Expert2Vec: Distributed Expert Representation Learning in Question Answering Community

Xiaocong Chen[1], Chaoran Huang[1], Xiang Zhang[1], Xianzhi Wang[2], Wei Liu[1], and Lina Yao[1]

[1] University of New South Wales, Sydney, NSW, 2052, Australia
{xiaocong.chen,chaoran.huang,wei.liu,lina.yao}@unsw.edu.au
xiang.zhang3@student.unsw.edu.au
[2] University of Technology Sydney, Sydney, NSW, 2007, Australia
xianzhi.wang@uts.edu.au

Abstract. Community question answering (CQA) has attracted increasing attention recently due to its potential as a de facto knowledge base. Expert finding in CQA websites also has considerably board applications. Stack Overflow is one of the most popular question answering platforms, which is often utilized by recent studies on the recommendation of the domain expert. Despite the substantial progress seen recently, it still lacks relevant research on the direct representation of expert users. Hence hereby we propose *Expert2Vec*, a distributed Expert Representation learning in question answering community to boost the recommendation of the domain expert. Word2Vec is used to preprocess the Stack Overflow dataset, which helps to generate representations of domain topics. Weight rankings are then extracted based on domains and variational autoencoder(VAE) is unitized to generate representations of user-topic information. This finally adopts the reinforcement learning framework with the user-topic matrix to improve it internally. Experiments show the adequate performance of our proposed approaches in the recommendation system.

Keywords: Stack Overflow · Expertise finding · Question answering · Embedding · Recommendation System · Reinforcement Learning

1 Introduction

Recommender Systems are software applications that support users in finding items of interest within a larger number of objects in a personalized way. Community question answering(CQA), such as Quora, Stack Overflow and so on, is a type of web application which contains a large number of open-end questions and answers. The major challenge existing on CQA is that the question can not be answered on time and as such users can not get the expected answers very soon. The expert recommendation problem, which recommends some expert users to an unsolved question such that the question can be answered in an acceptable time block after it was proposed, is a problem which fits this scenario.

The challenge existing on the CQA is that there exist many questions which can not be answered in a reasonable time, meaning that it lacks a sufficient algorithm to make the match between users and questions. The common strategy that is used to solve this problem is to find previous questions which may be similar to the new question and make recommendations, or to make recommendation by using user's browsing history. However, there still exists the problem that it may not have similar problems previously[14]. For example on Stack Overflow website, assume there is a user who clicks a question named "Why is it faster to process a sorted array than an unsorted array ?". The system behind will record this action and analyse the topic. It will extract the key word "sorted array","unsorted array" and "fast". Based on these keywords, it makes a recommendation to a new question which is "Is '==' in sorted array not faster than unsorted array". This strategy has the same problem which was mentioned before; there may not be any other questions with the same keywords, and so the recommendation system may not work properly.

In recent years, reinforcement learning has achieved many impressive processes in learning representation[4], improving generative adversarial network[5] and so on.Reinforcement learning can be applied in many areas such as recommendation system[7], general game playing and many other areas. In this work, we adopted the reinforcement learning into our model to improve the accuracy of our embedding. The normal reinforcement learning based recommendation systems treat the recommendation system as sequential actions between the users and the system(agent) and try to figure out a optimal strategy to maximise the reward[27]. Different from the normal reinforcement learning based recommendation system, we use the reinforcement learning to determine the best policy to optimize our embedding instead of finding a optimal recommending strategy. The major contributions we made in this paper present as follows:

- We acquire some idea about the distributed representation from word2vec which is applied on the CQA problem as well as the expert recommendation. Based that, we propose a new distributed representation for user expertise which does not have many research before.
- Evolution on the big and complex data set - Stack Overflow where we got a acceptable result on several measure metrics among a few state-of-art models.
- We apply the reinforcement learning to improve the embedding so that it can get a better performance in recommendation.

The remainder of this paper has the following structure. It will follow the related work which discusss the history and recent year's progress on expert recommendation. After discussing the related work, we explain the methodology in detail. which includes how to pre-process the data, how to utilize the data set to get the input and the learning step of the distributed representation. The result analysis is presented, as well as the conclusion and the future work at the end.

