Paper Review: Neural Collaborative Filtering Explanation & Implementation

Kung-Hsiang, Huang (Steeve) Sep 3, 2018 · 6 min read

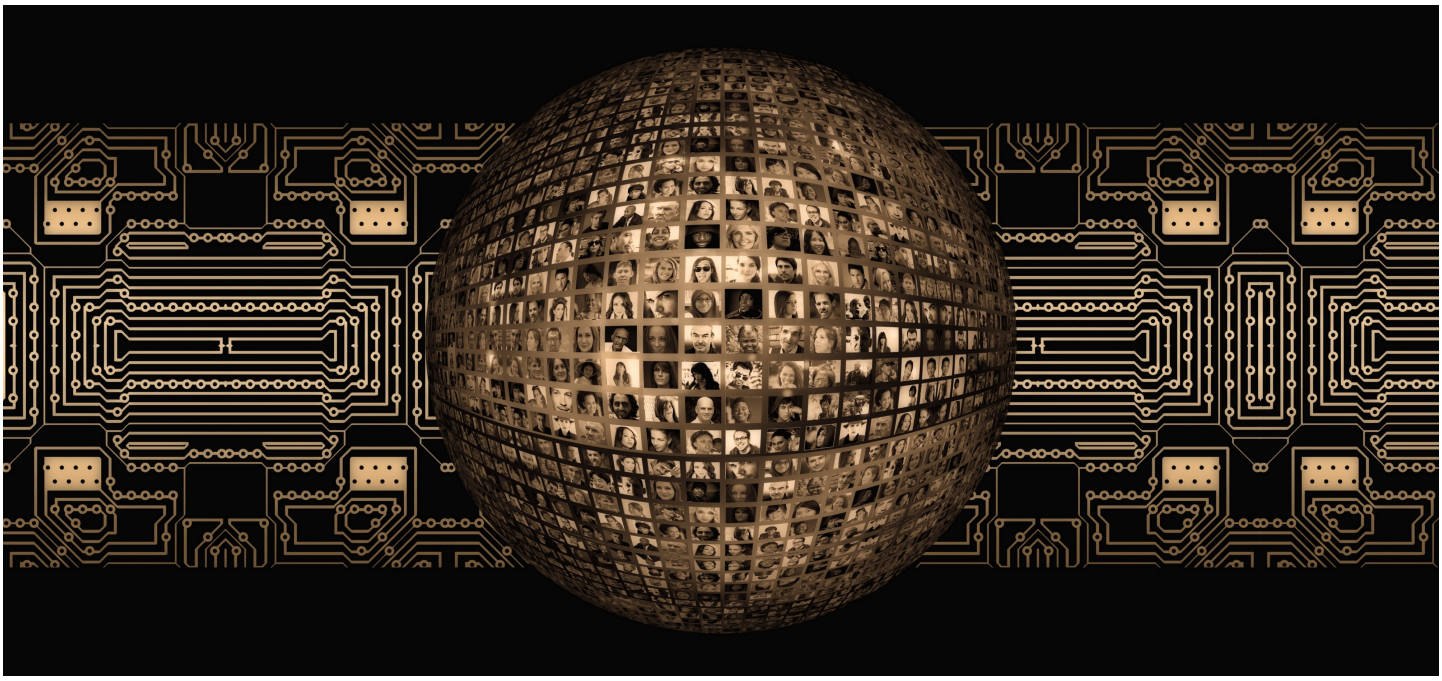


Figure Source: Pixabay

Neural Collaborative Filtering (NCF) is a paper published by National University of Singapore, Columbia University, Shandong University, and Texas A&M University in 2017. It utilizes the flexibility, complexity, and non-linearity of Neural Network to build a recommender system. It proves that Matrix Factorization, a traditional recommender system, is a special case of Neural Collaborative Filtering. In addition, it shows that NCF outperforms the state-of-the-art models in two public datasets. This

article will explain the concept of NCF, and demonstrate how to implement it in Pytorch.

Video version:

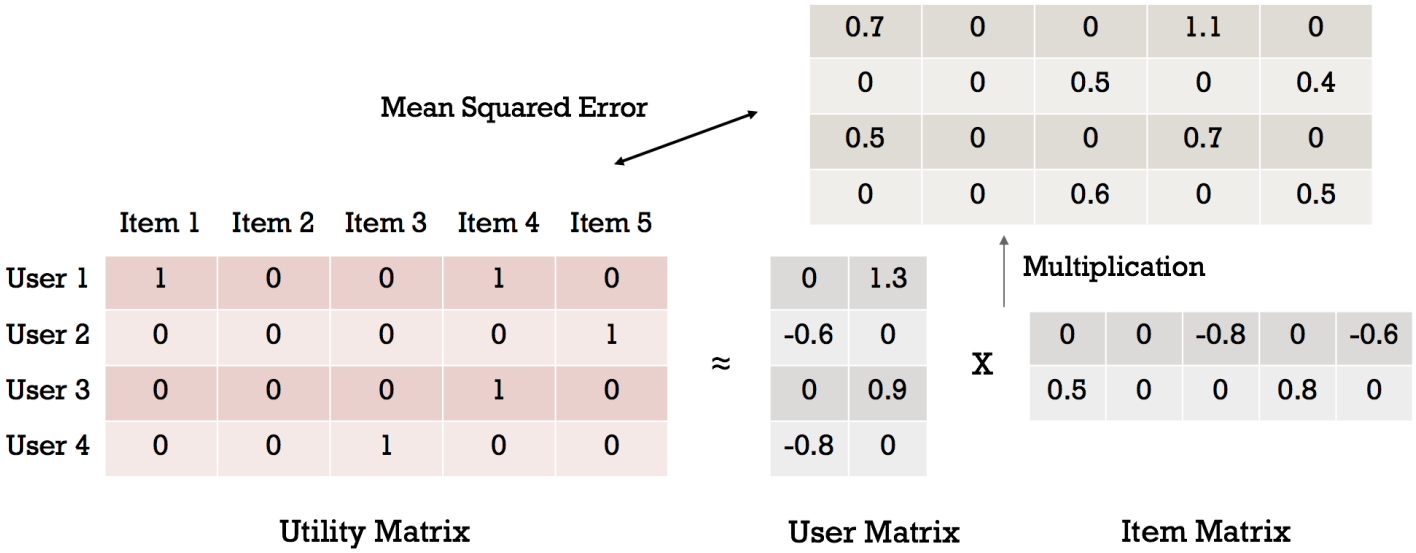
Video

Prerequisite

Before diving into the paper, you should know what a recommender system is, and what are some basic recommender systems. You can read my previous article to quickly equip yourself with the relevant knowledge.

Matrix Factorization

Let's begin with Matrix Factorization. It decomposes the utility matrix into two sub matrices. During prediction, we multiply the two sub-matrices to reconstruct the predicted utility matrix. The utility matrix is factorized such that the loss between the multiplication of these two and the true utility matrix is minimized. One commonly used loss function is mean-squared error.



Essentially, each user and item is projected onto a latent space, represented by a latent vector. The more similar the two latent vectors are, the more related the corresponding users' preference. Since we factorize the utility matrix into the same latent space, we can measure the similarity of any two latent vectors with cosine-similarity or dot product. In fact, the prediction for each user/ item entry is computed by the dot product of the corresponding latent vectors.

$$\hat{y}_{ui} = f(u, i|\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^{K} p_{uk} q_{ik}$$

The prediction equals the inner product of latent vectors

However, the paper argued that dot product limits the expressiveness of user and item latent vectors. Let's consider the following case. We first focus on the top three rows of this utility matrix.

Let S{x,y} denotes the similarity between user x and user y. By computing the cosine-similarity between users 1, 2, and 3, we know that $S\{2, 3\} > S\{1, 2\} > S\{1, 3\}$. Without loss of generality, we map the users onto a latent space of two dimensions as the following.

Now, we take into account user 4. Comparing the similarity with the others, we obtain $S\{1,4\} > S\{3,4\} > S\{2,4\}$. However, no matter we place the latent vector P4 to the right or left of P1, it will inevitably be closer to P2 than P3.

Therefore, this example shows the limitation of inner product to fully model the interactions between user and items in the latent space.

Neural Collaborative Filtering

The paper proposed Neural Collaborative Filtering as shown in the graph below. In the input layer, the user and item are one-hot encoded. Then, they are mapped to the hidden space with embedding layers accordingly. The Neural FC layer can be any kind neuron connections. Multiple layer perceptron, for example, can be placed here. It claims that with the complicated connection and non-linearity in the Neural CF layers,

this model is capable of properly estimating the complex interactions between user and item in the latent space.

NCF architecture

So, how NCF is a generalization of Matrix Factorization? Let me show you in the graph below. We first replace Neural CF layer with a multiplication layer, which performs element-wise multiplication on the two inputs. Then, we set the weight from the multiplication layer to the output layer to be a fixed unit matrix (matrix of all ones) of dimension K by 1 with linear activation function.

Then , we have the following equations.

The prediction of the unobserved interaction $\hat{y}_{ui}$ denotes the predicted value of the (u, i) entry on the reconstructed utility matrix. L is the linear activation function, while $\odot$ denotes the element-wise multiplication operation. p_u and q_i are the latent vector of user and item, respectively, and J is the unit matrix of dimension (K,1). Since J is a unit matrix, inside the linear function becomes the inner product between latent vector p_u and q_i. Furthermore, because the input and output are the same for linear function, it boils down to the last line. The predicted label is the inner product of the corresponding user and item latent vector. This equation is identical to that shown in the Matrix Factorization section. Thus, this proves that Matrix Factorization is a special case of NCF.

