Neural Collaborative Filtering
PyTorch implementation & comparison with Traditional Collaborative Filtering

Rajasvi Vinayak Sharma
UC San Diego
PID: A59012988
rvsharma@ucsd.edu

Abstract

Our project will focus on analyzing and implementing a Neural Collaborative Filtering model from scratch in PyTorch, using fundamentals of Linear Algebra. We then look into the comparison of Neural Collaborative Filtering with other Collaborative Filtering techniques and summarize our findings. Through this process we set out to build Generalized Matrix Factorization, Multi-Layer Perceptron and Neural Matrix Factorization (NMF) model. We shall then use our implementation to perform prediction and comparative evaluation on the MovieLens dataset.

1 Background

1.1 Aim & Importance

Neural Collaborative Filtering (NCF)[1] is a paper published by National University of Singapore, Columbia University, Shandong University, and Texas AM University in 2017. It utilizes the flexibility, complexity, and non-linearity of a 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. In this project we implement three collaborative filtering models: Generalized Matrix Factorization (GMF) [2], Multi-Layer Perceptron (MLP) [3], and Neural Matrix Factorization (NeuMF) [4] in PyTorch, and benchmark their performance on the MovieLens [5] dataset.

1.2 Existing relevant works

Currently there exists an implementation by Microsoft [6] for NCF in Tensorflow. Otherwise, there are no notable implementations of NCF using PyTorch which encompass all 3 models i.e. GMF, MLP and combined NeuMF. Prior to NCF, existing works which employed deep learning for recommendation, primarily used it to model auxiliary information, such as textual descriptions of items and acoustic features of musics. When it comes to model the key factor in collaborative filtering — the interaction between user and item features, they still resorted to matrix factorization and applied an inner product on the latent features of users and items. NCF is generic and can express and generalize matrix factorization under its framework. To supercharge NCF modelling with non-linearities, the original paper proposed to leverage a multi-layer perceptron to learn the user–item interaction function.

1.3 Proposed implementation

We will first train and generate features using GMF; followed by MLP. The ensemble of these two methods yields the NeuMF. The paper then elaborates how to learn NCF with a probabilistic model.

Preprint. Under review.

Finally, it can be shown that MF can be expressed and generalized under NCF. To explore DNNs for collaborative filtering, we then propose an instantiation of NCF, using a MLP to learn the user–item interaction function. Lastly, we present a new neural matrix factorization model, which concatenates MF and MLP under the NCF framework; it unifies the strengths of linearity of MF and non-linearity of MLP for modelling the user–item latent features or embeddings.

2 Proposed Methods

2.1 Preliminaries - Matrix Factorization

Matrix Factorization(MF) associates each user and item with a real-valued vector of latent features and models the two-way interaction of the above two feature vectors. Let $y_{u,i}$ denote the interaction between user u and item i, p_u and q_i denote the embedding vector for user u and item i, respectively; MF estimates an interaction $y_{u,i}$ as the inner product of p_u and q_i:

$$\hat{y}_{u,i} = f(u, i|p_u, q_i) = p_u^T.q_i = \sum_{k=1}^{K} p_{uk} q_{uk} \tag{1}$$

where K denotes the dimension of the embedding space. As such, MF can be deemed as modeling the linear relationship between the embedding vectors.

2.2 Neural Collaborative Filtering

We first present the general NCF framework, elaborating how to learn NCF with a probabilistic model that emphasizes the binary property of implicit data. We then show that MF can be expressed and generalized under NCF. To explore DNNs for collaborative filtering, we then propose an instantiation of NCF, using a multi-layer perceptron (MLP) to learn the user-item interaction function. Lastly, we present a new neural matrix factorization model, which ensembles MF and MLP under the NCF framework; it unites the strengths of linearity of MF and non-linearity of MLP for modelling the user-item latent structures.

