; revised 14 March, 2025; date of current version 14 March, 2025.

Exploring Methods of Optimizing User Item Recommendation Systems

MIE 424 Survey-Oriented Project

Jiacheng (Jason) Chen [1], Max Zhang[1], Ken Kato[1], Dav Vrat Chadha[1], Punyaphat Sukcharoenchaikul[1]

[1]Department of Engineering Science, University of Toronto, Toronto, ON Canada

I. INTRODUCTION

USER-item recommendation systems are crucial in modern digital platforms, driving personalized experiences in domains such as e-commerce, streaming services, and social media. These systems leverage machine learning to predict user preferences based on historical interactions, ensuring that users receive relevant content, products, or services. However, optimizing recommendation systems remains a significant challenge due to the need for high accuracy, computational efficiency, and adaptability to evolving user behavior.

In this project, we aim to study different optimization techniques for user-item recommendation systems, focusing on methods that enhance recommendation accuracy and efficiency. Specifically, we investigate the effectiveness of Collaborative Filtering, Content-enriched recommendation, temporal models, and Deep Reinforcement Learning (DRL) in improving recommendation performance. Mathematically, all these methods are trying to optimize the following problem:

$$\hat{R}_{ui} = f(u, i \mid \Theta) \tag{1}$$

where $\hat{R}_{ui}$ is the predicted rating for user u on item i, f is the optimization function, and Θ represents the model parameters. The goal is to minimize the prediction error between the predicted ratings and the actual ratings, typically using a loss function such as Mean Squared Error (MSE) or Mean Absolute Error (MAE).

TODO: fix the explaining sentences by reflecting the new section separations more

Different learning-based optimization techniques approach this problem in distinct ways. Autoencoders treat recommendation as a latent representation learning problem, compressing user-item interactions into a lower-dimensional space to extract meaningful embeddings that reveal hidden relationships. Convolutional Neural Networks (CNNs) take a spatial feature extraction approach, applying convolutional filters to structured representations of user-item interactions to capture local patterns and shared item attributes. Neural Attention Mechanisms cast recommendation as a context-aware learning problem, dynamically weighting past interactions to focus on the most relevant items and personalize recommendations. Recurrent Neural Networks (RNNs) frame recommendation as a sequential prediction problem, leveraging temporal dependencies in user behavior to anticipate future preferences. Meanwhile, Deep Reinforcement Learning (DRL) treats recommendation as a decision-making problem under uncertainty, optimizing long-term user engagement by learning policies that maximize cumulative satisfaction.

By systematically comparing these approaches, we aim to provide insights into their respective strengths, weaknesses, and practical trade-offs.

II. Related Works and methodologies

A. Collaborative Filtering

Collaborative Filtering (CF) builds recommendation systems by leveraging a user-item interaction matrix $R \in \mathbb{R}^{u \times i}$, where u and i denote users and items. Traditional CF methods like matrix factorization and neighborhood models struggle to capture complex, nonlinear user-item relationships. They often suffer from high computational cost, parameter inefficiency, and limited generalization due to linear assumptions. RBM-CF, for instance, requires slow contrastive divergence training and separately models discrete ratings, increasing memory use. To address these issues, the AutoRec paper proposes a deep learning-based CF model that improves expressiveness and efficiency [1].

1) AutoRec [1]

AutoRec is an autoencoder-based framework for collaborative filtering. At its core, it projects the partially observed user-rating vector $r^{(u)}$ or item-rating vector $r^{(i)}$ to a lower-dimensional latent space and aims to reconstruct the original vector in the output space. Mathematically, the method minimizes the MSE loss between the original rating vector and the re-constructed output vector as shown in equation 2:

$$\mathcal{L}(\Theta) = min \sum_{r \in S} ||r - h(r; \theta)||_2^2 \tag{2}$$

where $h(r;\theta)$ is the output of the autoencoder after projecting the input r to the latent space and reconstructing it back to the original space.

$$h(r;\theta) = f(W_2 \cdot g(W_1 \cdot r + b_1) + b_2) \tag{3}$$

where $\theta = \{W_1, W_2, b_1, b_2\}$ are the parameters of the autoencoder and f and g are activation functions. In t The parameters learned during this training process can then be used to predict un-observed ratings in the user-item interaction matrix. Because the we can obtain two kinds of vectors from the rating matrix(user-based and item-based) the paper proposes two variants of AutoRec: Item-based AutoRec (I-AutoRec), which models item embeddings and reconstructs user ratings, and User-based AutoRec (U-AutoRec), which models user embeddings and reconstructs item ratings. Compared to previous models, AutoRec introduces several key improvements. First, it offers more efficient training by using gradient-based backpropagation instead of the computationally expensive contrastive divergence required by RBM-CF. Second, it reduces parameter count as it does not require separate parameter estimation for different rating levels, unlike RBM-CF. Finally, AutoRec enables nonlinear representation learning through activation functions such as sigmoid, allowing it to capture more complex user-item interactions compared to traditional matrix factorization, which assumes a linear latent representation.

