Hybrid Recommendation Systems using Neural Networks

M.Sc. Thesis of Michael Bizimis

We will talk about…

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    3.1. Content-based profiles
    3.2. Proposed models
        3.2.1. Basic NCF
        3.2.2. Attention NCF
        3.2.3. Graph NCF
    3.3. Solving the prediction vs the ranking problem
4. Experiments
    4.1. Dataset used
    4.2. Train-val-test split
    4.3. Evaluation metrics
    4.4. Experiments
5. Conclusions
6. Future Work

Motivation

The problem…

Overchoice

- People faced with an overwhelming amount of options.
- Difficult to choose amongst them.
- Too many factors to consider.

The solution…

Recommendation Systems

- Automatically discard irrelevant options.
- Present only a smaller subset of relevant ones.
- Personalized to each user based on his past choices.

The Recommendation Task

Having access to certain **users**, **items** and **recorded interactions** between them, predict potential future interactions.

Store recorded interactions in the **utility matrix**.

	item 1	item 2	item 3	item 4
user 1	2	5	1	3
user 2	4	?	?	1
user 3	?	4	2	?
user 4	2	4	3	1
user 5	1	3	2	?

Rating interactions

	item 1	item 2	item 3	item 4
user 1	0	1	0	1
user 2	1	?	?	0
user 3	?	1	0	?
user 4	0	1	1	0
user 5	0	1	0	?

Binary interactions

The Recommendation Task

Prediction version

Given a user *u* and an item *i*, **predict their interaction value** as *f(u, i)*.

In other words…

Fill in the blanks of the utility matrix by using the known cells and by estimating function *f*.

Ranking version

Make **top-k recommendation** of items to users.

In other words…

Rank all the items per user by descending user preference.

The prediction version is more general.

The two main categories of methods

Collaborative Filtering (CF) methods

Typically solve the **prediction** version.

Rely on **correlations** between users and items.

Different methods depending on choice for *f*, e.g.:

- *Neighborhood-based CF*

  f(u, i) = average rating for item *i* from *k* most similar users to user *u*

- *Model-based CF*

  f(u, i) is learned from known interactions using a machine learning model.

Content-based methods

Typically solve the **ranking** version.

Rely on **item features** as content.

Calculate:

- Item profiles from item features.
- User profiles by aggregating interacted item profiles.
- A similarity / distance metric between item and user profiles.

Make top-k recommendation based on **similarity** / **distance**.

Hybrid methods combine both!

We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
   3.1. Content-based profiles
   3.2. Proposed models
      3.2.1. Basic NCF
      3.2.2. Attention NCF
      3.2.3. Graph NCF
   3.3. Solving the prediction vs the ranking problem
4. Experiments
   4.1. Dataset used
   4.2. Train-val-test split
   4.3. Evaluation metrics
   4.4. Experiments
5. Conclusions
6. Future Work

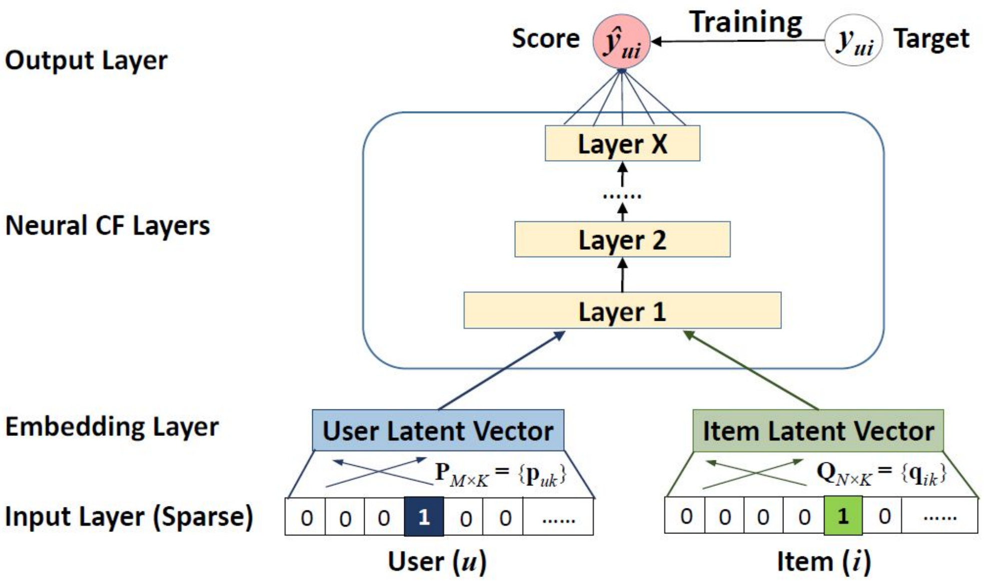
The Neural Collaborative Filtering (NCF) framework

A model-based CF method that extends **Matrix Factorization**.

Estimates *f(u, i)* with a **neural network**.

Many possible **vector representations** for users and items.

Many possible training objectives via different **loss functions**.



We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    3.1. Content-based profiles
    3.2. Proposed models
        3.2.1. Basic NCF
        3.2.2. Attention NCF
        3.2.3. Graph NCF
    3.3. Solving the prediction vs the ranking problem
4. Experiments
    4.1. Dataset used
    4.2. Train-val-test split
    4.3. Evaluation metrics
    4.4. Experiments
5. Conclusions
6. Future Work

Our methodology

We combine NCF with content-based methods for creating item and user profiles to get a **hybrid** Recommendation System that:

- Avoids the **cold-start problem** of CF.
- Achieves **better performance** by learning patterns from features.

We examine three increasingly complex neural network architectures.

