and normalization of continuous features.
- Adding features and layers significantly improves the accuracy.



e^{Wx+b}
weighted logistic
serving
training
ReLU
ReLU
ReLU
$\sqrt{\tilde{x}}$ $\tilde{x}$ $\tilde{x}^2$
$\sqrt{\tilde{x}}$ $\tilde{x}$ $\tilde{x}^2$
average
language embedding
normalize
normalize
video embedding
user language
video language
previous impressions
time since last watch
impression video ID
watched video IDs

Add-ons

- Such design is still not optimal (often only relevant/similar items are recommended and users can get bored).
- Google then introduced REINFORCE algorithm (reinforcement learning) with goal to maximize users' engagement over time by predicting which recommendations would expand their tastes.
- YouTube is constantly updating their machine learning algorithms and has multiple A/B tests in production.

Conclusion - Key Takeaways

- Offline metrics are a good indicator, but they **do not** always map perfectly to the online setting.
- A/B testing in production is a must.
- Continuous improvements and optimizations.
- Although deep networks help, manual feature engineering can still vastly improve the performance.
- Adding features and layers usually improve the accuracy (but has higher infrastructure costs and/or diminishing returns).
- Optimize for user satisfaction.

TensorFlow

What is TensorFlow?

TensorFlow

- *"An end-to-end open source machine learning platform"*
- At its core: general purpose ML framework, focused on DL
- Core implemented in C / C++ / Python, wrappers for Java / Go / Swift
- Peripherals: TF Serving (microservice for serving models), TensorBoard (model visualization / profiling), etc.

Pros & cons of TF

+ A lot of implemented operations and models

+ Optimized for speed, CPU, GPU support

+ Ecosystem (users, libraries, end-to-end platform) - most widely used DL framework

- Complexity

- Poor documentation + awkward API in some places

Code time

- Repository: github.com/janhartman/recsystf
- Option 1: run the notebooks on Google Colab
  - Colab: https://colab.research.google.com/
  - Click GitHub and enter *janhartman/recsystf*
  - All dependencies are preinstalled
  - The colab notebooks are stored on your GDrive
- Option 2: run the notebooks locally
  - Clone the repo and open it
  - Create virtualenv and install dependencies
  - Run *jupyter-lab*

Stuff we left out

- Saving and loading models
- TF Serving, production usage
- TF in other programming languages
- Custom op creation
- Estimators and tf.data pipelines

Sources (TF part)

- TensorFlow guide
- Deep Learning With Keras: Recommender Systems
- Collaborative Filtering for Movie Recommendations
- NVIDIA/DeepRecommender: Deep learning for recommender systems
- An implementation of DeepRecommender in Tensorflow & Keras

Challenge

Data

- Data: Goodreads book ratings, filtered
- 84k train, 21k test examples
- *train.csv* & *test.csv*: user ID, book name & rating
- Bonus: additional book data in *books.csv* (match by book name) - try to find a way to add it to the model
- *books.csv*: authors, publish date, publisher, rating, number of reviews, number of pages

Challenge

- Goal: implement a recommender system in TF to accurately predict a user's rating of a book
- Use mean squared error as your loss function & a validation dataset
- Play around with different algorithms, add extra book data into the model...
- Try to achieve the lowest MSE on the test set (mine: 0.84)
- Make sure to use additional data from *books.csv*

Conclusion

We are hiring!

- Always looking for interesting candidates
- Various positions are open
- Hard and interesting problems
- Big data
- outbrain.com/careers

Thank you for your attention!

Martin Jakomin

mjakomin@outbrain.com

linkedin.com/in/martin-jakomin

Jan Hartman

jhartman@outbrain.com

linkedin.com/in/jan-hartman