Recommender Systems Practice

Data AI Lab

School of Electrical Engineering

KAIST

Outline

1. **<u>Introduction</u>**

2. Latent Factor Model Practice (LF) - NCF

3. Graph Collaborative Filtering Practice (GCF) – NGCF

Introduction - Dataset

- Today, we will use **MovieLens** dataset.
- Data link: https://grouplens.org/datasets/movielens/latest/

rating : [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]

user : 610

movie : 9724

interaction : 100836

Introduction - Packages

- We will use **packages** below.

python 3.10.12
numpy 1.25.2
pandas 2.0.3
matplotlib 3.7.1
pytorch(torch) 2.3.1+cu121
scikit-learn(sklearn) 1.2.2

torch-scatter 2.1.2+pt25cu121
torch-sparse 0.6.18+pt25cu121
torch-cluster 1.6.3+pt25cu121
torch-spline-conv 1.2.2+pt25cu121
torch-geometric 2.6.1

Introduction - Model Evaluation

[Evaluation – How well the model predict ratings for items]

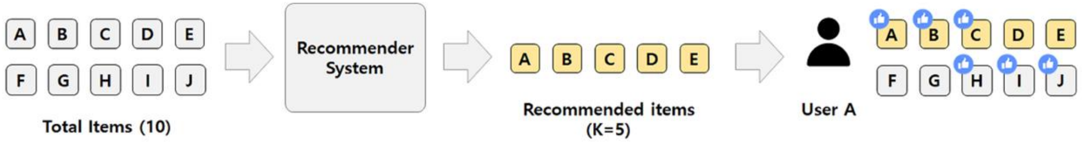
- If we want to focus on rating prediction:
  - **(RMSE)** Root Mean Square Error
    - $\boldsymbol{RMSE} = \sqrt{\frac{1}{|test\ data|}\sum_{i=1}^{|test\ data|}(y_i - pred_i)^2}$ (y_i = rating for test data)
  - **(MAE)** Mean Absolute Error
    - $\boldsymbol{MAE} = \frac{1}{|test\ data|}\sum_{i=1}^{|test\ data|}|y_i - pred_i|$

Ref: https://sungkee-book.tistory.com/11

Introduction - Model Evaluation

[Evaluation – How well the item list reflect users' taste]

- If we want to focus on item list prediction:
  - **Recall@K**
    - **Recall@K** = (Relevant item in top-K items) / Total number of relevant items
  - **Precision@K**
    - **Precision@K** = (Relevant item in top-K items) / K



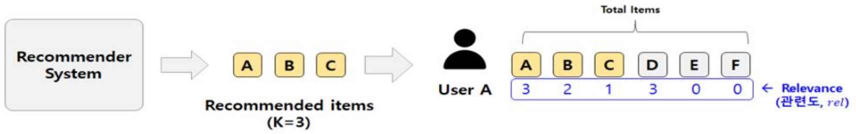
$$Precision@5 = \frac{3}{5} = 0.6 \qquad Recall@5 = \frac{3}{6} = 0.5$$

Ref: https://sungkee-book.tistory.com/11

Introduction - Model Evaluation

[Evaluation – How well the item list reflect users' taste]

- **NDCG** (Normalized Discounted Cumulative Gain)$\boldsymbol{@K}$
  - Cumulative Gain = $\sum_{i=1}^{K} relevance\ of\ item_i$
  - **NDCG**$\boldsymbol{@K}$ **=** Discounted Cumulative Gain (**DCG**) / Ideal Discounted Cumulative Gain (**IDCG**)
    - **DCG** = $\sum_{i=1}^{K} \frac{relevance\ of\ item_i}{\log_2(i+1)}$
    - **IDCG** =$\sum_{i=1}^{K} \frac{relevance\ of\ \ item_i\ in\ idea\ list}{\log_2(i+1)}$



$$CG_3 = \sum_{i=1}^{K} rel_i = rel_1 + rel_2 + rel_3 = 3 + 2 + 1 = 6$$

$$DCG_3 = \sum_{i=1}^{K} \frac{rel_i}{log_2(i+1)} = \frac{3}{log_2(1+1)} + \frac{2}{log_2(2+1)} + \frac{1}{log_2(3+1)} = \frac{3}{1} + \frac{2}{1.58} + \frac{1}{2} = 4.78$$

$$IDCG_3 = \sum_{i=1}^{K} \frac{rel_i^{opt}}{log_2(i+1)} = \frac{3}{log_2(1+1)} + \frac{3}{log_2(2+1)} + \frac{2}{log_2(3+1)} = \frac{3}{1} + \frac{3}{1.58} + \frac{2}{2} = 5.89$$

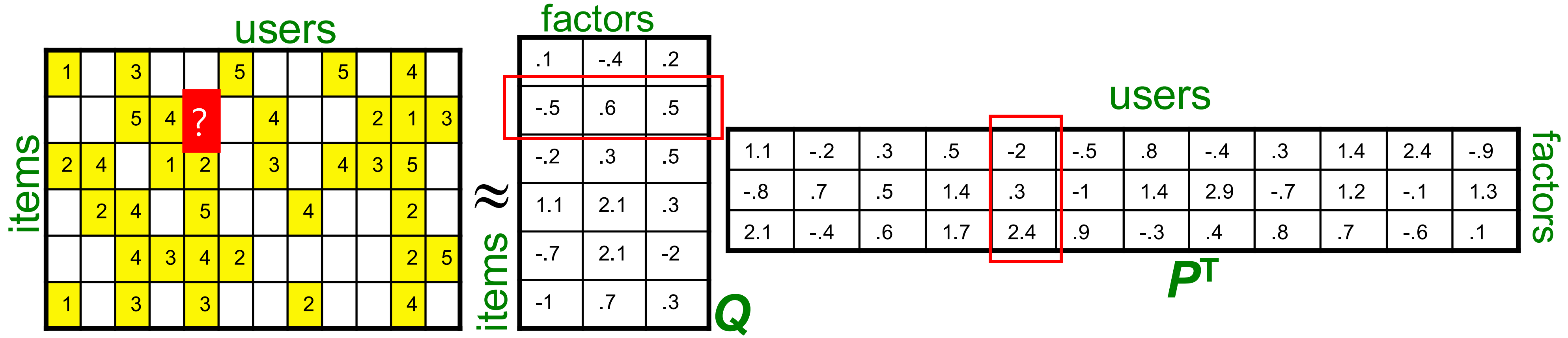
$$NDCG_3 = \frac{DCG}{IDCG} = \frac{4.78}{5.89} = 0.81$$

Outline

1. Introduction
2. **<u>Latent Factor Model Practice (LF) - NCF</u>**
3. Graph Collaborative Filtering Practice (GCF) – NGCF

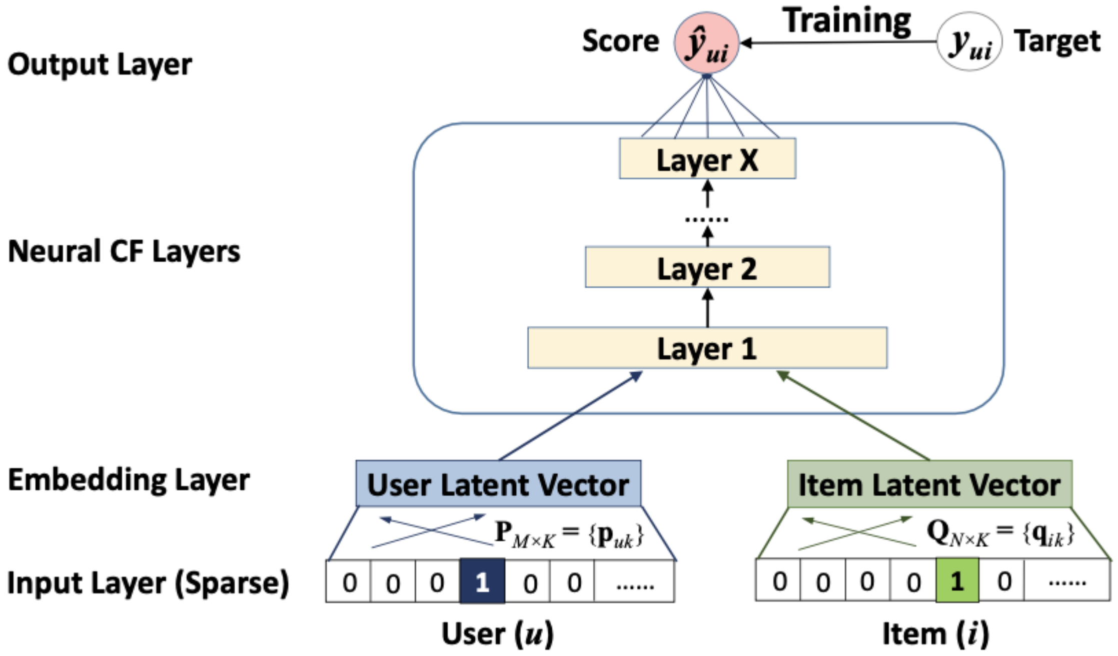
Recap - Neural Collaborative Filtering

- For learning latent factor model, use **inner product** as prediction(past).