2 Related Work

2.1 CQA and Expert recommendation

As the expert recommendation acquire some good research works and the CQA problem can gain the experience from the expert recommendation[24]. So that apply expert recommendation on CQA problem comes more common in recent years[20]. Query likelihood language (QLL) is most popular learning model which used on this question which use the hidden markov model[16]. After the QLL model which get a good result on CQA problem, Zheng et al.[30] proposed a expertise-aware QLL model which based on QLL and combine the answers quality to increase the recommend accuracy.The QLL model are focus on the language side which is majority on natural language processing.

Another type of models were focus on the topic, those models try to mine the information behind the topic such as Latent Dirichlet Allocation (LDA), Probabilistic Latent Semantic Analysis (PLSA) and so on [22]. Segmented Topic Model is a hierarchical topic model based on LDA which proposed by Du et al.[6]. Also, there is type of method is network-based which build a user-user network and try to use the relation between users to make recommendation such as PageRank[2].

As the expert recommendation is a type of recommendation problem, there are some models which are based on collaborative filtering(CF) or matrix factorization(MF). Yang et al. proposed a model which uses the probabilistic matrix factorization(PMF)[23], an extension of the basic MF. As the neural network acquires outstanding results on many areas including expert recommendation. Liu et al. proposed a model which combine the QLL and the LDA to compute the relevance and make recommendation by assuming that good respondents will give better answers[13]. Zheng et al. find that the convolutional neural network(CNN) can be used to combine the user features and question features together to make recommendation[28] more accurate. There are some hybrid methods which can combine topic model and apply the classification methods[10] or Network-based method with clustering [3]. However, the accuracy is still lower than expectation, the reason is that the NLP technique still needs be improved and the dataset used in CQA is extremely sparse.

2.2 Auto-Encoder

Autoencoder is a type of neural network which is normally used in dimension reduction and feature learning[21, 26]. Autoencoder is a multi-layer neural network which has an encoding layer, hidden layer and a decoding layer. It uses backpropagation to make the output as same as the input. There are many works that show that the autoencoder can get a good result in sentence encoding, image encoding and many other areas[12, 17]. Sarath et al. state that the autoencoder can do the feature learning from the word representation[1].

2.3 Reinforcement learning based recommendation system

Reinforcement learning(RL) in recommendation systems attracts many interests in recent years. Yash et al. proposed a new way to represent the actions during the learning process to improve RL's training efficiency and robustness[4]. As the generative adversarial network(GAN) obtains a promising result on computer vision and recommendation system area, Chen et al. proposed a GAN model for RL based recommendation system[5] which combines the GAN and the RL by cascading. Zhao et al. proposed a model-based deep RL for a sequential recommendation especially in whole-chain recommendation [27] which uses user's feedback as the reward and adopting the auto-encoder. Zheng et al. proposed a deep reinforcement learning framework for the news recommendation which based on the deep Q-learning[29]. Expert recommendation is similar with the normal recommendation, but more specific. Traditional recommendation system aims to recommend users some items based on users' purchase history or browse history. The common strategy is: assign an id to a user and label the items with id as well, then the recommendation system will based on this two-dimension user-item matrix to make recommendation. There are some works which attempt to add more temporal information to boost the performance like review[25] and had some improvement. However, expert recommendation requires a focus on the answers which will be a higher dimension vector than the normal items. If we follow the common strategy to construct a user-item matrix, it will have an extremely high dimension matrix and the traditional recommendation system will get stuck on it.

3 Background

Recommendation system can recommend k items to users in a single page, the user can provide some feedback by clicking one of those choices or switch the pages. After the feedback is provided, the system will record it and recommend another k items based on user's feedback. RL have two different branches which are model-based RL and model-free RL. The model-free RL based recommendation systems require a big dataset which contain a large number of user actions so that it can figure out a good policy. A larger sized dataset may lead to problems such as the model being hard to converge, recommendation system needing to be complex enough to handle those information and so on. RL initially comes from the Markov decision process(MDP), which defined as:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R) \text{ Where } P \in [0, 1]$$

Where $\mathcal{S}$ represents the set of states, $\mathcal{A}$ represents the set of actions, P is the probability of transition which is normally written as $P(s_{t+1}|s_t, a_t)$ where $s_{t+1}, s_t \in \mathcal{S}, a_t \in \mathcal{A}$ which represents the probability of action a_t transfer from state s_t to state s_{t+1} from a certain timestamp t to timestamp $t+1$, the R is the reward function. If we consider the discount factor γ, the MDP can be written as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$.

