Recommendation Systems Course - Final Project Report

by Gil Levy and Liad Levi-Raz

AutoRec Paper Summary

The anchor paper is: **AutoRec: Autoencoders Meet Collaborative Filtering (2015)**[1]

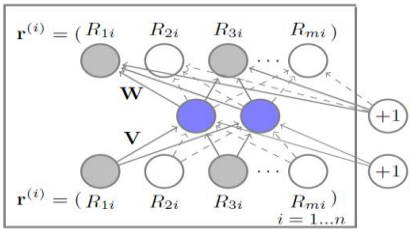
The main objective of the paper was to present the "AutoRec", which was a novel **Autoencoder** approach for a **Collaborative Filtering (CF)** recommendation system. This paper outperformed the state-of-the-art CF techniques when the paper was published (2015) on the Movielens and Netflix datasets. The paper describes a user based (U-AutoRec) and item-based (I-AutoRec) rating AutoEncoder (on **explicit ratings** dataset), which takes as input each sparse vector of a user's / Item's ratings, projects it into a low-dimensional latent (hidden) space, and then reconstructs in the output space to predict missing ratings for purposes of recommendation. With AutoRec the paper shows the following RMSE improvement (on different datasets):

- **2.7% - 5.3%** I-AutoRec vs baseline I-RBM*
- **1.7% - (-5%)** U-AutoRec vs baseline U-RBM*
- **1.7% - 2.7%** I-AutoRec vs state of the art Matrix Factorization with bias

(RBM stands for "Restricted Boltzmann Machine", it is not covered in this report)

Note: that the paper does not discuss **ranking** metrics. This is further addressed by our work. The architecture used in the paper was using a **single hidden layer** and an output layer to predict the ratings of user / Items to all existing items / users.

When mapped to the general autoencoder architecture there is a single linear (dense) layer for each of the encoder / decoder components. Experiments show best results for latent space = 500 with a moderate improvement over 200.



<< Figure1: From bottom to top, the users' rating vector r(i), is projected into a lower latent space (purple circles), and then reconstructed back to the output space (Image from the original paper(1)) Shaded nodes corresponding to observed ratings, and solid connections corresponding to weights that are updated for the input r(i).

The architecture can be formalized by the following formula: *f,g* are the Decoder and Encoder activation functions. The best results achieved with a Sigmoid for Encoder and an Identity for the Decoder:

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

Training: In a general autoencoder the loss (MSE) is the L2 distance between the original vector and the constructed vector. As in our case the input vectors are highly sparse, the loss incorporates RMSE only for the dimensions (ratings) that exist in the input vector. As in Matrix Factorization L2 regularization is applied to avoid overfitting. **I-AutoRec** achieved a lower RMSE compared to **U-AutoRec**. This is explained by a **higher number of ratings per Item than per user** (lower sparsity), and is in line with the baseline neighborhood algorithms results, where item to item CF results are superior over user to user CF.

Compared to Matrix Factorization (MF) approaches, which embed both users and items into a shared latent space, the I- AutoRec model only embeds items into the latent space. Further, while MF learns a linear latent presentation, AutoRec can learn a nonlinear latent representation through activation function (This was also achieved by later work such as NeuMF)

Innovations

We have decided to attempt in parallel **two innovations**:

1. **Hybrid architecture:** Added content / user data to the rating data. A hybrid approach (collaborative filtering and content) has the potential of solving the **cold start problem** and in general **improving the rating and ranking metrics**.
2. **Deep AutoRec architecture**. We have investigated the "next step" of AutoRec by applying deeper autoencoder architecture with and without constraining the total computational power. This is also addressed in the original AutoRec paper as the author's next stage.

General note: As indicated by the best works done for each of the datasets it seems that improving the metrics is very difficult. (Refer to results and conclusion sections).