2.2.1 General Framework

We use only the identity of a user and an item as the input feature, transforming it to a binarized sparse vector with one-hot encoding.Therefore input layer consists of two feature vectors v_u^U and v_i^I that describe user u and item i, respectively. The user embedding and item embedding are then fed into a multi-layer neural architecture, which we term as $Neural\ collaborative\ filtering$ layers, to map the latent vectors to prediction scores. Each layer of the neural CF layers can be customized to discover certain latent structures of user-item interactions.
The dimension of the last hidden layer X determines the model's capability. The final output layer is the predicted score $\hat{y}_{ui}$, and training is performed by minimizing the pointwise loss between $\hat{y}_{ui}$ and its target value y_{ui}.
NCF's predictive models is defined as:

$$\hat{y}_{u,i} = f(P^T v_u^U, Q^T v_i^I|P, Q, \theta_f) \tag{2}$$

where $\mathbf{P} \in \mathbb{R}^{M \times K}$ and $\mathbf{Q} \in \mathbb{R}^{N \times K}$, denoting latent factor matrix for users and items, respectively; and θ_f denotes the model parameters of the interaction function f. Since the function f is defined as a multi-layer neural network, it can be formulated as:

$$f(P^T, v_u^U, Q^T, v_i^I|P, Q, \theta_f) = \phi_{out}(\phi_X(...\phi_2(\phi_1(P^T v_u^U, Q^T v_i^I))...)) \tag{3}$$

where ϕ_{out} and ϕ_x respectively denote the mapping function for the output layer and x-th neural collaborative filtering (CF) layer, and there are X neural CF layers in total.
To learn model parameters, existing pointwise methods largely perform a regression with squared loss:

$$L_{sqr} = \sum_{(u,i) \in \mathcal{Y} \cup \mathcal{Y}^-} w_{ui} (y_{ui} - \hat{y}_{ui})^2 \tag{4}$$

where $\mathcal{Y}$ denoted set of observed interactions in $\mathcal{Y}^-$ denotes the set of negative instances, which can be all (or sampled from) unobserved interactions; and w_{ui} is a hyperparameter denoting the weight of training instance (u, i). Due to one-class nature of implicit feedback we can easily use probabilistic function (eg. Logistic or Probit function) as a activation function for output layer ϕ_{out}. With this in mind logarithm loss can be written as,

$$L = - \sum_{(u,i) \in \mathcal{Y} \cup \mathcal{Y}^-} y_{ui} log \hat{y}_{ui} + (1 - y_{ui}) log(1 - \hat{y}_{ui}) \tag{5}$$

This is the objective function to minimize for the NCF methods, and its optimization can be done by performing stochastic gradient descent (SGD).

2.2.2 Generalized Matrix Factorization (GMF)

We now show how MF can be interpreted as a special case of our NCF framework. Due to the one-hot encoding of user (item) ID of the input layer, the obtained embedding vector can be seen as the latent vector of user (item). Let the user latent vector p_u be $P^T v_u^U$ and item latent vector q_i be $Q^T v_i^I$. We define the mapping function of the first neural CF layer as

$$\phi_1(p_u, q_i) = p_u \odot q_i \tag{6}$$

where $\odot$ denotes the element-wise product of vectors. We then project the vector to the output layer:

$$\hat{y}_{ui} = a_{out}(h^T(p_u \odot q_i)) \tag{7}$$

where a_{out} and h denote the activation function and edge weights of the output layer, respectively.

Intuitively, if we use an identity function for a_{out} and enforce h to be a uniform vector of 1, we can exactly recover the MF model. Under the NCF framework, MF can be easily generalized and extended. For example, if we allow 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. And if we use a non-linear function for a_{out}, it will generalize MF to a non-linear setting which might be more expressive than the linear MF model.

2.2.3 Multi-Layer Perceptron (MLP)

Since NCF adopts two pathways to model users and items, it is intuitive to combine the features of two pathways by concatenating them. We propose to add hidden layers on the concatenated vector, using a standard MLP to learn the interaction between user and item latent features. In this sense, we can endow the model a large level of flexibility and non-linearity to learn the interactions between p_u and q_i, rather than the way of GMF that uses only a fixed element-wise product on them. More precisely, the MLP model under our NCF framework is defined as:

$$\begin{aligned} z_1 &= \phi_1(p_u, q_i) = \begin{bmatrix} p_u \\ q_i \end{bmatrix}, \\ \phi_2(z_1) &= a_2(W_2^T z_1 + b_2), \\ &\ldots\ldots \\ \phi_L(z_{L-1}) &= a_L(W_L^T z_{L-1} + b_L), \\ \hat{y}_{ui} &= \sigma(h^T \phi_L(z_{L-1})), \end{aligned} \tag{8}$$

where W_x, b_x, and a_x denote the weight matrix, bias vector, and activation function for the x-th layer's perceptron, respectively.