We solve both the *prediction problem* by using a **pointwise loss** (e.g. MSE, BCE) and the *ranking problem* directly by using a **pairwise ranking loss** (e.g. BPR).

We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    3.1. Content-based profiles
    3.2. Proposed models
        3.2.1. Basic NCF
        3.2.2. Attention NCF
        3.2.3. Graph NCF
    3.3. Solving the prediction vs the ranking problem
4. Experiments
    4.1. Dataset used
    4.2. Train-val-test split
    4.3. Evaluation metrics
    4.4. Experiments
5. Conclusions
6. Future Work

Content-based profiles

Item profiles

Vector representations of **item features**.

Categorical Features

- One-hot / multi-hot encoded
- Take up multiple dimensions (sparse)

Numeric

- Usually normalized to [0, 1] or [-1, 1]
- Take up one dimension

Embeddings

User profiles

Vector representations of **user preferences**.

Users with similar preferences → similar user profiles → similar recommendations

Many possible ways to aggregate interacted items profiles…

Our method for creating user profiles

We construct user profiles $\vec{v_u}$ as:

$$\vec{v_u} = \frac{1}{|N(u)|} \sum_{i \in N(u)} (r_{ui} - \overline{r_u})\, \vec{v_i} \quad \text{where} \quad \overline{r_u} = mean\left(\frac{1}{|N(u)|} \sum_{i \in N(u)} r_{ui}\, , m \right)$$

and:

- r = rating (e.g. 0-5)
- $N(u)$ = rated items for user u
- m = neutral rating (e.g. for 0-5 ratings it would be 2.5)

Positive interaction if $r_{ui} - \overline{r_u} > 0$, negative otherwise.

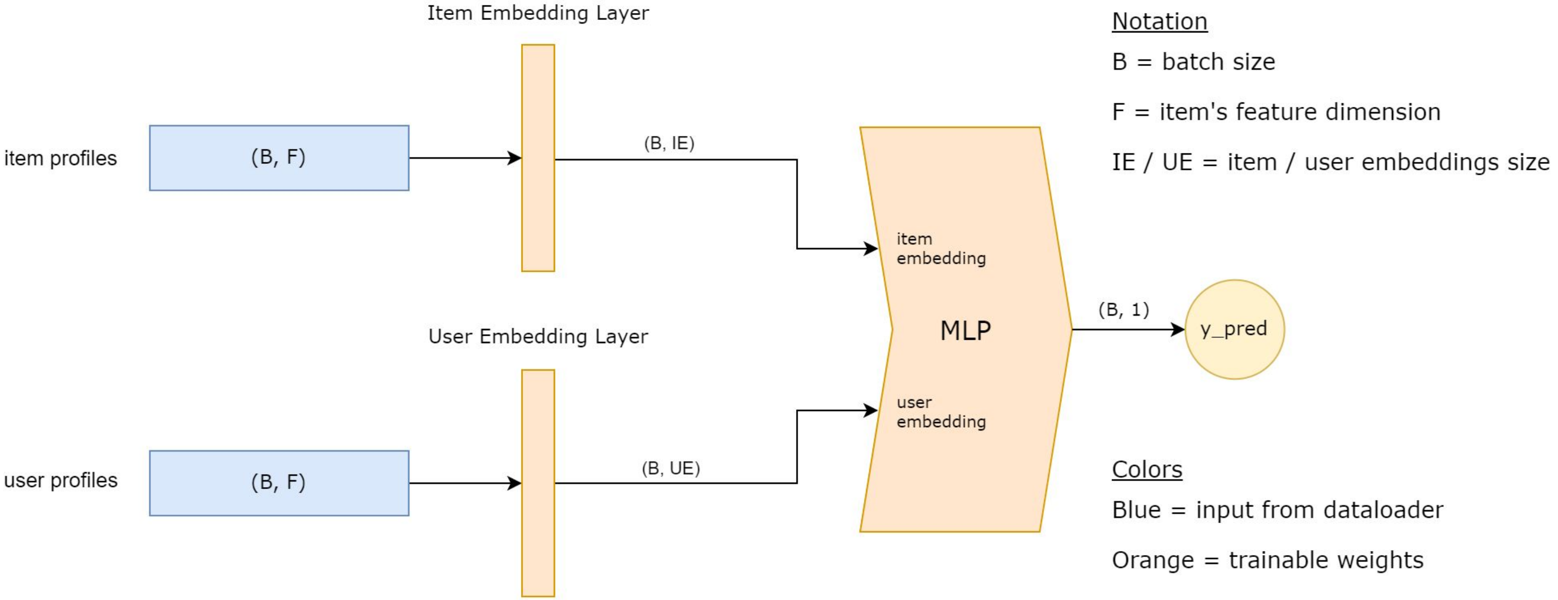
Positive interactions move users *towards* the item profile, negative ones move them *away*.

We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    3.1. Content-based profiles
    3.2. Proposed models
        3.2.1. Basic NCF
        3.2.2. Attention NCF
        3.2.3. Graph NCF
    3.3. Solving the prediction vs the ranking problem
4. Experiments
    4.1. Dataset used
    4.2. Train-val-test split
    4.3. Evaluation metrics
    4.4. Experiments
5. Conclusions
6. Future Work

Basic NCF

Vanilla NCF, but with **fixed** (precalculated) item and user profiles as input vectors.



We will talk about…

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    3.1. Content-based profiles
    3.2. Proposed models
        3.2.1. Basic NCF
        3.2.2. Attention NCF
        3.2.3. Graph NCF
    3.3. Solving the prediction vs the ranking problem
4. Experiments
    4.1. Dataset used
    4.2. Train-val-test split
    4.3. Evaluation metrics
    4.4. Experiments
5. Conclusions
6. Future Work

Attention NCF

Calculates **user profiles dynamically** during the forward pass.