However, the model-based RL methods will suffer from the computation difficulty when the $\mathcal{S}, \mathcal{A}$ becomes large. So we prefer to use the model-free RL methods. The model-free RL methods have two different approach which are value-based methods and policy-based methods. The traditional value-based RL methods is the Temporal difference(TD) which are trying to find the optimal value V^* by iteration:

$$V(s) = (1-\eta)V(s) + \eta(R(s) + \gamma V(s_{t+1})$$

In some cases, the value will depend on both states and actions, so we have the Q-learning. In Q-learning, we use the Q-value $Q(s,a)$ to determine the optimal policy π^*, different from the MDP, the Q-value baed on the pair of action a and state s instead of using state s only. The definition of π^* can be written as:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$$

the Q^* is the optimal Q-value where can be defined as:

$$Q^*(s,a) = R(s) + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1} \in \mathcal{A}} Q^*(s_{t+1}, a_{t+1})$$

This formula was used when known the certain state-action pair (s,a), and the γ is the discount factor which used in a long-term RL. During the training process the Q-value will be updated iteratively based on:

$$Q(s,a) \leftarrow (1-\eta)Q_o(s,a) + \eta(R(s) + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a))$$

The Q-learning is an off-policy learning which means it will learn from different policy and try to learn the value. However, the TD method is the on-policy learning which means it can only learn different values in the same policy. Benefiting from the neural network, the Q-learning was extended to the deep Q-learning(DQN). The DQN will pass the state s into a neural network and find out many q-values at once. The DQN have a similar target with the normal Q-learning, the DQN try to do:

$$\min[R(s) + \gamma \max_{a'} Q_w(s_{t+1}, a') - Q_w(s_t, a_t)]^2$$

where the $Q_w(s,a)$ is parametrized by the neural network weight w. During the optimizing the neural network, we only need to apply the gradient descent into the term $Q_w(s_t, a_t)$ which is the current q value for current state s_t and action a_t. In addition, to help the converge, the DQN uses the technique called experience replay that can store recent j experience pairs $(s_t, a_t, R(s), s_{t+1})$ with a replay batch with size j. The DQN will choose action based on the greedy algorithm from the reply batch and the current value.

We will use the Q-learning in our proposed model, here are some key aspects used in our model:

- **Environment**: Is the system which user can select on the top k items which provided by the recommendation system.
- **State** $s \in \mathcal{S}$: will defined as the match from the recommended items and user's exactly choose, in simple it represent the value changed in the embedding matrix which will detailed discussed in section 4.3.
- **Action** $a \in \mathcal{A}$:is defined as a subset $\mathbb{A} \subset k$ which those k items is the possible experts/topics show to the user. Also, the $\mathcal{A} \in (\genfrac{}{}{0pt}{}{I_t}{k})$ where the I_t is the whole possible topics/experts which may be recommended,the $(\genfrac{}{}{0pt}{}{I_t}{k})$ means that we select top k items from the item-set I_t at timestamp t.
- **State Transition Probability** $P(s_{t+1}|s_t, a_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0,1]$:it corresponds to a user behavior a_t which will give a probability from current state s to next state s_{t+1} at the timestamp t.
- **Reward Function** $R(S)$:$\in [0,1]$: Unlike the normal RL method, we do not have a mapping function used for reward. The reward value used in our model is the accuracy between the decoded embedding and the original data as we are aiming to use the RL to improve our embedding.
- **Policy** $\pi(s)$:is defined as the strategy on how to optimize our embedding which generated by the auto-encoder.
- **Discount Factor** γ **and Learning Rate** η : $\eta, \gamma \in [0,1]$: is the hypeparameter in this model, and need to be adjusted manually depends on the measure metric. Where when the $\gamma = 0$ which means the long-term reward will not be considered, only the current reward will be take into account. In opposite, $\gamma = 1$ means all the reward from previous can be fully considered in current state s.

