Autoencoder neural networks as recommendation engines

Filip Wójcik

Senior Data Scientist
PhD Candidate, UE Wroc

filip.wojcik@outlook.com
https://filip-wojcik.com

presentation agenda

business problem definition

What is the problem to solve?

recommendation systems

What are they and how do they work? Why do we use tchem?

classic approaches

What approaches have been used so far?

presentation agenda

autoencoders

Specific type of neural network used to rebuild the matrix

a case study

Comparison between classic algorithms and the autorec system

Business problem

business problem

Given an existing user base and a set of products, how we can make recommendation systems better – to improve hints quality, stability of the system as well as to make it more robust to rapidly changing environment?

business problem

01 How Company can improve recommendations?

Given a large number of ratings / users, how the Company can improve suggestings/recomnnedations given to tchem?

02 How a system can become more inteligent?

Given a constantly changing environment, user comments and interaction with others – how a recommendation system can accept new data without complete redesign?

03 How to make system more flexible and universal?

Given a new data sources, how system can be adjusted to use them without a need to reimplement it?

executive summary

Autoencoder networks can be used as a recommendation engines

Properly designed autoencoder neural networks can be used as a recommendation engines – learning **hidden (latent)** patterns from users' behaviour

Autoencoders outperform classical approaches

Studies on artificial/benchmark datasets as well as on real cases show, that autorencoder recommendaation engines can outpuerform calssical approaches like **collaborative filtering** or **matrix decomposition**

Autoencoders are more flexible and able to use other data sources

Autoencoders can be designed using different NN architectures, including also „static data" processing – e.g. **additional information** about the client or the product. Therefore they can **combine features of content-based and collaborative recommendation engines.**

Recommendation Systems

What are they? How do they work and what they are used for? What are the most common issues they can fall into?

01

recommendation systems

Preference analysis

- Consumer behaviour analysis
- Attempting to find behaviour patterns
- Search for similarities between people and products
- A difficult task at a time when the product offer is very wide

Representation

- Users ' or customers ' preferences are most often seen as ratings
- User ratings are the base material for a recommendation engine
- The System does not have access to most of the variables describing people and products

Goal

- Attempting to reconstruct hidden (latent) factors influencing decisions
- On the basis of such reconstruction, anticipating future behaviour
- Recommend products that conform to users’ preferences
- This overall idea can be implemented in many ways

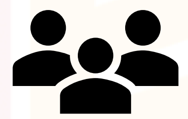
recommendation systems

01 moderate activity of users

The amount of goods purchased by customers is usually small in relation to the entire offer, and often people make individual purchases in a particular store

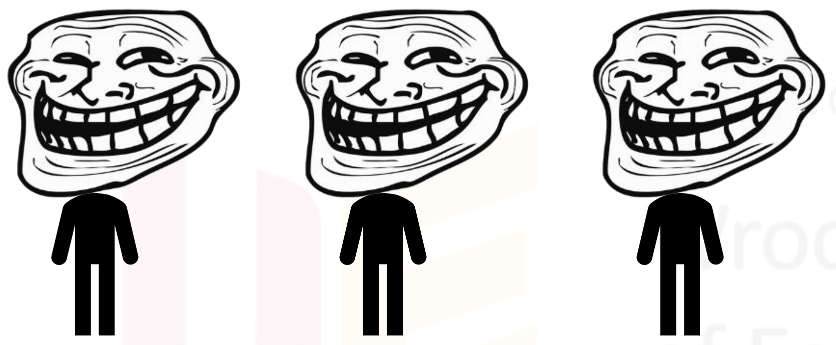
02 problem of cold start

It's hard to recommend anything to new users if their preferences are unknown.



03 significant bias of ratings

The problem of negative reviews on the internet is widely known. People are often very critical, or just don't want to leave any positive feedback.



Classic Algorithms

There are many architectures of recommendation engines. The concepts used in them are also used by autoencoders.

02

classic algorithms

approach based on models + *content based filtering*

An attempt to frame recommendation problem as a classic machine learning task. Requires knowledge of attributes that characterising users and products. Then they are connected with each other to make a prediction

collaborative filtering

The rating matrix and distance measure are sufficient to operate these types of systems. An algorithm searches for vectors similar to a given user/product. Recommends items that are „missing" from the currently processed using the appropriate formula

latent factors model

An approach based on matrix decomposition and analysis of hidden (latent) factors. The matrix decomposition is intended to reveal invisible connections between users and latent features (factors) as well as products and latent features (factors). On this basis, new elements are suggested. Mathematical decomposition of matrices – e.g. Svd, NNMF, etc.

classic algorithms

approach based on models + *content based filtering*

An attempt to frame recommendation problem as a classic machine learning task. Requires knowledge of attributes that characterising users and products. Then they are connected with each other to make a prediction

User feature 1	User feature 2	...	User feature n	Item feature 1	Item feature 2	...	Item feature m	Rating

$$
\mathbf{D} = \left(\begin{array}{c|cccc}
 & X_1 & X_2 & \cdots & X_d \\
\hline
\mathbf{x}_1 & x_{11} & x_{12} & \cdots & x_{1d} \\
\mathbf{x}_2 & x_{21} & x_{22} & \cdots & x_{2d} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\mathbf{x}_n & x_{n1} & x_{n2} & \cdots & x_{nd}
\end{array}\right)
$$

classic algorithms

collaborative filtering

The rating matrix and distance measure are sufficient to operate these types of systems. An algorithm searches for vectors similar to a given user/product. Recommends items that are „missing” from the currently processed using the appropriate formula

KNN

$$s_{ij}(x_i, x_j) = \text{cosine}(\theta)$$

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5		?	5	?	4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4	?		2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- unknown rating - rating between 1 to 5