Hybrid Architecture

We have taken the advantage that both of our datasets have additional content-based information and user-based information. For each of the original paper algorithms (U-AutoRec, I-AutoRec) we have proposed an improvement by a side band information. The side band information that describes the items / users is critical for solving the cold start problem where for a specific item we do not have any ratings (or too few) and must rate it based on the most popular that is not personalized. We also aimed to improve the general rating/ranking metrics however found it tough despites massive configurations tested. During our work we have found a reference to our approach that suggested an improvement to our

basic approach. We tried this paper proposal as an additional improvement: "Hybrid collaborative filtering with autoencoders."[5] and report both approaches results.

Architecture

After adding side band information, the rating matrixes are of the following form:

I-AutoRec Content

	Item Id				
User Id	**1**	**2**	**3**	**....**	**N**
1					
2	1		3		
3		2			4
....			5		
M					
Drama	1				
Kids		1			
.....			1		
Comedy				1	

U-AutoRec Content

	Item Id							
User Id	**1**	**2**	**3**	**...**	**N**	**Gender**	**Age Vector**	**Occupation vector**
1						1	0..1..0	0..1..0
2	1		3			0	0..1..0	0..1..0
3		2			4	1	0..1..0	0..1..0
....			5			1	0..1..0	0..1..0
M						0	0..1..0	0..1..0

Item based content: Includes the genres of the movie. Total of 18 different genres. An item can belong to more than one genre. We have also tried to incorporate much more content information as done in HW2 (words from movie title..) however finally results are reported only for genres as we did not achieve better results with more content info.

User based content: Includes gender , age and occupation (we have dropped the zip code that is also available in the dataset). Age was translated to a categorical feature by splitting to 12 non equal bins.)Younger ages bins are smaller as we believe that changes in taste are more frequent) and finally presented as a one hot vector. Occupation had a total of 21 values and translated to a one hot representation. Overall for a user 35 features of side information.

Train procedure: When applying the same AutoRec architecture to the updated rating matrix there is a challenge to provide the right weight to the new information in the loss function. This is critical as the side band information is way smaller than the rating information. We have tried two solutions:

1. Weighted loss: We increase the loss of the content information portion by multiplying its loss by a factor. This is denoted as content gain. By this we apply a higher loss to the decoder construction related to the side band information. We have defined the new loss by the following formula:

Loss = alpha*rating_loss + (1-alpha)*sideband_loss*contain_gain , where **alpha** = **raiting_info_size/total_size** and content gain is a hyper parameter. The main idea behind this approach is to force the model to map items / users that have similar content to closer values in the embedded space. The content gain is a hyper parameter.

2. A more advanced solution is suggested by the following paper: "Hybrid collaborative filtering with autoencoders"[5]. Inject the side band information at each layer of the encoder. In our case when only a single hidden layer exists, we inject the side ban information at both the input stage and to the hidden layer.

Note: This paper also suggests an additional improvement that we have not tested (The main reason is that it relates to Autoencoders and not specifically to adding sideband information where we have decided to focus): A few of the available ratings are ignored at the input (replaced with zero) while still required to be constructed correctly by the decoder. This somehow resembles dropout but for inputs. The final loss is a weight average RMSE on both normal inputs and those that were zeroed.

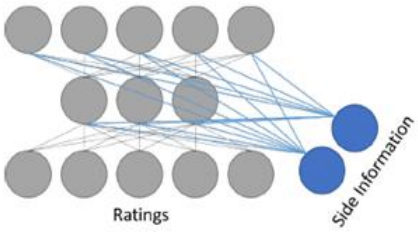


Figure 2: Integrating side information. The Network has two inputs: the classic Autoencoder rating input and a side information input. Side information is wired to every neurons in the Network.