B. Convolutional Sequence Embedding Recommendation Model(Caser) [2]

Traditional recommender systems mainly model a user's general preferences based on past interactions and fail to account for sequential dependencies, where more recent interactions are stronger indicators of future behavior. Top-N sequential recommendation addresses this by predicting the next items a user will likely engage with based on their recent activity. The problem is framed as having a set of users $\mathcal{U} = \{u_1, u_2, \cdots, u_{|\mathcal{U}|}\}$ and items $\mathcal{I} = \{i_1, i_2, \cdots, i_{|\mathcal{I}|}\}$, with each user u having a sequence of interactions $S^u = (S^u_1, \cdots, S^u_{|S^u|})$, where $S^u_i \in \mathcal{I}$ over time. The goal is to recommend the top N items taking the user's recent interaction as well as their general preference into account. This is different from the conventional recommendation system which considers user behaviour as a set of items and not as a sequence of items. In the paper, the authors pointout that the previous Markov Chain-based models that aim to tackle the above problem are not sufficient in doing so for multiple reasons. They discuss that these model successfully lern the point-level influences but are not sufficient to model union-level influences where a group of interactions taken by the user could influence the users behaviour in the future. Also, Markov-Chain-based models cannot account for skip-behaviours by users where a sequence of actions that occurred not in the immediate past could have a strong influence on the user's actions.

To address these gaps, the authors propose Caser (Convolutional Sequence Embedding Recommendation Model), which takes advantage of Convolutional Neural Networks (CNNs) to model sequential patterns in recommendation. Caser represents a sequence of recent items as an embedding matrix of size $L \times d$ where L represents the number of past items taken into consideration and d is the number of latent dimensions for the items. By appling a horizontal and vertical convolutional filter to the "image" of user-item interaction, they aim to detect point-level and union-level interaction of the users as well as skip-behaviors. Additionally, it incorporates user embeddings to personalize recommendations and allows for skip behaviors, making it more flexible in comparison to traditional Markov models.

The Caser architecture(see figure 5 in appendix) takes the item sequence matrix and user embedding as input and outputs a probability distribution over the items, which represents the probability of the user iteracting with the item at the timestep we are predicting for. The output of the model hence can be expressed as:

$$p(S^u_t \mid S^u_{t-1}, S^u_{t-2}, \cdots, S^u_{t-L}) = \sigma(y^{(u,t)}_{S^u_t}),$$

where $y^{(u,t)}_{S^u_t}$ is the output of the model for the item S^u_t given the previous items $S^u_{t-1}, S^u_{t-2}, \cdots, S^u_{t-L}$ at time t and σ is the sigmoid function. The model is trained using a binary cross-entropy loss function, which minimizes the difference between the predicted probabilities and the actual interactions In the paper, the overall loss is written as:

$$\ell = \sum_u \sum_{t \in C^u} \sum_{i \in \mathcal{D}^u_t} -\log(\sigma(y^{(u,t)}_i)) + \sum_{j \neq i} -\log(1 - \sigma(y^{(u,t)}_j))$$

where C^u is the set of all items that user u has interacted with and $\mathcal{D}^u_t$ is the set of all items that user u has interacted with at time t, and beyond to account for skip behaviours. Hence $C^u = \{L+1, L+2, \cdots, |S^u|\}$ and $\mathcal{D}^u_t = \{S^u_t, S^u_{t+1} \cdots S^u_{t+T}\}$.