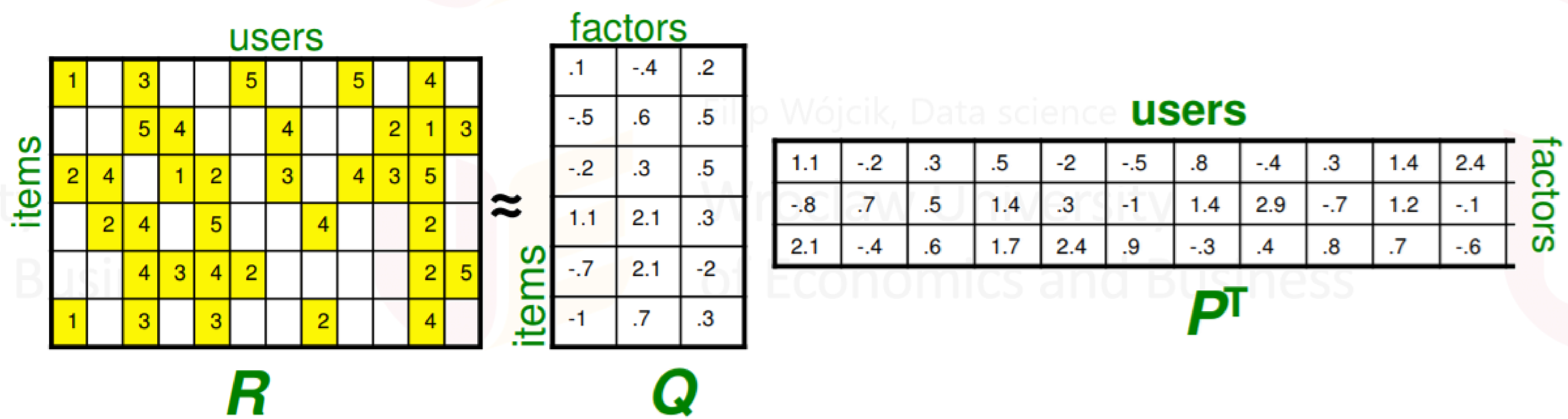
$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

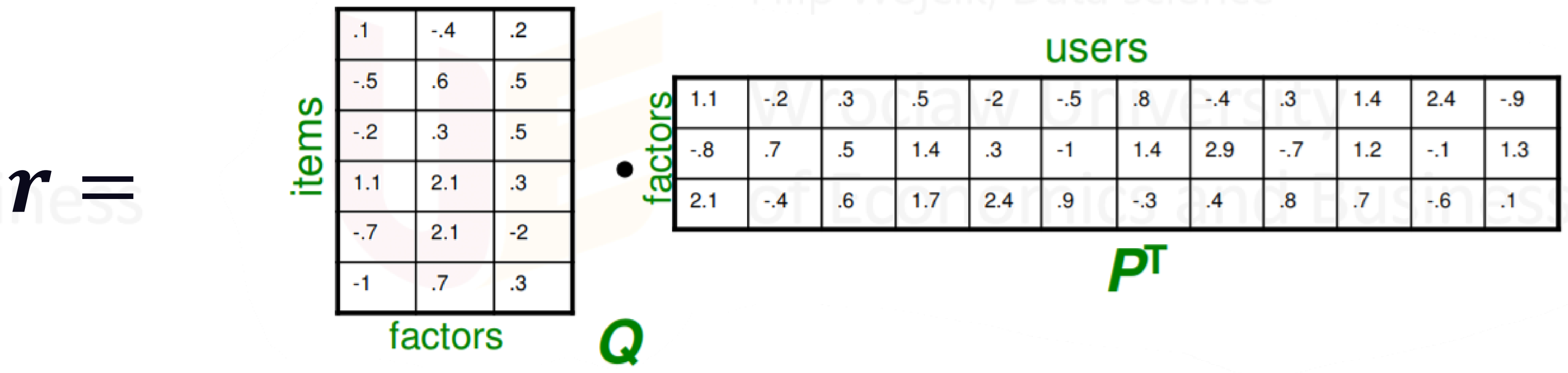
Rajaraman, A. and Ullman, J.D., 2011. *Mining of massive datasets*. Cambridge University Press.

classic algorithms

latent factors model

An approach based on matrix decomposition and analysis of hidden (latent) factors. The matrix decomposition is intended to reveal invisible connections between users and latent features (factors) as well as products and latent features (factors). On this basis, new elements are suggested. Mathematical decomposition of matrices – e.g. Svd, NNMF, etc.





Rajaraman, A. and Ullman, J.D., 2011. *Mining of massive datasets*. Cambridge University Press.

03

Neural networks with proper structure can be used to express latent factors, just like matrix decomposition. This approach is widely used in e.g. image processing.

autoencoders

Autoencodery rekomendacyjne

AutoRec: Autoencoders Meet Collaborative Filtering

Suvash Sedhain[†*], Aditya Krishna Menon[†*], Scott Sanner[†*], Lexing Xie[*†]
[†] NICTA, [*] Australian National University
suvash.sedhain@anu.edu.au, { aditya.menon, scott.sanner }@nicta.com.au,
lexing.xie@anu.edu.au

ABSTRACT
This paper proposes AutoRec, a novel autoencoder framework for collaborative filtering (CF). Empirically, AutoRec's compact and efficiently trainable model outperforms state-of-the-art CF techniques (biased matrix factorization, RBM-CF and LLORMA) on the Movielens and Netflix datasets.

Categories and Subject Descriptors D.2.8 [**Information Storage and Retrieval**]Information Filtering

Keywords Recommender Systems; Collaborative Filtering; Autoencoders

1. INTRODUCTION
Collaborative filtering (CF) models aim to exploit information about users' preferences for items (e.g. star ratings) to provide personalised recommendations. Owing to the Netflix challenge, a panoply of different CF models have been proposed, with popular choices being matrix factorisation [1, 2] and neighbourhood models [5]. This paper proposes *AutoRec*, a new CF model based on the autoencoder paradigm; our interest in this paradigm stems from the recent successes of (deep) neural network models for vision and speech tasks. We argue that AutoRec has representational and computational advantages over existing neural approaches to CF [4], and demonstrate empirically that it outperforms the current state-of-the-art methods.

2. THE AUTOREC MODEL
In rating-based collaborative filtering, we have m users,

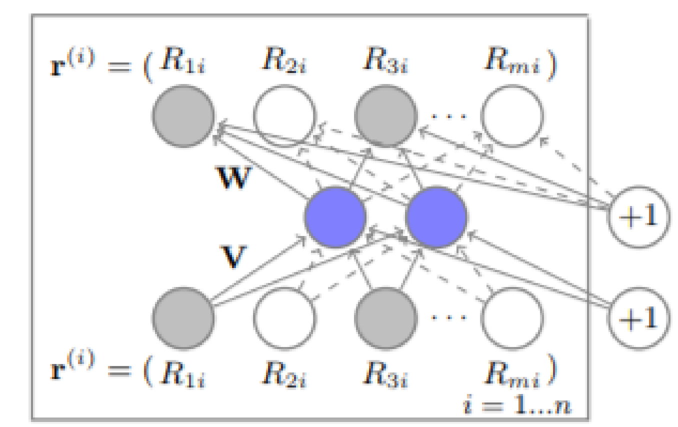


Figure 1: Item-based AutoRec model. We use plate notation to indicate that there are n copies of the neural network (one for each item), where $\mathbf{W}$ and $\mathbf{V}$ are tied across all copies.

where $h(\mathbf{r};\theta)$ is the *reconstruction* of input $\mathbf{r} \in \mathbb{R}^d$,

$$h(\mathbf{r};\theta) = f\left(\mathbf{W} \cdot g(\mathbf{V}\mathbf{r} + \boldsymbol{\mu}) + \mathbf{b}\right)$$

for *activation functions* $f(\cdot), g(\cdot)$. Here, $\theta = \{\mathbf{W}, \mathbf{V}, \boldsymbol{\mu}, b\}$ for transformations $\mathbf{W} \in \mathbb{R}^{d \times k}, \mathbf{V} \in \mathbb{R}^{k \times d}$, and biases $\mu \in \mathbb{R}^k, \mathbf{b} \in \mathbb{R}^d$. This objective corresponds to an auto-associative neural network with a single, k-dimensional hidden layer. The parameters θ are learned using backpropagation.