The overview of a simple RL based recommendation system is (where the recommend agent represent the whole recommendation system):

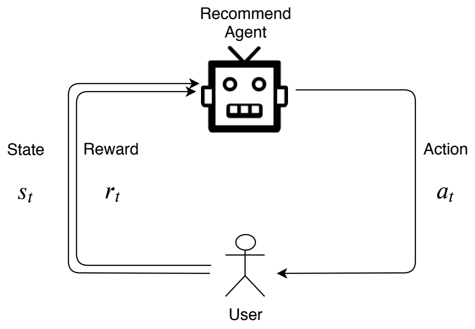


Fig. 1: Flaw chart for RL

4 Methodology

In this section, we will briefly illuminate our approach and the model structure. We will discuss the pre-processing step, how to obtain the original representation, how to get the embedding we are looking for, how to measure the accuracy between the embedding and the original representation, how reinforcement learning works in our model and how to used in a recommendation system.

4.1 Pre-process of the data

The original dataset provided by Stack Overflow contain all questions and the corresponding answers which are plain text. We use the *SEWordSim* [19] which is a word similarity database for the Stack Overflow dataset which can make it more reliable and reduce some edge effects. In addition, we delete the questions which have zero response to overcome the cold-start problem for recommendation system. After deleting the unnecessary words, we extract all the users and users' answered questions and its corresponding vote received. Furthermore, the sentences are in higher dimension space, in order to reduce the dimension, we would convert all the topics and answers into vectors by using word2vec[15]. As the word2vec is a distributed representation of words which can retain the relation in the sentence, so we can use the vector form directly in the following steps.

4.2 Generate the User-Topic Matrix

After the pre-processing of the data, we get the formatted data which is needed for the matrix generation and the vector for each word. To keep all the information which is needed for ranking, we store all the voting and its corresponding user and topic together. As the topic is vector based and has high dimension which is hard to put it together with the topic information into the matrix. We build a hashing function $f : \mathbb{R} \mapsto \mathbb{R}$ which can simply map the topicID to its corresponding original vector, and we initialise a hash table to store all the mapping relations based on our hash function f. Thus our user-topic matrix can contain all the voting information by ordering. If user don't have action with specific topic, the value of this cell with be null. In some cases the userId may not be continuous, and we convert all the userId with a list of continuous id so that it can easily determine the size. Also, it's easy to roll back to the original id. In addition, the userId is not important in the user-topic matrix as we only need to know the relationship between user and the topic.

The most important thing is how to rank the topic for each user. We cannot use the original voting information because less-popular questions may have very small view counts which can lead to a good answer receiving only one or zero vote. Also, the number of answers is varied for different questions which means we are unable to compare those answers through the same measurement metric due to the different number of competing answers. Therefore, we require a consistent measurement metric to help us determine the order, which means we need to make sure the votes in popular questions and less-popular questions are equivalent. To overcome this problem, we use the percentage vote(PVote) to compare answers, where we transfer all the votes receive for a certain answer j in a question q into a percentage mark:

$$\mathrm{PVote}_q^j = \frac{V_{\mathrm{j}}}{\sum_{i=0}^{n} V_i}$$

Where V_j means the vote that jth answer get on question q, n means number of answers we have on the question q. The denominator is the sum of all answers' vote. In addition, PVote can restrict the value in range $[0, 1]$ which means we do not need further processing or normalization. We do not need to consider about the case which denominator is zero as we delete all the topics which have zero response. Then, we convert all the topicId back to it's vector form as we need the topic information. After those operations we get a User-Topic Matrix $M : U \times \overrightarrow{T} \in \mathbb{R}^{T \times U}$, where U, T is the number of users and topics, $\overrightarrow{T}$ is the ranked topics.

