Project Evaluation Plan

Neural Collaborative Filtering for Expert Recommendation

Mina Mousavifar, sem311, 11279515

Architecture

The main task in personalizing recommender systems is modelling users' preference for items based on their past interactions, known as collaborative filtering[2]. Matrix Factorization(MF)[3] method is the most popular technique in collaborative filtering which projects users and items into a shared latent space, using a vector of latent features to represent a user or an item. Then a user's interaction on an item is modelled as the inner product of their latent vectors. MF assumes that each dimension of the latent space is independent of each other and linearly combining them with the same weight.

The model consists of two separate models, Generalized Matrix Factorization(GMF)[1] and Multi-Layer Perceptron(MLP)[1] to generalize the MF method and capture the non-linearity in the latent space.

Generalized Matrix Factorization (GMF)

For the input layer, expert and question IDs will be represented by one-hot encoding. So, this embedding vector would represent the latent vector of the expert or question. The first layer of this model would be

$$\phi(\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u \odot \mathbf{q}_i \qquad \hat{y_{ui}} = a_{out}(\mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i))$$

Where the output layer, uses sigmoid function and uses the same loss function defined in the Loss section. This is the generalization of the MF method because it allows $\mathbf{h}$ to be learnt from data without the uniform constraint, it will result in a variant of MF that allows varying importance of latent dimensions.

Multi-Layer Perceptron (MLP)

GMF concatenate experts' and questions' features. However, this concatenation may not capture the interactions between expert and question features. So, I utilize a standard multilayer perceptron to learn the interaction between experts and questions latent features.

For learning the interaction between expert and question latent features $\mathbf{p}_u$ and $\mathbf{q}_i$, instead of learning the element-wise product in GMF, the following method will be applied:

$$\mathbf{z}_1 = \phi(\mathbf{p}_u, \mathbf{q}_i) = \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix}$$

$$\phi_2(\mathbf{z}_1) = a_2(\mathbf{W}_2^T \mathbf{z}_1 + \mathbf{b}_2)$$

...

$$\phi_2(\mathbf{z}_{L-1}) = a_L(\mathbf{W}_L^T \mathbf{z}_{L-1} + \mathbf{b}_L)$$

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \phi_L(\mathbf{z}_{L-1}))$$

Where $\mathbf{W}_x$, $\mathbf{b}_x$, and a_x denote the weight matrix, bias vector, and activation function for the x-th layer's neural network. Furthermore, I will use ReLu[4] for activation.

As for network structure design, a common solution is to adopt a tower model, where the bottom layer is the largest and each successive layer has fewer neurons to learn more abstractive features of the data by using a small number of hidden units for higher layers. I empirically implement the tower structure, halving the layer size for each successive higher layer. Three hidden layers will be employed for this model and the factors of [8, 16, 32, 64] will be tested for the number of neurons for the last layer.

The Fusion of GMF and MLP(NeuMF)

So far I have developed two models, GMF that applies a linear kernel to model the latent feature interactions and MLP that uses a non-linear kernel to learn the interaction function from data. For adding more flexibility to the fused model, I allow GMF and MLP to learn separate embeddings and combine the two models by concatenating their last hidden layer.

$$\phi^{GMF} = \mathbf{p}_u^G \odot \mathbf{q}_i^G$$

$$\phi^{MLP} = a_L(\mathbf{W}_L^T(a_{L-1}(...a_2(\mathbf{W}_2^T \begin{bmatrix} \mathbf{p}_u^M \\ \mathbf{q}_i^M \end{bmatrix} + \mathbf{b}_2)...)) + \mathbf{b}_L)$$

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix})$$

Where $\mathbf{p}_u^G$ and $\mathbf{p}_u^M$ are expert embeddings for GMF and MLP model, and $\mathbf{q}_i^G$ and $\mathbf{q}_i^M$ are for question embeddings. Finally, each model parameter can be calculated with standard back-propagation.

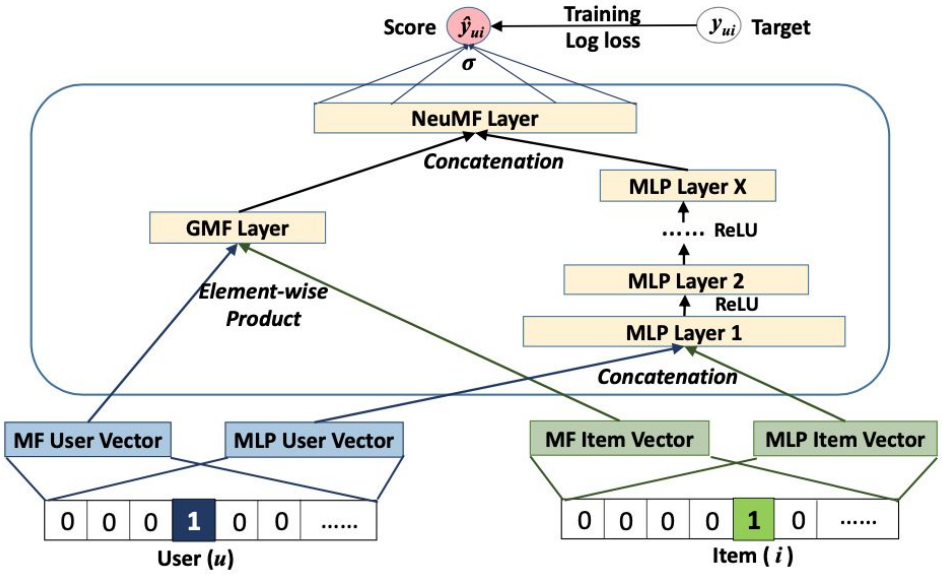

Figure 1. NeuMF architecture[1]

Loss Function

The training is performed by minimizing the pointwise loss between the predicted score $\hat{y}_{ui}$ and its target value y_{ui}, where ui is the interaction between a specific expert u and a question i. The target value y_{ui} is a binarized 1 or 0 which can be seen as 1 means expert u has answered question i, and 0 otherwise. The prediction score $\hat{y}_{ui}$ then represents how likely question i will be answered by the expert u.