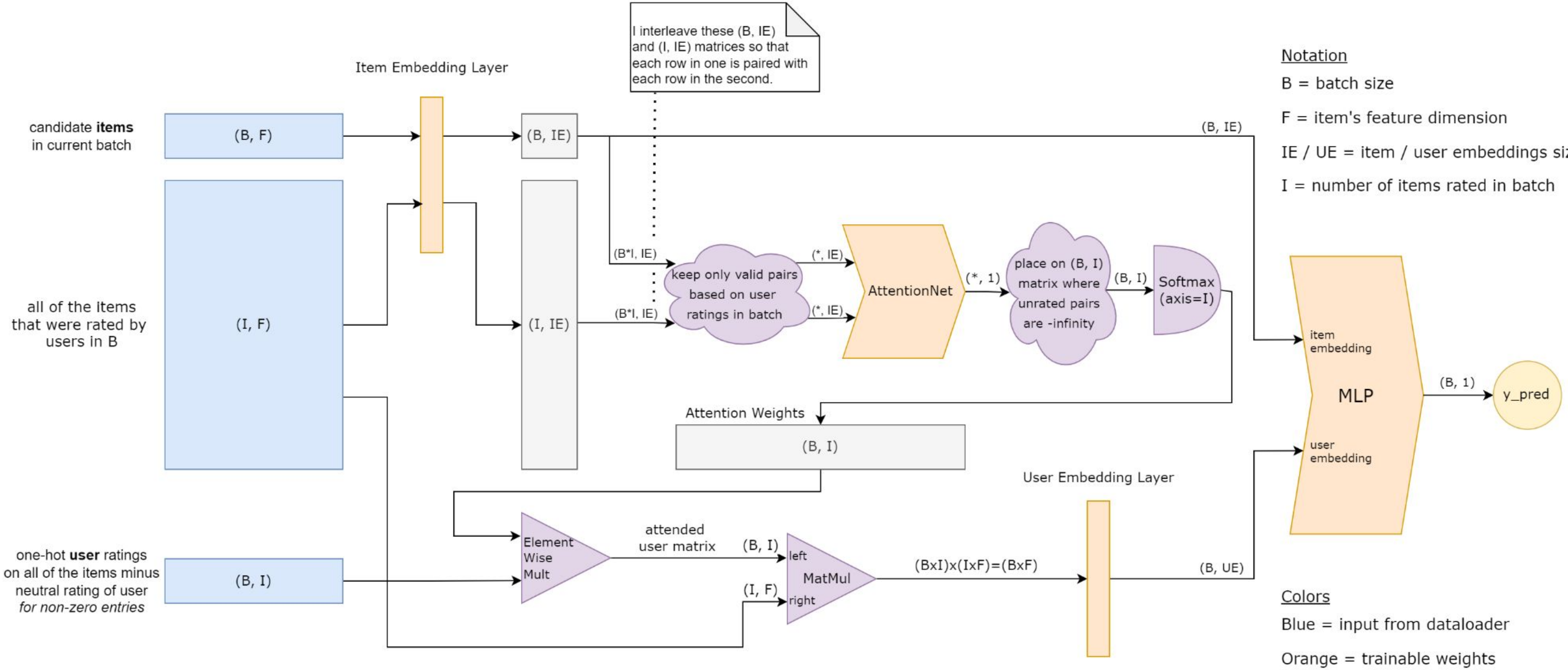
Adds an **item-item attention mechanism** to them.

Calculate an attention weight a_{ci} based on the **rated item** i and the **candidate item** c:

$$\vec{v_u} = \sum_{i \in N(u)} a_{ci} \left(r_{ui} - \overline{r_u} \right) \vec{v_i} \qquad \text{where} \qquad \sum_{i \in N(u)} a_{ci} = 1$$

We learn these in an **end-to-end** way, using a secondary neural network we call **AttentionNet**.

Attention NCF



Notation

B = batch size

F = item's feature dimension

IE / UE = item / user embeddings size

I = number of items rated in batch

Colors

Blue = input from dataloader

Orange = trainable weights

Purple = non-trainable operations

Important caveat about user profiles

We are using the same known user-item interactions as both:

- Training samples.
- Part of the user profiles.

Thus, during training, the candidate item is also part of the user profile.

Fixed user profiles → problem is contained → candidate item with fixed $1/|N(u)|$ weight.

Learnable attention weights → can drastically lead to **overfitting** → candidate item gets all the attention!

To avoid this, we **mask out** the candidate item from being used as a rated item at training.

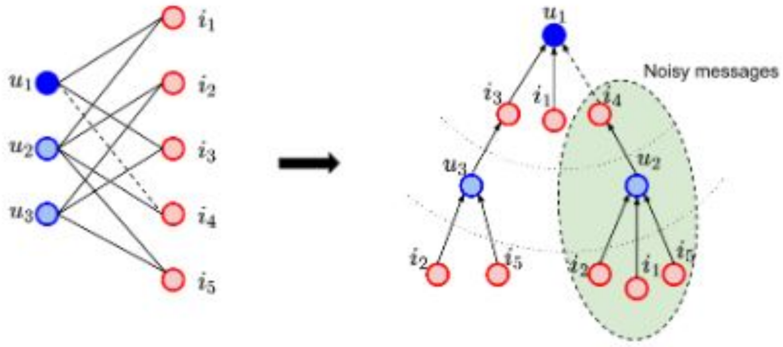
We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    - 3.1. Content-based profiles
    - 3.2. Proposed models
        - 3.2.1. Basic NCF
        - 3.2.2. Attention NCF
        - 3.2.3. Graph NCF
    - 3.3. Solving the prediction vs the ranking problem
4. Experiments
    - 4.1. Dataset used
    - 4.2. Train-val-test split
    - 4.3. Evaluation metrics
    - 4.4. Experiments
5. Conclusions
6. Future Work

Graph NCF

So far, we have been capturing the collaborative signal between users and items **implicitly**, through our training objective.

