COMP4434 Big Data Analytics Project

Recommendation System based on the MoiveLens Dataset

ZHANG Haolin, LIU Chenxi, BAI Haoran

1. Introduction

Predicting the probability that a target user will click through the recommended item is known as the click-through rate (CTR) prediction problem. Driven by the increasing industrial demand, CTR prediction has become one of the top concerns for large-scale recommendation systems [1]. These large-scale recommendation systems can be deployed in platforms, including e-business and social media, to feed advertisements or content catering to the preferences of target users [2].

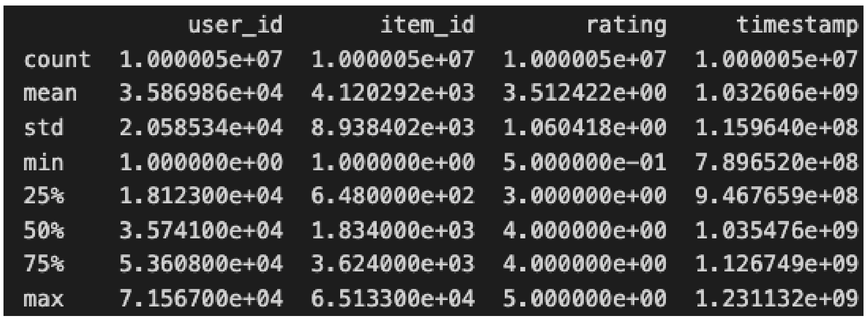
Although the user behavior in a click-through context can be simplified as a binary value, either positive or negative, the billion-scale data and the industrial requirement of real-time reaction prevent some of the high-performance models from being utilized directly. Also, the challenges of CTR prediction can be raised by cold start due to the lack of user-item interactions of new items [3] and the sparsity of the user-item interaction matrix [4]. To overcome these challenges and achieve efficient predictions on CTR tasks, multiple models like statistical models, factorization machines [5], and deep learning neural networks like DIN [6] have been explored before. Also, as we will discuss in the paper reading part, the Graphical Neural Network (GNN) can be applied to this CTR task, but it has limitations to overcome.

For the evaluation, we chose the broadly accepted CTR metric AUC, which is also utilized by the paper we chose. This AUC metric is the area of the ROC curve ranged [0, 1], with a higher value to indicate better performance.

2. Dataset Analysis and Processing

Considering the billion-scale data of CTR predictions in real-life application, we utilized MovieLens-10M [7] dataset to conduct the experiments on our proposed models and evaluate the performance. This MovieLens-10M dataset consists of 10000054 ratings from 71567 users on 10681 movies. We analyzed and visualized the dataset to investigate the properties of it [8].

The overall summarization of the ratings is:



```
           user_id       item_id        rating     timestamp
count  1.000005e+07  1.000005e+07  1.000005e+07  1.000005e+07
mean   3.586986e+04  4.120292e+03  3.512422e+00  1.032606e+09
std    2.058534e+04  8.938402e+03  1.060418e+00  1.159640e+08
min    1.000000e+00  1.000000e+00  5.000000e-01  7.896520e+08
25%    1.812300e+04  6.480000e+02  3.000000e+00  9.467659e+08
50%    3.574100e+04  1.834000e+03  4.000000e+00  1.035476e+09
75%    5.360800e+04  3.624000e+03  4.000000e+00  1.126749e+09
max    7.156700e+04  6.513300e+04  5.000000e+00  1.231132e+09
```

Fig2.1

The overall distribution of the ratings is shown in Fig2.2, we can see that the majority of users tend to give positive ratings (greater or equal to 3.0 out of 5).

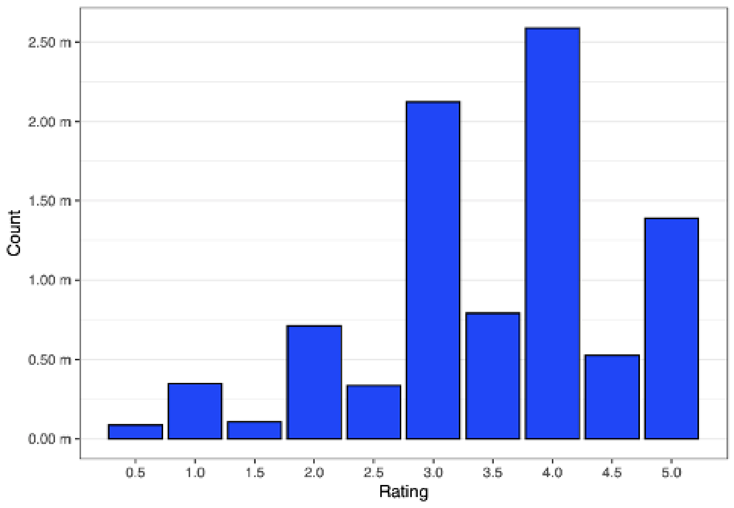


Figure 2.2: The overall distribution of the ratings

From Figure 2.3, we can see that the majority of the movies received 10 ~ 1000 ratings.

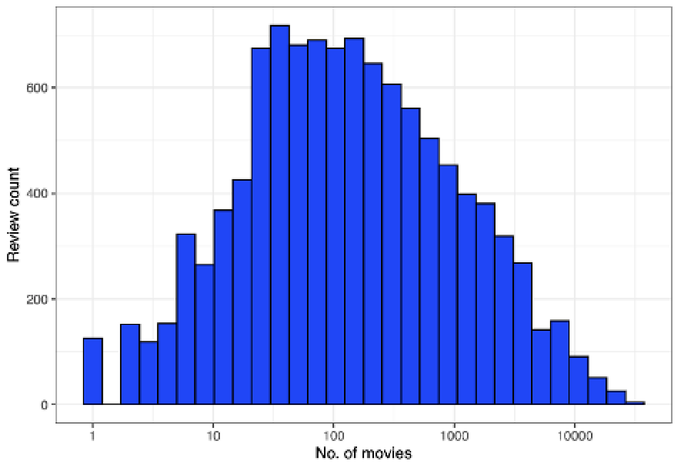


Figure 2.3: ratings count over the movies

After constructing the user-item matrix, which fills in the rating values as entries, we calculated the sparsity with the number of ***zeros*** divided by the total number of entries as *98.67%.* This shows that in the real life, the sparsity of the high-dimension user-item matrix can be quite high.

Aligning with the paper we chose and intending to predict users' CTR based on history data of user-item interaction, we split the training set and test set based on the 90% quantile of *timestamp* attribute value. Specifically, the training dataset is the rating records whose timestamp is before 90% quantile of all timestamps, while the test dataset is the remaining 10% of the records. Also, we follow the experimental settings of the paper we chose to map the [0.5, 5] rating range into binary values, where the rating values of at least 4 are positive (1) otherwise negative (0). This can be regarded as the clicked or not relations between the user and items, which fits in the semantics of the CTR prediction problem.

3. Paper Reading

The paper *Macro Recommendation Graph and Macro Graph Neural Networks for Billion-Scale Recommender Systems* [9] introduces a novel framework aimed at addressing the computational complexity and sampling bias challenges in large-scale recommender systems. The authors propose the use of ***macro nodes*** to efficiently compute and aggregate information at a broader level, enhancing the performance of predictive models in extensive digital environments.

Traditional ***Graph Neural Networks*** (*GNNs*) struggle to manage billion-scale datasets due to their inherent computational demands and the limitations imposed by sampling biases. The proposed ***Macro Recommendation Graph*** and ***Macro Graph Neural Networks*** (*MacGNN*) framework seeks to overcome these issues by improving scalability and accuracy in handling vast amounts of data.