Test procedure: We ignore the results of the side band predictions

Cold Start : As the main advantage of the side band information relates to the cold start case we have performed an additional special testing for this. The experiment is reported in detail in the Experimental section

Deep AutoRec Architecture

The final words on the "AutoRec paper" include a remark about a potential improvement using **Deeper Autoencoder networks**, see: "Do deep extensions of AutoRec help?". We have explored this suggestion by experimenting with various AutoEncoder architectures on Movielens 1M and 100K.
Our assumption was that using deeper models would achieve a better performance (spoiler… the results were rather disappointing…). We were inspired by **two additional papers** that addressed Deep AutoEncoders, "Training deep autoencoders for collaborative filtering." Kuchaiev, Oleksii, and Boris Ginsburg. (2017)[2] and "Movie Recommendations Using the Deep Learning Approach,"

(J. Lund and Y. -K. Ng, 2018)[3], from which we learned that the **expected improvement** in performance (lower Test RMSE), using the Deep AutoEncoder architecture, **is quite small**, even when building more complex, deeper architectures with larger hidden dimension sizes (Full details about the experiments and evaluation in "Experiments and Evaluation of the suggested improvements")

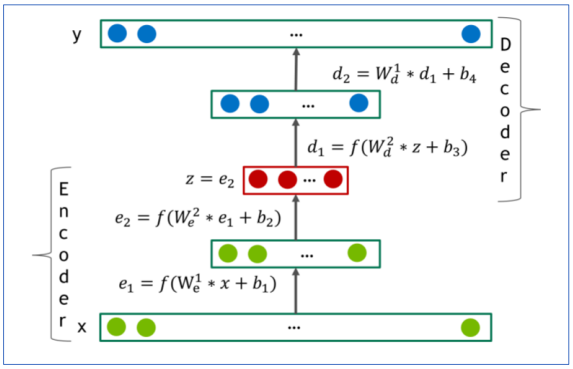


Figure 5: A "Deep" AutoEncoder architecture from the paper "Training deep autoencoders for collaborative filtering.">>

Baseline Algorithm

As our baseline algorithm we selected the **Matrix Factorization,** implemented as a simple **Neural Network** in PyTorch (also in HW3), this baseline was found to be quite powerful and hard to beat, scoring 0.864 Test RMSE on the MovieLens 1M dataset.

The architecture is very simple, yet powerful, and includes only 2 Embedding layers, one for the users and one for the items, we selected the latent dim of 20 (as was found to be optimal in HW3):

Matrix Factorization NN architecture - for ML-1M>>

```
MF_NN(
  (embedding_user_mf): Embedding(6041, 20)
  (embedding_item_mf): Embedding(3953, 20)
)
```

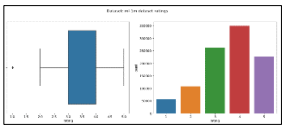
Datasets

We selected to work with the **MovieLens 100K and MovieLens-1M**, for a few reasons:

- The ML1M was used in the original paper
- We faced computational constraints when attempted to use larger datasets (such as the ML10M)
- We had 5folds ready for the ML100K, and for the 1M we created 5-folds using the provided **split_ratings.sh** (provided with the MovieLens datasets)

Dataset validation: for both datasets we verified that we have > 10 ratings per user, and that every user in training set fold also appears in the matching test set fold (folds were well balanced - more details about the validation in the attached notebook)
ML-100K summary: **Counts: users=943, items=1682, ratings=100,000, Sparsity=93.7%**

ML-1M summary: Counts: users=6040, items=3952, ratings=1,000,209, Sparsity=95.53%



Note: Full datasets EDA and additional plots are available in the attached Jupyter notebook

Metrics

We used the original metrics from the paper **RMSE**, and in addition measured the following **ranking metrics MRR@5,MRR@10,nDCG@5 and nDCG@10** - as a general comment the results of the ranking metrics are quite impressive , usually > 0.9, which means that all the models were recommending quite well on the test set. For the ranking metrics we have used relative threshold = 3 (Note that average rating is higher than 3 which may also explain the high ranking metrics we achieved).