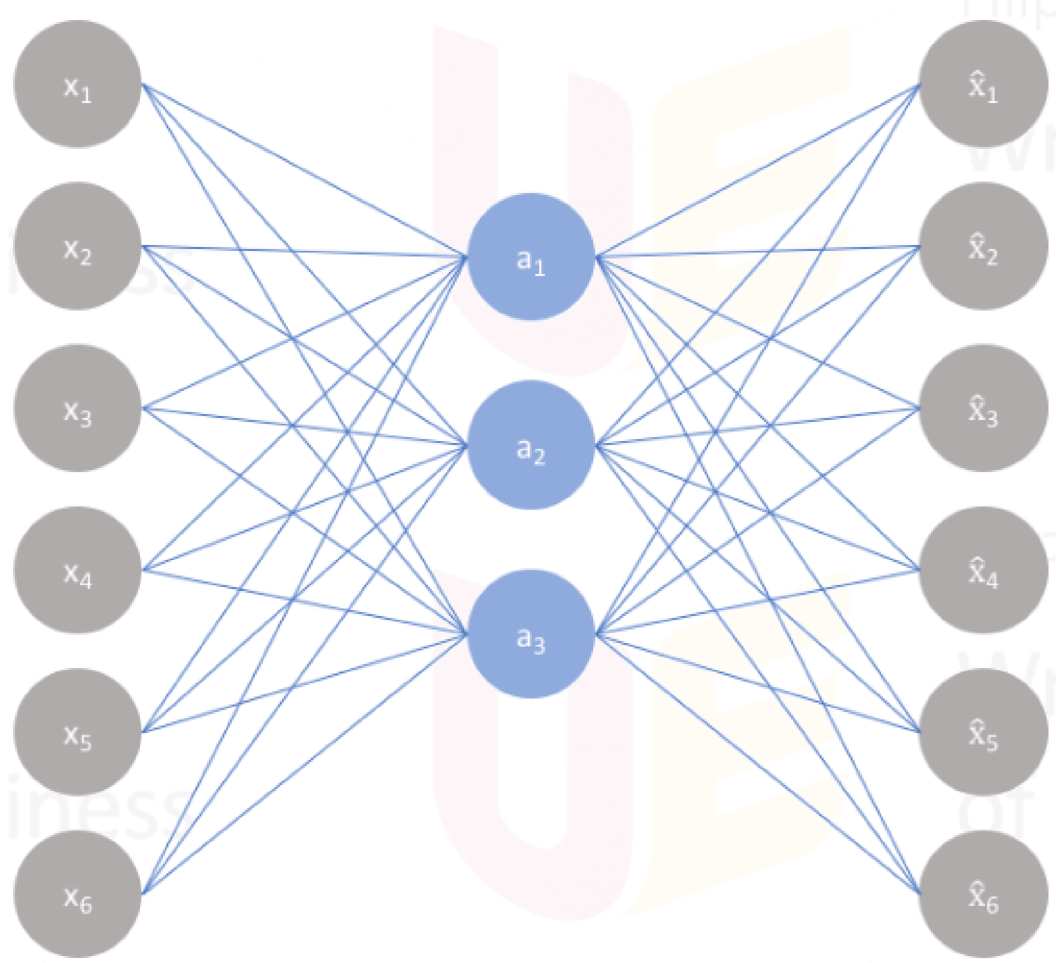
The item-based AutoRec model, shown in Figure 1, applies an autoencoder as per Equation 1 to the set of vectors $\{\mathbf{r}^{(i)}\}_{i=1}^{n}$, with two important changes. First, we account for the fact that each $\mathbf{r}^{(i)}$ is partially observed by only updating during backpropagation those weights that are associated with observed inputs, as is common in matrix factorisation

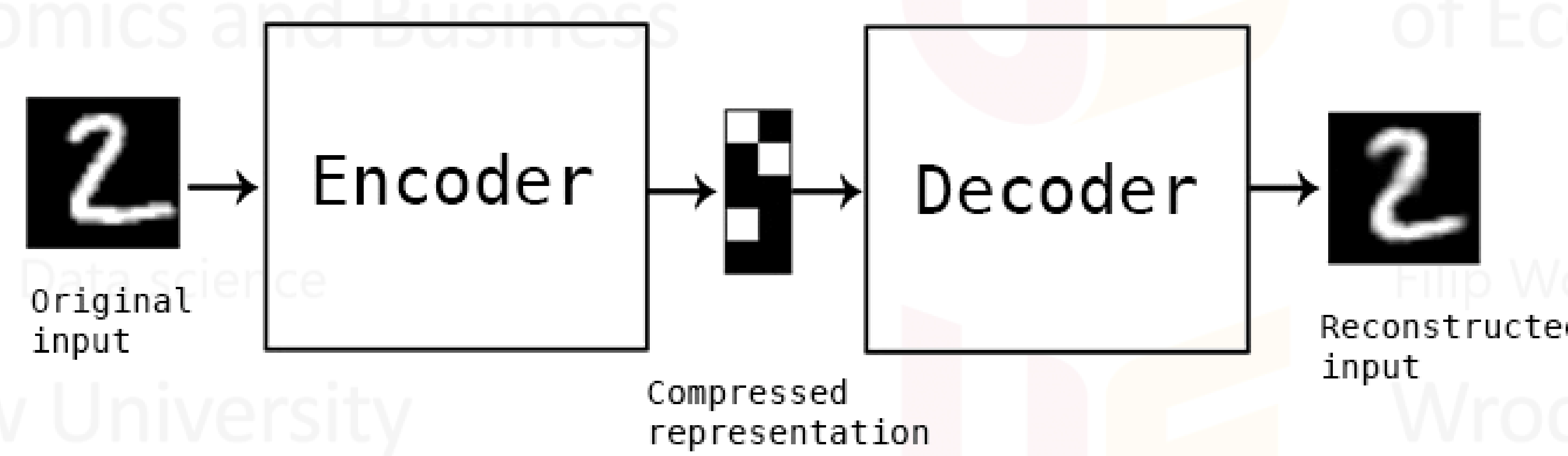
2015

autoencoders
characteristics

input reconstructed on output

Autoencoders accept input and map them to the output. So there's no classic classification or regression – it's about recreating. A classic example of use is the image denoising.