The MacGNN novelty could be summarized as the followings:

3. 1. ***Macro Recommendation Graph*** (*MAG*):

- The MAG is designed to reduce the computational complexity of processing billion-scale datasets by grouping micro nodes (users and items) with similar behavior patterns into macro nodes. This grouping allows for more efficient computation by reducing the node count from billions to hundreds.
- The MAG aims to capture a higher-level representation of user-item interactions by forming macro nodes based on behavioral patterns, enabling more effective information aggregation and embedding refinement.
- The MAG addresses the limitations of traditional micro-recommendation graphs by providing a more suitable structure for billion-scale recommendations, thereby mitigating sampling bias and improving the coverage of user-item behavioral patterns.

3. 2. ***Macro Graph Neural Networks*** (*MacGNN*):

- The MacGNN framework is tailored to operate on the Macro Recommendation Graph, facilitating information aggregation and embedding revision at a macro level.
- MacGNN leverages macro weight modeling to identify target user/item preferences over macro neighbors based on the weights of connected macro edges. This modeling approach helps capture the importance of macro neighboring nodes in the historical interactions of target users/items.
- The architecture of MacGNN includes components such as macro weight modeling, user/item macro neighbor aggregation, recent behavior modeling, and a CTR prediction layer to predict click-through rates in billion-scale recommender systems.
- By aggregating information at a macro level and revising embeddings of macro nodes, MacGNN aims to enhance the performance of recommendation models while maintaining computational efficiency in large-scale environments.

Furthermore, the implementation detail of the MacGNN is available in the Fig. [3.1].

Overall, the proposed framework offers a comprehensive solution to the challenges faced in billion-scale recommender systems. By introducing a macro-level representation and leveraging tailored neural network models, the framework aims to improve recommendation systems operating on massive datasets' accuracy, efficiency, and scalability.

The paper utilizes publicly available datasets (such as ***MovieLens*** [7]) and a billion-scale industrial dataset from a central e-commerce platform for offline evaluation. The datasets are preprocessed to suit the ***click-through rate*** (*CTR*) prediction task, transforming interactions into clicked and non-clicked relationships for training and testing purposes.

In terms of experimental settings, the paper conducts comprehensive experiments on benchmark datasets and real-world online recommendation systems to evaluate the performance of the MacGNN framework. The experiments aim to answer research questions related to the performance comparison with state-of-the-art models, the efficiency of the proposed framework, the impact of different components in MacGNN, and the performance on billion-scale real-world recommendation platforms.

Marco Edges is generated each time when querying a pair of User and Item through their subgraph

Recent Behavior is considering the recent objects as its Neighbors

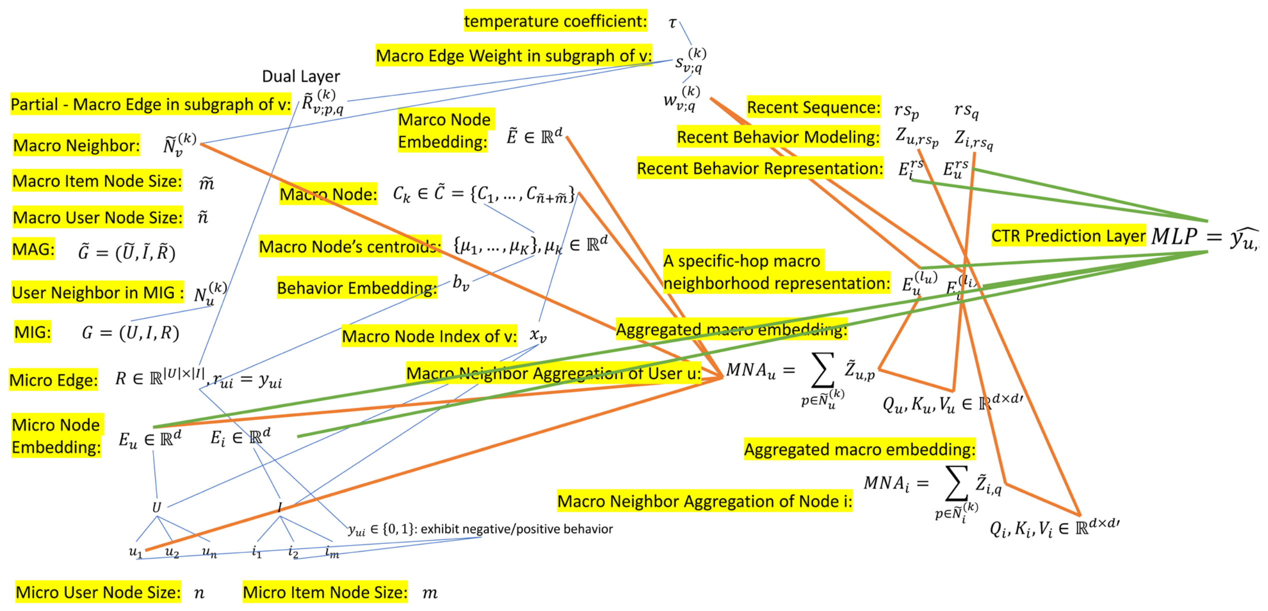

temperature coefficient: τ
Macro Edge Weight in subgraph of v: $s_{v;q}^{(k)}$
$w_{v;q}^{(k)}$
Dual Layer
Partial - Macro Edge in subgraph of v: $\tilde{R}_{v;p,q}^{(k)}$
Marco Node Embedding: $\tilde{E} \in \mathbb{R}^d$
Recent Sequence: rs_p rs_q
Recent Behavior Modeling: Z_{u,rs_p} Z_{i,rs_q}
Recent Behavior Representation: E_i^{rs} E_u^{rs}
Macro Neighbor: $\tilde{N}_v^{(k)}$
Macro Item Node Size: $\tilde{m}$
Macro User Node Size: $\tilde{n}$
Macro Node: $C_k \in \tilde{C} = \{C_1, \dots, C_{\tilde{n}+\tilde{m}}\}$
MAG: $\tilde{G} = (\tilde{U}, \tilde{I}, \tilde{R})$
Macro Node's centroids: $\{\mu_1, \dots, \mu_K\}, \mu_k \in \mathbb{R}^d$
CTR Prediction Layer $MLP = \widehat{y_{u,i}}$
A specific-hop macro neighborhood representation: $E_u^{(l_u)}$ $E_i^{(l_i)}$
User Neighbor in MIG : $N_u^{(k)}$
Behavior Embedding: b_v
MIG: $G = (U, I, R)$
Macro Node Index of v: x_v
Aggregated macro embedding:
Micro Edge: $R \in \mathbb{R}^{|U| \times |I|}, r_{ui} = y_{ui}$
Macro Neighbor Aggregation of User u: $MNA_u = \sum_{p \in \tilde{N}_u^{(k)}} \tilde{Z}_{u,p}$
$Q_u, K_u, V_u \in \mathbb{R}^{d \times d'}$
Micro Node Embedding: $E_u \in \mathbb{R}^d$ $E_i \in \mathbb{R}^d$
Aggregated macro embedding:
$MNA_i = \sum_{p \in \tilde{N}_i^{(k)}} \tilde{Z}_{i,q}$
Macro Neighbor Aggregation of Node i:
$Q_i, K_i, V_i \in \mathbb{R}^{d \times d'}$
U I
$y_{ui} \in \{0, 1\}$: exhibit negative/positive behavior
u_1 u_2 u_n i_1 i_2 i_m
Micro User Node Size: n
Micro Item Node Size: m

Figure 3.1: MacGNN Implementation Detail

4. Methods and Experiments

4.1. Traditional Machine Learning Algorithms

For the traditional machine learning algorithms to be applied to this CTR research problem, we utilized 3 models: matrix factorization, user-user collaborative filtering, and item-item collaborative filtering.