- Inner product may not be sufficient to capture the complex structure of interaction.
- Using **deep neural network** as interaction function!

Recap - Neural Collaborative Filtering



[Loss function : MSE]

$$\frac{1}{|Train|} \sum_{(u,i) \in Train} (\hat{y}_{ui} - y_{ui})^2$$

[Evaluation : RMSE, Recall, Precision]

Data Download & GPU Setting

- For faster model training, we use free GPU provided from Google Colab.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

- Download the dataset.

```
ratings_path = './ml-latest-small/ratings.csv'
df = pd.read_csv(ratings_path)
print(df.head())
```

Data Processing

- For PyTorch setting, change the raw data as data frame format .

```
class MovieLens:
  def __init__(self, users, movies, ratings):
    self.users = users
    self.movies = movies
    self.ratings = ratings

  def __len__(self):
    return len(self.users)

  def __getitem__(self,item):
    users = self.users[item]
    movies = self.movies[item]
    ratings = self.ratings[item]
    return {'users': torch.tensor(users, dtype = torch.long).to(device),
            'movies': torch.tensor(movies, dtype = torch.long).to(device),
            'ratings': torch.tensor(ratings, dtype=torch.long).to(device)}
```

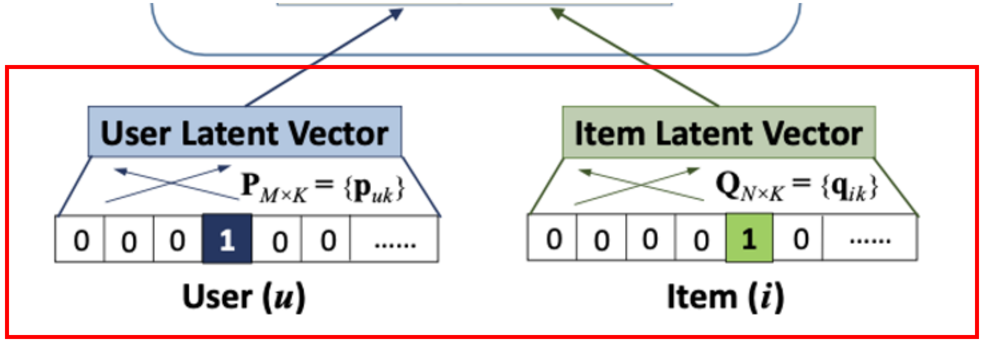
Data Processing

- Encode input as integer and split data.

```
# for setting dataframe column(userId, movieId) data type
lbl_user = preprocessing.LabelEncoder()
lbl_movie = preprocessing.LabelEncoder()

df.userId = lbl_user.fit_transform(df.userId.values)
df.movieId = lbl_movie.fit_transform(df.movieId.values)

# devide original dataframe into train, test dataframe
df_train, df_test = model_selection.train_test_split(df, \
    test_size=0.1, random_state=42, stratify=df.rating.values)
```

Model Setting

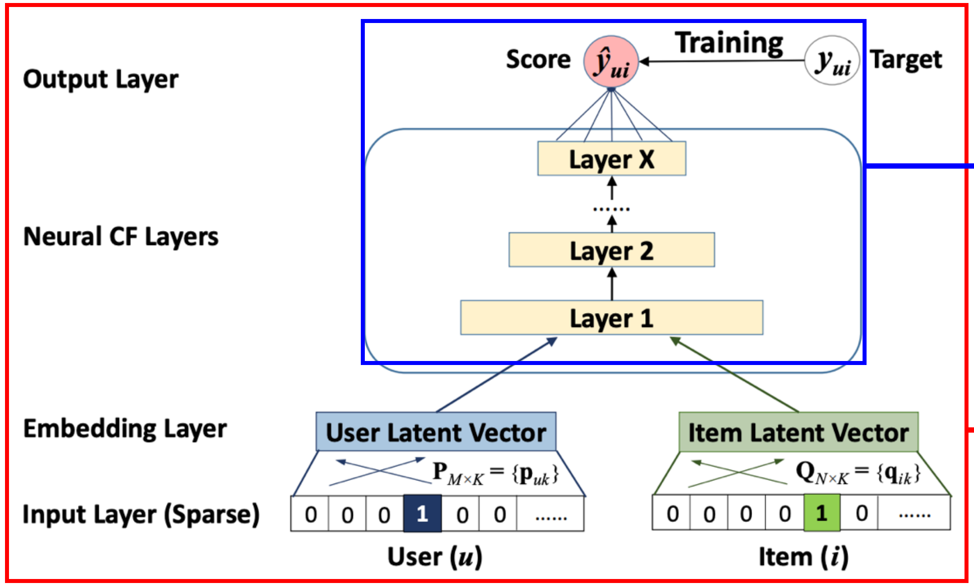
1. Embedding layer(Embedding dimension: 32)




```
# embedding layer(map each user & item to different embedding : will be also trained)
self.user_embedding = nn.Embedding(n_users, 32)
self.movie_embedding = nn.Embedding(n_movies, 32)
```

Model Setting

2. NCF layer(2 layer (including output layer) + 1 activation function)




```
self.fc1 = nn.Linear(64, 32)
self.relu = nn.ReLU()
self.fc2 = nn.Linear(32, 1)
```

```
def forward(self, users, movies, ratings = None): # (users, movies, ratings)
  user_embedding = self.user_embedding(users)
  movie_embedding = self.movie_embedding(movies)
  input_embedding = torch.cat([user_embedding, movie_embedding], dim = 1)

  hidden_feature = self.fc1(input_embedding)
  hidden_feature = self.relu(hidden_feature)
  output = self.fc2(hidden_feature)

  return output
```

Training

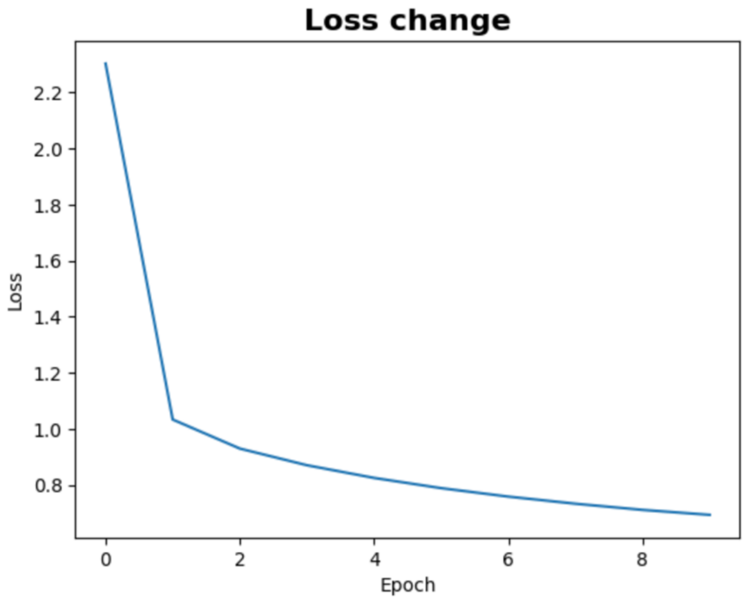
- Check the training settings:
  - Batch size: 128
  - Optimizer: Adam with learning rate = 1e-3
  - Loss function: MSELoss

```
# convert each dataset(numpy -> tensor)
train_dataset = MovieLens(users = df_train.userId.values, \
                          movies = df_train.movieId.values, ratings = df_train.rating.values)
test_dataset = MovieLens(users = df_test.userId.values, \
                         movies = df_test.movieId.values, ratings = df_test.rating.values)
```

```
# model create
model = Collaborative_Filtering(n_users = len(lbl_user.classes_), \
                                n_movies = len(lbl_movie.classes_)).to(device)

# Optimizer, Objective function
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
loss_func = nn.MSELoss(reduction= 'mean')
```