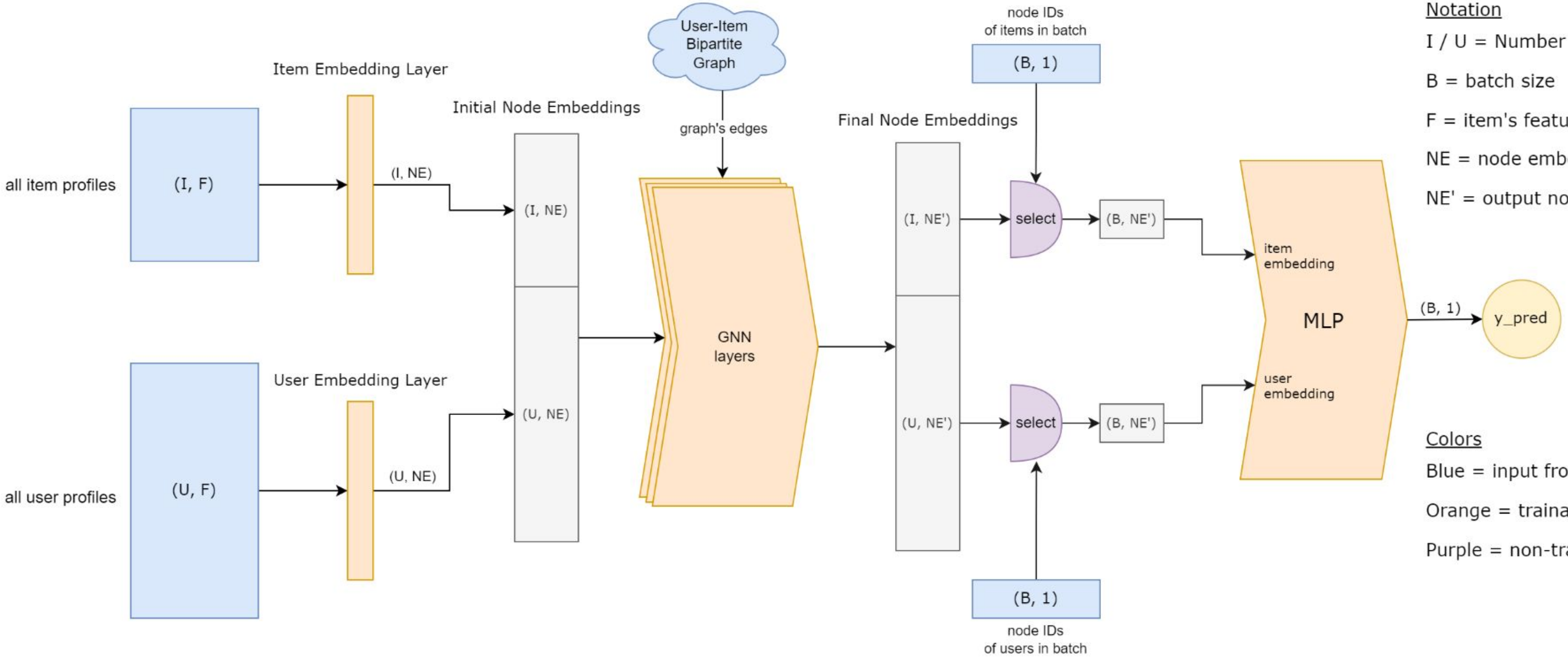
Instead, we can try to **explicitly** capture it by using **Graph Neural Networks** for **message passing** on the user-item bipartite graph.



User-item bipartite graph

Message passing for node u_1

Graph NCF



<u>Notation</u>

I / U = Number of all items / users

B = batch size

F = item's feature dimension

NE = node embeddings size

NE' = output node embeddings size

<u>Colors</u>

Blue = input from dataloader

Orange = trainable weights

Purple = non-trainable operations

Graph NCF

Account for **ratings** → **edge weights** in user-item graph:

- $r_{ui} - \overline{r_u}$ for user-to-item edges
- $r_{ui} - \overline{r_i}$ for item-to-user edges

Account for graph's **heterogeneity** → separate learnable matrices W_u, W_i for each node type.

Message Construction

$$m_{u\to i}^{(t)} = \frac{r_{ui} - \overline{r_u}}{\sqrt{|N_u||N_i|}} W_u^{(t)} \vec{e_u}^{(t-1)}$$

$$m_{i\to u}^{(t)} = \frac{r_{ui} - \overline{r_i}}{\sqrt{|N_u||N_i|}} W_i^{(t)} \vec{e_i}^{(t-1)}$$

Message Aggregation

$$\vec{e_u}^{(t)} = \sum_{i\to u} m_{i\to u}^{(t)}$$

$$\vec{e_i}^{(t)} = \sum_{u\to i} m_{u\to i}^{(t)}$$

Graph NCF

To get the **final node embeddings**, after T GNN layers, we either:

- Concatenate
- Average (must have same dim)

all T+1 node embeddings (including the starting ones).

Message and **node dropout** for regularization during training.

Again, we **mask out** the target user-item interaction from the graph's edges during training.