Reproducing the results from the AutoRec paper

To reproduce the results we used the paper AutoEncoder architecture with a **single linear** layer + **Sigmoid** for the Encoder, and a **single linear** layer for the Decoder (**relevancy threshold rating >=3**)

We experimented with various hidden layers sizes, and searched for the best hyper-parameters by using **5-Fold Cross Validation**, on the MovieLens 1M and 100K datasets, using permutations of the following hyper parameters:

AutoRec User Architecture - for ML-1M	**AutoRec Item Architecture- for ML-1M**
AutoRec((autorec_sequential): Sequential((0): Linear(in_features=3952, out_features=100, bias=True) (1): Sigmoid() (2): Linear(in_features=100, out_features=3952, bias=True) (3): Identity()))	AutoRec((autorec_sequential): Sequential((0): Linear(in_features=6040, out_features=400, bias=True) (1): Sigmoid() (2): Linear(in_features=400, out_features=6040, bias=True) (3): Identity()))
AutoRec User Experiments on the ML-1M	**AutoRec Item Experiments on the ML-1M**
Best Test RMSE=0.876 (**bold** indicates selected values): 1. Hidden Layer sizes=32,64,**100**,128,250,500 2. Learning rates = 0.1,0.01,0.001,**0.003**,0.0001,1e-5 3. Batch size = 16,32,128,512,**1024** 4. Activation = **Sigmoid**,ReLU,Tanh	Best Test RMSE=0.839 (**bold** indicates the selected values): 1. Hidden Layer sizes=32,64,100,128,250,**400**,500 2. Learning rate = 0.1,0.01,0.001,**0.003**,0.0001,0.0003,1e-5 3. Batch size = 16,32,128,512,**1024** 4. Activation = **Sigmoid**,ReLU,Tanh

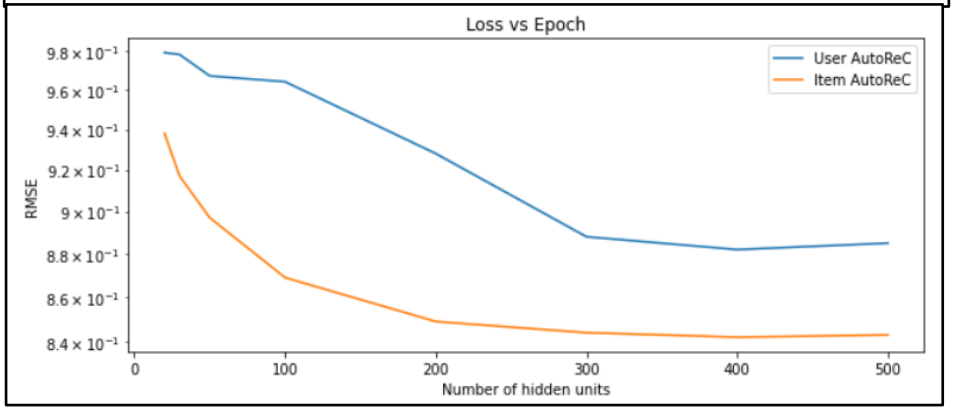
Evaluation

Here are the results for the baseline MF NN and the two original AutoRec models, we got very close to the original results:

	Method	RMSE	MRR_5	MRR_10	NDCG_5	NDCG_10	Time	Params
0	MF_NN	0.864	0.987	0.987	0.967	0.971	167.265877	199880
1	AutoRec_User	0.879	0.988	0.988	0.968	0.972	118.607457	794452
2	AutoRec_Item	0.838	0.974	0.974	0.965	0.970	16.486842	4838440

```
Diff from paper results RMSE Item:0.0070
Diff from paper results RMSE User:0.0050
```

<<Figure 4: Reproduced results of baseline



<<Figure 5:Hidden Dim size vs. Test RMSE

Here are the relevant ML-1M results of the original paper:

U-AutoRec	0.874
I-AutoRec	**0.831**

Experiments and Evaluation of the suggested improvements

Hybrid models

The evaluation of the hybrid models is divided to two sections

1. Standard RMSE evaluation on Total test dataset
2. Cold start : The test data set was adopted to report this case.

For both evaluation we have used the same training dataset

Standard RMSE Experiments

We have optimized by CV the hyper parameters of each mode .l We report the best configuration for the 1M dataset for User / Item content. Additional info is documented in the Notebook