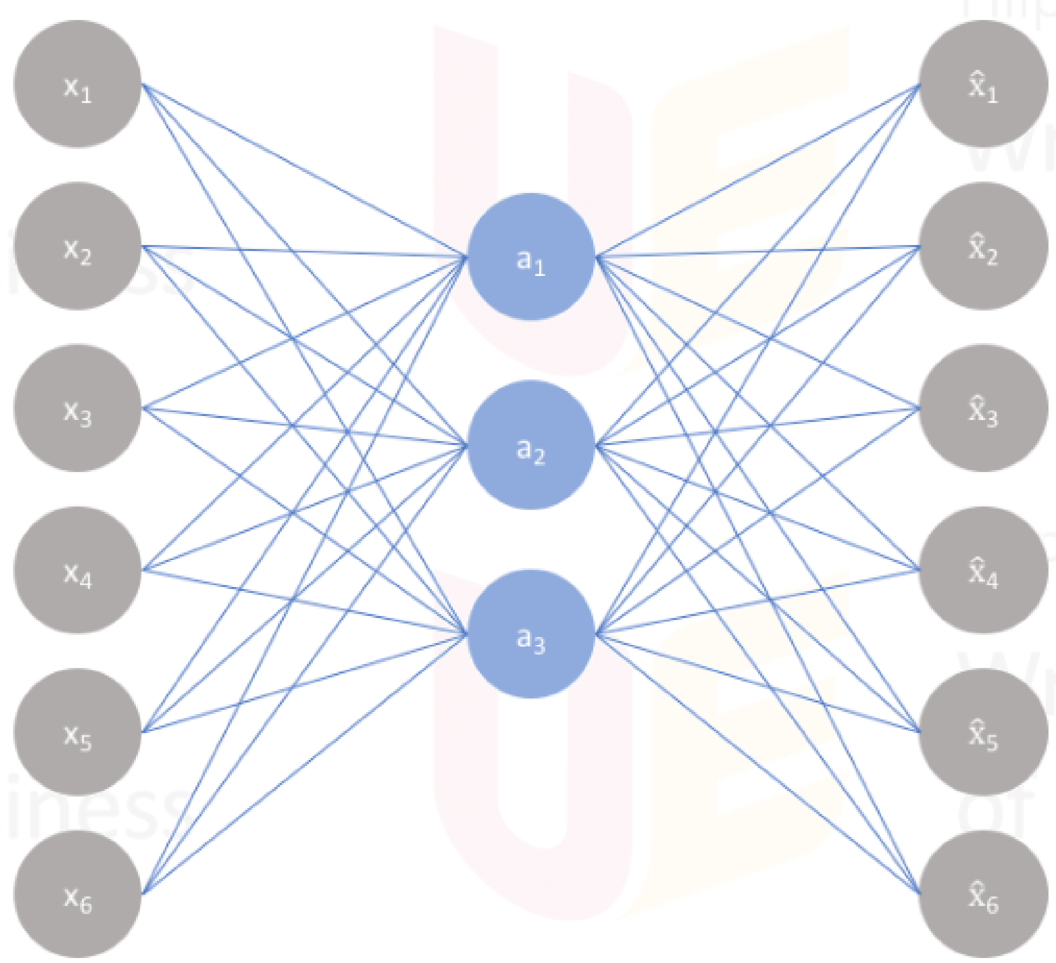


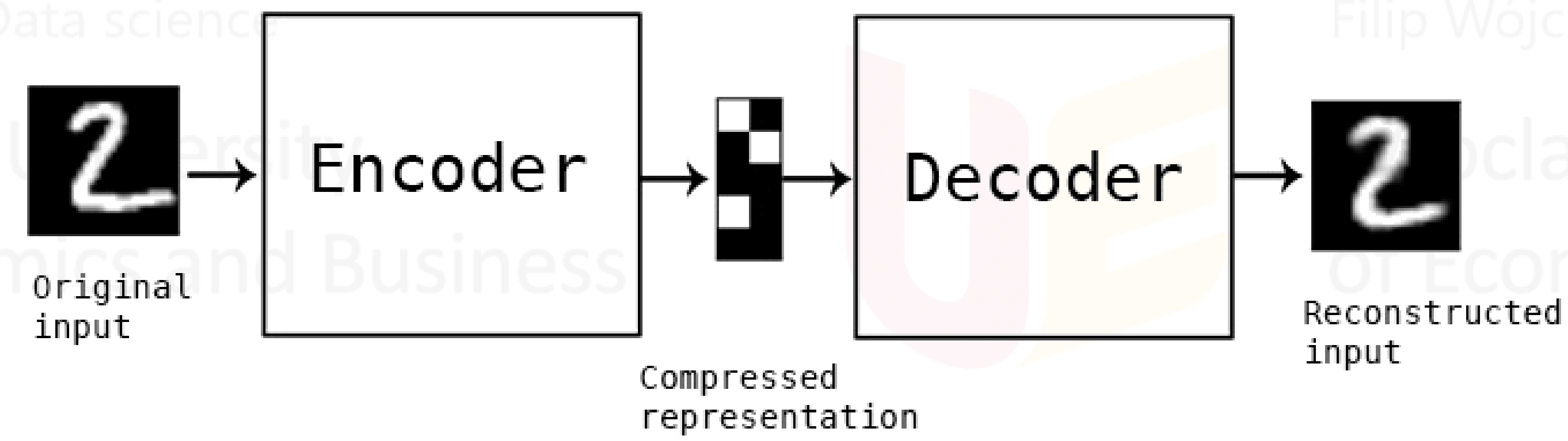
https://blog.keras.io/building-autoencoders-in-keras.html

autoencoders
characteristics

compression of latent representation

When processing input into output, Autoencoder perform compression. This is the same as making a non-linear reduction in dimensionality. Such a compressed dimension can be interpreted as latent variables/factors!