Training

- Train the model with 10 epochs.
- Observe how loss changes.




```
epochs = 10
total_loss = 0
iter_cnt = 0
all_losses_list = []

model.train()

for epoch in range(epochs):
  total_loss = 0
  epoch_check = 0
  for i, train_data in enumerate(train_loader):
    batch_size = len(train_data['users'])
    output = model(train_data['users'],train_data['movies'])
    rating = train_data['ratings'].view(batch_size, -1).to(torch.float32)
    loss = loss_func(output, rating)
    total_loss = total_loss + (loss.item() * batch_size)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Evaluation

Evaluation – RMSE

```
model.eval()

with torch.no_grad():
  for i, batched_data in enumerate(test_loader):
    model_output = model(batched_data['users'],batched_data['movies'])
    model_output_batch = model_output.cpu().numpy().squeeze(axis=1).tolist()
    model_output_list += (model_output_batch)

    target_rating = batched_data['ratings']
    target_rating_batch = target_rating.cpu().numpy().tolist()
    target_rating_list += target_rating_batch

mse = mean_squared_error(target_rating_list, model_output_list)
rms = np.sqrt(mse)
```

Evaluation

Evaluation – Recall@10, Precision@10

```
with torch.no_grad():
  for i, batched_data in enumerate(test_loader):
    users = batched_data['users']
    movies = batched_data['movies']
    ratings = batched_data['ratings']

    model_output = model(batched_data['users'], batched_data["movies"])

    for i in range(len(users)):
      user_id = users[i].item()
      movie_id = movies[i].item()
      pred_rating = model_output[i][0].item()
      true_rating = ratings[i].item()

      user_est_true[user_id].append((pred_rating, true_rating))
```

Make dictionary {user: rating(pred), rating(true)}

Evaluation

Evaluation – Recall@10, Precision@10

```
for user_id, user_ratings in user_est_true.items():
  user_ratings.sort(key=lambda x: x[0], reverse =True)

  # get the number for real relevant items = denominator of recall@k
  n_real_relevant= sum((true_r >= threshold) for (_, true_r) in user_ratings)

  # k recommended ratings
  recommended_k = user_ratings[:k]

  # get the number of recommented item that is actually relevant with real relevant.
  n_real_relevant_in_top_k = sum((true_r >= threshold) for (est, true_r) in recommended_k)

  # precision@k
  precisions[user_id] = n_real_relevant_in_top_k / k

  # recall@k
  recalls[user_id] = n_real_relevant_in_top_k / n_real_relevant if n_real_relevant != 0 else 0
```

Precision@10: (relevant item in top 10) /10

Recall@10: (relevant item in top 10) / total relevant item

Evaluation

Result

```
rms = mean_squared_error(target_rating_list, model_output_list, squared=False)
print(f"rms: {rms}")
```

```
rms: 0.93895540227349
```

```
# Precision and recall can then be averaged over all users
print(f"precision @ {k}: {sum(prec for prec in precisions.values()) / len(precisions)}")

print(f"recall @ {k} : {sum(rec for rec in recalls.values()) / len(recalls)}")
```