Before using matrix factorization to predict the ratings in the test dataset, we constructed the user-item matrix R with dimension ($u*m$), where u is the number of users and m is the number of movies. The entry of this matrix is filled in by ratings scaled to [0,1] as the probability of click-through. The matrix factorization method introduced in COMP4434 Lab is used to learn the latent factors directly. Specifically, we use 3 latent factors for both items matrix Q with dimension ($m*3$) and users matrix P with dimension ($u*3$) so that the user-item matrix R can be recovered as $R = P * Q^T$.

For the cross-validation, we randomly masked 20% of the rated values as 0 in the training matrix and moved these masked values to the same entry in the validation matrix, with all other values in the validation matrix as 0. After each epoch, we can calculate the root of the mean square error (RMSE) on both the training set and the validation set.

After running 50 epochs, the RMSE metrics converged to 0.2134 and 0.2360 for training data and validation data, respectively. In the evaluation with test data, the AUC score is 0.5305842439432419.

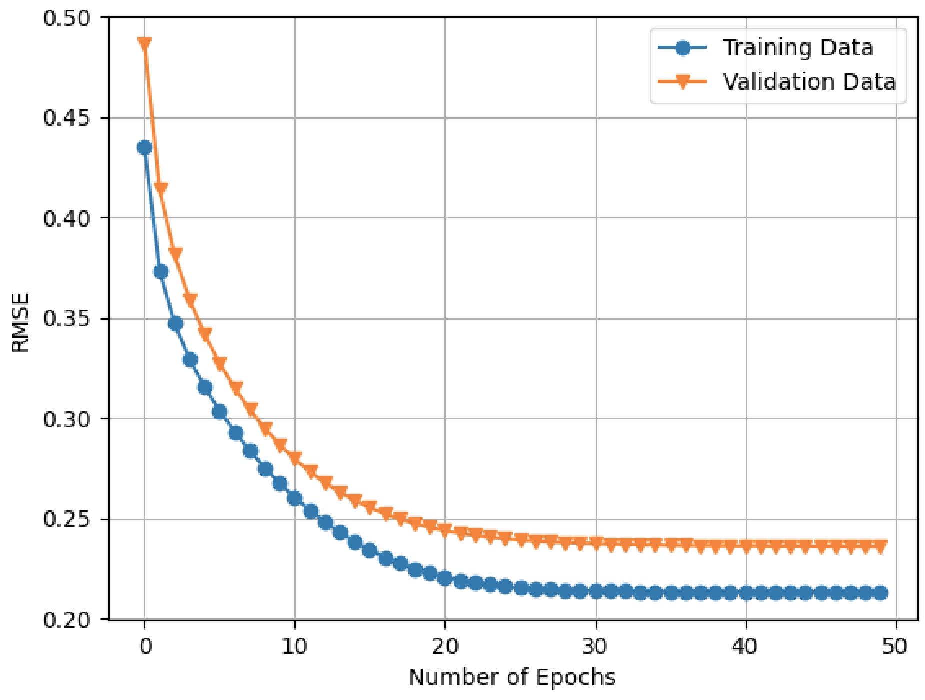


Figure 4.1.1. Matrix Factorization Implementation Detail

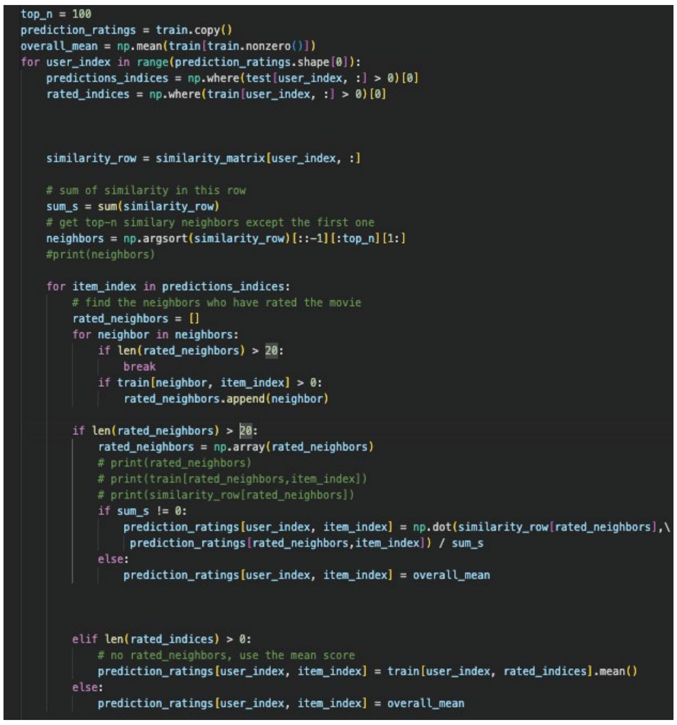
Besides the matrix factorization, we also explored the user-user and item-item collaborative filtering based on the similarity matrix. These two methods are interesting to explore in the CTR problem because they don't require the training of a large-size model, which is attractive given the large amount of data in CTR task. This is based on the assumption that similar users (or items) will have similar user-item interaction results (like ratings) so that we can use the aggregation of the most similar users (or items) who ***have rated*** the item to estimate the missing ratings. However, since the sparsity of user-item interaction is quite high (in the MovieLens-10M dataset, 98.67%), it is not easy to find sufficient similar users who have rated a specific item. Also, the similarity calculation is quite challenging since 98.67% of entries are missing values in the user-item interaction matrix.

Since these two methods are quite similar, except that the similarity matrices are constructed on different dimensions, we will focus on user-user as an *example* to elaborate on how we conducted the experiment and try to overcome these challenges. In this case, we compared and decided to use Pearson Correlation as the similarity metric to construct the similarity matrix (dimension *u * u, u* is the number of users) to lower the impact of missing values. After constructing this similarity matrix, we can create the neighborhood set *N* of top-*100* similar users. Then, we can predict the unrated value for each user. To achieve this, we will use the formula as the following formula:

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Here, *x* is the target user we want to predict, *i* is the item (movie) whose rating is missing on user *x, N* is the similar users in the neighborhood who *have rated i*, *Sxy* is the similarity value between *x* and *y.*

To deal with a large amount of missing values, we will use the average ratings of a user to estimate the *rxi* if there is less than 20 users in *N who have rated (i.e., the number of y is less than 20).* In the worst case, if a user never rated any movie, which is the cold start problem, we will use the average rating of *i* by all other users who have rated *i* to estimate *rxi.*



```python
top_n = 100
prediction_ratings = train.copy()
overall_mean = np.mean(train[train.nonzero()])
for user_index in range(prediction_ratings.shape[0]):
    predictions_indices = np.where(test[user_index, :] > 0)[0]
    rated_indices = np.where(train[user_index, :] > 0)[0]



    similarity_row = similarity_matrix[user_index, :]

    # sum of similarity in this row
    sum_s = sum(similarity_row)
    # get top-n similary neighbors except the first one
    neighbors = np.argsort(similarity_row)[::-1][:top_n][1:]
    #print(neighbors)

    for item_index in predictions_indices:
        # find the neighbors who have rated the movie
        rated_neighbors = []
        for neighbor in neighbors:
            if len(rated_neighbors) > 20:
                break
            if train[neighbor, item_index] > 0:
                rated_neighbors.append(neighbor)

        if len(rated_neighbors) > 20:
            rated_neighbors = np.array(rated_neighbors)
            # print(rated_neighbors)
            # print(train[rated_neighbors,item_index])
            # print(similarity_row[rated_neighbors])
            if sum_s != 0:
                prediction_ratings[user_index, item_index] = np.dot(similarity_row[rated_neighbors],\
                 prediction_ratings[rated_neighbors,item_index]) / sum_s
            else:
                prediction_ratings[user_index, item_index] = overall_mean



        elif len(rated_indices) > 0:
            # no rated_neighbors, use the mean score
            prediction_ratings[user_index, item_index] = train[user_index, rated_indices].mean()
        else:
            prediction_ratings[user_index, item_index] = overall_mean
```