C. Attention Networks

D. Recurrent Neural Networks [3]

Recommender systems rely on long-term user histories to generate personalized recommendations, but in many real-world scenarios—such as e-commerce sites and media platforms—user tracking is unreliable or infeasible due to privacy concerns and short user engagement. In these cases, session-based recommendation is important, where recommendations are made based only on a user's recent interactions within a session. Existing methods, including item-to-item similarity models and Markov Chain-based approaches, are limited because they either consider only the last interaction or struggle to model long-term dependencies in user behavior. To address these shortcomings, the authors of propose a Recurrent Neural Network (RNN)-based model with Gated Recurrent Units (GRUs), which captures sequential dependencies by processing an entire session as a sequence. Unlike traditional models, GRUs dynamically update their hidden state with each user interaction, learning richer session representations. In the paper, they represent

the events as a one-hot encoding of the interacted item or a weighted sum of the items the user interacted with so far. When discussing the loss function used for training the RNN, the authors point out that training the model to independently estimate the ranking of events did not yield a stable learning and hence chose a pairwise ranking objective. The RNNs were trained using two types of pairwise ranking loss functions: the Bayesian Personalized Ranking (BPR) loss and the TOP1 loss. The BPR loss is defined as:

$$L_{BPR} = -\frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \log\left(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j})\right) \tag{4}$$

This loss function is defined for a given point at one session where N_S is the sample size, $\hat{r}_{s,k}$ is the score on item k at given point of the session, i is the desired item and j are the negative samples. The TOP1 loss is defined as:

$$L_{TOP1} = \frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \mathbb{I}\{\hat{r}_{s,j} > \hat{r}_{s,i}\} \tag{5}$$

where $\mathbb{I}$ is an indicator function which they approximate using a sigmoid function. In order to forc the scores of negative samples to be around zero, the authors also add a regularization term to the loss function. The final loss function is defined as:

$$L_s = \frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \sigma\left(\hat{r}_{s,j} - \hat{r}_{s,i}\right) + \sigma\left(\hat{r}_{s,j}^2\right) \tag{6}$$

E. Deep Reinforcement Learning Frameworks [4]

This paper introduces a Deep Reinforcement Learning framework for news recommendation, addressing the limitations of previous methods that have:

- Focuseed on modelling only the current reward, ignoring the long-term impact of recommendations on user.engagement.
- Did not consdider forms of user engagement beyond click / no click labels.
- Kept recommending same items to users, leading to a lack of diversity in recommendations and users to get bored.

By introducing a Deep Q-Learning (DQN) based framework, the authors have addresed above limitations and modelled the future reward explicitly. They have also considered more feedback information beyond click/no-click labels allowing for more nuanced user engagement information. Finally, they also incorporate an effective exploration strategy to improve the diversity of the recommendations.

In their framework, the model has two stages of training: offline and online. In the offline stage, the model is trained using the user-news clicks logs where the model takes news features, user features, user-news features and context features as input (see full description in table 1). Then the model is trained online, interacting with users and updating the network in a 4 step process (details discussed in table 2). At each timestep t the model predicts the total reward for taking an action a given the state s with below equation:

$$y_{s,a,t} = r_{a,t+1} + \gamma Q(s_{a,t+1}, \arg\max_{a'} Q(s_{a,t+1}, a'; W_t); W'_t) \tag{7}$$

where $r_{a,t+1}$ is the reward for taking action a at time t, γ is the discount factor, and W_t and W'_t are two different sets of weights for the DQN. With W_t the model selects the future action a' that maximizes the Q-value, and with W'_t the model estimates the future reward given thate $s_{a,t+1}$ and a' are the next state and action. Every few iterations, W_t and W'_t are switched to reduce overoptimistic estimation of the Q value. Now these Q-networks are split into two subnetworks: V and A where V is the value function and A is the advantage function. The value function takes in teh state features of User features and Context features while the advantage function takes in all of the features. This is due to the observation of the author's that the reward of taking an action a at a certain state s is closely related to all the features but there is a portion of the reward that is solely related to the state features.(e.g. whether this user is active, whether they have read enough news, etc.)

III. Datasets and Experiments

A. AutoRec [1]

The paper utilizes popular benchmark datasets from the collaborative filtering domain, namely Movielens (with both 1M and 10M ratings) and Netflix. The experiments involve splitting the data into 90% for training and 10 for testing, with a further 10% of the training set reserved for hyperparameter tuning, and repeating this process multiple times to obtain average RMSE values. They compare different model variants—item-based and user-based AutoRec—against several baseline methods such as RBM-based collaborative filtering, biased matrix factorization, and LLORMA. Moreover, they conduct experiments to examine the effects of different activation functions (linear versus non-linear) and varying numbers of hidden units, as well as exploring a deeper version of the model with multiple hidden layers.