items × factors: **Q**

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

•

factors × users: P^T

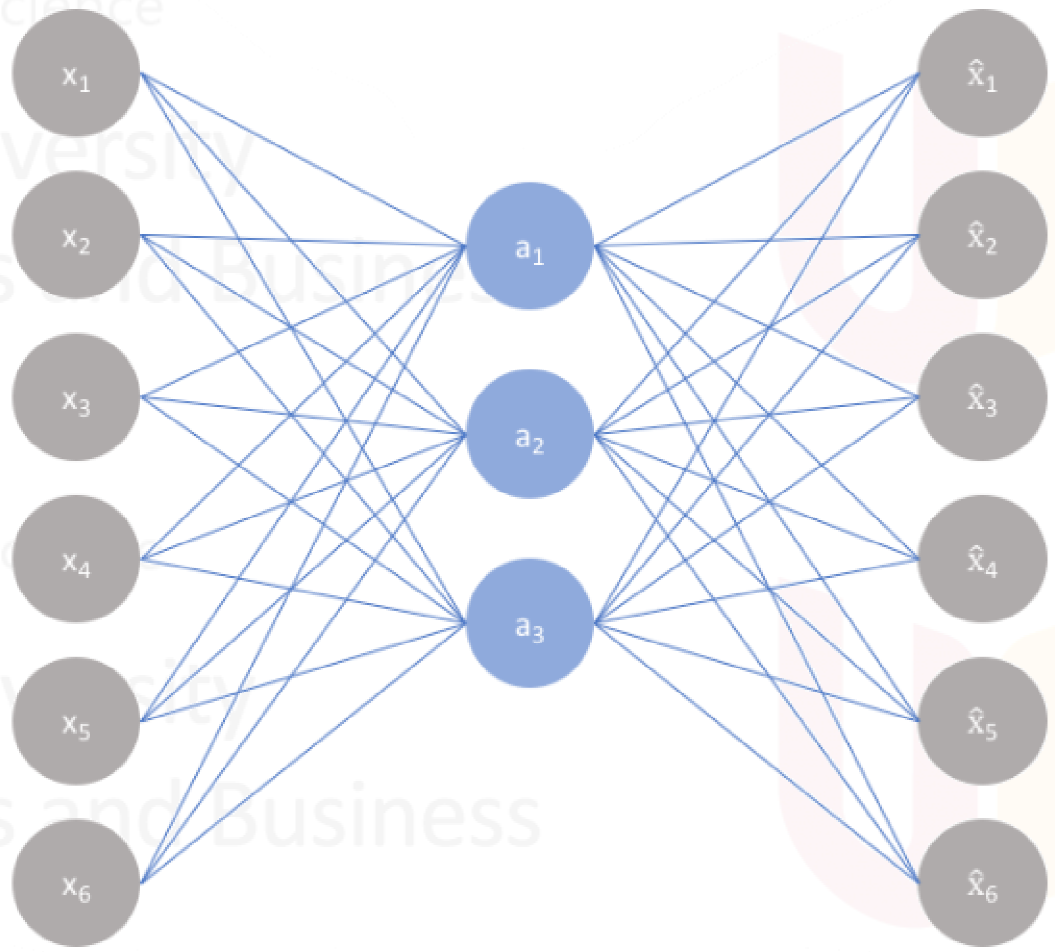
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

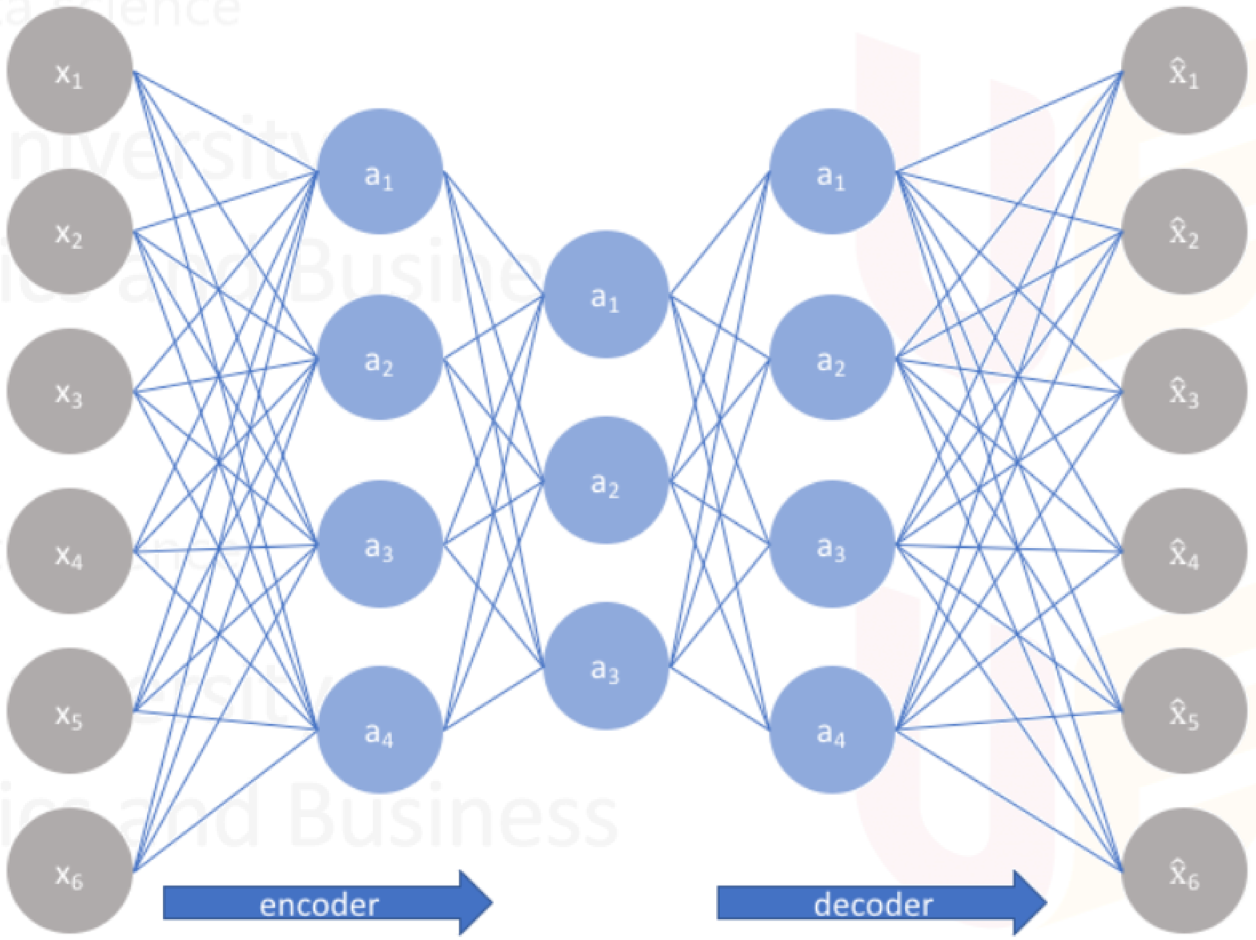
https://blog.keras.io/building-autoencoders-in-keras.html

autoencoders characteristics

flexible architecture

Autoencoders can take any form – from simple networks with one hidden layer, to deep networks with multiple layers of compression, to deep stacked autoencoders (autoencoders compiled independently into one network)