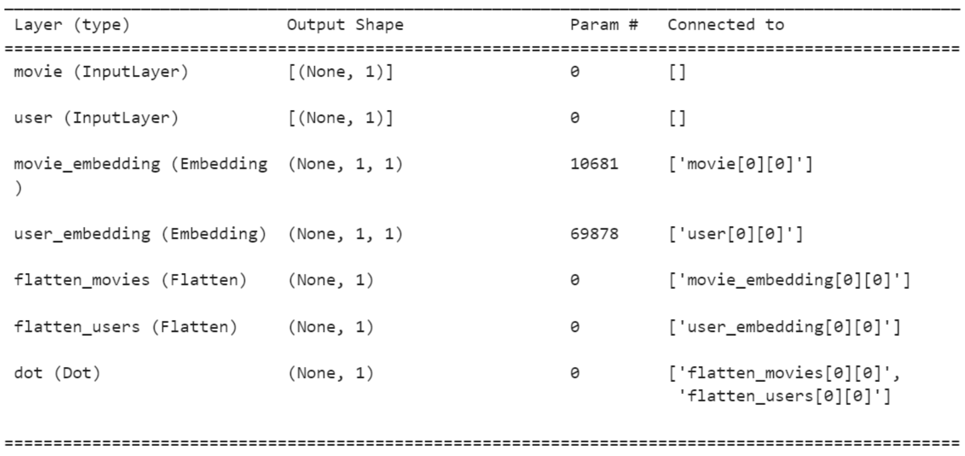
Figure 4.1.2. User-User Collaborative Filtering Implementation Detail

With the evaluation on test set, the AUC score of user-user collaborative filtering is 0.5053460733080465, while the AUC of item-item collaborative flitering is 0.500010264621954.

4.2. Basic Neural Network Algorithms

By observation, Neural nets are fundamentally matrix operations. Moreover, Matrix factorization techniques for recommendation systems also do something similar. For example, in SVD, we find matrices using weights calculated by SGD, which is similar to Deep Learning. This inspires us to use Deep Learning for Collaborative Filtering in movie recommendation systems.

In collaborative filtering, the similarity is calculated based on a user's rating of an item. Two users can be considered similar if they give the same ratings to ten movies. In this way, our first basic model, ***Simple Model***, is a collaborative filtering model that utilizes embeddings for users and movies. It consists of two input layers for user and movie IDs, followed by embedding layers for each input. The embeddings are then flattened and concatenated before passing through dense layers with dropout for regularization. The model learns the latent representations of users and movies in a collaborative filtering setting to predict movie ratings accurately. The model structure is depicted in the Fig. [4.2.1].



```
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 movie (InputLayer)             [(None, 1)]          0           []

 user (InputLayer)              [(None, 1)]          0           []

 movie_embedding (Embedding     (None, 1, 1)         10681       ['movie[0][0]']
 )

 user_embedding (Embedding)     (None, 1, 1)         69878       ['user[0][0]']

 flatten_movies (Flatten)       (None, 1)            0           ['movie_embedding[0][0]']

 flatten_users (Flatten)        (None, 1)            0           ['user_embedding[0][0]']

 dot (Dot)                      (None, 1)            0           ['flatten_movies[0][0]',
                                                                  'flatten_users[0][0]']

==================================================================================================
```

Figure 4.2.1: The Simple Model structure.

The AUC for this simple model is 0.5168418398041632 on the test set. The result is not satisfying. It may lack the complexity needed to capture the intricate relationships between users and movies. The Simple Model's architecture, with a basic structure of embedding layers and a few dense layers, may not be sufficient to learn the nuanced patterns in user preferences and movie characteristics.

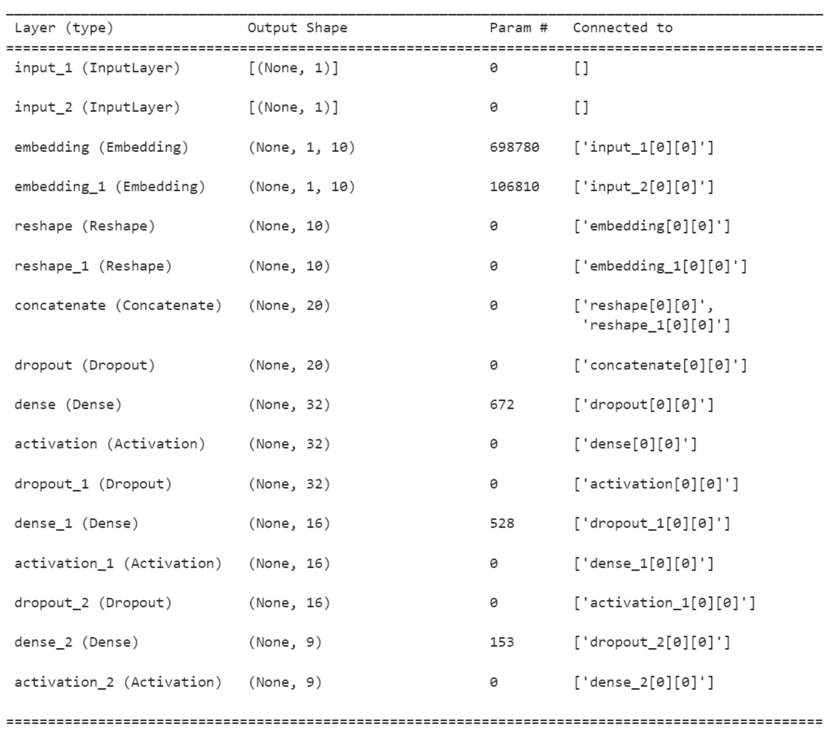
To capture more complex relationships between users and movies. To enhance the representation learning capabilities of the model by introducing non-linear transformations and deeper layers for improved prediction accuracy. We improved the previous simple model by introducing additional complexity, known as ***Deep Model***. Also, to reserve more features of the original data, we convert the output from the CTR (1/0) to 9 possible rating classes.

The Deep Model takes user and movie IDs as inputs and aims to predict user-movie ratings. Users and Movies are one-hot encoded and fed into the Deep Model as different distinct inputs and ratings are given as output. Due to the flexibility of the network's input layer, such a DNN structure can easily incorporate query features and item features, which can help capture the specific interests of a user and improve the relevance of recommendations. The Deep Model is built by extracting the latent features of Users and movies with the help of embedding layers. Then, dense layers with dropouts were stacked at the end, and finally, a dense layer with nine neurons (one for each possible rating from 1 to 5) with a softmax activation function was added The model is also listed in the Fig. [4.2.2].

Hyperparameters of the Deep Model were carefully tuned, and many loss functions and optimizers were tried with minimum validation loss as a metric to build the model and get the weights. During the training and the evaluation, we have the following consideration:

- Each model was trained using a subset of the dataset and evaluated based on the validation loss. The training process involved multiple epochs with batch processing and monitoring of validation loss for model checkpointing.
- Hyperparameter tunings, such as adjusting learning rates, batch sizes, and optimizer selection, were crucial for optimizing model performance.
- The model is tested on the test set, with AUC as the measurement. Since the train test split follows the data processing in the paper, the AUC can be compared with the method proposed in the selected paper.