B. Caser [2]

The paper experiments with several publicly available datasets that capture sequential user behavior. In particular, they use datasets from MovieLens (movie ratings), Gowalla and Foursquare (location-based check-ins), and Tmall (e-commerce purchase data). These datasets differ in terms of sequential signal intensity, which the authors quantify by mining sequential association rules. For their experiments, user action sequences are divided into training, validation, and test sets, and the models are evaluated using top-N recommendation metrics such as Precision@N, Recall@N, and Mean Average Precision (MAP). The proposed convolutional sequence embedding model (Caser) is compared against a variety of baseline methods—including popularity-based, matrix factorization with Bayesian personalized ranking,

and both Markov chain-based and recurrent neural network models—with further analyses performed on hyperparameter settings and the contribution of different model components.

C. RNN [3]

The paper evaluates its GRU-based model on two real-world datasets. One dataset, from the RecSys Challenge 2015, is composed of click-stream data from an e-commerce website collected over approximately six months. This dataset includes millions of sessions and tens of millions of click events on tens of thousands of items, with sessions of length one filtered out to focus on meaningful user behavior. The second dataset comes from a video streaming service, capturing watch events over a shorter period (just under two months) and involving a much larger catalog of videos, though similarly processed to remove anomalies such as excessively long sessions. The experiments compare the proposed model against popular baselines like item-to-item similarity methods, session-based popularity predictors, and matrix factorization techniques. Additionally, the study explores various architectural choices—such as session-parallel mini-batches, output sampling strategies, and different ranking loss functions—and examines the impact of hyperparameter tuning and network configurations on the model's performance.

D. DRL [4]

The paper evaluates its proposed deep reinforcement learning framework using a dataset collected from a commercial news recommendation application. The offline dataset spans six months and includes over half a million users and more than one million news articles, while the online dataset covers one month with tens of thousands of users and hundreds of thousands of news items. In the offline experiments, the data is split temporally—using the most recent two weeks for testing—and the click/no-click ratios are down-sampled for effective model training. The authors conduct experiments to assess recommendation accuracy using metrics such as CTR, nDCG, and Precision@k, and they also analyze the model's convergence behavior. Additionally, the framework is deployed in a live production environment to evaluate its online performance, where both recommendation accuracy and diversity (measured by an ILS metric) are examined.

The proposed method and its enhanced variant, Double DQN (DDQN) + all tricks, outperform all baseline methods, including Logistic Regression (LR), Factorization Machines (FM), Wide & Deep (W&D), and bandit-based approaches (LinUCB, HLinUCB). The DDQN model achieves the highest CTR (0.1662) and nDCG (0.4877), demonstrating the effectiveness of reinforcement learning in optimizing long-term engagement. The dueling network architecture improves user-news interaction modeling, while future reward consideration in DDQN leads to additional gains in recommendation accuracy. However, in the offline setting, incorporating user activeness and exploration does not show significant improvements due to dataset limitations, as interactions are static, preventing effective exploration. The authors note that naïve exploration strategies like ε-greedy can harm recommendation accuracy, suggesting that exploration should be designed more carefully. The authors propose evaluating the model in an online setting, where user engagement evolves dynamically, allowing better assessment of exploration strategies and activeness-aware recommendations. Additionally, integrating more advanced exploration methods and personalized reward functions could further enhance long-term engagement optimization.

IV. Results and Comparison

A. Formulation Comparison

Autorec - reconstructing user-item matrix using autoencoder while minimizing MSE

Caser - using CNN to model sequenctial dependencies and user-item interaction.

RNN - using GRU to model sequential dependencies and user-item interaction.

DRN - using DQN to maxiize the reward fucntion dependant on user engagement

B. Results comparison

TODO:move to later part

AutoRec has several limitations. It is a relatively simple model, and while a deeper version has been tested, the performance improvements are minimal, suggesting that deeper models may require further adjustments to be effective. Additionally, AutoRec does not address the cold-start problem, making it challenging to provide recommendations for new users or items that lack prior ratings. also time series data.

V. Conclusion and Future work

REFERENCES

[1] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th international conference on World Wide Web*, 2015, pp. 111–112.

[2] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 565–573.