4.3 AutoEncoder

As the user-topic matrix R has already been generated, the dimension of R is acceptable but the size $U \times T$ is relatively large. So we use the matrix R as the input of the variational autoencoder(VAE)[11]. Then, the VAE can learn a lower-dimension representation during the training which is the embedding E. The reason why we use the VAE is that the autoencoder is used widely on dimension reduction and features learning. Our user-topic matrix R have a high dimension topic embedding, we need to figure out a way to reduce the dimension and retain the necessary information to conduct analysis. The representation E we get from the autoencoder is the rough version of the embedding we are looking for. Then we need to calculate the similarity of the representation E. We pass the E into the decoder so that we can get a decoded matrix D_e which is supposedly the as same as the original matrix R. We use the "accuracy" to measure the similarity between D_e and R:

$$accuracy = \frac{\sum_i^n D_e^i \odot R^i}{n}$$

where n is the number of elements inside matrix R and D_e. The $\odot$ is the XNOR which used to calculate how many elements are exactly samethe D_e^i, R^i is the i-th element in matrix D_e and R. The XNOR operator has following property:

$$a \odot b = \begin{cases} 0 & \text{if } a \neq b \\ 1 & \text{if } a == b \end{cases}$$

4.4 Reinforcement Learning and Recommendation

As the accuracy was defined in previous section, we will use this accuracy as the reward in our reinforcement learning framework. We use the Q-learning here, the training algorithm was described in the Algorithm 1. We will use the Q-learning to allow our model to improve the embedding E by itself. The n in algorithm refers to the number of episode. The strategy is to find a best direction of the value change in the embedding E which can acquire the highest Q-value. Once the optimal Q-value reached, it means we have figured out an optimal policy π^* which can improve our embedding representation E. Finally, we obtain an optimal embedding E^* which we can use in the recommendation system. In the

Algorithm 1: Q-learning

```
Initialize Q-table,Q(s,a) randomly;
Initialize embedding $E$ comes from the VAE;
Initialize $\eta \leftarrow \eta_{init}, \gamma \leftarrow \gamma_{init}$ ;
for $i = 0$ to $n$ do
    Initialize $s$;
    $r$ = accuracy($E$,R);
    for each step in episode $i$ do
        choose $a$ from $s$ using policy derived from $Q$ ;
        $Q(s_t, a) \leftarrow Q(s_t, a) + \eta(r + \gamma \max_a Q(s_{t+1}, a))$ ;
        use the q-value find the policy: $\pi \leftarrow Q(s, a)$;
        $s_t \leftarrow s_{t+1}$
    end
    use the policy update the embedding: $E \leftarrow E'$;
end
```

real recommendation system, what we will use is the optimal embedding E^*. The embedding cannot be used for recommendation directly as the embedding does not have any valuable information for recommendation system. So, we need to recover the embedding E^*, through the decoder, into a matrix R' that contains the user-topic information and the ranking information. The recommendation method we used is the collaborative filtering(CF). So, the overall structure for our proposed model in figure 2.

5 Experiment

5.1 Experiment Setup

The data set used for experiment is the Stack Overflow which was flattened by removing all the XML markups and converted into the json format. The original data set contained 14,768,990 records including answers and questions. After filtering, the dataset was changed as 'userID:Topic' format that was described in section 4.1. After that, we had 99,220 users and 118,320 questions in total. As we conduce some data cleaning technique with the dataset, it leads to the userID not being consecutive as some users are considered inactive users. If we use the original userId as the axis it will make our matrix extremely big

