IAAML: Performance Analysis of Neural Collaborative Filtering (NCF) Architecture

Michał Filipiuk
University of Warsaw
Ludwig-Maximilian-University Munich
Michal.Filipiuk@campus.lmu.de

Bagrat Ter-Akopyan
Ludwig-Maximilian-University Munich
bagrat.terakopyan@campus.lmu.de

Abstract

In this paper, as the part of Infrastructure for Advanced Analytics and Machine Learning final project, we would like to analyze how a deep learning model like Neural Collaborative Filtering performs, when trained on Central Processing Unit with a small number of cores and how their number would influence the training process. Besides the introduction to problem definition as well as explaining the fundamentals of collaborative filtering, leaving out the details of the math, we decided to focus on analyzing the performance regarding strong and weak scaling.

1. Introduction

This paper comes to life as part of the "Infrastructure for Advanced Analytics and Machine Learning" course. The aim of this paper is to showcase the learned techniques to train any machine learning model under various configurations and analytically evaluate, compare and visualize the models' performance regarding the these configurations as well as the achieved results, respectively. For this purpose we decided to focus on the already existing system for collaborative filtering - more precisely the Neural Collaborative Filtering [5].

In the following we provide a detailed explanation of the problem of collaborative filtering as well as the fundamental difference between two kinds of feedback that is essentially important for any recommendation system based on collaborative filtering.

We then take a closer look on the used model architecture and provide basic mathematical representation of the model emphasizing the key points of the system and its' benefits. Afterwards we describe the data the model is trained on and provide two other possible data sets to show the scope of the applied model.

Finally, we define the performance metrics we are going to evaluate the model and depict the architecture infrastructure we train the model on. After describing the experimental settings we perform experiments on previously described real-world dataset and provide a detailed performance analysis including the visualizations of the results regarding the performance metrics.

1.1. Problem Overview

In this section we first formalize our motivation regarding the problem and discuss the problems of collaborative filtering, followed by pointing out the difference between implicit and explicit feedback.

1.1.1 Motivation for choosing the problem

We decided to showcase a neural network solving a recommendation problem as it is an area of deep learning which is not that commonly known by machine learning enthusiasts like e.g. natural language processing or computer vision, although it may be much closer related to our everyday lives: almost everywhere on the Internet, we can find some kind of recommendation functionality provided on the website: Google ads, products suggestions on Amazon, next videos to watch on YouTube, new songs on Spotify etc.

Neural Collaborative Filtering network is a very lightweight architecture that was one of the first deep learning approaches in this area; precisely to solve an problem called collaborative filtering. For a few years, it was also featured in MLPerf[9] competition which gathers the best deep learning engineering teams from companies like NVIDIA, Google or Intel, where they run neural networks on their own dedicated hardware. However, this year it was replaced with a much bigger neural network called Deep Learning Recommendation Model[10] created by Facebook AI team with roughly 5 000 times bigger dataset[3], as it is more similar to real-life problems in the industry. Obviously we would like much more to present you DLRM performance, but it would require us to get an access to much better compute powers (this model is so big, that it does not even fit on a single GPU without very strong restriction on

the data – even on the latest NVIDIA A100 GPU with 40GB of memory on board. On the other hand, you can easily fit NCF with a big chunk of its dataset on a single V100 GPU.)

1.1.2 Collaborative filtering – Problem definition

Collaborative filtering can be simply described as a method of making automatic predictions about the interests of a user by collecting preferences or taste information from many users. [1]

Collaborative filtering is only one of a few possible approaches to the problem of recommendation. The others worth mentioning are content-based recommenders, where decisions are taken based on information about items and users, and hybrid models, taking advantage of both approaches.

1.1.3 Implicit vs. explicit feedback

Since collaborative filtering can be applied on two types of feedback, explicit and implicit, we want to clarify the difference between those.

The explicit feedback, used in works like [7][14], include i.e. ratings, reviews, likes, pins, etc. This kind of feedback offers the possibility to automatically measure users satisfaction on specific items, which in following can be used to improve the whole recommendation system.