[3] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[4] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "DRN: A Deep Reinforcement Learning Framework for News Recommendation," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. Lyon, France: ACM Press, 2018, pp. 167–176. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3178876.3185994

Appendix

A. Autorec

1) Visual representation of the AutoRec model

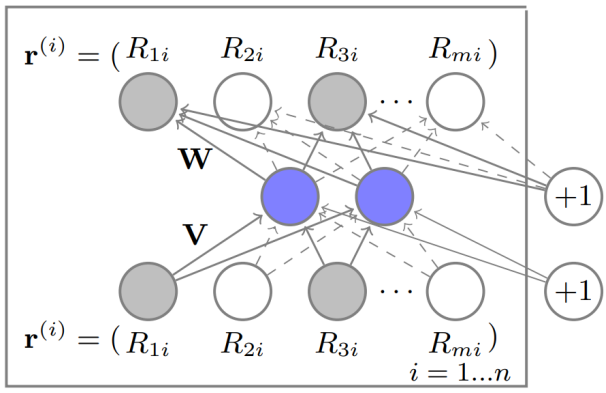

FIGURE 1. **AutoRec model architecture.**

2) AutoRec model performance

	ML-1M	ML-10M
U-RBM	0.881	0.823
I-RBM	0.854	0.825
U-AutoRec	0.874	0.867
I-AutoRec	**0.831**	**0.782**

(a)

$f(\cdot)$	$g(\cdot)$	RMSE
Identity	Identity	0.872
Sigmoid	Identity	0.852
Identity	Sigmoid	**0.831**
Sigmoid	Sigmoid	0.836

(b)

	ML-1M	ML-10M	Netflix
BiasedMF	0.845	0.803	0.844
I-RBM	0.854	0.825	-
U-RBM	0.881	0.823	0.845
LLORMA	0.833	**0.782**	0.834
I-AutoRec	**0.831**	**0.782**	**0.823**

(c)

FIGURE 2. **Comparison RMSE of AutoRec and other models on different datasets.**

B. Caser

1) User behaviour visualization discussed in Caser paper

In the Caser paper, the authors argue that previous Markov Chain-based models fails to capture the union-level influences and skip behaviours.

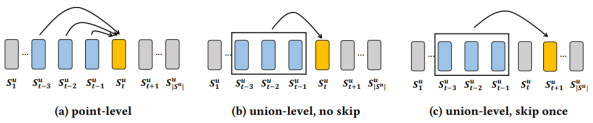

FIGURE 3. **Visualization of point-level, union-level, and skip behaviours.**

2) Caser model architecture

The Caser model architecture is shown in figure 5. The model takes the user-item interaction matrix as input and applies horizontal and vertical convolutional filters to capture point-level and union-level interactions. The latent representation of the item-sequence is then concatenated with the user-embedding. The output of the model is a probability distribution over the items, which represents the probability of the user interacting with the item at the timestep we are predicting for.