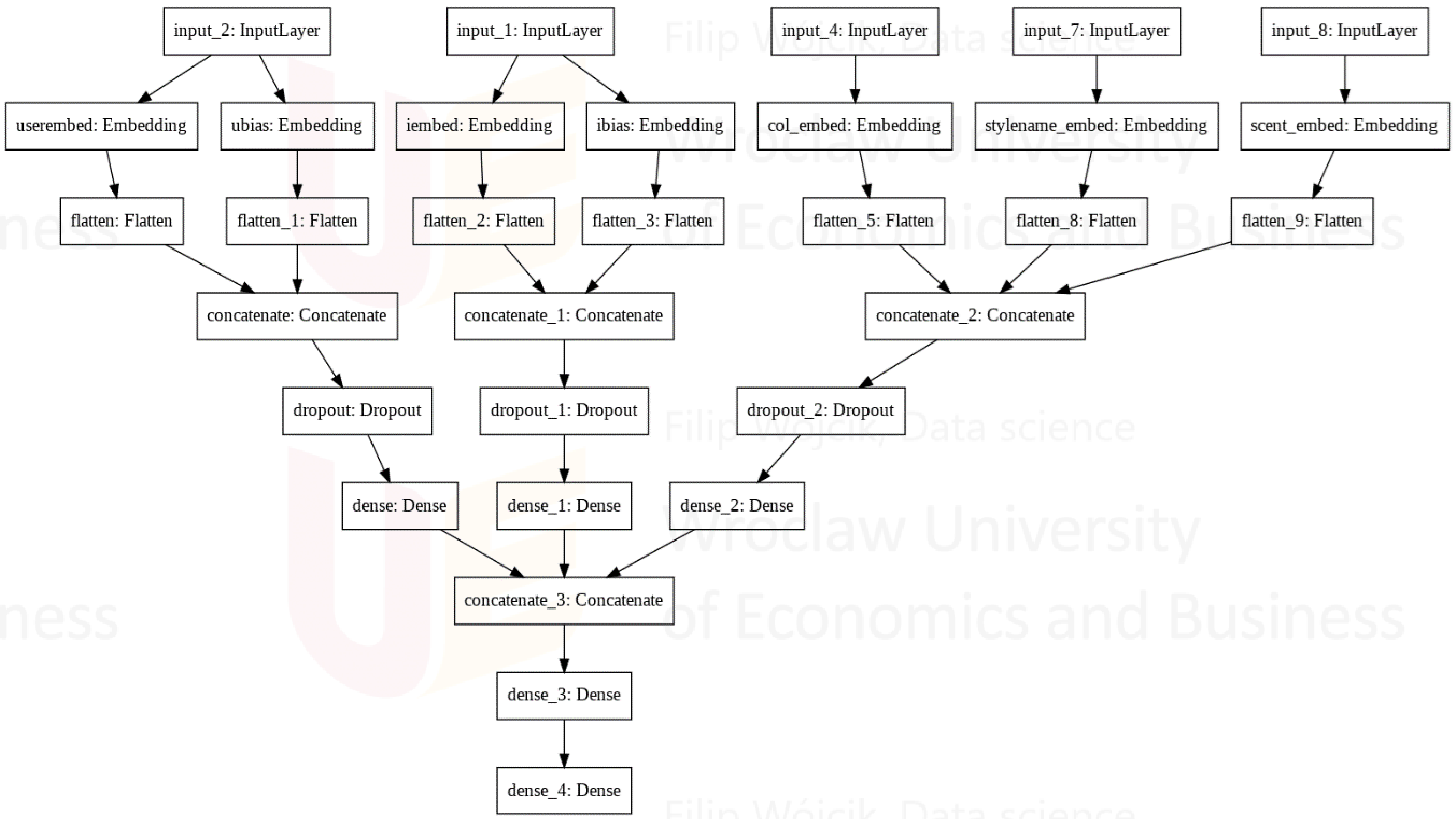


https://www.jeremyjordan.me/autoencoders/

autoencoders characteristics

hybrid networks

Hybrid autoencoders can use additional information as input – not only ratings matrix, but also item/user descriptions or features. This makes them similar to model-based approaches, where a system can utilize external information.



autoencoders characteristics

simple and flexible training

Autoencoders can be trained like any other deep neural netowork or using **greedy layer-wise pretraining,** where every partial autoencoder is trained separately and then added to the stack.
Autoencoders can be also trained with shared input/output weights (one of the most popular methods initially).

$$l \ -\ loss\ function$$
$$o \ -\ autoencoder's\ input$$
$$\hat{a} \ -\ activation\ function\ on\ output$$
$$h(x) -\ activation\ function\ on\ input$$
$$W \ -\ weights\ (can\ be\ shared)$$

$$o(\hat{a}(x)) = o(c + W^T h(x)) = o(c + W^T \sigma(b + Wx))$$

$$\frac{\partial l}{\partial W_{ij}} = \frac{\partial l}{\partial \hat{a}_j}\frac{\partial \hat{a}_j}{\partial W_{ij}} = \frac{\partial l}{\partial \hat{a}_j}(h_i + W_{ij}\frac{\partial h_i}{\partial W_{ij}}) = \frac{\partial l}{\partial \hat{a}_j}h_i + \frac{\partial l}{\partial a_i}x_j$$

training schedule for autoencoders

In the case of a recommendation system, the mere reconstruction of a matrix from latent variables is an intermediate task.
The real goal is to check the generalization of the system.