- **learning rate:** 2e-4 , 5e-4
- **L2 regularization** 2e-4 for both
- **batch_size -** 256 , 128. Note: By nature of the AutoRec architecture a Fix batch size does not result by fix number of rating samples per batch.
- **optimizer** - Adam for both

In addition have evaluated for each of the two models: U-AutoRec content and I-AutoRec content the following two parameter for both datasets:

- The optimal size of the latent space. reported in graphs below
- The optimal content gain (Unique to our arch proposal): This parameter controls the loss portion related. to the content information. The optimal numbers found are: for U-AutoRec it is 30 , 100 for 100K/1M for I-AutoRec 100 for both datasets

We also report the results of the I-AutoRec content advance which incorporates the suggestion of the referred paper for injecting the content data at every layer of the autoencoder

Standard RMSE Evaluation

RMSE: Table below indicates small to negligible improvement:

1. 100K- ML: U-AutoRec content 0.938 bs 0.941 (-0.03)
2. 100K- ML: I-AutoRec content 0.887 bs 0.894 (-0.07)
3. 1M-ML: U-AutoRec content 0.877 bs 0.879 (-0.02)
4. 1M-M: I-AutoRec content advance 0.838 vs 0.839 (-0.01)

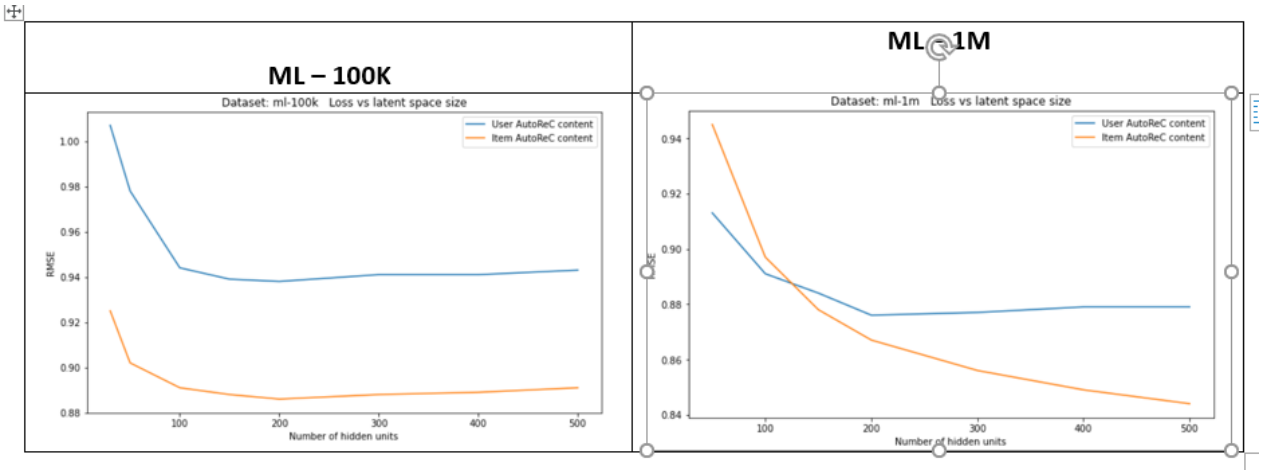
ML – 100K

	Method	RMSE	MRR_5	MRR_10	NDCG_5	NDCG_10
0	MF_NN	0.925	0.977	0.977	0.959	0.964
1	AutoRec_User	0.941	0.974	0.974	0.958	0.963
2	AutoRec_Item	0.894	0.967	0.967	0.967	0.971
3	AutoRec_User_Content	0.938	0.976	0.976	0.958	0.963
4	AutoRec_Item_Content	0.887	0.966	0.966	0.968	0.971
5	AutoRec_Item_Advance	0.897	0.965	0.965	0.967	0.971