```
precision @ 10: 0.4173333333333326
recall @ 10 : 0.7479948056108262
```

Question

- Can we adapt these models for other platforms? **Yes!**
  - **Yelp** (Business venues), **LastFM** (Music), **Gowalla** (Locations) etc.
  - The data above is open-source, You can adapt our model mechanism for these datasets.

- Deeper understanding:
  - In the real world, rating information is **very sparse** (Too expensive) → Hard to train ML model.
  - Implicit data (such as clicks and dwell time) is generally used to model user preferences.
  - Numerous studies are continuously being conducted to create better latent vectors from implicit data beyond simple latent factor modeling (**Graph**, Social network, etc.).

Outline

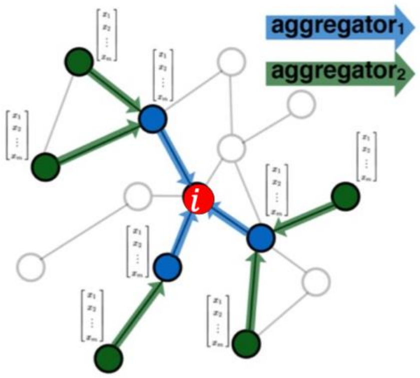
1. Introduction
2. Latent Factor Model Practice (LF) - NCF
3. **<u>Graph Collaborative Filtering Practice (GCF) – NGCF</u>**

Recap - Graph Collaborative Filtering

[Graph neural networks (GNNs)]

- GNNs are neural networks for graphs (G=(V, E))
  - V : Node set
  - E : Edge set
  - G can be represented as an adjacency matrix $\mathbf{A} \in \{0,1\}^{|V|\times|V|}$
  - For each node $\in V$, it has its own feature and stored in the feature matrix $\mathbf{X} \in \mathbb{R}^{|V|\times d}$
    - $\mathbf{X}$ can be a learnable matrix

- From $\mathbf{X}$ and $\mathbf{A}$, network is trained and it generates useful representation (vector) for node/graph

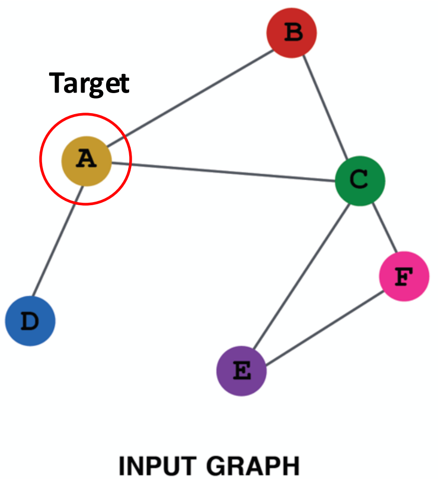


Propagate and transform information

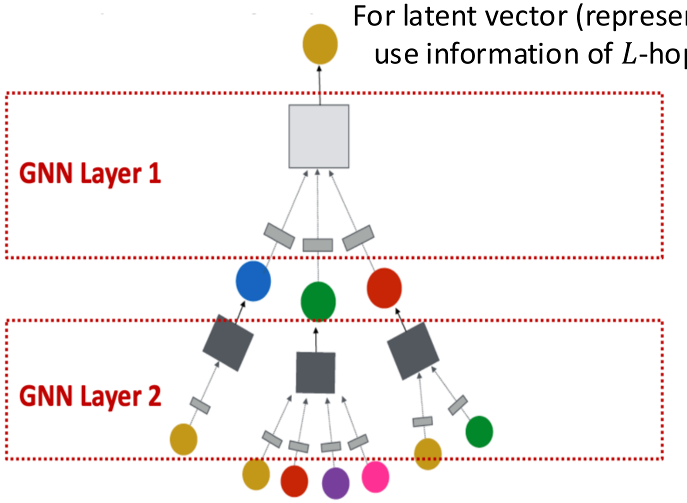
Recap - Graph Collaborative Filtering

[Graph neural networks (GNNs)]

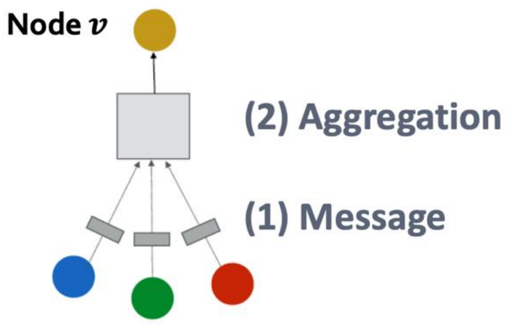
- For each GNN layer:
  - Message Construction
  - Aggregation of Neighbors
  - Update Target Node Vector



For latent vector (representation) of A, use information of L-hop neighbors



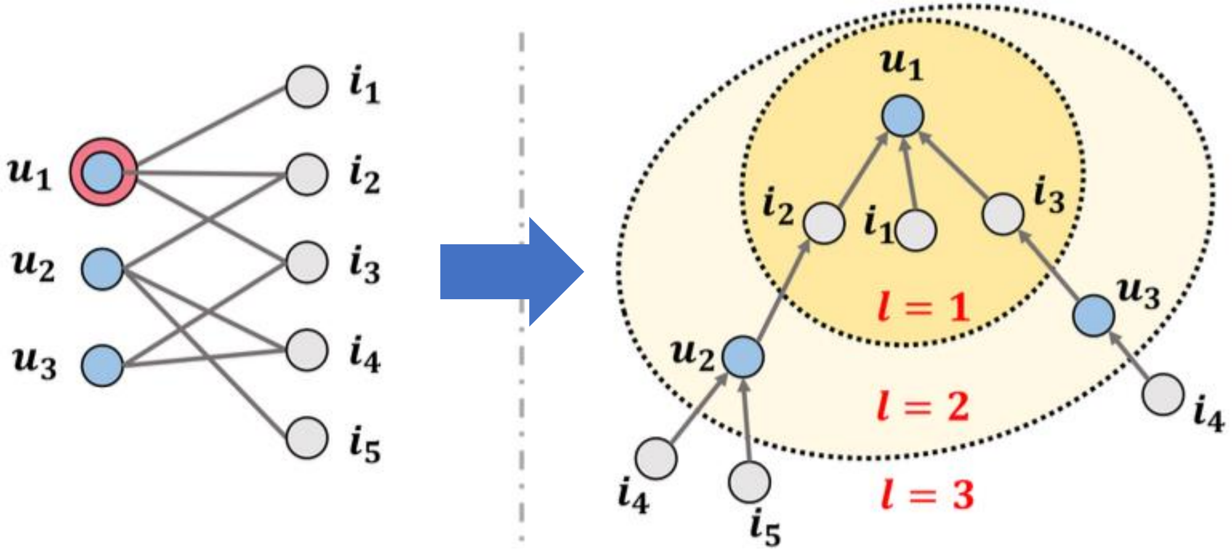
2-hop neighbors



Recap - Graph Collaborative Filtering

[Graph neural networks (GNNs)]

- Graph structure and GNN can be used for recommender systems.
  - Users and items to nodes
  - Interaction between users and items to edges

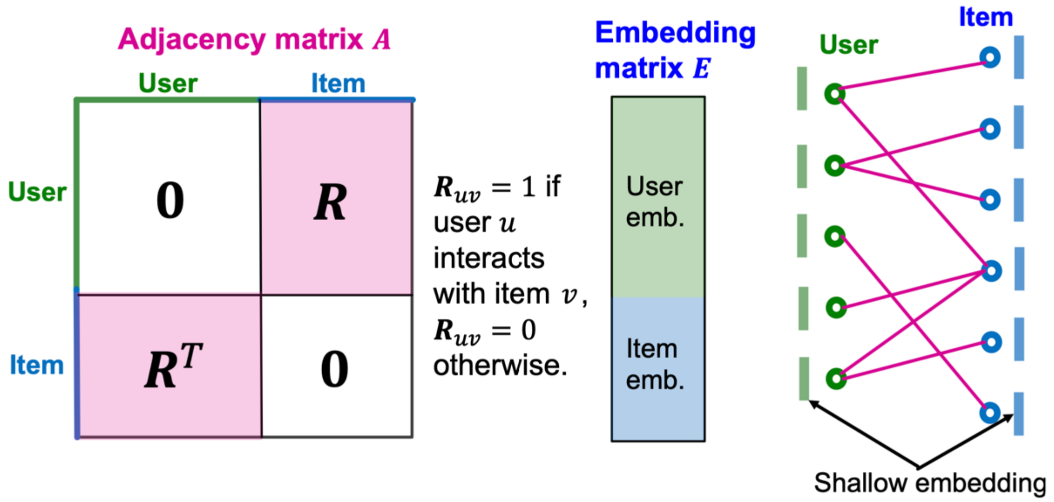


Recap - Graph Collaborative Filtering

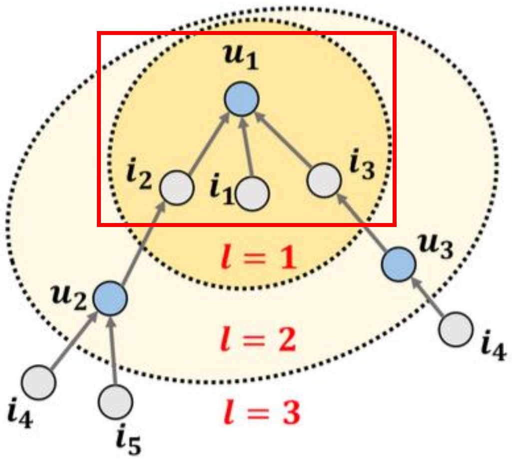
[Assumption]

- Dataset – Utility matrix ($\mathbf{R} \in \mathbb{R}^{m \times n}$) with implicit interaction
    - m : # of users n : # of items
    - $\mathbf{R}_{uv} = 1$ if user u interacts with item v, else $\mathbf{R}_{uv} = 0$

- Setting for GNNs – Adjacency matrix ($\mathbf{A}$), Feature (Embedding) matrix ($\mathbf{X} = \mathbf{H}$)
    - Utility matrix -> Adjacency matrix ($\mathbf{A}$)
    - Learnable embedding matrix ($\mathbf{H}_U^{(0)}, \mathbf{H}_I^{(0)}$)
        - Embedding for user u : $h_u^{(0)}$
        - Embedding for item i : $h_i^{(0)}$



Recap - NGCF



- Message Construction ($m_{u\leftarrow i}$, $m_{i\leftarrow u}$)
  - $m_{u\leftarrow i} = \frac{1}{\sqrt{N(u)}\sqrt{N(i)}} (\mathbf{W}_1^l h_i^{(l-1)} + \mathbf{W}_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right))$
  - $N(\cdot)$ = Number of neighbors (ex - $N(u_1) = 3$, $N(i_2) = 2$)
  - $\mathbf{W}_1^l, \mathbf{W}_2^l$ = Learnable weights for each lth GNN layer

- Message Aggregation & Update ($\mathrm{COMBINE}(m_{u\leftarrow u},\ \mathrm{AGG}(\{m_{u\leftarrow i} | i \in N(u)\}))$)
  - $\mathrm{COMBINE}(\cdot) = \sigma(m_{u\leftarrow u} + \sum_{i\in N(u)} m_{u\leftarrow i})$ ($m_{u\leftarrow u} = \mathbf{W}_1^l h_u^{(l-1)}$)
  - $h_u^{(l)}$ = Result of $\mathrm{COMBINE}(\cdot)$ ($h_u^{(0)} \in \mathbf{H}_U^{(0)}, h_u^{(1)} \in \mathbf{H}_I^{(0)}$)

Recap - NGCF

- **Matrix Form**
  - For calculation efficiency, Each GNN-based layer is implemented by **matrix multiplication.**
  - **NGCF**
    - $\mathbf{H}^{(l+1)} = \sigma((\mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5} + \mathbf{I})\mathbf{H}^{(l)}\mathbf{W}_1^{(l+1)} + \mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}\mathbf{H}^{(l)} \odot \mathbf{H}^{(l)}\,\mathbf{W}_2^{(l+1)})$
    - $\mathbf{D} \in \mathbb{R}^{(m+n)\times(m+n)}$ = Degree matrix of $\mathbf{A}$ ($\mathbf{D}_{aa} = N(a)$ else 0)
    - $\mathbf{I}$ = Identity matrix

Recap - NGCF

- **Score Prediction**
  - After L layers, generate final representations of users and items.
  - **NGCF**
    - $h_u^{final} = h_u^{(0)} | \cdots | h_u^{(L)},\ h_i^{final} = h_i^{(0)} | \cdots | h_i^{(L)}$
  - Using final representations, predict the interaction between user and item.
    - $\hat{r}_{ui} = (h_u^{final})^{\mathrm{T}}(h_i^{final})$

Loss function

- We use BPR loss to optimize our models.
  - To maximize the scores of positive pairs, and minimize those of negative pairs.
  - Positive pairs (real interactions in train data) / Negative pairs (non-interacted pairs)
- For convenience, we unify the loss function for both models.
  - Strictly, we should add L2-norm of $\mathbf{W}$ as a regularization in NGCF loss.