2.2.4 NeuMF (GMF+MLP)

We let GMF and MLP share the same embedding layer, and then combine the outputs of their interaction functions. This way shares a similar spirit with the well-known Neural Tensor Network (NTN). Specifically, the model for combining GMF with a one-layer MLP can be formulated as,

$$\hat{y}_{ui} = \sigma(h^T a(p_u \odot q_i + W \begin{bmatrix} p_u \\ q_i \end{bmatrix} + b)) \tag{9}$$

To provide more flexibility to the fused model, we allow GMF and MLP to learn separate embeddings, and combine the two models by concatenating their last hidden layer. Figure 11 illustrates our proposal, the formulation of which is given as follows,

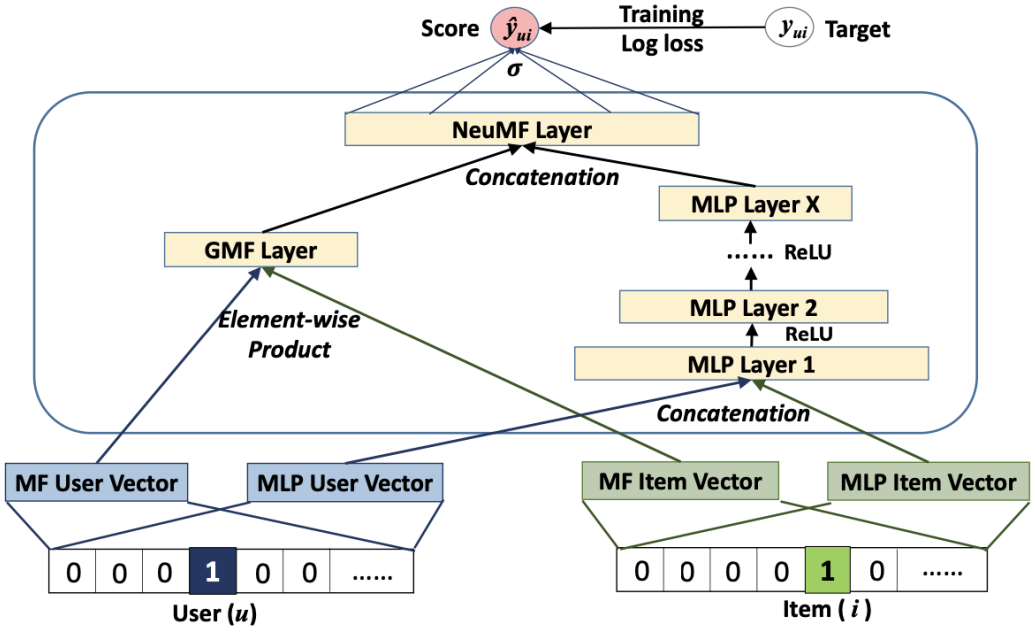

Figure 1: NeuMF architecture

$$\begin{aligned} \phi^{GMF} &= p_u^G \odot q_i^G, \\ \phi^{MLP} &= a_L(W_L^T(a_{L-1}(...a_2(W_2^T \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix} + b_2)...)) + b_L), \\ \hat{y}_{ui} &= \sigma(h^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix}) \end{aligned} \tag{10}$$

where p_u^G and p_u^M denote the user embedding for GMF and MLP parts, respectively; and similar notations of q_i^G and q_i^M for item embeddings.This model combines the linearity of MF and non-linearity of DNNs for modelling user-item latent structures. We dub this model "NeuMF", short for $NeuralMatrixFactorization$.