We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    3.1. Content-based profiles
    3.2. Proposed models
        3.2.1. Basic NCF
        3.2.2. Attention NCF
        3.2.3. Graph NCF
    3.3. Solving the prediction vs the ranking problem
4. Experiments
    4.1. Dataset used
    4.2. Train-val-test split
    4.3. Evaluation metrics
    4.4. Experiments
5. Conclusions
6. Future Work

Solving the prediction vs the ranking problem

Solving the prediction problem

Use (user *u,* item *i* , target rating *y*) samples.

MSE loss: $L(u, i) = (y_{ui} - \widehat{y_{ui}})^2$

BCE loss: $L(u, i) = -y_{ui} \log \widehat{y_{ui}} - (1 - y_{ui}) \log (1 - \widehat{y_{ui}})$

Solving the ranking problem

Use (user *u*, pos item *i*, neg item *j*) samples, where $r_{ui} > r_{uj}$.

BPR loss: $L(u, i, j) = -\log \sigma(\widehat{y_{ui}} - \widehat{y_{uj}})$

But **too many possible triplets** → **sample negative** item *j* at random.

Give higher probability to **hard negatives** (items with higher rating).

We will talk about...

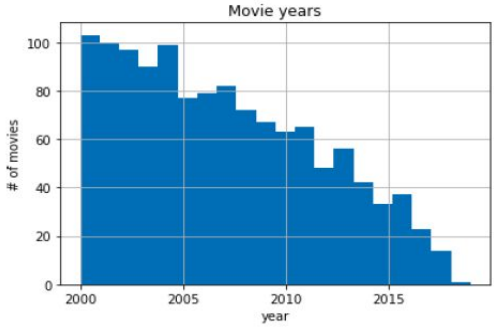
1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    - 3.1. Content-based profiles
    - 3.2. Proposed models
        - 3.2.1. Basic NCF
        - 3.2.2. Attention NCF
        - 3.2.3. Graph NCF
    - 3.3. Solving the prediction vs the ranking problem
4. Experiments
    - 4.1. Dataset used
    - 4.2. Train-val-test split
    - 4.3. Evaluation metrics
    - 4.4. Experiments
5. Conclusions
6. Future Work

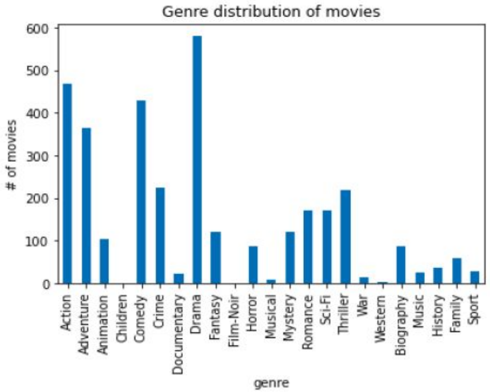
Dataset used

We combined:

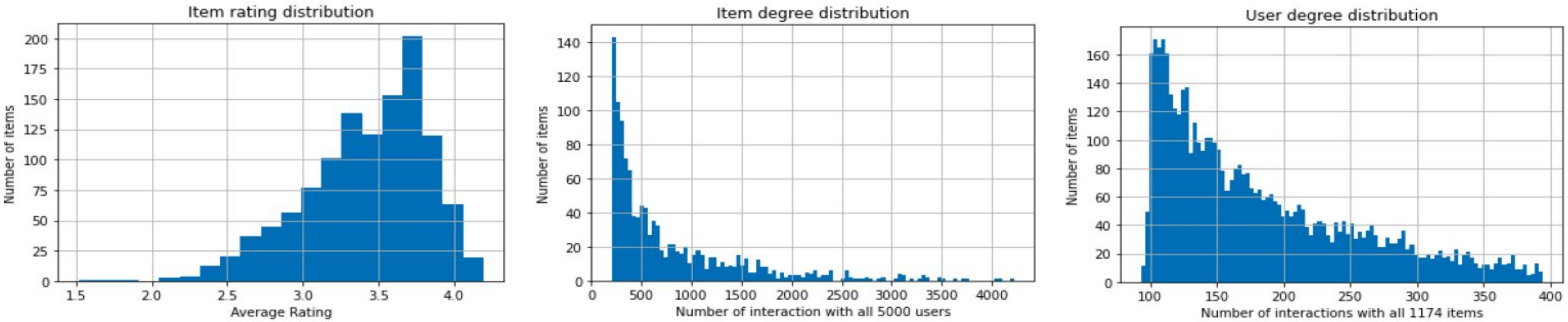
- The **25M MovieLens dataset** with 1-5 ratings from 160000 users to 60000 movies.
- The **IMDb database** for **metadata** as item features.
- The 1100 **genome tags** from MovieLens also as item features.

After some filtering we ended up with **910891 ratings** between **1174 movies** and **5000 users**.





Dataset used



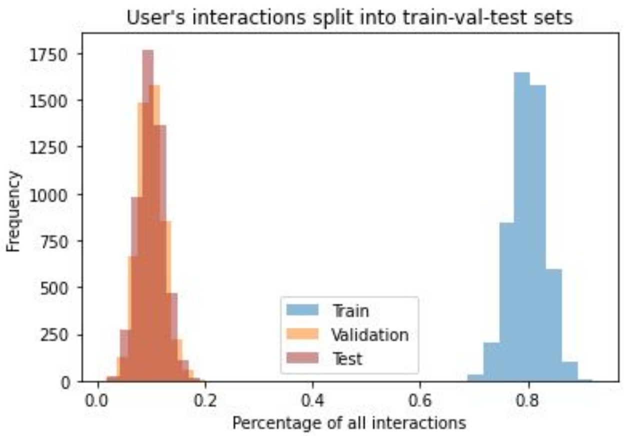
We will talk about...