NeuMF

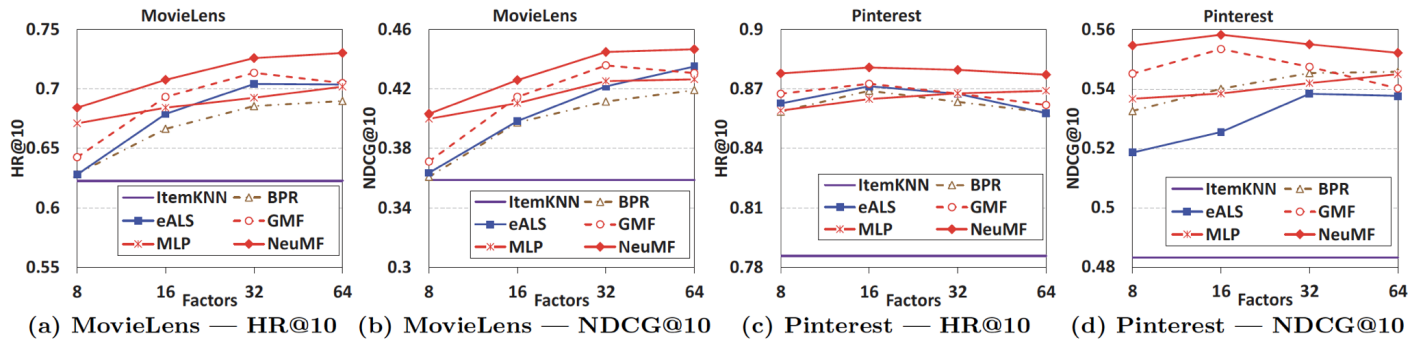
In order to introduce additional non-linearity, the final model proposed, NeuMF, includes a Mutliple-layer Perceptron (MLP) module apart from the Generalized Matrix Factorization (GMP) layer.

The output of GMF and MLP modules are concatenated and connected with the sigmoid activation output layer.

Performance

The paper evaluates NCF models and other models with the Leave-One-Out schema. That is, the last interaction for each user is held out for evaluation. Two evaluation metrics are being considered, Hit Ratio @ 10, and NDCG @ 10. Hit Ratio @ K denotes the fraction of prediction hits given 10 recommendation for each user. Let's say if we recommend 10 items for each user, and 4 of the 10 users interact the items matching our recommendation, then Hit Ratio@ 10 = 0.4. NDCG, on the other hand, can be viewed as an extension of Hit Ratio, except that it considers the order of the hit. This means that if you the hit occurs at the higher recommendation, NDCG will be higher.

The performance comparison is shown in the figure below. In all cases, NeuMF performs better than the other models.



(a) MovieLens — HR@10 (b) MovieLens — NDCG@10 (c) Pinterest — HR@10 (d) Pinterest — NDCG@10

Figure 4: Performance of HR@10 and NDCG@10 *w.r.t.* the number of predictive factors on the two datasets.

In addition, the paper demonstrates the effectiveness of pre-training the individual modules for NeuMF. After training GMF and MLP separately, they set the weight of the trained GMF and MLP as the initialization of NeuMF.

Table 2: Performance of NeuMF with and without pre-training.

	With Pre-training		Without Pre-training	
Factors	HR@10	NDCG@10	HR@10	NDCG@10
MovieLens				
8	0.684	0.403	**0.688**	**0.410**
16	**0.707**	**0.426**	0.696	0.420
32	**0.726**	**0.445**	0.701	0.425
64	**0.730**	**0.447**	0.705	0.426
Pinterest				
8	**0.878**	**0.555**	0.869	0.546
16	**0.880**	**0.558**	0.871	0.547
32	**0.879**	**0.555**	0.870	0.549
64	**0.877**	**0.552**	0.872	0.551

Implementation

In this section, I will show you how to implement NeuMF in Pytorch easily. You only need to specify two functions for each model, the **__init__()** function specifying the model structure, and the **forward()** function defining how the input tensors are being fed forward.

The paper claimed that using separate embedding layer for MLP and GMF yield better performance. Therefore, we define the designated embedding layer for these two modules. In addition, ModuleList is leveraged for building multiple layers of perceptron.

The **forward()** function is rather simple. We just let the user and item indices flow through the network defined in **__init__()**. One special thing to note is that I added Dropout layers at the end of both GMP and MLP module as I found out that this help regularize the network and improve the performance.

Conclusion

You have learned the concept of Neural Collaborative Filtering and how to implement it in Pytorch. Should you have any question, check out the Youtube video attached at the top, or leave your comments below.

Machine Learning Deep Learning Recommender Systems Neural Networks Pytorch