In comparison to the above, implicit feedback, used in [5], is tracked automatically just based on the interaction between users and items. It makes it easier to collect data, even though it turns out that implicit feedback makes it much more difficult to quantify users satisfaction as well as to collect negative feedback.

1.2. Model Architecture

In the original NCF paper[5] two advanced techniques are combined to develop the new neural matrix factorization model:

- Matrix factorization (MF)[6][7]:
  over the past years MF has become the most popular technique for modelling personalized recommender systems - predicting users' future behaviour based on their past interactions with items (e.g. likes, clicks). Therefore users and items are projected into a shared latent space using a vectorized latent feature representation. The performance of such a system is strongly depending on the choice of the interaction function to model the latent feature interactions between users and items.

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

  For the sake of completeness we provide the predictive model of MF in (1), but highly recommend to take a look at the NCF paper [5] to completely understand the formula and the limitations of MF. The NCF framework allows to generalize and to extend the original MF calling it Generalized Matrix Factorization (GMF). By defining the user latent vector $\mathbf{p}_u$ as $\mathbf{P}^T \mathbf{v}_u^U$ and the item latent vector $\mathbf{q}_i$ as $\mathbf{Q}^T \mathbf{v}_i^I$, the mapping function of the first CF layer is defined as

$$\phi_1(p_u, q_i) = p_u \odot q_i, \quad (2)$$

  where the two vectors p_u and q_i are computed element-wise. The output layer then is defined as

$$\hat{y}_{ui} = a_{out}(h^T(p_u \odot q_i)), \quad (3)$$

  with a_{out} and h^T being the activation function and edge weights of the output layer. Due to the fact that h can be learnt from data and that it is possible to use a non-linear activation function for a_{out} to overcome the traditional MF linear setting, the provided model above can be seen as the generalization of the MF and can lead to more expressive results.

- Multi-Layer Perceptron (MLP):
  in contrast to widely adopted multimodal deep learning work [15][12], where combining the features of users and items are achieved by just concatenating the vectors, [5] proposes to use standard MLP to learn the interaction between user and item latent features. The MLP mode in the original NCF framework paper [5] is defined in the following way:

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

  with W_x, b_x, a_x denoting the weight matrix, bias vector and the activation function for x-th layer. For more information on MLP in context of NCF and detailled information on choosing the activation function please refer to the original paper [5].

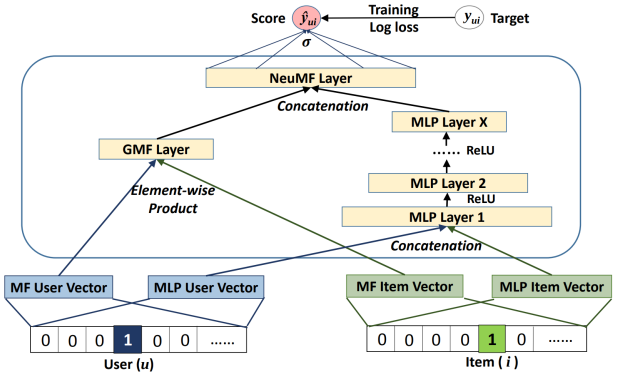

Figure 1. Neural Collaborative Filtering architecture overview. Diagram taken from NCF paper [5]

After mentioning the two key points of the NCF model, it is important to understand how these two instantiations can be fused, to achieve better modelling of the user-item interactions. The Figure 1 illustrates the fused model and how exactly GMF and MLP are combined. First of all GMF and MLP learn separate embeddings and then the models are combined by concatenating their last hidden layer of each part. For detailed mathematical formulation please refer to the original paper [5].

1.3. MovieLens-20M

The dataset that we feature in our project is MovieLens-20M[4]. It contains around 20 million user-item ratings, coming from a movie recommendation service called MovieLens, which is a scientific project led by researchers from University of Minnesota.

Except of just ratings, it contains also basic information about the rated movies, their genres and tags. However in a collaborative filtering approach, we do not use any other data than ratings.