```python
def bpr_loss(self, user_emb, pos_item_emb, neg_item_emb, reg_weight=1e-4):
    pos_scores = torch.sum(user_emb * pos_item_emb, dim=1)
    neg_scores = torch.sum(user_emb * neg_item_emb, dim=1)
    loss = -torch.mean(F.logsigmoid(pos_scores - neg_scores))
    reg_loss = reg_weight * (user_emb.norm(2).pow(2) + pos_item_emb.norm(2).pow(2) \
                             + neg_item_emb.norm(2).pow(2)) / user_emb.size(0)
    return loss + reg_loss
```

$$Loss = \sum_{(u,i,j)\in O} -\ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda \|\Theta\|_2^2$$

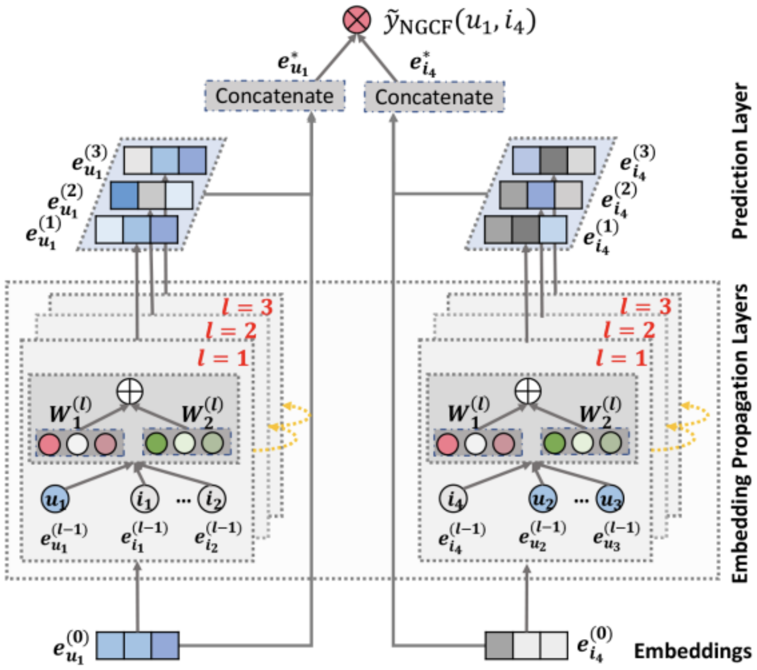
Create Graph from Data

- To utilize the graph structure to recommender system
- Users and movies will be used as **nodes** for graph
- We generate edge between users and movies
  - If user rates movie with more than 1, we generate edge between them

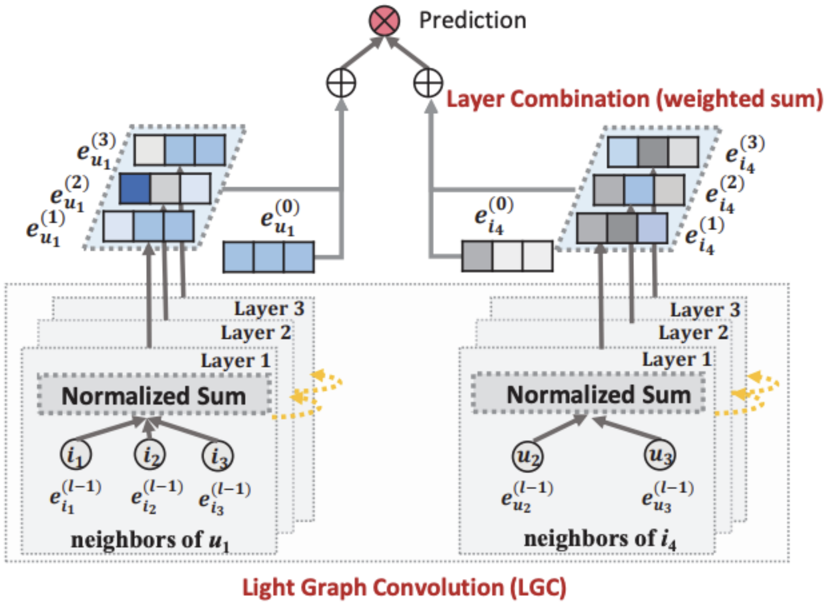
```python
# Create edge_index
def create_edge_index(df, rating_threshold=1.0):
    src, dst = [], []
    for _, row in df.iterrows():
        if row['rating'] >= rating_threshold:
            src.append(row['userId'])
            # item indices after user indices
            dst.append(row['movieId'] + num_users)
    return torch.tensor([src, dst], dtype=torch.long)