3) Caser model performance

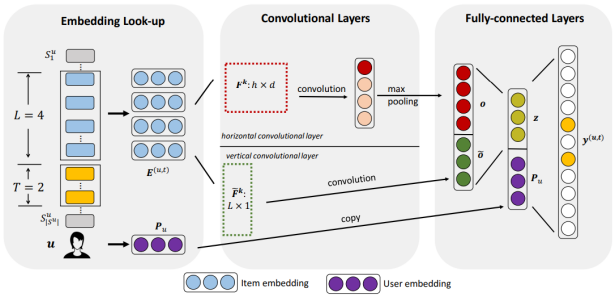

FIGURE 4. **Caser model architecture.**

Dataset	Metric	POP	BPR	FMC	FPMC	Fossil	GRU4Rec	Caser	Improv.
MovieLens	Prec@1	0.1280	0.1478	0.1748	0.2022	0.2306	**0.2515**	0.2502	-0.5%
	Prec@5	0.1113	0.1288	0.1505	0.1659	0.2000	0.2146	**0.2175**	1.4%
	Prec@10	0.1011	0.1193	0.1317	0.1460	0.1806	0.1916	**0.1991**	4.0%
	Recall@1	0.0050	0.0070	0.0104	0.0118	0.0144	**0.0153**	0.0148	-3.3%
	Recall@5	0.0213	0.0312	0.0432	0.0468	0.0602	0.0629	**0.0632**	0.5%
	Recall@10	0.0375	0.0560	0.0722	0.0777	0.1061	0.1093	**0.1121**	2.6%
	MAP	0.0687	0.0913	0.0949	0.1053	0.1354	0.1440	**0.1507**	4.7%
Gowalla	Prec@1	0.0517	0.1640	0.1532	0.1555	0.1736	0.1050	**0.1961**	13.0%
	Prec@5	0.0362	0.0983	0.0876	0.0936	0.1045	0.0721	**0.1129**	8.0%
	Prec@10	0.0281	0.0726	0.0657	0.0698	0.0782	0.0571	**0.0833**	6.5%
	Recall@1	0.0064	0.0250	0.0234	0.0256	0.0277	0.0155	**0.0310**	11.9%
	Recall@5	0.0257	0.0743	0.0648	0.0722	0.0793	0.0529	**0.0845**	6.6%
	Recall@10	0.0402	0.1077	0.0950	0.1059	0.1166	0.0826	**0.1223**	4.9%
	MAP	0.0229	0.0767	0.0711	0.0764	0.0848	0.0580	**0.0928**	9.4%
Foursquare	Prec@1	0.1090	0.1233	0.0875	0.1081	0.1191	0.1018	**0.1351**	13.4%
	Prec@5	0.0477	0.0543	0.0445	0.0555	0.0580	0.0475	**0.0619**	6.7%
	Prec@10	0.0304	0.0348	0.0309	0.0385	0.0399	0.0331	**0.0425**	6.5%
	Recall@1	0.0376	0.0445	0.0305	0.0440	0.0497	0.0369	**0.0565**	13.7%
	Recall@5	0.0800	0.0888	0.0689	0.0959	0.0948	0.0770	**0.1035**	7.9%
	Recall@10	0.0954	0.1061	0.0911	0.1200	0.1187	0.1011	**0.1291**	7.6%
	MAP	0.0636	0.0719	0.0571	0.0782	0.0823	0.0643	**0.0909**	10.4%
Tmall	Prec@1	0.0010	0.0111	0.0197	0.0210	0.0280	0.0139	**0.0312**	11.4%
	Prec@5	0.0009	0.0081	0.0114	0.0120	0.0149	0.0090	**0.0179**	20.1%
	Prec@10	0.0007	0.0063	0.0084	0.0090	0.0104	0.0070	**0.0132**	26.9%
	Recall@1	0.0004	0.0046	0.0079	0.0082	0.0117	0.0056	**0.0130**	11.1%
	Recall@5	0.0019	0.0169	0.0226	0.0245	0.0306	0.0180	**0.0366**	19.6%
	Recall@10	0.0026	0.0260	0.0333	0.0364	0.0425	0.0278	**0.0534**	25.6%
	MAP	0.0030	0.0145	0.0197	0.0212	0.0256	0.0164	**0.0310**	21.1%

FIGURE 5. **Comparison of Caser and other models on different datasets.**

C. RNN

1) RNN model architecture

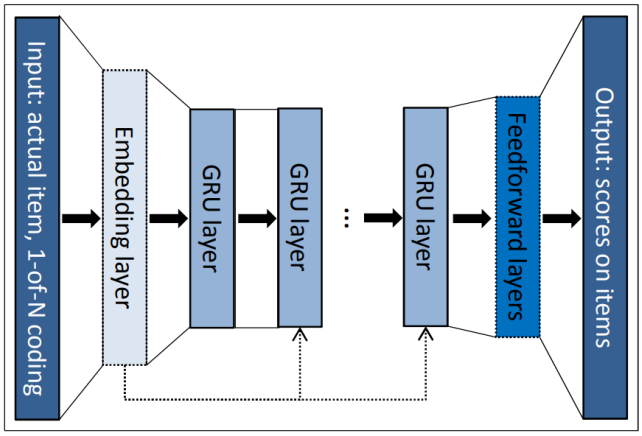

FIGURE 6. **RNN model architecture.**

2) RNN model performance

Loss / #Units	RSC15 Recall@20	RSC15 MRR@20	VIDEO Recall@20	VIDEO MRR@20
TOP1 100	0.5853 (+15.55%)	0.2305 (+12.58%)	0.6141 (+11.50%)	0.3511 (+3.84%)
BPR 100	0.6069 (+19.82%)	0.2407 (+17.54%)	0.5999 (+8.92%)	0.3260 (-3.56%)
Cross-entropy 100	0.6074 (+19.91%)	0.2430 (+18.65%)	0.6372 (+15.69%)	0.3720 (+10.04%)
TOP1 1000	0.6206 (+22.53%)	**0.2693 (+31.49%)**	**0.6624 (+20.27%)**	**0.3891 (+15.08%)**
BPR 1000	**0.6322 (+24.82%)**	0.2467 (+20.47%)	0.6311 (+14.58%)	0.3136 (-7.23%)
Cross-entropy 1000	0.5777 (+14.06%)	0.2153 (+5.16%)	–	–