The ratings are spanning from 1 to 5 stars and the dataset contains 20 000 263 of them, given by 138 493 unique users during the period between January 09, 1995 and March 31, 2015.

NCF model is created to work on implicit feedback data, however this dataset is obviously of the feedback that is explicit. Although these two types of data seem to be totally different, we do not treat them as such – the only difference for us are values of data: in implicit we naturally have only two possible values, in explicit like movie ratings we have continuous values. So what we do during dataset preprocessing we just set values to one if they exists (so if user had rated some movie) and missing values are treated as zeros. Depending on approach, this step may vary: other idea is e.g. to set a threshold for values: values greater or equal are set to 1, the rest are set to 0. You can find it in other paper on collaborative filtering called Variational autoencoders for collaborative filtering [8].

1.3.1 Other datasets

The other available dataset that was used in the original NCF paper [5] is the Pinterest dataset[1]. Because of the sparse nature of the original dataset, the data is first filtered retaining only users with at least 20 interactions (pins), resulting in a subset with 55,187 users and 1,500,809 interactions, denoting whether the user has pinned the image.

Another dataset, which suprisingly is not featured in NCF paper is a Netflix dataset, released during probably one of the first data science competition called Netflix Prize in 2006. It is main price was 1 000 000$, which even nowadays is one of the biggest prizes ever. It is the biggest existing dataset of ratings (There exists MovieLens-1B, however it is a synthetically enlarged dataset), containing over 100 000 000 ratings of roughly 18 000 movies, given by almost 500 000 users.

1.4. Metric and evaluation of the model

Evaluating models using such data is a quite tricky as there is no way of simply splitting dataset into training and testing subset as you cannot recommendation for a user, that has never been seen by the model (that's require the model to be retrained with every new user what definitely is a drawback of this approach). To overcome this problem, the authors decided to adopt a *leave-one-out* evaluation: for every user in the dataset, one item that he interacted with is being held out to be used during evaluation. In evaluation step, the one positive sample is being mixed with 100 negative samples (random items that our user didn't interact with) to create a ranking that would be a subject for metrics.

1.4.1 Hit Ratio @ K

Hit Ratio @ K (also known as Recall @ K) is a very simple metric – it returns a number of positive samples in the top K rated items. In our case, naturally the highest possible value is 1 as we have only one positive sample, however it's not a problem to adapt it to multiple positive items. One clear drawback that this metric has, is a fact that it's not able to differentiate between cases where the positive sample is on different positions in top K items.

$$HR@K = \frac{|\{relevant\ items\} \cap \{top\ K\ items\}|}{min(|\{relevant\ items\}|, K)}$$

The formula shown above is a general formula for Hit Ratio, which would also work for a case with multiple positive items.

1.4.2 Normalized Discounted Cumulative Gain @ K

Normalized Discounted Cumulative Gain @ K (NDCG@K) is a metric that fixes a shortcoming of

[1] https://sites.google.com/site/xueatalphabeta/academic-projects

Hit Ratio of not differentiating between positions in top K selected items. Each position in top K position has a weight associated with it, which decreases as an position increases. The weight that's used in the metric is $\frac{1}{log_2(item\ position+1)}$, however depending on our needs we can change this function to any that would be monotonously decreasing. The final value is a sum of the weight of relevant items in top K selected items.

The metric defined that way would have unfortunately one drawback unpleasant to work with: its range of values is based on the weights used and number of all positive samples. Of course it does not prohibit us to use it, in such form we call it Discounted Cumulative Gain @ K (rel_i takes value of 1 if the item at index i is relevant for us, otherwise 0).

$$DCG@K = \sum_{i=1}^{K} \frac{rel_i}{log_2(i+1)}$$

What we have to do to solve this issue is to normalize the output and we will do it by dividing by the maximal value that this metric can achieve. We can easily get this value by assuming a perfect ordering by our model (on the top we have all the relevant items and then irrelevant) and calculating the value of Discounted Cumulative Gain.

$$iDCG@K = \sum_{i=1}^{min(K,|\{relevant\ items\}|)} \frac{1}{log_2(i+1)}$$

$$NDCG@K = \frac{DCG@K}{iDCG@K}$$

1.5. Other approaches

However these days, due to limitations inherited from the collaborative filtering approach, companies are moving towards models that are able to process more heterogeneous data, containing different number of features, both categorical and continuous like in DLRM[10].