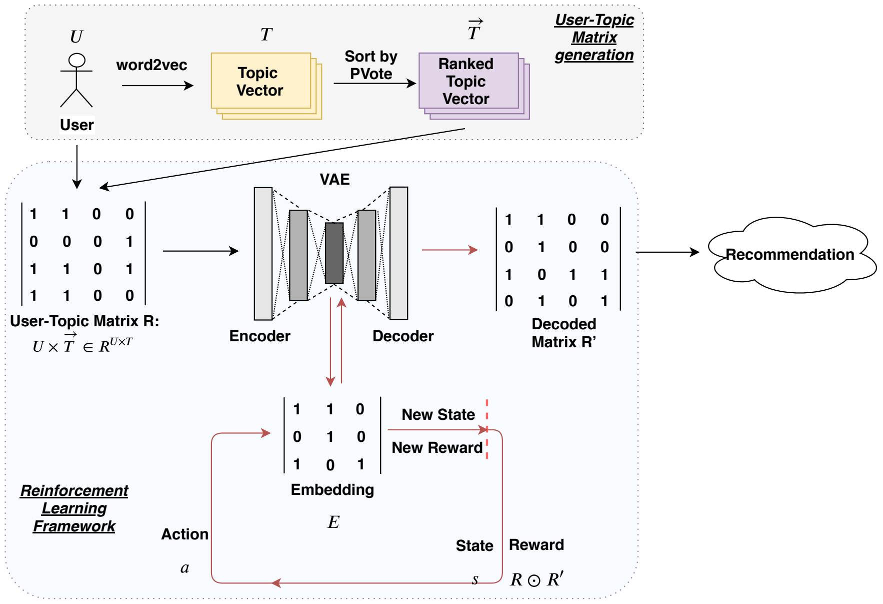


Fig. 2: Model Structure,where the red line represent the work flow of RL. The new state is s_{t+1}, new reward is $R_{t+1} \odot R'$, t is the timestamp

as the userID comes from 0 to 6,454,151, but we only need the 99,220 active users. Using the original userId as the index of matrix will get a matrix with shape [6454151,118320] which will take a huge amount of memory of computer. To overcome this problem, we replace the normal userId with our customised userId by using a hash table can map from [1,99220] to the [1,6454151]. By using the customised userId we can save $\frac{6454151-99220}{6454151} = 98.5\%$ run-time memory. So we have a user-topic matrix R which has shape [99,220,118,320] with the values 1, meaning have action, and 0, meaning no action. Also, topic we used in R is the topic id which is mapped as well(See section 4.2 for detail). The methodology we used for getting the vector is the word2vec, we use the pre-trained word2vec to transfer the topic into vector. What we do is that we firstly generate all single word's vector by using word2vec, so we get two lists which have word name and its vector. After that, the topic will be convert into vector with the dimension of 300. As the data is pretty big for training, to vertify the correctness of our approach, we just select top 20% of the samples from the dataset based on the reputation which still have over 2,000,000 records.

After finished the pre-process step, we just put the user-topic matrix R into the VAE to get a reconstructed representation E. To verify this representation is valid and have the necessary information we need, we recover it back to user-topic matrix R' by using the decoder. Then we calculate the accuracy between R and R' by using the formula mentioned in section 4.3. Then we put the embedding E and the accuracy R into the Q-learning framework to improve it. For each episode

i, we take the improved embedding E_i and compare with the original matrix R to get the new accuracy and transfer back to the Q-learning. After this optimize process, we passing the optimal embedding E' into a normal recommendation system. Then, using the Accuracy and the nDCG as the measure metric in our model where the accuracy is defined previously, and the nDCG is defined as:

$$nDCG_p = \frac{\sum_{i=1}^{p} x}{\sum_{i=1}^{|REL|} x} \text{ where } x = \frac{2^{rel_i} + 1}{\log_2(i+1)}$$

the rel_i is the real result which i supposed to be. We will use the nDCG@k, accuracy@k and the recall@k as the major measurement metric.

5.2 Experiment Results

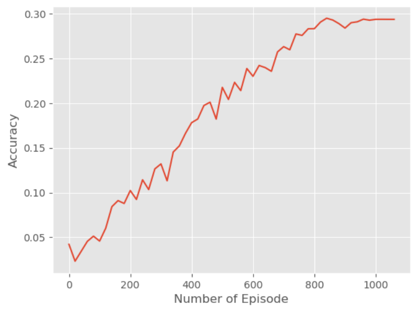
As the expert recommendation is not a popular area, the state-of-art model is hard to figure out[20]. So, the baseline we used here is the probabilistic matrix factorization(PMF)[23],Bayesian probabilistic matrix factorization(BPMF), the segmented topic model(STM)[6], GRE4Rec[9], Convolutional Sequence Embedding Recommendation Model(Caser)[18] and Adversarial Personalized Ranking for recommendation (APR)[8]. The result can be found in figure 3.