Finally, ***Nadam*** for the optimizer and ***Sparse Categorical Cross entropy*** for the loss function was picked. The validation error and model loss during the training process is shown as in the Fig [4.2.3]. And the AUC for the Deep Model is 0.6034482431991461 on the test set.



```
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 1)]                  0         []

 input_2 (InputLayer)        [(None, 1)]                  0         []

 embedding (Embedding)       (None, 1, 10)                698780    ['input_1[0][0]']

 embedding_1 (Embedding)     (None, 1, 10)                106810    ['input_2[0][0]']

 reshape (Reshape)           (None, 10)                   0         ['embedding[0][0]']

 reshape_1 (Reshape)         (None, 10)                   0         ['embedding_1[0][0]']

 concatenate (Concatenate)   (None, 20)                   0         ['reshape[0][0]',
                                                                     'reshape_1[0][0]']

 dropout (Dropout)           (None, 20)                   0         ['concatenate[0][0]']

 dense (Dense)               (None, 32)                   672       ['dropout[0][0]']

 activation (Activation)     (None, 32)                   0         ['dense[0][0]']

 dropout_1 (Dropout)         (None, 32)                   0         ['activation[0][0]']

 dense_1 (Dense)             (None, 16)                   528       ['dropout_1[0][0]']

 activation_1 (Activation)   (None, 16)                   0         ['dense_1[0][0]']

 dropout_2 (Dropout)         (None, 16)                   0         ['activation_1[0][0]']

 dense_2 (Dense)             (None, 9)                    153       ['dropout_2[0][0]']

 activation_2 (Activation)   (None, 9)                    0         ['dense_2[0][0]']

==================================================================================================
```

Figure 4.2.2: The Deep Model structure.

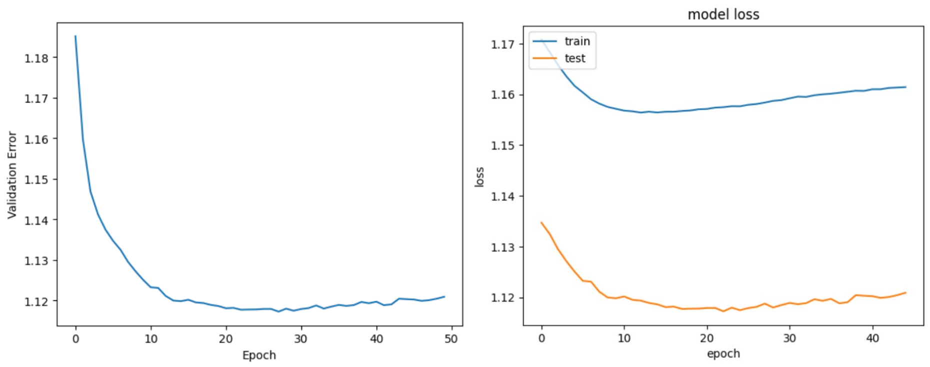
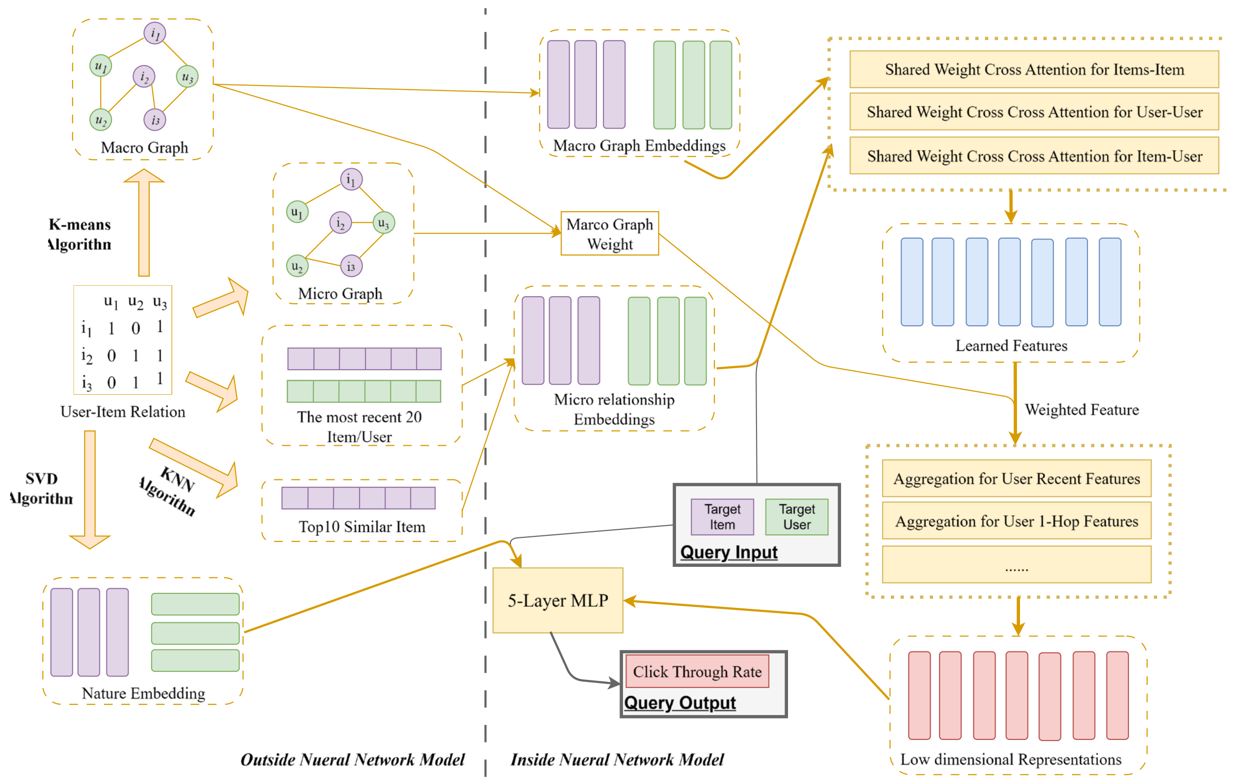


Figure 4.2.3: The Deep Model training, validation, and test error.

4.3. Advanced Model

In this section, we introduce an innovative ***Graph Neural Network*** (*GNN*) recommendation method that synthesizes elements of both ***collaborative filtering*** and ***content-based*** approaches alongside traditional methodologies. This hybrid model incorporates the Macro Graph technique [9] to extract collaborative filtering data and introduces toolkits for assimilating content-based information, enhancing the system's effectiveness in addressing complex recommendation scenarios. An illustrative overview of our method is presented in Fig. [4.3.1].



Macro Graph
K-means Algorithn
User-Item Relation
u1 u2 u3
i1 1 0 1
i2 0 1 1
i3 0 1 1
SVD Algorithn
KNN Algorithn
Nature Embedding
Micro Graph
The most recent 20 Item/User
Top10 Similar Item
Outside Nueral Network Model
Macro Graph Embeddings
Marco Graph Weight
Micro relationship Embeddings
Target Item
Target User
Query Input
5-Layer MLP
Click Through Rate
Query Output
Inside Nueral Network Model
Shared Weight Cross Attention for Items-Item
Shared Weight Cross Cross Attention for User-User
Shared Weight Cross Cross Attention for Item-User
Learned Features
Weighted Feature
Aggregation for User Recent Features
Aggregation for User 1-Hop Features
......
Low dimensional Representations

Figure 4.3.1: The overview of the proposed advanced model.

The process begins with extracting ***Micro Graph*** and ***Macro Graph*** data, representing ***User-Item*** and ***Item-Item*** relationships, utilizing ***K-means*** and ***KNN*** algorithms, respectively. This data is then processed through multiple ***cross-attention layers*** designed to uncover latent relationships that reflect ***user preferences*** and ***item characteristics***. These relationships are further refined by a ***self-attention*** layer and aggregated using either ***a channel-wise weighted summarization*** or a s***equence decoder*** approach. The final recommendation—whether a user is likely to engage with an item—is determined by a ***Multilayer Perceptron*** *(MLP)*, which evaluates the consolidated latent relationships.