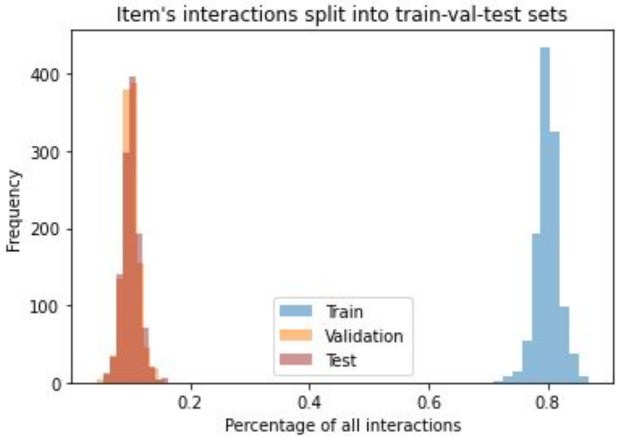
1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
   3.1. Content-based profiles
   3.2. Proposed models
      3.2.1. Basic NCF
      3.2.2. Attention NCF
      3.2.3. Graph NCF
   3.3. Solving the prediction vs the ranking problem
4. Experiments
   4.1. Dataset used
   4.2. Train-val-test split
   4.3. Evaluation metrics
   4.4. Experiments
5. Conclusions
6. Future Work

Train-val-test split

Train-val-test splitting a CF dataset is not straight forward. Based on users? Based on items?

The simplest approach of **uniformly splitting** all 900k interactions worked best → even distribution of **both** user and item interactions.





We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
   - 3.1. Content-based profiles
   - 3.2. Proposed models
     - 3.2.1. Basic NCF
     - 3.2.2. Attention NCF
     - 3.2.3. Graph NCF
   - 3.3. Solving the prediction vs the ranking problem
4. Experiments
   - 4.1. Dataset used
   - 4.2. Train-val-test split
   - 4.3. Evaluation metrics
   - 4.4. Experiments
5. Conclusions
6. Future Work

Evaluation Metrics

Regression Metrics

We use the MSE loss (lower is better).

Ranking Metrics

We use the NDCG@k metric (higher is better).

$$NDCG@k = \frac{DCG@k}{iDCG@k} \qquad DCG@k = \sum_{i=1}^{k} \frac{relevance_i}{\log_2(i+1)}$$

For the test set, we calculate these metrics:

- Once, using **only the training** interactions for the user profiles (Test).
- Once, using **the training and the validation** interactions for the user profiles (Test+).

If the model has learned something meaningful, we **expect the Test+ metrics to be better**.

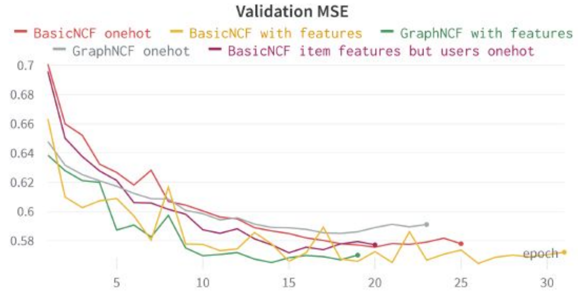
We will talk about...

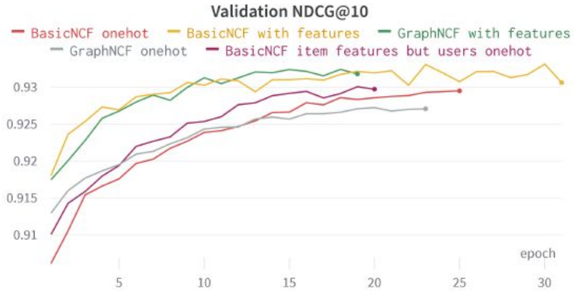
1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
   3.1. Content-based profiles
   3.2. Proposed models
      3.2.1. Basic NCF
      3.2.2. Attention NCF
      3.2.3. Graph NCF
   3.3. Solving the prediction vs the ranking problem
4. Experiments
   4.1. Dataset used
   4.2. Train-val-test split
   4.3. Evaluation metrics
   4.4. Experiments
5. Conclusions
6. Future Work

One-hot vs Features

Content-based profiles solve the cold-start problem. But do they also increase performance?

Model	Test MSE	Test+ MSE	Test NDCG@10 / adjusted	Test+ NDCG@10 / adjusted
Basic NCF one-hot	0.5749	–	0.9287 / 0.7712	–
Basic NCF with features but users one-hot	0.5739	–	0.9287 / 0.7706	–
Basic NCF with features	**0.5657**	0.5618	0.9320 / 0.7786	0.9327 / 0.7817
Graph NCF one-hot	0.5858	–	0.9277 / 0.7690	–
Graph NCF with features	0.5667	0.5632	**0.9321 / 0.7804**	0.9324 / 0.7817