2. Workplace setup

2.1. Infrastructure

To run all the experiments, we created a VM in LRZ Compute Cloud. We used Ubuntu (precisely 16.04) as our operating system of choice as it's used widely throughout the ML community, even on the most powerful servers like DGX A100. It is also the system that we are most used to. The VM size was `xlarge` featuring 10 cores and 45GB of RAM as it was the biggest that we could use at the moment at LRZ CC. To track the results of trainings in different runs, we decided to use MLflow[13], which is a tool for managing machine learning models lifecycle. It gives us also an opportunity to quickly compare the results of different runs without exporting any data to python to be analyzed and visualised.

2.2. Code

Unfortunately, we did not have that much time to make this project, so we could not implement the model on our own. We got the implementation from NVIDIA's repository[2] in PyTorch[11], where you can find state-of-the-art implementations of multiple neural networks in different frameworks, and we have adapted it to our infrastructure: we added MLflow tracking, support for CPU training and arguments for running scripts with different number of threads.

3. Experiments

In this section, we describe our experiments with the aim of evaluating the performance of the model regarding the aspects like training time, speedup and efficiency depending on number of threads used for the training.

We first present the experimental settings, followed by analyzing and visualizing the above questions.

3.1. Experimental Settings

3.1.1 Strong scaling

To measure how strong scaling performs for this model, we decided to run a single epoch (training + evaluation) for different numbers of cores, from 1 to 10. The metric that we've been measuring was a total time of training.

3.1.2 Weak scaling

In case of weak scaling we have had 10 runs of training for each core count with a corresponding number of epochs. The metric that we've been measuring was a total time of training.

3.2. Performance Comparison

3.2.1 Strong scaling

The total runtime (total time of training) can be seen as a function depending on number of threads: $T(n) = s + \frac{p}{n}$, where p denotes the parallel part and s the sequential part of the software, that is not amenable to parallelization. Based on this fact, the speedup of the software is limited by the serial part. This knowledge is better known as Amdahls' Law. This is what we call strong scaling.

In Figure 2 we can observe the decreasing total training time with increasing number of cores used for the training. It definitely looks as expected, however human minds are the best in seeing linear dependencies (here, we can say that decreases and approaches some value but nothing more), so

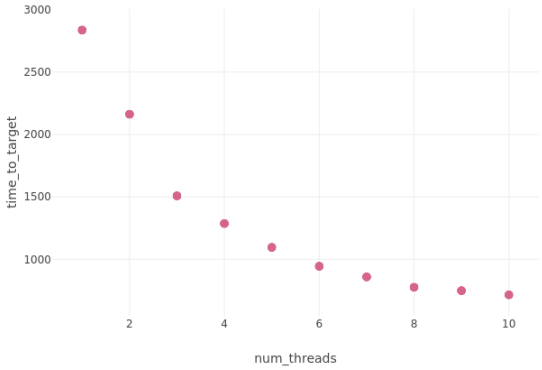

Figure 2. Time to target: T(1) = 2836.4, T(10) = 716

we will change the plot a bit and instead of plotting raw time, we will plot speedup (time taken by a single core divided by time taken by n cores) of our setup.