Our methodology employs a graph-based representation ***Micro Graph*** to describe the relationships $r_{\{u,i\}}$ between users $u \in U$ and items $i \in I$ in the ***collaborative filtering***. Given the high dimensionality and sparsity of the relationship matrix, directly applying a GNN to the Micro Graph presents substantial challenges [9]. To address these, we utilize the ***Macro Graph*** approach, which effectively preserves most of the relational information while minimizing the introduction of biases commonly seen in other sampling-based methods. It is constructed using the ***K-means*** algorithm to standardize features and categorize them into graph nodes for all items and users. In this configuration, items and users are organized into 20 and 21 groups, respectively. For instance, similar items, such as everyday tools, are aggregated into a single Macro node, as illustrated in Fig. [4.3.2]. Within the Macro Graph, edge weights are evaluated in each query and calculated based on the cumulative relationships of neighboring nodes. For example, the two-hop Macro weight for a user is defined as Eq. [1]

$$r_{marco;u}^{(2)} = \sum_{a \in \tilde{C}^{(1)}(u),\ b \in \tilde{C}^{(2)}(u)} r_{\{a,b\}} \tag{1}$$

where $\tilde{C}^{(k)}(u) = N_{micro}^{(k)}(u) \cap N_{macro}^{(k)}(u)$ represents user u's k-hop neighboring in the joint set of both Micro and Macro Graph. Further, we adopt the ***logarithmic smoothing*** and ***temperature-based softmax activation*** techniques proposed in [9] to maintain numerical stability. By extracting both one-hop and two-hop relationships for users and items within the Macro Graph, we ensure that our model learns from a comprehensive collaborative filtering representation in a low dimensionality and without bias.

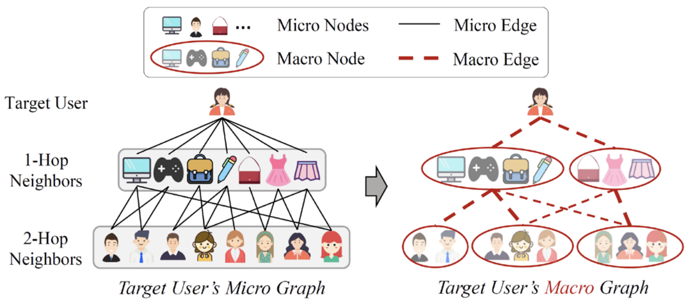


Figure 4.3.2: An example of the Macro Graph. [1]

In our model, ***content-based*** information serves as contextual data drawn from the Micro Graph to enhance the representation of queried items. Specifically, we consider ***the most recent 20 items or users*** and ***top 10 similar items*** previously interacted with by the user as the content-based context information. To ensure that the calculated similarity provides adequate guidance, we evaluate items using both genre and rating metrics. Genre similarity is quantified using the ***Pearson similarity coefficient*** based on text-token information extracted from the dataset, denoted as $sim_p(i_a, i_b) \in [-1, 1]$. Additionally, the ***absolute rating differences*** are normalized through min-max scaling to compute the distance, $dis_r(i_a, i_b) \in [0, 1]$. Rating similarity is calculated as $sim_r(i_a, i_b) = 1 - dis_r(i_a, i_b)$. These metrics are combined in the following way, Eq. [2],

$$Sim(i_a, i_b) = sim_p(i_a, i_b) + sim_r(i_a, i_b) \tag{2}$$

to establish a thorough similarity score used in the ***K-nearest neighbors*** (*KNN*) algorithm to find the top 10 similar items.

To effectively model the complex interactions between users and items, we introduce several types of learnable embeddings. These embeddings capture essential and nuanced features of user and item identities. The fundamental attributes of users and items in the Micro graph are encapsulated by ***the user nature embedding*** E_u^n and ***the item nature embedding*** E_i^n. These embeddings initially came from the latent concepts identified by the ***Singular Value Decomposition*** (*SVD*) algorithm, which decomposes the Micro Graph matrix R into three components: $R = U\sum V$, where U represents the latent concepts for items and V for users. Additionally, the ***Macro Graph embedding*** is depicted through embeddings $\tilde{E}_u$ for user Macro nodes and $\tilde{E}_i$ for item Macro nodes. At the same time, we define ***Micro relationship embedding*** for ***user preferences*** E_u^r and ***item characteristics*** E_i^r, which reflect the connectivity and affinity across the Micro Graph and the Macro graph. These embeddings collectively form a comprehensive representation that integrates both Micro and Macro graph information, providing a robust framework for our recommendation system.

To synthesize the various embeddings, we employ ***cross-attention layers*** to articulate ***the interaction features*** between a queried user and associated items. In our model, the cross-attention mechanism utilizes two primary inputs: the key K and the query Q. Here, the query serves as a guide to refine the key's embedding. The cross-attention operation can be described mathematically as, Eq. [3]:

$$CrossAttention(Q, K) = softmax\left(\frac{Feature_Q(Q) \cdot Feature_K(K)^T}{\sqrt{feature\ length}}\right) \cdot Feature_V(K) \tag{3}$$

where $Feature$ represents a linear transformation layer applied without bias terms. Notably, our architecture includes three distinct cross-attention components tailored to user-user, item-item, and item-user relationships. For instance, to model the user-item relationship, the user preference embedding E_u^r function as the query inputs, while the Marco node embeddings $\tilde{E}$ or the Micro item characteristics E_i^r, act as the key. At the last, the feature will be scaled by their Marco edge weight. This configuration enables the generation of unique features that accurately reflect user preferences and item-specific features, thereby enhancing the predictive accuracy of our recommendation system.

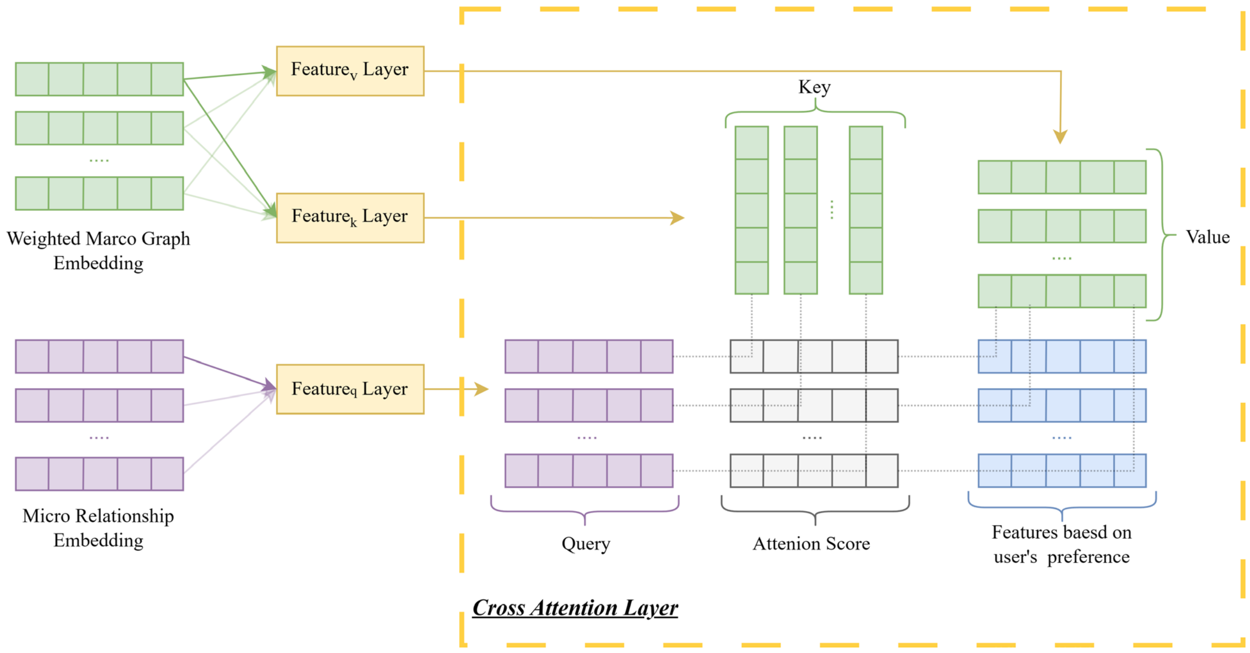


Figure 4.3.3: The interaction features extraction based on cross attention layer.