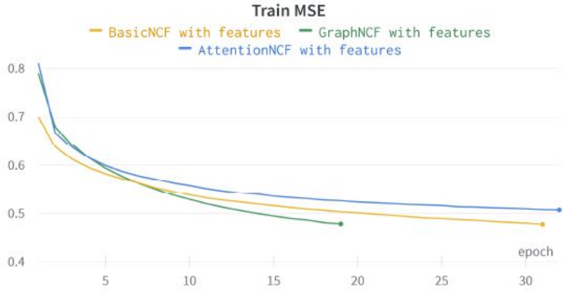


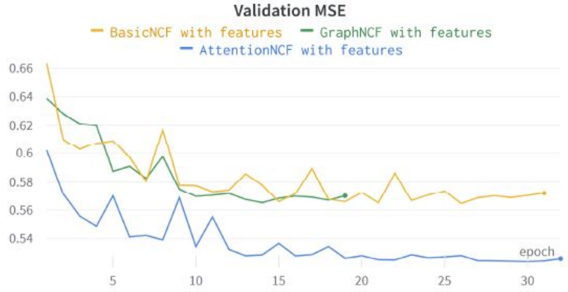
Kind of…

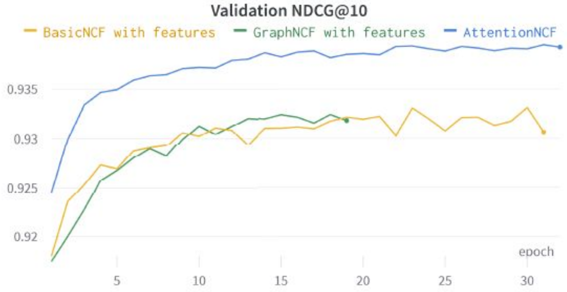
Basic NCF vs Attention NCF vs Graph NCF

Which architecture performs better on our dataset?

Model	Test MSE	Test+ MSE	Test NDCG@10 / adjusted	Test+ NDCG@10 / adjusted
Basic NCF	0.5657	0.5618	0.9320 / 0.7786	0.9327 / 0.7817
Attention NCF	**0.5244**	**0.5185**	**0.9387 / 0.8009**	**0.9396 / 0.8039**
Graph NCF	0.5667	0.5632	0.9321 / 0.7804	0.9324 / 0.7817
Basic NCF one-hot	0.5749	–	0.9287 / 0.7712	–
Graph NCF one-hot	0.5858	–	0.9277 / 0.7690	–

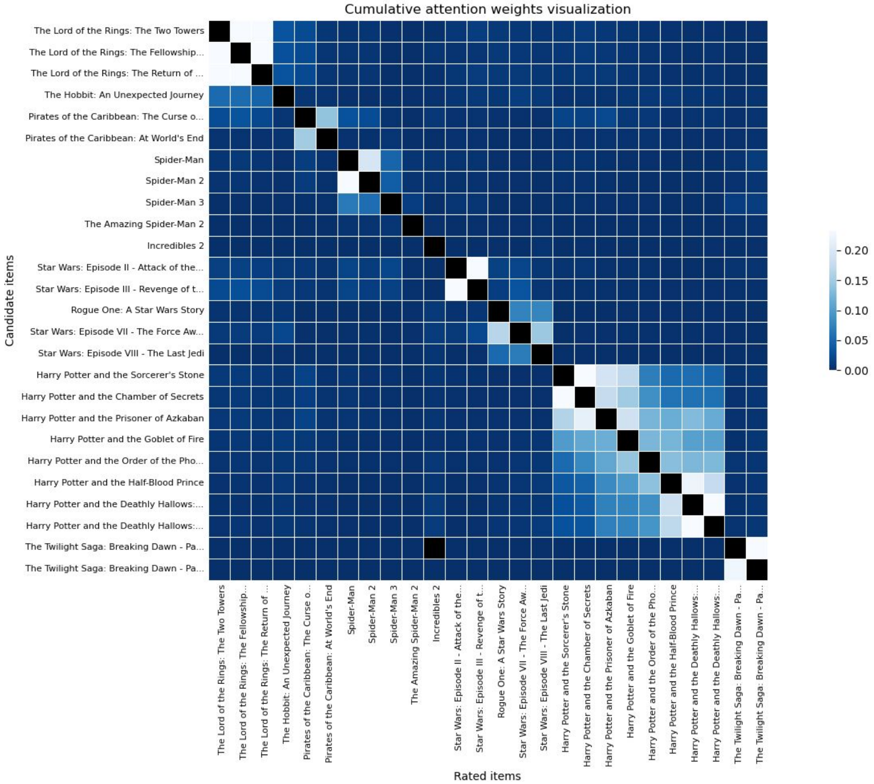


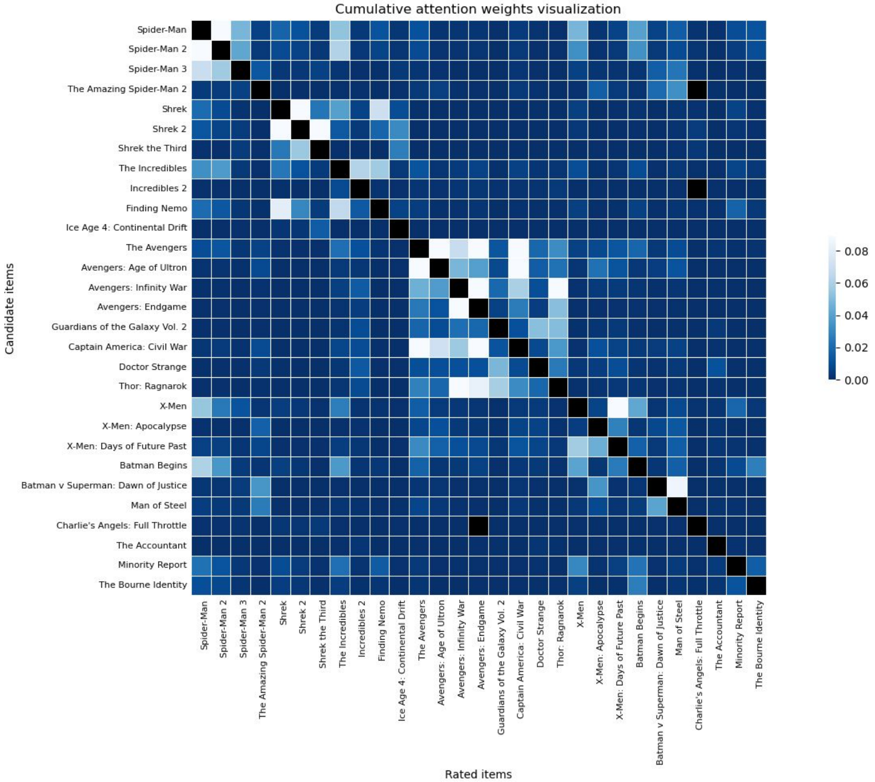




Definitely Attention NCF!