5.3 Evaluation

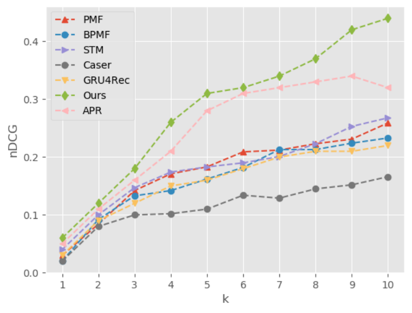
It is obvious that when the accuracy increases the nDCG increases as well, which means the reinforcement learning is improving the embedding E by itself. However the nDCG@k is still not good enough which the highest value can reach 0.4767834. The reason is due to data sparsity. Even if the number of records are reduced and all the active users in the dataset are selected, it is still too sparse for the recommendation system to recommend a topic for a user. But we can see that our model is better than the others. The accuracy which is passed into the reinforcement learning is stable after a few episodes and it is stable at around 0.3, which means much information is lost during the encoding and decoding process. However, we still obtain a competitive result on the recommendation which means that our model meets the expectation. The caser is a state-of-art model which used in recommendation system area, it's sensitive with the sparsity data which we can find that the result is not good enough.

5.4 Possible improvement and future work

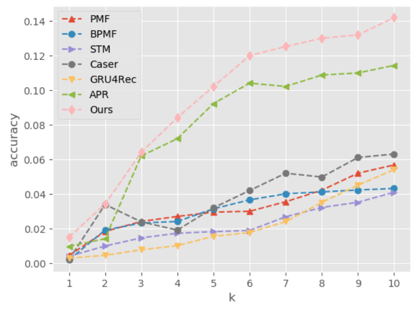
As discussed in section 4.1, we mentioned that in some questions they only have one answer which means user can only vote this answer or answer one. It may lead to some edge effects which will make the recommendation less efficient. Furthermore, due to the limitation of NLP technique, we may still lose information during the word2vec, and the answers are normally a paragraph which contains many sentences. We need a more efficient way to capture the relation between the word in word level and sentence level. That is the possible improvement



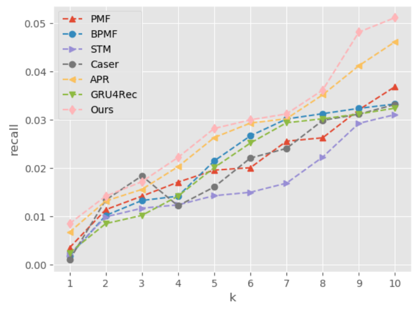
a) (a)



b) (b)



c) (c)



d) (d)

Fig. 3: Graph (a) is the accuracy during the RL process.(b) is the model comparison result in nDCG ,(c) is the comparison result in accuracy, (d) is the comparison result in recall.

can be done in the dataset side. From the model perspective, we can make some improvements in the recommendation system and the reinforcement learning system. For example, we can change the CF to the matrix factorization(MF) based recommendation system or a more complex model. But the challenge is that all the state-of-art recommendation systems are not working properly in CQA, in the future we may can adopt the state-of-art recommendation systems to the CQA problem so that it can make recommendation through our embedding.

As the neural network get some surprising result on reinforcement learning, we may change our reinforcement learning framework to deep reinforcement learning. One typical example is that change the Q-learning to DQN which discussed in section 3. But consider about the dataset's complexity, it will be tough to employ the complex recommendation system and the DQN framwork.

6 Conclusion

In this study we proposed a new distributed representation (expert2vec) for expert which is used on solving the CQA problem and the expert recommendation problem. Expert2vec is the distributed representation which contain the information about user and topic and its corresponding rank. We innovatively adopt the reinforcement learning framework into the expert recommendation problem to let the model to improve the embedding by itself. Our model(Expert2Vec) got a promising result among the current expert recommendation state-of-art model.