ML – 1M

	Method	RMSE	MRR_5	MRR_10	NDCG_5	NDCG_10
0	MF_NN	0.868	0.987	0.986	0.967	0.971
1	AutoRec_User	0.879	0.988	0.988	0.968	0.972
2	AutoRec_Item	0.839	0.974	0.974	0.965	0.970
3	AutoRec_User_Content	0.877	0.987	0.987	0.968	0.973
4	AutoRec_Item_Content	0.844	0.976	0.976	0.965	0.970
5	AutoRec_Item_Advance	0.838	0.977	0.977	0.966	0.971

The graphs below show the optimized latent space size for the hybrid models for each dataset



Cold Start Experiment

For testing cold start the dataset needs to contain items / users that have none or very few ratings. Our datasets already include items/users with only few ratings. For example, the ML 100K which has 1682 items has 333 items with less than 5 ratings. A possible reason that we failed to report significant RMSE importance for the normal RMSE evaluation is that RMSE is the mean average of the error evaluated on ratings in the test hence item / users with only few ratings have negligible impact on the RMSE. One way could define the RMSE to be calculated by an equal weight to all items / users (instead of equal weight to all ratings). Another way we took it is to change the test dataset to include only a single rating per item / user. (The selection was random. Note: Training set is not changed !). We found this as an adequate criterion to evaluate cold start for datasets that already have users / items with only few ratings. (Might be that this is already in the literature…). However our results showed only improvement for the 100K dataset (Detail results are in the notebook)

Deep AutoRec Models

First we tried to take the approach of maintaining **the same number of model parameters** (800K for AutoRec User, 4M for AutoRec Item), but creating a deeper AutoEncoder (results were not as good as the single layer models), then we tried the approach of creating deeper networks with larger hidden layer dimensions.
We experimented with various architectures, such as:

- Maintaining the same number of parameters:
  - AutoRec User from a single hidden layer in dim:[100] →[100,40,100] - where 40 is the dimension of the bottleneck layer
  - AutoRec Item from a single hidden layer in dim:[400] → [400,100,400] - where 100 is the dimension of the bottleneck layer ("diamond" shape)
- Deeper architectures:
  - We experimented with adding more (odd number of…) layers with more neurons, such as:
  - [512,**256**,512] and [256,**512**,256] - an interesting observation was that the "diamond" shape architectures have the less parameters, but maintain the same performance (thus they are preferred)
  - [4096,2048,1024,**512**,1024,2048,4096]
- KFold CV for hyper params search:
  - For every model we ran dozens of experiments using 5-fold CV for hyper params search on the ML-1M, every model had **Early Stopping and Checkpointing** for saving the best weights and saving training times.

Deep AutoRec Models Architectures

In the final deep model architecture of **Deep AutoRec User and Item**, we added an non linear **activation** after every linear layer, and for the **Deep AutoRec User**, also a **dropout** after the middle bottleneck layer (to fight overfitting)

```
AutoRec(
  (autorec_sequential): Sequential(
    (0): Linear(in_features=3952, out_features=256, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=256, out_features=512, bias=True)
    (3): ELU(alpha=1.0)
    (4): Dropout(p=0.8, inplace=False)
    (5): Linear(in_features=512, out_features=256, bias=True)
    (6): ELU(alpha=1.0)
    (7): Linear(in_features=256, out_features=3952, bias=True)
    (8): Identity()
  )
)

Model parameters: 2290544
```

Deep AutoRec User Architecture- for ML-1M

```
AutoRec(
  (autorec_sequential): Sequential(
    (0): Linear(in_features=6040, out_features=400, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=400, out_features=200, bias=True)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=200, out_features=400, bias=True)
    (5): ELU(alpha=1.0)
    (6): Linear(in_features=400, out_features=6040, bias=True)
    (7): Identity()
  )
)

Model parameters: 4999040
```