FIGURE 7. **Comparison of RNN and other models on different datasets.**

D. DRN

1) DRN model architecture

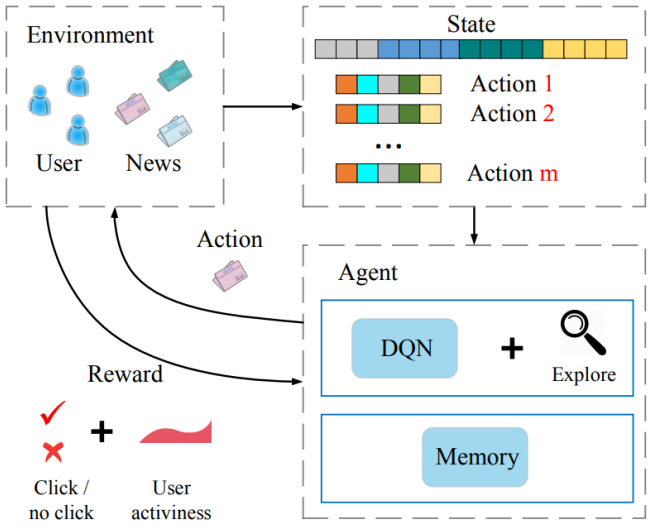

FIGURE 8. **DRL model architecture.**

2) DRN model feature description

TABLE 1. Summary of Feature Dimensions

Feature Type	Dimension	Description
News features	417	One-hot features describing properties of news such as headline, provider, ranking, entity name, category, topic category, and click counts in the last 1 hour, 6 hours, 24 hours, 1 week, and 1 year.
User features	2065	Features describing user-clicked news properties (headline, provider, ranking, entity name, category, topic category) across 1 hour, 6 hours, 24 hours, 1 week, and 1 year, plus total click counts per time granularity.
User news features	25	Features describing user-news interactions, specifically frequencies of entities, categories, topic categories, and providers in user's reading history.
Context features	32	Features describing context at news request time, including time, weekday, and freshness of news (time gap between request and publish time).

3) DRN model online learning process

TABLE 2. Overview of Recommendation Process Steps

Step	Description
(1) PUSH	At each timestamp $(t_1, t_2, t_3, \dots)$, when a user sends a news request to the system, the recommendation agent G takes the feature representation of the current user and news candidates, and generates a top-k list of news to recommend. This list is generated by combining exploitation of the current model and exploration of novel items.
(2) FEEDBACK	The user who received recommended news interacts with it by clicking. This generates feedback B.
(3) MINOR UPDATE	After each timestamp (e.g., after t_1), using the feature representations of the previous user and news list L, and feedback B, agent G compares performance of exploitation network Q and exploration network $\widetilde{Q}$. If $\widetilde{Q}$ outperforms Q, the model is updated accordingly; otherwise, Q remains unchanged. This update happens after every recommendation impression.
(4) MAJOR UPDATE	After a certain period (e.g., after t_3), agent G uses accumulated feedback and activeness scores to update the network Q via experience replay. G maintains a memory of recent historical clicks and user activeness. G samples a batch of records to update the model.
(5) REPEAT	Repeat steps (1)–(4).

4) DRN Q network architecture

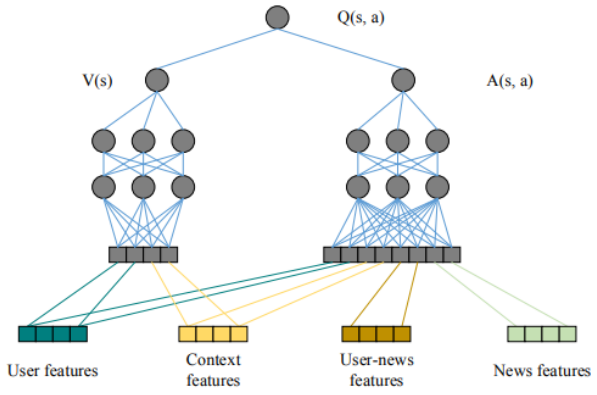

FIGURE 9. Q network architecture.