The features derived from the cross-attention layers are further refined through three individual ***shared-weight multi-head self-attention*** mechanism. In this mechanism, each feature is divided into four segments. A conventional self-attention layer processes each segment independently. The outputs are then concatenated to reconstruct the full embedding. This multi-head self-attention framework scales the relational data, allowing for nuanced item comparisons and the signal gain enhancement.

Following the self-attention processing, the feature undergo aggregation to synthesize the learned features as a low dimensional representation. This is achieved through a 1D convolution layer, which performs ***a channel-wise weighted summarization*** of regular features, illustrated in Fig. [4.3.4.a]. Alternatively, for user-item or item-user recent interactions, a ***Gated Recurrent Unit*** (*GRU*) layer to perform as ***a sequence decoder*** to aggregate the feature information, as shown in Fig [4.3.4.b]. This dual representation aggregation approach ensures that both static and dynamic aspects of user preferences and item characteristics are effectively captured in the final recommendation model.

In the final stage of our recommendation system, all processed features are input into a 5-layer ***Multilayer Perceptron*** (*MLP*) to produce the decision. The MLP input includes

- User nature embedding
- Representations of user 1-hop and 2-hop neighbors
- Representation of user's recent item interactions
- Item nature embedding
- Representations of item 1-hop and 2-hop neighbors
- Representation of item's recent user interactions

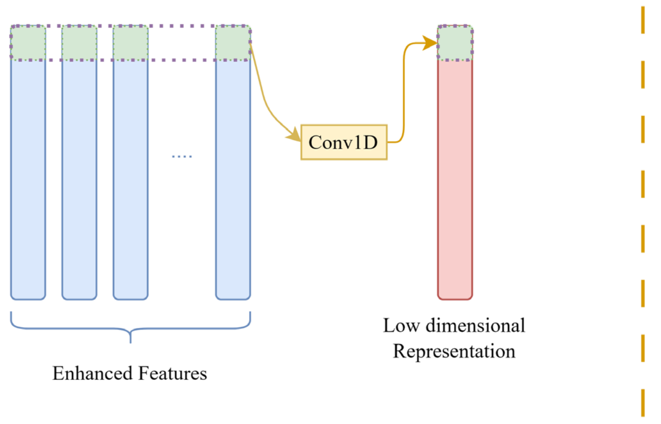


Figure 4.3.4.a: Channel-wise weighted summarization

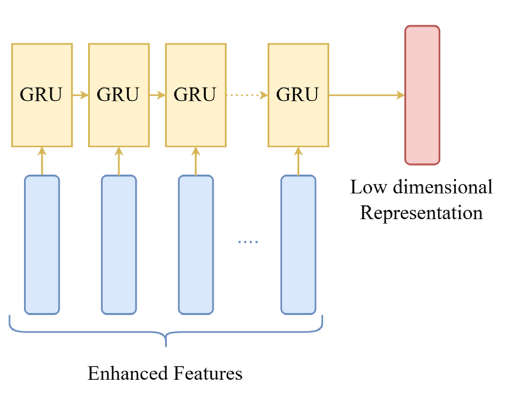


Figure 4.3.4.b: Sequence decoder

- Top 10 item representations based on item characteristics
- Top 10 item representations based on user preferences

These inputs are designed to encapsulate a holistic view of both user and item characteristics and neighboring relationships, ensuring that the MLP can make informed predictions. The output of the MLP is processed through a Softmax layer, which normalizes the results to produce a binary output, indicating the likelihood of a user engaging with an item. The model employs L2 regularized binary cross-entropy loss for training, defined mathematically as:

$$\mathcal{L} = \sum y\log(\hat{y}) + (1 - y)\log(1 - \hat{y}) + \lambda \parallel \theta \parallel^2 \quad (4)$$

To optimize our model further, we implemented a 5 random seed to do validation for refining several hyperparameters, and the results are presented accordingly. The best AUC of this approach reaches 0.737566.

- *Embedding Dimension*: We varied the embedding dimension across a range of values from 8 to 40, with 38 emerging as the optimal size (see Fig. [4.4]). This dimension crucially influences the model's capacity and the representational space of latent patterns.

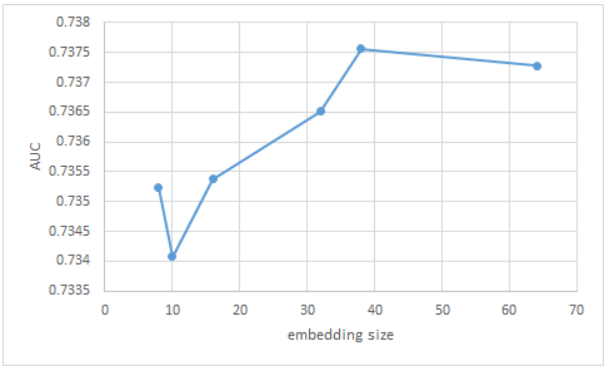


Figure 4.4: The relation of AUC and the embedding size

- *Optimization Parameters*: We experimented with different combinations of batch size, learning rate, L2 regularization coefficient, and optimizers (see Fig. [4.5]). The best results were achieved using the *NAadm* optimizer with a learning rate of 1e-4, a weight decay of 5e-5, and a batch size of 512.