Deep AutoRec Item Architecture- for ML-1M

AutoRec User Best Hyper Params

We run dozens of experiments using 5-fold CV for hyper params search on the ML-1M, and found best hyper params: epochs=850, activation=ELU(), hidden_dims=[256,512,256], learning_rate=0.01, batch_size=100, dropout=0.8, optimizer=SGD(momentum=0.9), weight_decay=2*1e-4

AutoRec Item Best Hyper Params

We run dozens of experiments using 5-fold CV for hyper params search on the ML-1M, and found best hyper params: hidden_dims=[400,200,400], epochs=1500, lr=0.01, activation=ELU(),optimizer=SGD(momentum=0.9, wd=0.001

Experiments

In addition to the selected hyper params for each model above, during the experiments, we also tried many permutations for the following hyperparams, here are some examples and interesting observations:

- **hidden_dims** - we noticed that deeper models , or larger hidden dim sizes, did not really improve the results - compared to the selected hidden dims. although **"diamond shape"** architectures (where the middle layer has the largest hidden dim size) had half the number of parameters and converged faster.
- **learning rate** (0.1 , 0.01, 0.001 ,0.0001,...) the LR didn't have a significant impact on the result, just on the training time
- **epochs** - For some permutations with a smaller learning rate, slower convergence, we had to use large number of epochs (up to 2000)
- **dropout** - For the "User" a relatively large dropout was used to fight overfitting, for the "Item" adding dropout (even 0.1…) caused worse results, indicating the model was not overfitting
- **batch_size -** we tried with smaller and larger batch sizes, the main difference was on training time, so we picked the largest we value that performed well
- **activation - (**Sigmoid(), ReLU,Tanh,LeakyReLU , **ELU) -** we attempted to run with a few activation functions, ELU and ReLU were the most successful, ELU was slightly better.
- **optimizer** - Adam and SGD(mom=0.9) , for both optimizers, for the **"User"** the **weight_decay** was set to **2*1e-4** and for the **Item** to **0.001**

- **BatchNormalization** - we tried to use BatchNormalization between the linear layers, and it seemed that the training and validation losses were converging more smoothly, but performance was not improved (probably as there was no problem of vanishing gradients in our model)

Evaluation

Compared to the baseline **AutoRec User** Test RMSE of 0.879, the Deep AutoRec user performed **better** achieving **Test RMSE of 0.874**. This is matching our expectations and the best RMSE on MovieLens 1M from paperswithcode.com

Surprisingly, compared to the baseline AutoRec Item Test RMSE of 0.838, the **Deep AutoRec Item** performed slightly **worse**, achieving **Test RMSE of 0.858**. Although this is matching our expectations and the best RMSE on MovieLens 1M from paperswithcode.com, we were a bit disappointed by this result.

With the deeper architectures, for Deep Item AutoRec, we were not able to reproduce the 0.827 RMSE result o the ML-1M, which was mentioned at the end of the paper (we managed to get 0.856 which is not as good as the basic I-AutoRec that got 0.838). There are no details in the paper how this result was achieved. Also based on the SOTA website paperswith code), it appears that such a good result was only achieved by another architecture CF-NADE (which is not covered in our report)

One interesting enhancement we attempted was to work with **Normalized rating data**, inspired by the paper "Sparsity Normalization: **Stabilizing the Expected Outputs of Deep Networks** [4] - we normalized the rating data between 0 and 1, using MinMaxScaler, and although we experienced slightly faster convergence times, we could not identify an improvement to the final Test RMSE with any of models.