edge_index = create_edge_index(rating_df)
```

Models

- We will build two RecSys classes, NGCF & LightGCN which are based on graph.

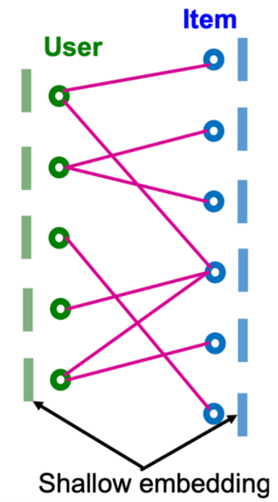


NGCF



LightGCN

NGCF – Implementation (1)



User (src)	U1	U1	U2	U2	U2	U3	U3	U4	U5	U5
Item (dst)	I1	I2	I3	I4	I5	I2	I1	I6	I7	I8

$$m_{u\leftarrow i} = \boxed{\frac{1}{\sqrt{N(u)}\sqrt{N(i)}}}(\mathbf{W}_1^l h_i^{(l-1)} + \mathbf{W}_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right))$$

$$m_{i\leftarrow u} = \boxed{\frac{1}{\sqrt{N(u)}\sqrt{N(i)}}}(\mathbf{W}_1^l h_u^{(l-1)} + \mathbf{W}_2^l \left(h_u^{(l-1)} \odot h_i^{(l-1)}\right))$$

Node	Deg
U1	2
U2	3
U3	2
U4	1
U5	2
…	
I1	2

[deg]

2	3	3	3	3	2	2	1	2	2
U1	U1	U2	U2	U2	U3	U3	U4	U5	U5
I1	I2	I3	I4	I5	I2	I1	I6	I7	I8
2	2	1	1	1	2	2	1	1	1

1/2	$1/\sqrt{6}$	$1/\sqrt{3}$	$1/\sqrt{3}$	$1/\sqrt{3}$	1/2	1/2	1	$1/\sqrt{2}$	$1/\sqrt{2}$

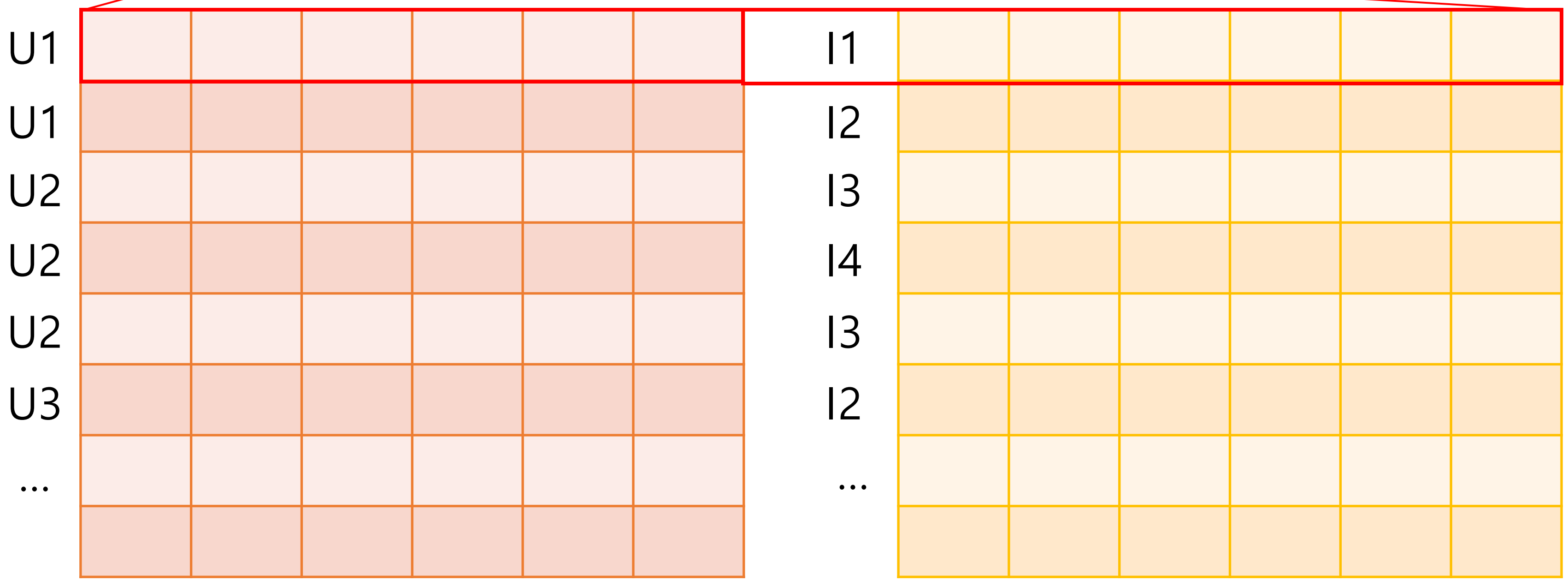
[norm]

NGCF – Implementation (2)

$$m_{u\leftarrow i} = \frac{1}{\sqrt{N(u)}\sqrt{N(i)}}\boxed{\left(\mathbf{W}_1^l h_i^{(l-1)} + \mathbf{W}_2^l\left(h_i^{(l-1)} \odot h_u^{(l-1)}\right)\right)}$$

$$m_{i\leftarrow u} = \frac{1}{\sqrt{N(u)}\sqrt{N(i)}}\boxed{\left(\mathbf{W}_1^l h_u^{(l-1)} + \mathbf{W}_2^l\left(h_u^{(l-1)} \odot h_i^{(l-1)}\right)\right)}$$

User (src)	U1	U1	U2	U2	U2	U3	U3	U4	U5	U5
Item (dst)	I1	I2	I3	I5	I3	I2	I1	I6	I7	I8

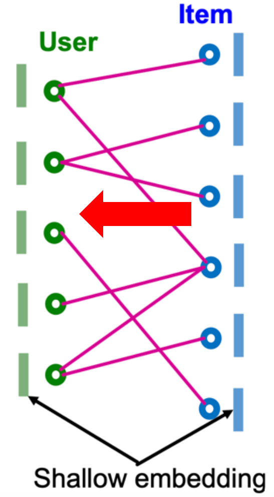


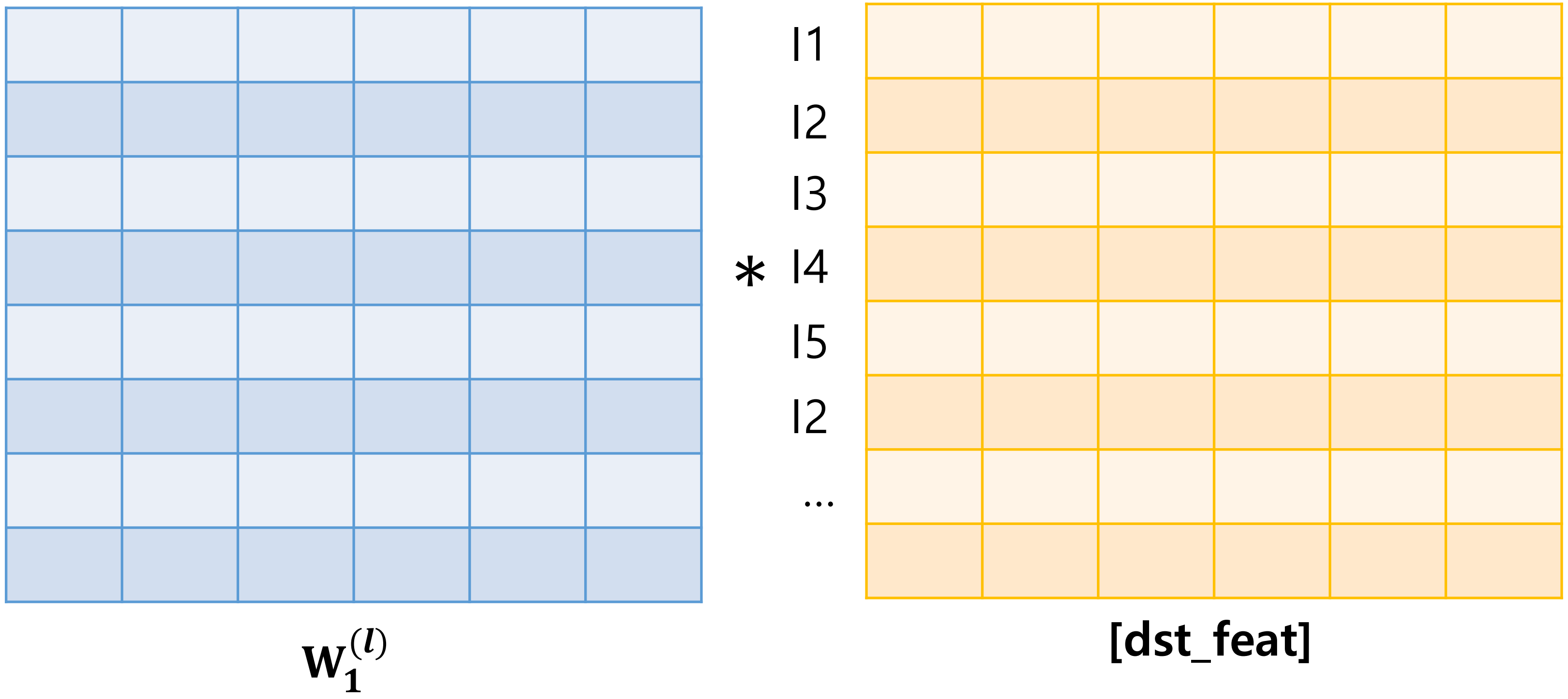
[src_feat] [dst_feat]

NGCF – Implementation (3)

User (src)	U1	U1	U2	U2	U2	U3	U3	U4	U5	U5
Item (dst)	I1	I2	I3	I4	I5	I2	I1	I6	I7	I8