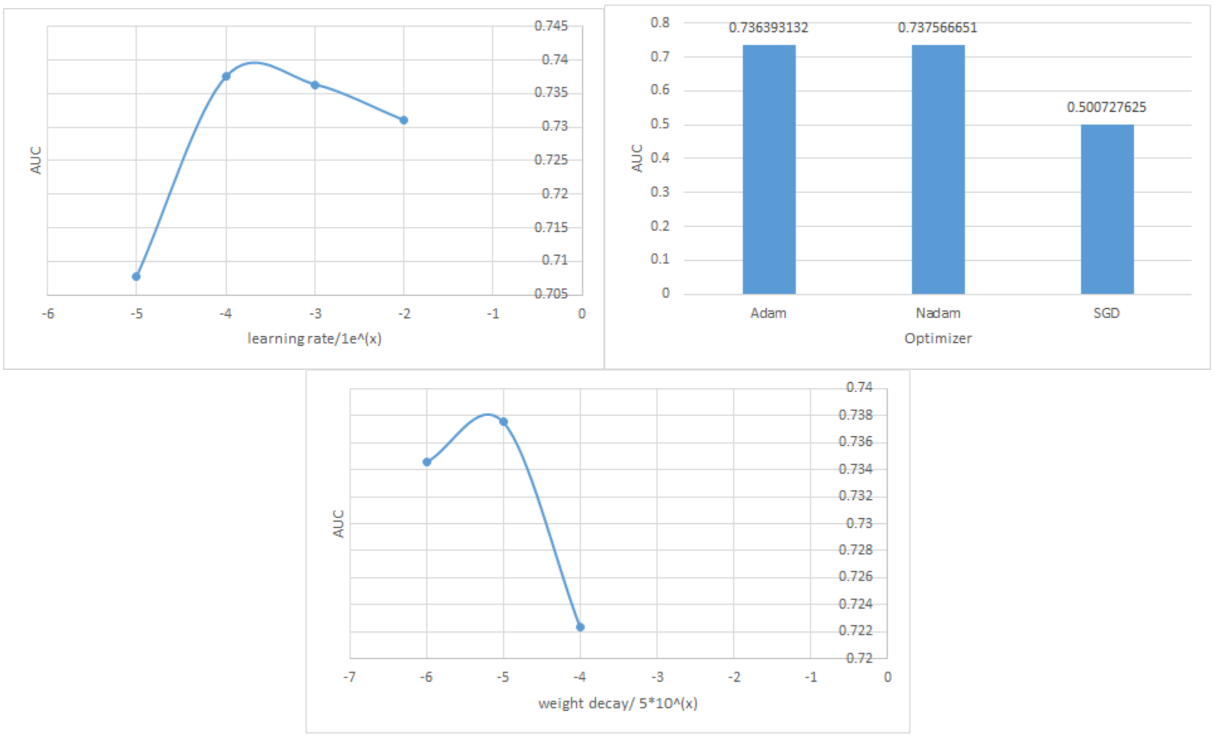


Figure 4.5: The relation of AUC and the optimization parameter

In the development of our final model, we also attempted to explore several alternative methods for embedding extraction and feature synthesis to assess their effectiveness. But none of them outperform the proposed methods.

- *Micro Nature Embeddings*: We initially experimented with using Micro Nature Embeddings to learn and represent neighbors, recent interactions, and similar items. However, these embeddings exhibited substantial numerical intra-differences that adversely affected the learning process, reducing performance by approximately 2%.
- *Linear Layer Aggregation and MLP depth*: A linear layer was also tested to synthesize learned features. Although this model incorporated more parameters than our proposed approach, it tended to overfit the data, leading to a decrease in the Area Under the Curve (AUC) by about 1%. In addition, we explore the impact of the depth of the very last MLP model. The result shows that the five layer MLP could reach the best performance, while the other configurations may lead to underfitting or overfitting.

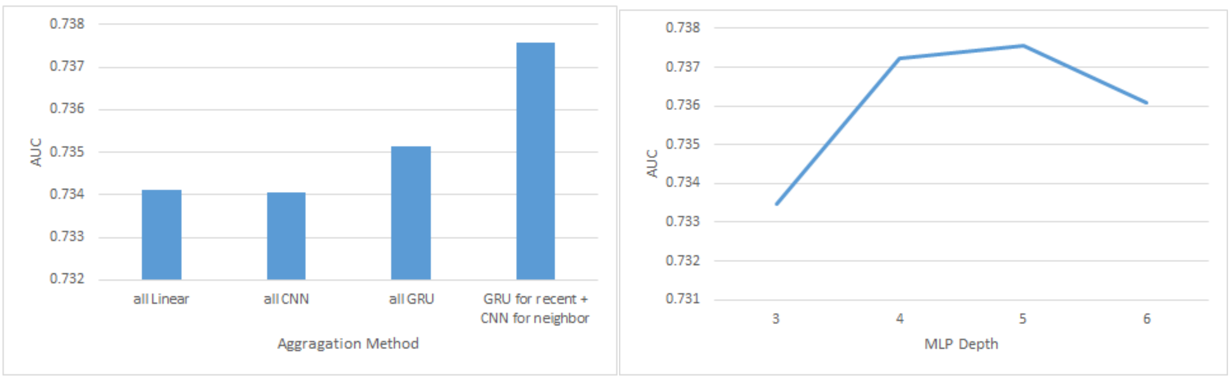


Figure 4.5: The relation of AUC and the aggregation layer type and MLP depth.

- *Separate Attention Layers*: Replacing our shared-weight attention mechanism with separate attention layers for each embedding type also increased the model's complexity. This change resulted in overfitting, like the linear model, and decreased overall accuracy.

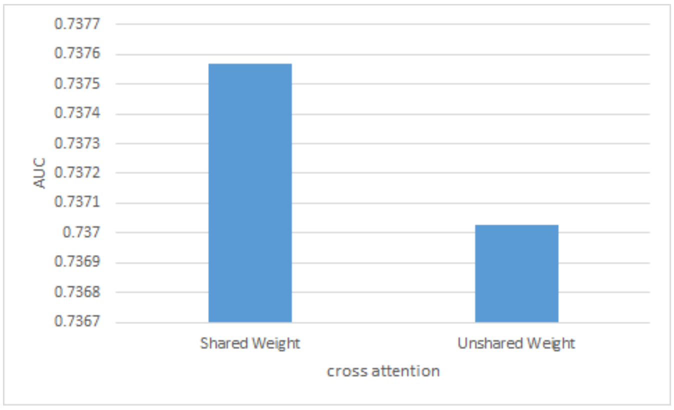


Figure 4.5: The relation of AUC and the attention weight type

5. Evaluation Results

We conducted the experiments on our models (3 traditional models, 2 basic neural network models, and 1 proposed advanced model) and compared the AUC score with the paper-proposed model MacGNN under the same test set. Based on the result, we can see that our advanced model has achieved a promising performance which is close to the result of MacGNN.

Models	AUC (4 digits precision)
Matrix Factorization	0.5306
User-user Collaborative Filtering	0.5053
Item-item Collaborative Filtering	0.5000
Simple Model	0.5168
Deep Model	0.6034
Advanced Model	**0.7376**
MacGNN	**0.7458**

6. Conclusion

In this project, we reviewed the literature on the CTR prediction problem and reproduced the state-of-the-art paper to gain insights from frontiers. Given the CTR problem, we explored and analyzed the suitable dataset MovieLens to conduct the experiments. When implementing and conducting experiments on our six models, we broadly explored and gradually improved the performance. Also, we integrated the feature of movie genre information in the user-item relationship learning to propose the hybrid method to mitigate the cold start and sparsity problems. This hybrid method is utilized in our advanced model, which achieved promising evaluation result.

7. References

[1] Y. Yang and P. Zhai, "Click-through rate prediction in online advertising: A literature review," *Information processing & management*, vol. 59, no. 2, pp. 102853-, 2022.

[2] Y. Yang, Y. C. Yang, B. J. Jansen, and M. Lalmas, "Computational Advertising: A Paradigm Shift for Advertising and Marketing?," *IEEE intelligent systems.*, vol. 32, no. 3, pp. 3–6, 2017.

[3] W. Ouyang *et al.*, "Learning Graph Meta Embeddings for Cold-Start Ads in Click-Through Rate Prediction," in *SIGIR '21 - PROCEEDINGS OF THE 44TH INTERNATIONAL ACM SIGIR CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL*, 2021, pp. 1157–1166.

[4] C. Yan, Y. Chen, Y. Wan, and P. Wang, "Modeling low- and high-order feature interactions with FM and self-attention network," *Applied intelligence (Dordrecht, Netherlands)*, vol. 51, no. 6, pp. 3189–3201, 2021.

[5] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International conference on data mining*. IEEE, 995–1000.

[6] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1059–1068.

[7] F. Harper and J. Konstan, "The MovieLens Datasets: History and Context," *ACM transactions on interactive intelligent systems*, vol. 5, no. 4, pp. 1–19, 2016.

[8] Joewellman, "Joewellman/Movielens," GitHub, https://github.com/joewellman/MovieLens [Accessed Apr. 30, 2024].

[9] H. Chen *et al.*, "Macro Graph Neural Networks for Online Billion-Scale Recommender Systems," *arXiv.org*, 2024, doi: 10.48550/arxiv.2401.14939.