Attention visualized

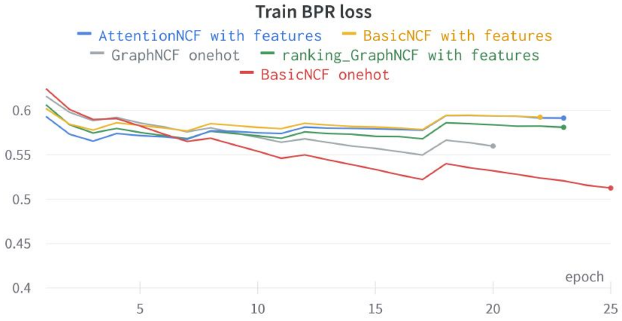


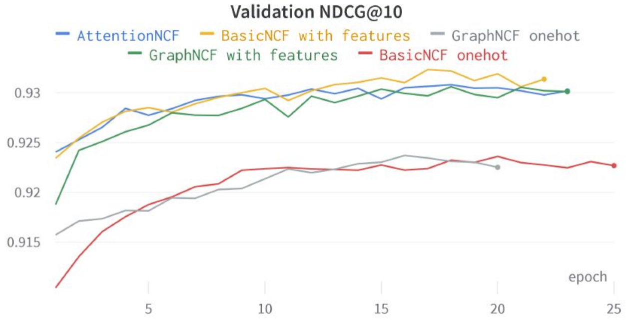


Solving the ranking problem

Solving the ranking problem works…

Model	Test NDCG@10 / adjusted	Test+ NDCG@10 / adjusted
Basic NCF	**0.9326 / 0.7817**	**0.9332 / 0.7839**
Attention NCF	0.9315 / 0.7786	0.9315 / 0.7783
Graph NCF	0.9304 / 0.7753	0.9310 / 0.7778
Basic NCF one-hot	0.9238 / 0.7558	–
Graph NCF one-hot	0.9232 / 0.7552	–





…but not as good, especially for Attention NCF.

Model cost comparison

Rough model cost estimation in our experiments:

Model	batch size	Average time per epoch (minutes / epoch)
Basic NCF	128	2.516
Basic NCF	512	2.037
Attention NCF	512	5.281
Graph NCF with 3×64 GNN layers	512	5.714
Graph NCF with 2×128 GNN layers	512	6.842

Model	batch size	Average time with MSE (minutes / epoch)	Average time with BPR (minutes / epoch)
Basic NCF	512	2.037	3.227
Attention NCF	512	5.281	7.913
Graph NCF	512	5.714	10.826

We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
   3.1. Content-based profiles
   3.2. Proposed models
      3.2.1. Basic NCF
      3.2.2. Attention NCF
      3.2.3. Graph NCF
   3.3. Solving the prediction vs the ranking problem
4. Experiments
   4.1. Dataset used
   4.2. Train-val-test split
   4.3. Evaluation metrics
   4.4. Experiments
5. Conclusions
6. Future Work

Conclusions

- Attention NCF is undoubtedly the best architecture of the three:
  - **Better performance** (roughly ~7.5% better Test and Test+ MSE).
  - Offers **explainability** through its attention mechanism.

- Graph NCF did not perform any better than Basic NCF:
  - **Weak collaborative signal** on our graph / data?
  - Generalization issue of **fixed user profiles**?

- Training with **MSE for prediction** > **BPR for ranking**.
  - More hyperparameters (negative sampling).
  - More expensive computationally.
  - Worse results (especially for Attention NCF).

We will talk about...

1. Introduction to Recommendation Systems
2. The Neural Collaborative Filtering framework
3. Our methodology
    3.1. Content-based profiles
    3.2. Proposed models
        3.2.1. Basic NCF
        3.2.2. Attention NCF
        3.2.3. Graph NCF
    3.3. Solving the prediction vs the ranking problem
4. Experiments
    4.1. Dataset used
    4.2. Train-val-test split
    4.3. Evaluation metrics
    4.4. Experiments
5. Conclusions
6. Future Work

Future work

- Improve performance by using **more relevant item features**, e.g.:
  - Embeddings from content (e.g. video, text, audio, etc.) learned separately (e.g. unsupervised representation learning).

- Try to further improve Attention NCF, e.g. by leveraging **more information in AttentionNet** (e.g. the given rating).

- Try to improve Graph NCF by **avoiding fixed user profiles**, e.g.:
  - Implement **dynamic user profiles** as the first graph convolution.
  - Only item nodes send messages.
  - User nodes with no initial node features.

The code

All the code (including a web app demo) is open-source at:

https://github.com/michaelbzms/DeepRecommendation

It should be **flexible** enough so that anyone can apply it to different tasks by simply:

- Extending some abstract classes for the content-based profiles.
- Changing any necessary data loading logic.

Questions?