3 Experiments

We plan to explore main question through our experiments:

1. How does NCF perform better as compared to other implicit collaborative filtering methods i.e. Generalized Matrix Factorization (GMF) and Multi-Layer Perceptron (MLP)?

3.1 Setup

Base Dataset: The experiments are conducted on the MovieLens dataset. This movie rating dataset has been widely used to evaluate collaborative filtering algorithms. It consists of 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens

in 2000. It comprises of User information such as gender, age, and occupation, as well as Movie information such as titles and genre. For our experiment we need user, ratings columns since we are solving for implicit feedback data. While the dataset consists of explicit feedback data, we have modified it to implicit data, where each entry is marked as 0 or 1 indicating whether the user has rated the item.

Negative Sampling: Since base data has only positive user-rating interactions (positive feedback) we generated 4 negative samples per positive interaction which is randomly sampled from set of movies a user didn't interact with.

3.2 Implementation Details

We will utilize PyTorch in our implementation. It is an open source machine learning framework invented by Meta (formerly Facebook) that allows high flexibility in terms of research as it is built on the concept of dynamic graphs made in realtime (unlike TensorFlow).

Baseline (GMF/MF): GMF class utilises torch embedding layers along with dynamic capability to control GMF layer as pure MF model using MF (i.e. unit activation function and linear layer). Furthermore to perform various experiments, we have given capability to set the "number of latent factors" for user/rating embeddings so we can compare the effect of embedding size over our model.

MLP: Our Multi-Layered Perceptron (MLP) class utilises concatenated torch embedding layers followed by a 3 layered neural network (ReLU as activation function) and sigmoid for prediction. Here, the predictive final layer corresponds to the "latent factors" of embedding which can be chosen while defining model as a parameter (similar to GMF class).

NeuMF: Our Neural Matrix factorization (NeuMF) class utilises pre-trained layers loaded from GMF and MLP models of equivalent latent factors. We only load the embedding layer from pre-trained models using state_dict in PyTorch. Apart from the capability to control latent factors as a parameter we have also added a parameter α which controls the relative weight given to GMF (α) and MLP ($1 - \alpha$) in final predictive layer after concatenation.

3.3 Evaluation

To evaluate the performance of item recommendation, the leave-one-out evaluation approach is used. For each user, the latest interaction is held-out as the test set and the remaining data is utilized for training. Since it is too time-consuming to rank all items for every user during evaluation, we followed the common strategy that randomly samples 100 items that are not interacted by the user, ranking the test item among the 100 items. The performance of a is measured by Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG) [7] primarily HR@10 and HDCG@10 where we truncated the ranked list at 10 for both metrics.

Parameter Settings: We implement above mentioned methods based on Pytorch. For hyper-parameters tuning, we randomly sampled one interaction for each user and used it as the validation set. All the models are learnt by optimizing the log loss of Equation 5. We will randomly initialize model parameters with a Gaussian distribution (with a mean of 0 and standard deviation of 0.01) for the GMF and MLP models that are to be trained from scratch, with Adam optimizer. For NeuMF we use the pretrained GMF and MLP models with optimizer as SGD. We did not use Adam here as it utilizes momentum to properly update parameters, but since we have used pretrained models (without saving their momentum) we cannot use the fast convergence property of Adam properly.

We iterate through a batch size of [128, 256, 512, 1024], and the learning rate of [0.001, 0.005,0.01 ,0.05] for hyperparameter analysis. Since the last hidden layer of NCF determines the model capability, we term it as predictive factors and evaluated the factors of [8; 16; 32; 64], where large factors may cause overfitting and degrade the performance. We tested with three hidden layers for MLP; for example, if the size of predictive factors is 8, then the architecture of the neural CF layers is $32 \rightarrow 16 \rightarrow 8$, and the embedding size is 16. Finally we use the Binary Cross Entropy loss for all our training.

3.4 Performance Comparison

(1) How does NCF perform better as compared to other implicit collaborative filtering methods i.e. Generalized Matrix Factorization (GMF) and Multi-Layer Percpetron (MLP)?