Error in reconstruction– informs how well autoencoder „understood” latent variables
Error in prediction – helps in assessing predictive power and generalization to unseen ratings

User \ Movie	M1	M2	...	Mn
U1	5.0	3.0		-
U2	3.5	-		3.5
U3	4.0	4.5		-
U4	1.0	1.5		5.0
U5	3.5	5.0		4.0
U6	3.5	4.5		3.5

autoencoders
characteristics

Train data
reconstruction

User \ Movie	M1	M2	...	Mn
U1	5.0	3.0		-
U2	3.5	-		3.5

Validation data
reconstruction

U3	4.0	4.5		-
U4	1.0	1.5		5.0

Test data
reconstruction +
prediction

U5	?	5.0		?
U6	3.5	?		3.5

autoencoders characteristics

W_{ih}, W_{ho} – *input / output weight (can be shared)*
b_{in}, b_{out} – *input / output bias*
θ – *set of all autoencoder params* (W, b)
$g(x)$ – *nonlinear inner activation function*
$f(z)$ – *nonlinear output activation function*
$\boldsymbol{r}$ – *a matrix of* ***observed*** *(really existing, not missing) ratings*
$\boldsymbol{y}$ – *vector of* **"hidden" ratings** *from test set*
$h(\boldsymbol{r};\boldsymbol{\theta})$ – *functional autoencoder description*

$$\mathrm{h}(\mathbf{r};\theta) = \mathrm{f}(W_{out} \cdot g(W_{in} \cdot \boldsymbol{r} + b_{in}) + b_{out})$$

Training data
reconstruction

Validation data
reconstruction

Test data
reconstruction+
prediction

$$\mathcal{L}_1(\boldsymbol{r}, h(\boldsymbol{r};\theta)) = \mathcal{L}_1(\boldsymbol{r}, \hat{\boldsymbol{r}}) = \frac{1}{|\boldsymbol{r}|}\sqrt{\sum_{i=1}^{r}(r_i - \hat{r}_i)^2}$$

Error in reconstruction: **MSE**

$$\mathcal{L}_2(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

Error in prediction: **any**

Goodfellow, I., Bengio, Y. i Courville, A. (2016) *Deep Learning*. MIT Press.

autoencoders
characteristics

Training data
reconstruction

Validation data
reconstruction

Test data
reconstruction+
prediction

$$\mathcal{L}_1(\boldsymbol{r}, h(\boldsymbol{r};\theta)) = \mathcal{L}_1(\boldsymbol{r}, \hat{\boldsymbol{r}}) = \frac{1}{|\boldsymbol{r}|}\sqrt{\sum_{i=1}^{\boldsymbol{r}}(r_i - \hat{r}_i)^2}$$

$$\mathcal{L}_2(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

Error in reconstruction: **MSE**

Error in prediction: **any**

$\boldsymbol{r}$

User \ Movie	M1	M2	...	Mn
U1	5.0	3.0		-
U2	3.5	-		3.5

$\hat{\boldsymbol{r}}$

User \ Movie	M1	M2	...	Mn
U1	4.874	4.001		-
U2	3.222	-		3.654

U5	?	5.0		?
U6	3.5	?		3.5

$$\boldsymbol{y} = [3.5, 4.5, 4.0\ ]$$

$$\hat{\boldsymbol{y}} = [3.223, 4.894, 3.999\ ]$$

Goodfellow, I., Bengio, Y. i Courville, A. (2016) *Deep Learning*. MIT Press.

04

Example use case

How Autorec system can be used as a recommendation engine in a real-world dataset?

a case study on Amazon 2018 dataset

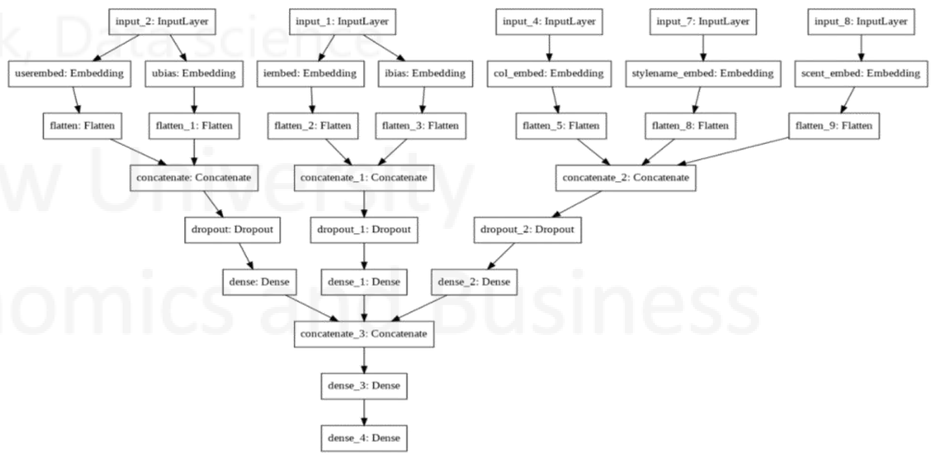
- Amazon sales dataset, category „All beauty” & „fashion”
- 5269 reviews in a whole dataset
- Biased ratings
- Experimented with 3x repeated 10-fold cross validation
- Additional product-relevant data:
  - Size/type/style/design
  - 400 unique text values

example
use case

a case study on Amazon 2018 dataset

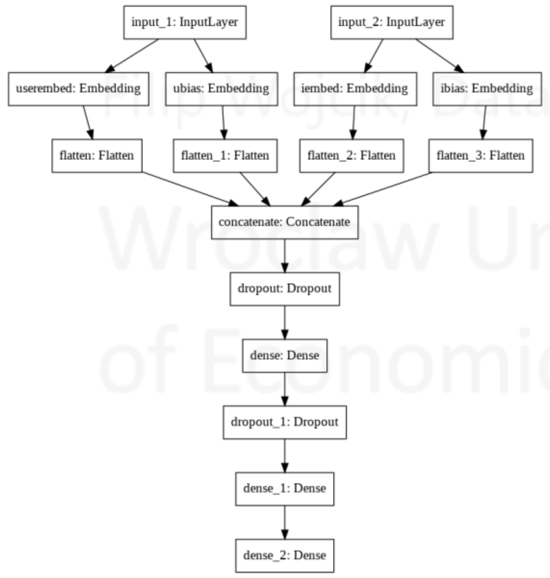
Deep Hybrid Collaborative Filtering [DHCF]

- (Rating input + user data input + content input)– Encoder – Decoder
- Relu activation function
- Dropout on input to simulate missing ratings
- 256 latent states