References

1. AP, S.C., Lauly, S., Larochelle, H., Khapra, M., Ravindran, B., Raykar, V.C., Saha, A.: An autoencoder approach to learning bilingual word representations. In: Advances in Neural Information Processing Systems. pp. 1853–1861 (2014)
2. Borodin, A., Roberts, G.O., Rosenthal, J.S., Tsaparas, P.: Link analysis ranking: algorithms, theory, and experiments. ACM Transactions on Internet Technology (TOIT) **5**(1), 231–297 (2005)
3. Bouguessa, M., Dumoulin, B., Wang, S.: Identifying authoritative actors in question-answering forums: the case of yahoo! answers. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 866–874. ACM (2008)
4. Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., Thomas, P.S.: Learning action representations for reinforcement learning. In: International Conference on Machine Learning (2019)
5. Chen, X., Li, S., Li, H., Jiang, S., Qi, Y., Song, L.: Generative adversarial user model for reinforcement learning based recommendation system. In: International Conference on Machine Learning. pp. 1052–1061 (2019)
6. Du, L., Buntine, W., Jin, H.: A segmented topic model based on the two-parameter poisson-dirichlet process. Machine learning **81**(1), 5–19 (2010)
7. Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., Coppin, B.: Deep reinforcement learning in large discrete action spaces. arXiv preprint arXiv:1512.07679 (2015)
8. He, X., He, Z., Du, X., Chua, T.S.: Adversarial personalized ranking for recommendation. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. pp. 355–364. ACM (2018)
9. Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939 (2015)
10. Ji, Z., Wang, B.: Learning to rank for question routing in community question answering. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 2363–2368. ACM (2013)
11. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
12. Li, J., Luong, M.T., Jurafsky, D.: A hierarchical neural autoencoder for paragraphs and documents. arXiv preprint arXiv:1506.01057 (2015)
13. Liu, M., Liu, Y., Yang, Q.: Predicting best answerers for new questions in community question answering. In: International Conference on Web-Age Information Management. pp. 127–138. Springer (2010)

14. Liu, Q., Agichtein, E., Dror, G., Maarek, Y., Szpektor, I.: When web search fails, searchers become askers: understanding the transition. In: Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval. pp. 801–810. ACM (2012)
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
16. Miller, D.R., Leek, T., Schwartz, R.M.: A hidden markov model information retrieval system. In: SIGIR. vol. 99, pp. 214–221 (1999)
17. Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., Carin, L.: Variational autoencoder for deep learning of images, labels and captions. In: Advances in neural information processing systems. pp. 2352–2360 (2016)
18. Tang, J., Wang, K.: Personalized top-n sequential recommendation via convolutional sequence embedding. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. pp. 565–573. ACM (2018)
19. Tian, Y., Lo, D., Lawall, J.: Sewordsim: Software-specific word similarity database. In: Companion Proceedings of the 36th International Conference on Software Engineering. pp. 568–571. ACM (2014)
20. Wang, X., Huang, C., Yao, L., Benatallah, B., Dong, M.: A survey on expert recommendation in community question answering. Journal of Computer Science and Technology **33**(4), 625–653 (2018)
21. Wang, Y., Yao, H., Zhao, S.: Auto-encoder based dimensionality reduction. Neurocomputing **184**, 232–242 (2016)
22. Xu, F., Ji, Z., Wang, B.: Dual role model for question recommendation in community question answering. In: Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval. pp. 771–780. ACM (2012)
23. Yang, B., Manandhar, S.: Tag-based expert recommendation in community question answering. In: 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014). pp. 960–963. IEEE (2014)
24. Yao, L., Wang, X., Sheng, Q.Z., Benatallah, B., Huang, C.: Mashup recommendation by regularizing matrix factorization with api co-invocations. IEEE Transactions on Services Computing (2018)
25. Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR) **52**(1), 5 (2019)
26. Zhang, X., Yao, L., Yuan, F.: Adversarial variational embedding for robust semi-supervised learning. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 139–147. KDD '19, ACM, New York, NY, USA (2019)
27. Zhao, X., Xia, L., Zhao, Y., Yin, D., Tang, J.: Model-based reinforcement learning for whole-chain recommendations. arXiv preprint arXiv:1902.03987 (2019)
28. Zheng, C., Zhai, S., Zhang, Z.: A deep learning approach for expert identification in question answering communities. arXiv preprint arXiv:1711.05350 (2017)
29. Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N.J., Xie, X., Li, Z.: Drn: A deep reinforcement learning framework for news recommendation. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web. pp. 167–176. W3C (2018)
30. Zheng, X., Hu, Z., Xu, A., Chen, D., Liu, K., Li, B.: Algorithm for recommending answer providers in community-based question answering. Journal of Information Science **38**(1), 3–14 (2012)