We ran the training for 10 epochs per model across different factors and as seen from Fig.2 it can be seen how with epochs the training loss converges. As training loss reaches convergence it can be seen that all model metrics reach convergence too. NeuMF is best model for factors = [8,16,32] whereas for factor=64 it is second best which can be accounted for overfitting as latent factors are high in number. Furthermore, it can also be observed that baseline GMF model underperforms for most factors (except 8 where it reaches close to NeuMF). Overall NeuMF can be declared as the best with MLP as second-best (better than NeuMF for 64 factors).

As a brief summary it can be seen from Fig. 3 that final metrics for NeuMF and MLP are better than GMF across all factors [8,16,32,64]. For large latent factors, NeuMF and GMF have substantially low metric values which can be due to overfitting. Optimum number of factors for achieving highest performance is 16 for which NeuMF showcases HR@10=0.575, and NDCG@10 = 0.261.

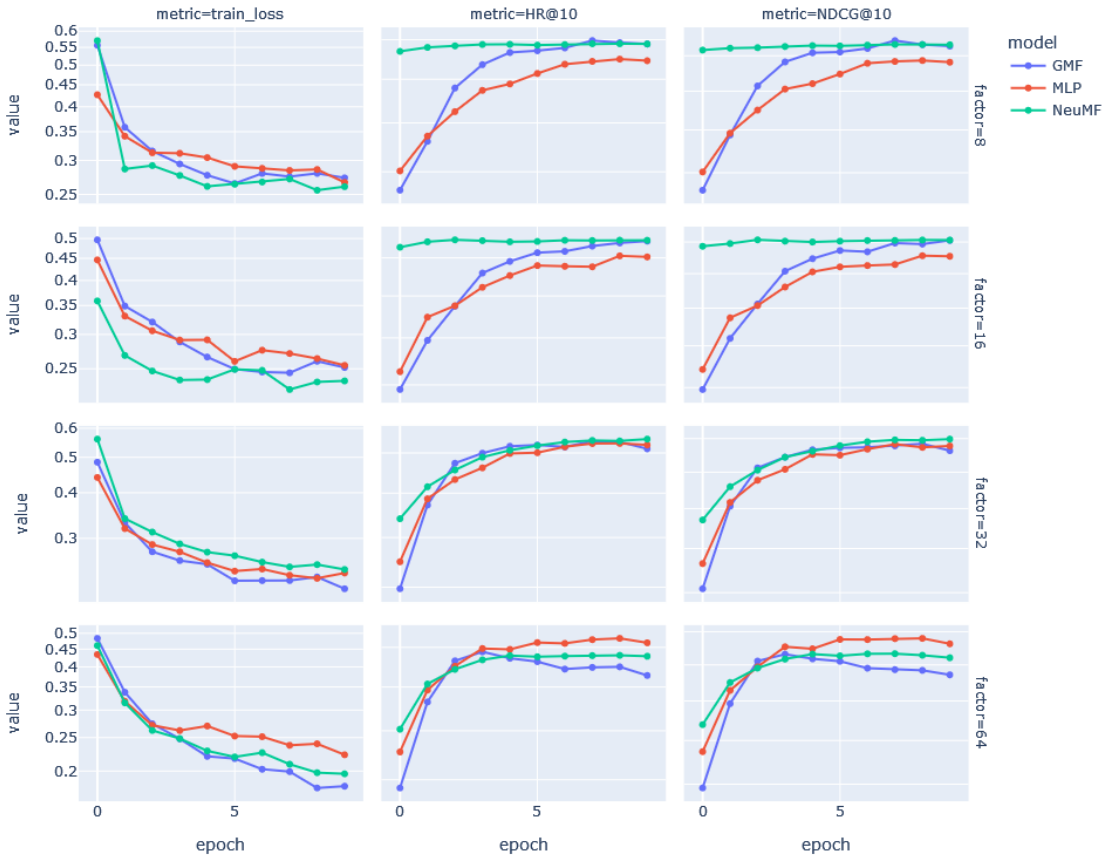

Figure 2: Epoch wise Training loss and Performance of HR@10 & NDCG@10 w.r.t number of predictive factors on Movie Lens dataset.