$$m_{u\leftarrow i} = \frac{1}{\sqrt{N(u)}\sqrt{N(i)}}\left(\boxed{\mathbf{W}_1^l h_i^{(l-1)}} + \mathbf{W}_2^l\left(h_i^{(l-1)} \odot h_u^{(l-1)}\right)\right)$$

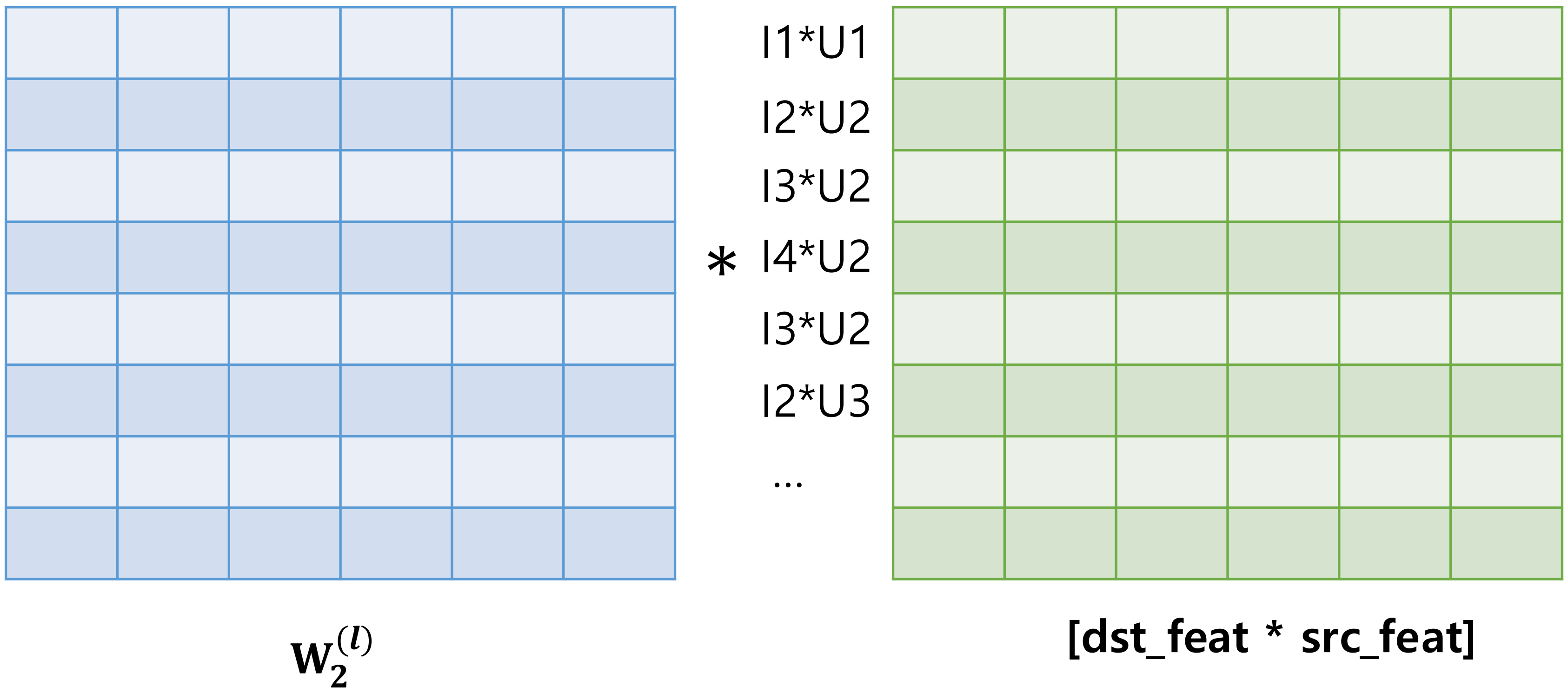




NGCF – Implementation (4)

User (src)	U1	U1	U2	U2	U2	U3	U3	U4	U5	U5
Item (dst)	I1	I2	I3	I4	I5	I2	I1	I6	I7	I8

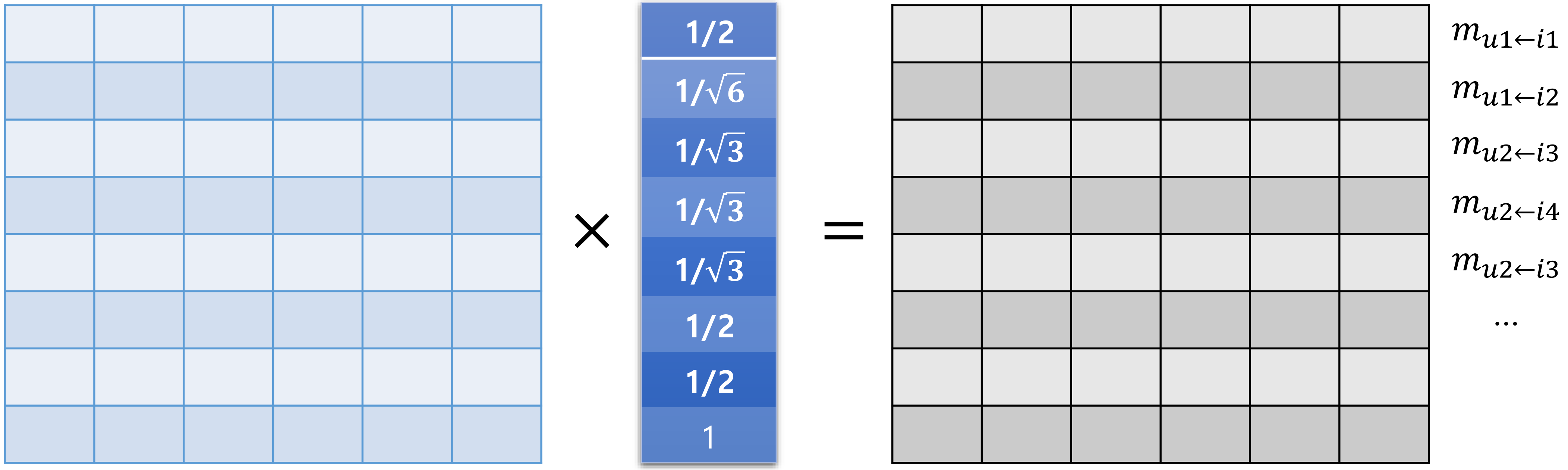
$$m_{u\leftarrow i} = \frac{1}{\sqrt{|N(u)|}\sqrt{|N(i)|}}(\mathbf{W}_1^l h_i^{(l-1)} + \boxed{\mathbf{W}_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right)})$$



NGCF – Implementation (5)

User (src)	U1	U1	U2	U2	U2	U3	U3	U4	U5	U5
Item (dst)	I1	I2	I3	I4	I5	I2	I1	I6	I7	I8

$$m_{u\leftarrow i} = \frac{1}{\sqrt{N(u)}\sqrt{N(i)}}(\mathbf{W}_1^l h_i^{(l-1)} + \mathbf{W}_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right))$$

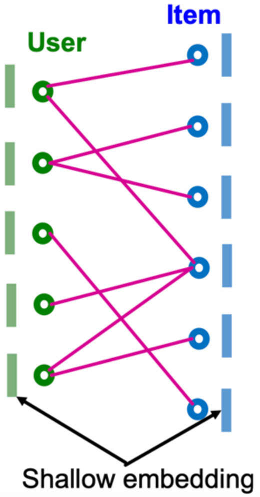


$\mathbf{W}_1^l h_i^{(l-1)} + \mathbf{W}_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right)$

[norm]

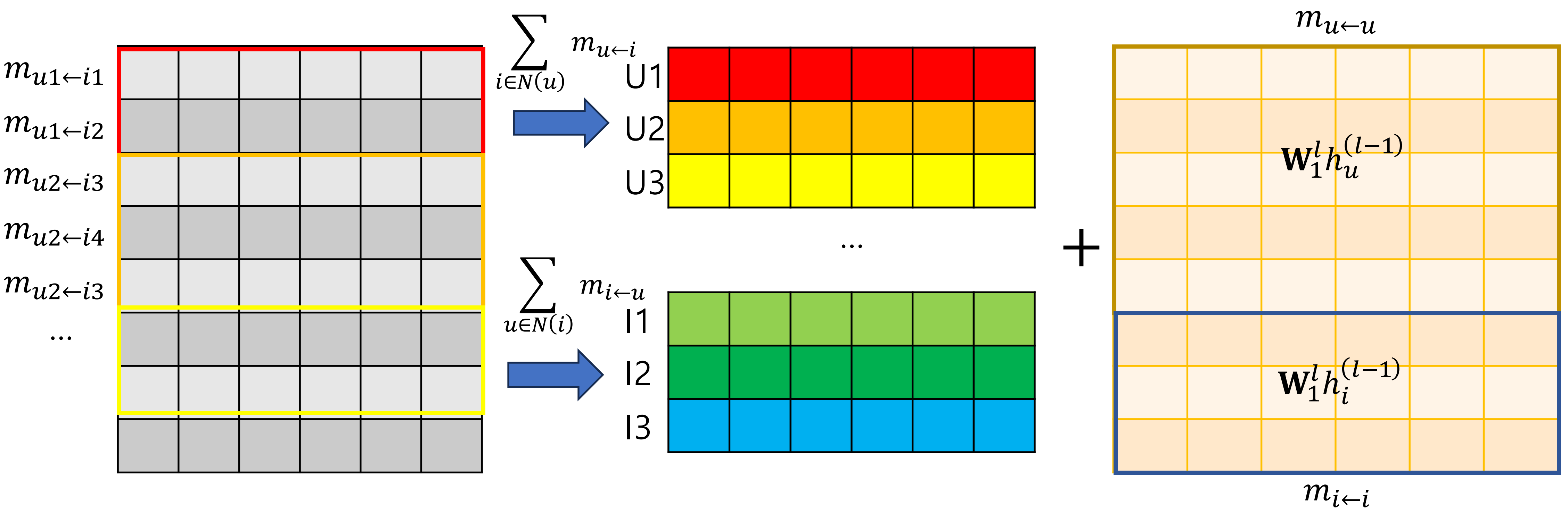
$\frac{1}{\sqrt{N(u)}\sqrt{N(i)}}(\mathbf{W}_1^l h_i^{(l-1)} + \mathbf{W}_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right))$

NGCF – Implementation (6)



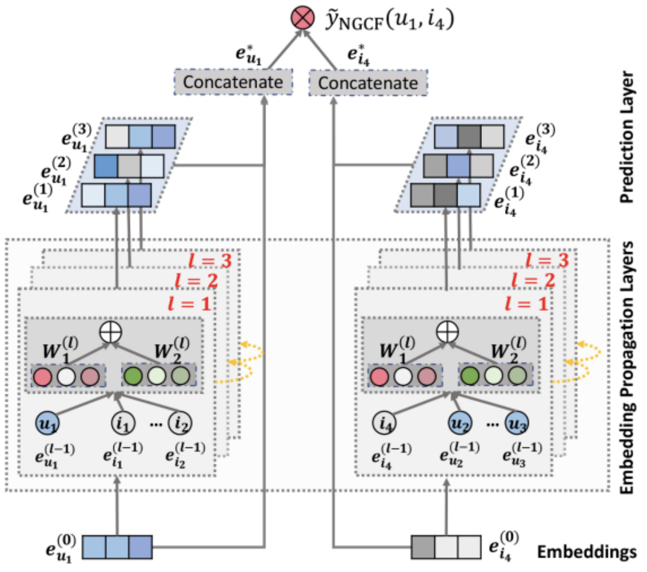
User (src)	U1	U1	U2	U2	U2	U3	U3	U4	U5	U5
Item (dst)	I1	I2	I3	I4	I5	I2	I1	I6	I7	I8

$$\sigma(m_{u\leftarrow u} + \textstyle\sum_{i\in N(u)} m_{u\leftarrow i})$$



Neural Graph Collaborative Filtering

- NGCF Layer class contains:
  - Degree calculation of each node.

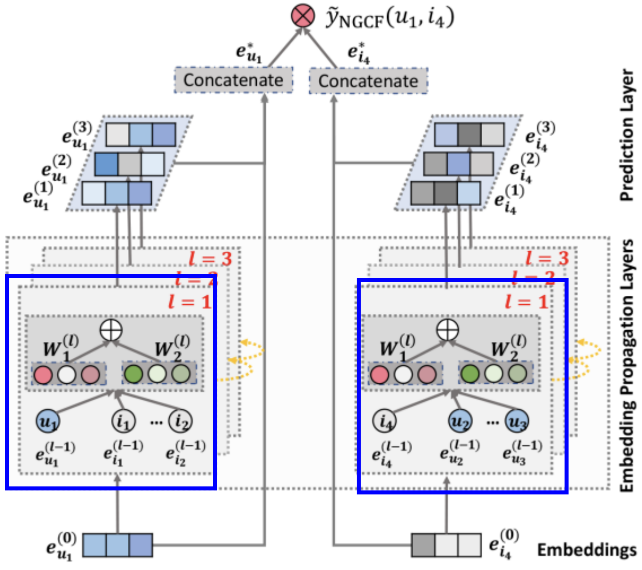


$$m_{u \leftarrow i} = \boxed{\frac{1}{\sqrt{N(u)}\sqrt{N(i)}}}\left(W_1^l h_i^{(l-1)} + W_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right)\right)$$