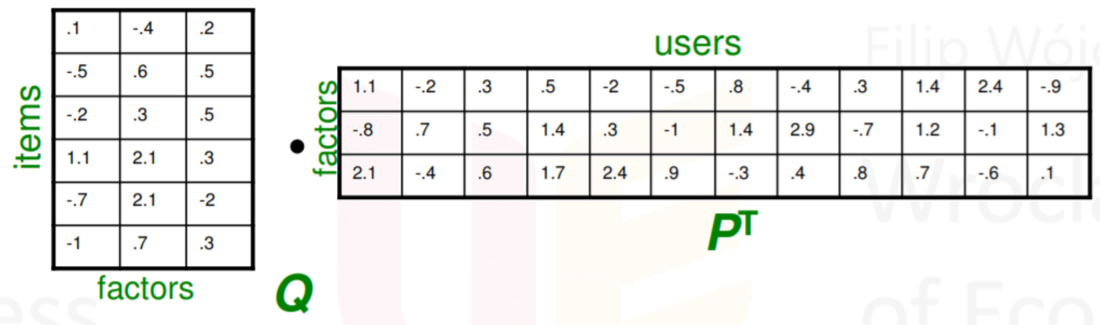
Deep Collaborative Filtering [DCF]

- Input – (Encoder –Encoder 2 – Decoder – Decoder2)
- Relu activation function
- Dropout on input to simulate missing ratings
- Regularization as well as intermediate dropout
- 256 latent states



Collaborative filtering [CF]

- Classic matrix decomposition model + user/item bias inclusion
- Builds user/item embeddings and performs decomposition
- 256 latent states



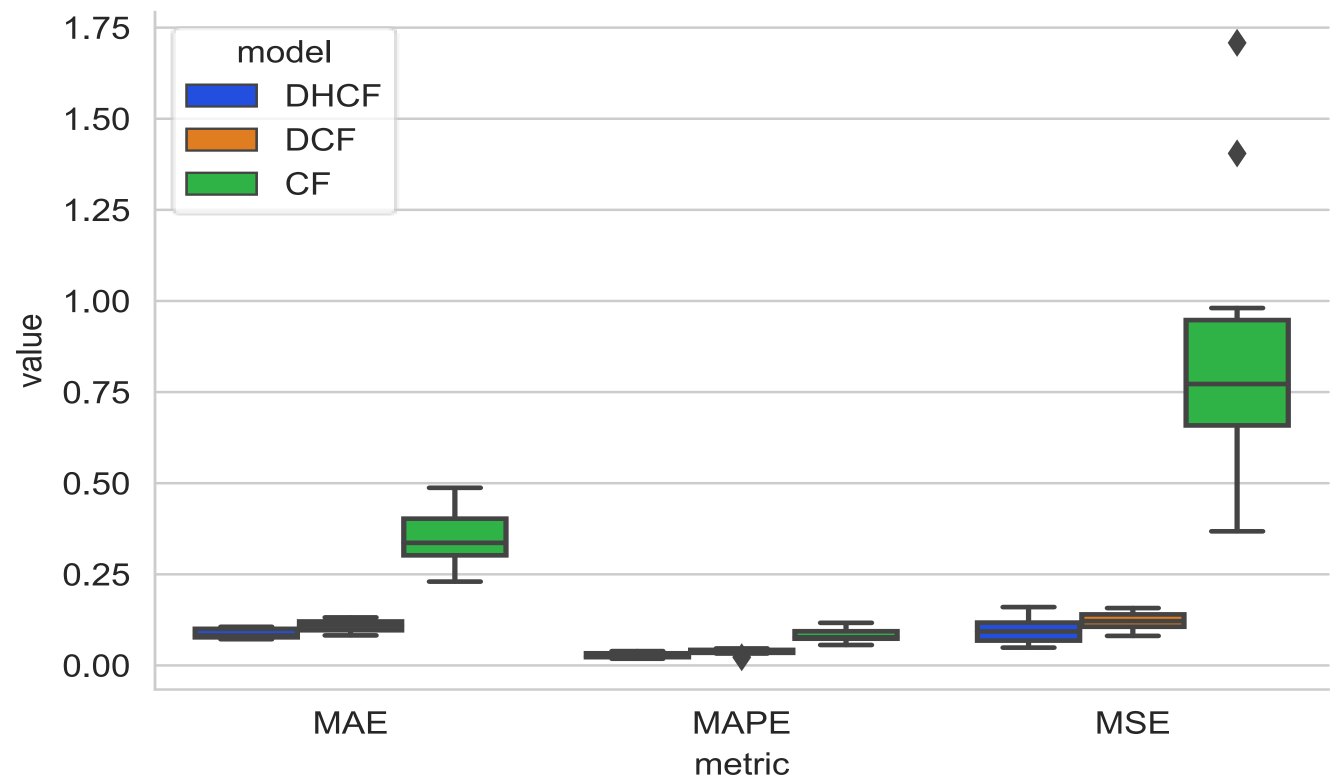
example
use case

a case study on Amazon 2018 dataset

Model 1	Model 2	Metrics diff, pval		
		MSE	MAE	MAPE
DHCF	DCF	0.055	**0.0023**	**0.002**
DHCF	CF	$< \mathbf{0.001}$	$< \mathbf{0.001}$	$< \mathbf{0.001}$
DCF	CF	$< \mathbf{0.001}$	$< \mathbf{0.001}$	$< \mathbf{0.001}$

Test Metrics comparison (std in brackets)

Model\Metric	Test MSE	Test MAE	Test MAPE
DHCF	**0.1698 (0.76)**	**0.1691 (0.375)**	**0.065 (0.24)**
DCF	0.5392 (2.29)	0.3592 (0.63)	0.139 (0.49)
CF	22.8573 (4.9)	4.7270 (0.71)	0.986 (0.02)



example
use case

a case study on Amazon 2018 dataset

- **Both Autorecommender** versions proven to be **significantly better** than Collaborative Filtering approach
- **Hybrid Autorecommender** proven to be marginally better than Deep Autorecommender
- Hybrid Autorecommender proven to **converge faster** although can become unstable during training
- Both autorecommender implementation provide **the same funcitonality** as Collaborative Filtering – latent states directly correspond to factorized matrix
- Additionally – autorecommender's **flexible architecture** makes them much more usable

research publications

Li, Sheng, Jaya Kawale, and Yun Fu. "Deep collaborative filtering via marginalized denoising auto-encoder." *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015.

Li, Xiaopeng, and James She. "Collaborative variational autoencoder for recommender systems." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2017.

Ouyang, Yuanxin, et al. "Autoencoder-based collaborative filtering." *International Conference on Neural Information Processing*. Springer, Cham, 2014.

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

Wang, Hao, S. H. I. Xingjian, and Dit-Yan Yeung. "Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks." *Advances in Neural Information Processing Systems*. 2016.

Wójcik, F., and Górnik, M. „Improvement of e-commerce recommendation systems with Deep Hybrid Collaborative Filtering with content: A case study”. Econometrics. Ekonometria. Advances in Applied Data Analysis, 24(3), 2020

Thank you