Performance of GMF, MLP and NeuMF

	GMF		MLP		NeuMF	
Factors	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
8	0.544	0.246	0.523	0.236	0.544	0.247
16	0.574	0.261	0.551	0.251	0.575	0.261
32	0.555	0.253	0.560	0.255	0.568	0.260
64	0.514	0.234	0.556	0.253	0.538	0.244

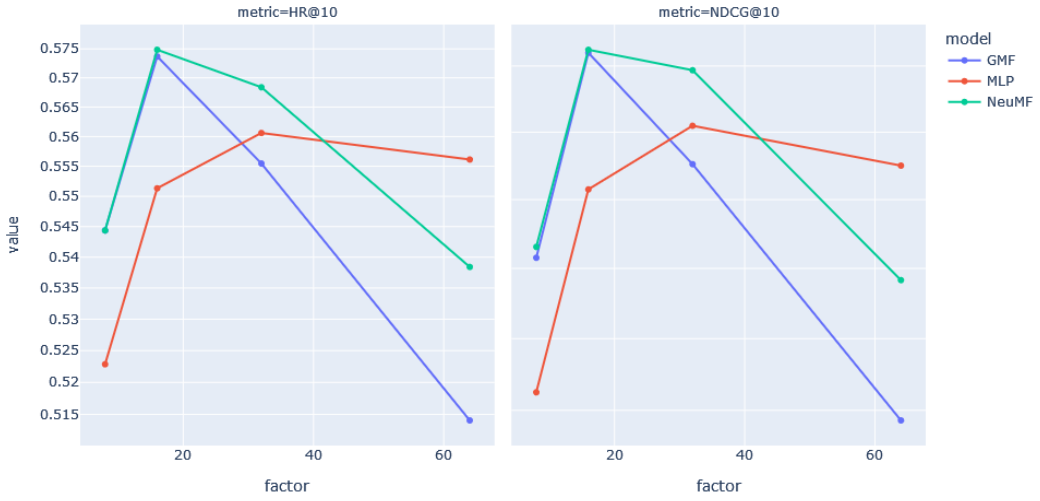

Figure 3: Performance metrics HR@10 and NDCG@10 w.r.t no. of predictive factors = [8,16,32,64]

4 Conclusion

In this paper we have evaluated the performance of Neural Collaborative Filtering in comparison to Collaborative Filtering techniques such as GMF and MLP. While GMF is a generalized version of the traditional Matrix Factorization technique, we can clearly see that addition of neural network brings a dynamic decision making process into the field of recommender systems and proved better than traditional MF. Neural CF models outperformed GMF by a margin across all factors which was clear from our experiments.

Primary motive of our project was the implementation of all Neural CF models and GMF using PyTorch which we have provided separately. Furthermore our code provides flexibility to test various datasets, batch-sizes and includes capability to vary important hyper-parameters like negative samples, number of factors, relative weight of Neural CF models w.r.t. GMF (α). It supports ability to oversample negative feedback samples using parameter - number of negative interactions per positive interaction. In future works, we wish to explore other Neural Collaborative Filtering architectures, expand current models to explicit feedback datasets, and adding more evaluation metrics.

References

[1] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[2] H. Shan and A. Banerjee, "Generalized probabilistic matrix factorizations for collaborative filtering," in *2010 IEEE International Conference on Data Mining*. IEEE, 2010, pp. 1025–1030.

[3] D. W. Ruck, S. K. Rogers, and M. Kabrisky, "Feature selection using a multilayer perceptron," *Journal of Neural Network Computing*, vol. 2, no. 2, pp. 40–48, 1990.

[4] G. K. Dziugaite and D. M. Roy, "Neural network matrix factorization," *arXiv preprint arXiv:1511.06443*, 2015.

[5] "Movie lens dataset," https://www.kaggle.com/odedgolden/movielens-1m-dataset, accessed: 2022-01-22.

[6] "Microsoft tf implementation," https://github.com/microsoft/recommenders, accessed: 2022-01-22.

[7] X. He, T. Chen, M.-Y. Kan, and X. Chen, "Trirank: Review-aware explainable recommendation by modeling aspects," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 1661–1670.