Final Results of all models

Note: additional plots available in the attached Jupyter notebook

Method	RMSE	MRR_5	MRR_10	NDCG_5	NDCG_10
AutoRec_Item	0.838	0.974	0.974	0.965	0.970
AutoRec_Item_Advance	0.839	0.977	0.977	0.966	0.971
AutoRec_Item_Content	0.843	0.974	0.974	0.965	0.969
AutoRec_User	0.879	0.988	0.988	0.968	0.972
AutoRec_User_Content	0.871	0.989	0.989	0.969	0.973
DeepAutoRec_Item	0.856	0.969	0.968	0.962	0.966
DeepAutoRec_User	0.874	0.986	0.986	0.967	0.971
MF_NN	0.864	0.987	0.987	0.967	0.971

Method	RMSE	MRR_5	MRR_10	NDCG_5	NDCG_10
AutoRec_Item	0.894	0.967	0.967	0.967	0.971
AutoRec_Item_Advance	0.895	0.963	0.963	0.967	0.971
AutoRec_Item_Content	0.887	0.967	0.967	0.968	0.971
AutoRec_User	0.941	0.974	0.974	0.958	0.963
AutoRec_User_Content	0.939	0.971	0.971	0.957	0.961
DeepAutoRec_User	0.978	0.978	0.978	0.956	0.961
MF_NN	0.925	0.977	0.977	0.959	0.964

Figure 6: Final results of all models vs. baselines (to update results)

Actually these results are backed up by the best RMSE results for MovieLens 1M shown in the SOTA website: https://paperswithcode.com/sota/collaborative-filtering-on-movielens-1m , the next best model to outperform the I-AutoRec is CF-NADE (0.829), Sparse-FC (0.824) and GLocal-K (0.82), which are not covered in this report.

Conclusions

Autoencoder is a powerful model for recommendation systems. The I-AutoRec beats state of the art Matrix factorization for explicit rating by a non-negligible margin on both datasets we have tested. . Moreover AutoRec is a very simple model. This adds to its attractiveness. It utilizes the non linearity of the neural networks models and therefore can capture nonlinear relations between users and items. We have encountered an interesting paper by Rendell that argues about the power of neural networks for recommendation systems. While we found it inline with our HW3 results (for implicit database and Neural collaborative filtering model) we do see a significant improvement for AutoRec (for explicit datasets) which is also based on Neural networks.
Adding content based information to the AutoRec model is very elegant and requires only minor changes to the model.
This Hybrid approach achieved a small improvement for RMSE metric over the original AutoRec proposals for some cases. Our conclusion is that when massive data exists collaborative filtering is very powerful in predicting ratings and side band information is not that critical. We have reached a similar conclusion for HW2 where we have applied a weighted approach for hybrid solution (this work was beyond the work expected by this assignment). That approach combined the ranks of both MF and content based approach with various weights options. However, that hybrid approach could not beat the collaborative filtering and for some weights was even worse. In our case we did observe some minor RMSE improvement. (This work was beyond the work expected by this assignment)
As to ranking metrics all models, show very good results.

With the **AutoEncoder architecture**, on the datasets we experimented with, deeper doesn't automatically mean better - same goes for **more complex** architectures (more neurons)
Diamond shaped architectures, where the bottleneck layer has a larger hidden dimension size, performs quite as well as the **Hourglass shaped architectures,** and with half the number of parameters.

References

1. “AutoRec: Autoencoders Meet Collaborative Filtering” Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, Lexing Xie (2015)
2. "Training deep autoencoders for collaborative filtering." Kuchaiev, Oleksii, and Boris Ginsburg. (2017)
3. "Movie Recommendations Using the Deep Learning Approach," (J. Lund and Y. -K. Ng, 2018)
4. “Sparsity Normalization: Stabilizing the Expected Outputs of Deep Networks” Joonyoung Yi1 , Juhyuk Lee1 , Sung Ju Hwang1,2 , Eunho Yang1,2 (2019)
5. "Hybrid collaborative filtering with autoencoders." Florian Strub (SEQUEL, CRIStAL), Jeremie Mary (CRIStAL, SEQUEL), Romaric Gaudel (LIFL) (2016)
6. Neural collaborative filtering vs Matrix Factorization revisited. Steffen Rendle, Walid Krichene, Li Zhang, Google Research, John Anderson

View on Github: https://github.com/lleviraz/recsys_final_project