Then the likelihood function is defined as binary cross-entropy loss, also known as log loss. So, the recommendation problem can be addressed as a binary classification problem.

$$p(\gamma, \gamma^{-}|\mathbf{P}, \mathbf{Q}, \theta_f) = \Pi_{(u,i)\in\gamma}\hat{y}_{ui}\Pi_{(u,i)\in\gamma^{-}}(1-\hat{y}_{ui})$$

By taking the negative logarithm of the likelihood:

$$L = -\sum_{(u,i)\in\gamma} log\hat{y_{ui}} - \sum_{(u,i)\in\gamma^{-}} log(1-\hat{y_{ui}}) = -\sum_{(u,i)\in\gamma\bigcup\gamma^{-}} y_{ui}log\hat{y_{ui}} + (1-y_{ui})log(1-\hat{y_{ui}})$$

Where $\mathbf{P} \in R^{M\times K}$ and $\mathbf{Q} \in R^{N\times K}$ denoting the latent factor matrix for experts and questions, respectively, and θ_f denotes the model parameters of the interaction function f.

Dataset Split

The dataset consists of one-year stack overflow data, with users who have a contribution in this system more than a certain threshold. There are approximately 500,000 questions, 500,000 answers and 5000 users in this dataset. I'll split the data randomly, by taking 80% as the training set and 20% for the testing. Furthermore, in order to add negative interaction to the data, I should randomly sample users who didn't answer the question and consider them as negative instances. So, I will sample four expert who hasn't answer the question, per question.

In order to evaluate the performance of the expert recommender, for each question in the test set, the model would rank all users, including the user who has actually answered the question based on prediction. For instance, in the labelling of our data, the expert who answered the question has label 1 and others are zero, so the model should rank it higher than others. Afterwards, rank-based metrics such as Hit Ratio(HR) and Normalized Discounted Cumulative Gain (NDCG)[5] can be applied to evaluate our model. Finally, I'll conduct this experiment 10 times, and take the average of such metrics.

Optimization

The proposed model consists of two separate models, Generalized Matrix Factorization(GMF)[1] and Multi-Layer Perceptron(MLP)[1], which was explained in the Architecture section.

For training GMF and MLP, mini-batch Adaptive Moment Estimation (Adam) [6] will be used, which adapts the learning rate for each parameter by performing smaller updates for frequent, and larger updates for infrequent parameters.

For the final model, the Fusion of GMF and MLP(NeuMF), Stochastic Gradient Descent(SGD)[7] will be used for optimizing.

Initialization
The GMF and MLP models will use random initializations from a Gaussian distribution (with a mean of 0 and a standard deviation of 0.01) until convergence. Then, the NeuMF parameters will be initialized by pre-trained models of GMF and MLP. So, the weights of the two models are concatenated with:

$$\mathbf{h} \leftarrow \begin{bmatrix} \alpha \mathbf{h}^{GMF} \\ 1 - \alpha \mathbf{h}^{MLP} \end{bmatrix}$$

where $\mathbf{h}^{GMF}$ and $\mathbf{h}^{MLP}$ denote the $\mathbf{h}$ vector of pre-trained GMF and MLP model and α is a hyper-parameter showing the trade-off between the two models.

Hyper-parameters
The hyper-parameters are α, *batch size*, *learning rate*. The batch size of [128, 256, 512, 1024], and the learning rate of [0.0001 ,0.0005, 0.001, 0.005] for NeuMF will be compared. Parameter α will be set to 0.5, allowing the pre-trained GMF and MLP to contribute equally to NeuMF's initialization.

Regularization
Dropout will be experimented to regularize the model. I will report the final dropout value in the final report.

Baseline
I'll use Matrix Factorization method[3], as the baseline for my model, which is the de-facto approach to latent factor model-based recommendation, which was used as the baseline for the related paper. This model performance highly relies on the choice of user-interaction function. In this project, NeuMF aims to enhance this model by generalizing this user-interaction function using deep learning in the expert recommendation domain.
Moreover, my primary objective is to try to reproduce the NeuMF result on Stackoverflow dataset, and afterwards explore the impact of adding users embedding to the model on its performance.

Related Paper
[1]Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. Proceedings of the 26th International Conference on World Wide Web - WWW 17 (2017). DOI:http://dx.doi.org/10.1145/3038912.3052569

References

[2]Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. 2001. Item-based collaborative filtering recommendation algorithms. Proceedings of the tenth international conference on World Wide Web - WWW 01 (2001). DOI:http://dx.doi.org/10.1145/371920.372071

[3]Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. Computer 42, 8 (2009), 30–37. DOI:http://dx.doi.org/10.1109/mc.2009.263

[4]Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. (2010). Retrieved March 3, 2020, from http://www.cs.toronto.edu/~fritz/absps/reluICML.pdf

[5]Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. 2015. TriRank. Proceedings of the 24th ACM International on Conference on Information and Knowledge Management - CIKM 15 (2015). DOI:http://dx.doi.org/10.1145/2806416.2806504

[6]Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. (December 2014). Retrieved March 2, 2020, from https://arxiv.org/abs/1412.6980

[7]Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. The Annals of Mathematical Statistics 22, 3 (1951), 400–407. DOI:http://dx.doi.org/10.1214/aoms/1177729586