```python
def forward(self, edge_index, node_features):
    src, dst = edge_index

    deg = torch.zeros(node_features.size(0), device=node_features.device)
    deg.index_add_(0, src, torch.ones_like(src, dtype=torch.float))
    deg.index_add_(0, dst, torch.ones_like(dst, dtype=torch.float))

    ##### To do #####
    norm =
```

Neural Graph Collaborative Filtering

- NGCF Layer initialization & forward contains:
  - Parameter initialization.
  - MSG & AGG function.

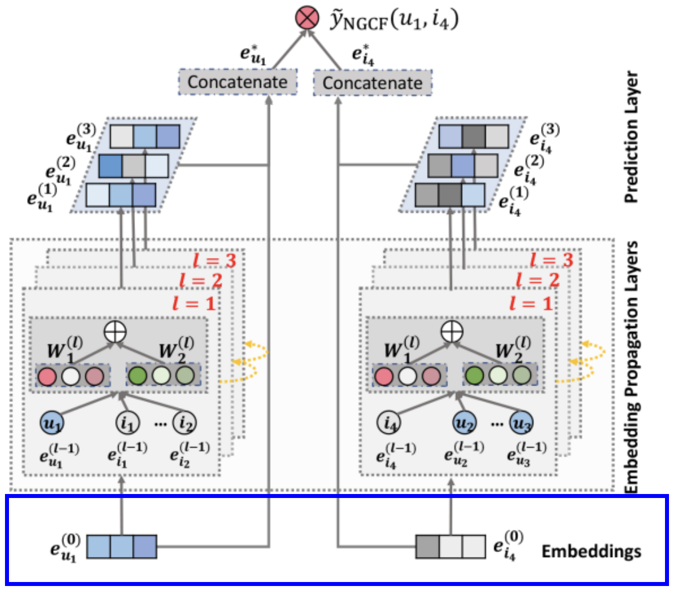


$$m_{u\leftarrow i} = \frac{1}{\sqrt{N(u)}\sqrt{N(i)}}\boxed{\left(W_1^l h_i^{(l-1)} + W_2^l \left(h_i^{(l-1)} \odot h_u^{(l-1)}\right)\right)}$$

```
# edge_messages for user(src) = m_(u<-i)) 결과 저장.
# Hint: step1. self.W1(h_i) + self.W2(h_u * h_i) 계산
# Hint: step2. 최종 m_(u<-i)를 위해선 앞선 norm을 앞서 계산한 message에 곱하기
edge_messages_for_src = self.W1(dst_feat) + self.W2(src_feat * dst_feat)
edge_messages_for_src *= norm.unsqueeze(1)

# edge_messages for movie(dst) = m_(i<-u)) 결과 저장.
# Hint: step1. self.W1(h_u) + self.W2(h_i * h_u) 계산
# Hint: step2. 최종 m_(i<-u)를 위해선 앞선 norm을 앞서 계산한 message에 곱하기
edge_messages_for_dst = ## fill this part ##
edge_messages_for_dst *= ## fill this part ##
```

Neural Graph Collaborative Filtering

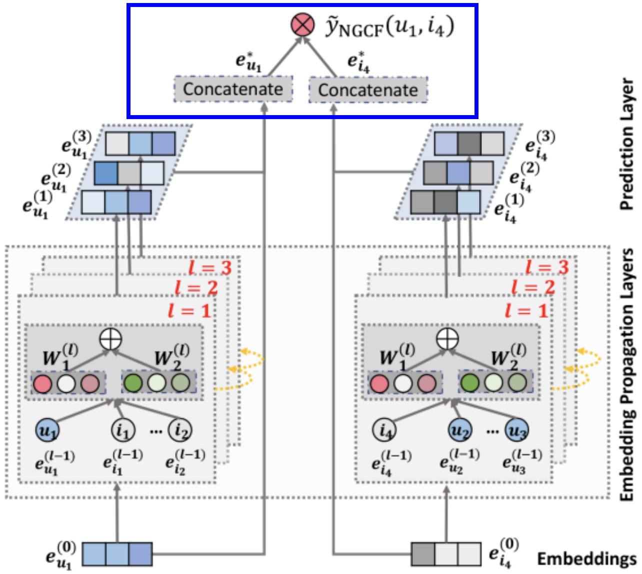
- NGCF class initialization contains:
  - Initialization and updating embeddings of users & items.




```
### self.node_embeddings = H_(0) = learnable embedding matrix ###
self.node_embeddings = nn.Embedding(self.num_users+self.num_items,self.embedding_dim)
nn.init.xavier_uniform_(self.node_embeddings.weight)
```

Neural Graph Collaborative Filtering

- NGCF class forward contains:
  - Concatenation of embeddings from each layer.




```python
def forward(self, edge_index):
    node_features = self.node_embeddings.weight
    layer_outputs = [node_features]
    for layer in self.layers:
        node_features = ## fill this part ##
        layer_outputs.append(node_features)

    # Hint: NGCF의 final feature(representation)은 layer 별 feature에 대한 concatenated vector
    # Hint: 최종 final feature matrix에는 [feuture_vector for users + feature_vector for items]가 들어있음.

    final_features = ## fill this part ##
    user_features = ## fill this part ##
    item_features = ## fill this part ##
    return user_features, item_features
```

Evaluations

- For evaluation, we use the metrics:
  - Recall@10
  - Precision@10
  - NDCG@10

```python
def evaluate(user_features, item_features, test_edge_index, k):
    user_pos_items = defaultdict(list)
    E_test = test_edge_index.size(1)
    for i in range(E_test):
        u = test_edge_index[0, i].item()
        it = test_edge_index[1, i].item() - num_users
        user_pos_items[u].append(it)

    recalls, precisions, ndcgs = [], [], []
    for user, pos_items in user_pos_items.items():
        user_emb = user_features[user]
        scores = torch.matmul(item_features, user_emb)
        topk_scores, topk_indices = torch.topk(scores, k=k)
        topk_indices = topk_indices.cpu().numpy().tolist()

        hits = 0
        dcg = 0.0
        idcg = 0.0
        n_pos = len(pos_items)

        for rank, item_idx in enumerate(topk_indices):
            if item_idx in pos_items:
                hits += 1
                dcg += 1.0 / math.log2(rank + 2)
        for rank in range(min(n_pos, k)):
            idcg += 1.0 / math.log2(rank + 2)

        recall_u = hits / n_pos
        precision_u = hits / k
        ndcg_u = dcg / idcg if idcg > 0 else 0.0

        recalls.append(recall_u)
        precisions.append(precision_u)
        ndcgs.append(ndcg_u)

    recall = np.mean(recalls)
    precision = np.mean(precisions)
    ndcg = np.mean(ndcgs)
    return recall, precision, ndcg
```

Results

- Create your model object.
- Train the model and test the performance.

```
print("===== Train NGCF =====")
train(
    model=ngcf_model,
    optimizer=optimizer_ngcf,
    train_edge_index=train_edge_index,
    val_edge_index=val_edge_index,
    num_epochs=30,
    batch_size=1024,
    device=device,
    k=10
)

print("===== Test NGCF =====")
test(
    model=ngcf_model,
    train_edge_index=train_edge_index,
    test_edge_index=test_edge_index,
    k=10,
    device=device
)
```

References

- [WWW ‘17] Neural Collaborative Filtering (NCF)
  - https://arxiv.org/abs/1708.05031

- [SIGIR ‘19]Neural Graph Collaborative Filtering (NGCF)
  - https://arxiv.org/abs/1905.08108