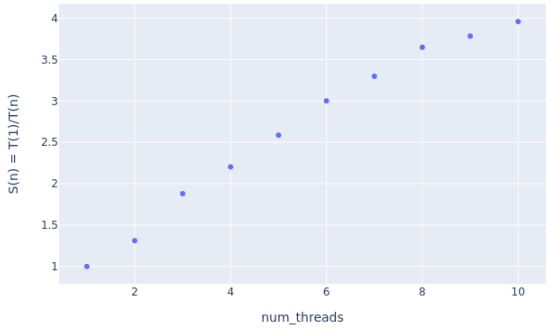

Figure 3. Speedup plot

Ok, now we have a plot with a clearly linear relationship, at least in the beginning, as in the end it's plateauing, what should go perfectly with Amdahl's law – let's see on a plot how perfectly (the values of s and p for Amdahl's law were estimated by minimizing LSE):

This plot is really intriguing. Due to the fact that Amdahl's Law measures the theoretical maximum speed up, that in practice is almost never achieved, we intuitively were expecting the red line in Figure 4 being completely above the measurements from the speedup.

The last step will be taking a look at efficiency, however you may already know what to expect. It's not really impressive and honestly, we expected more from such a deep learning framework.

3.2.2 Weak scaling

The problem with weak scaling in deep learning is a very iterative definition of the problem. The only way of scaling

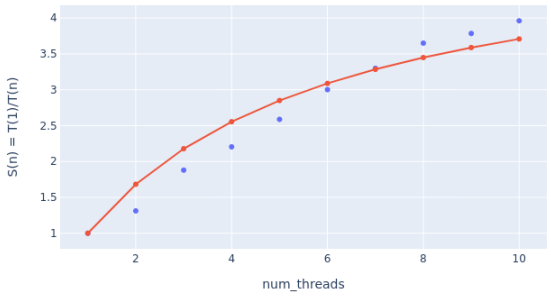

Figure 4. Amdahl's law

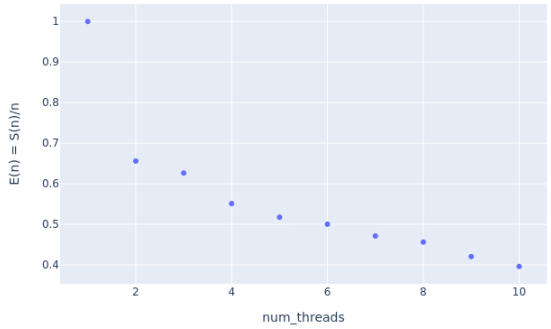

Figure 5. Efficiency plot

the problem is by increasing a number of epoch which is basically a sequential increasing of problem (it's the same case like with scaling horizontally and vertically, if we want to keep the neural network the same, then it can only be scaled vertically). So if we increase number of threads, then they firstly have to deal with the first epoch to move to the next one – the same as it is going in the experiment for the strong scaling. Nevertheless, take a look at the plot of times:

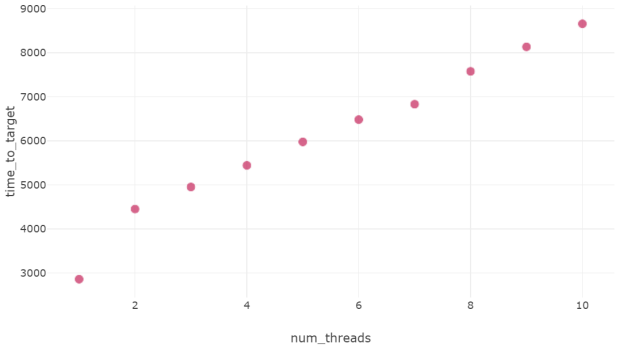

Figure 6. Weak scaling test (number of epochs = number of cores)

Basically as expected, according to Amdahl's law, each step should take $n \times s + p$, as every epoch is sequential (we have n epoch, each of them takes $s + \frac{p}{n}$ time). That means the interval between consecutive runs should be equal to s.

If you cannot get better results with a problem, try to change it – that's a very common idea in some fields of mathematics like numerical methods and that's exactly that we will try to do: the problem before was very sequential. However changing a batch size value should give us an opportunity to scale the problem horizontally, giving more operation to be calculated in parallel. Nonetheless we should not treat it as a perfect solution to this problem, changing batch size totally changes the way how a neural network is learning, so such change would require us to revisit every hyperparameter and there still isn't a guarantee that we will be able to achieve the same results as before. We are just showcasing it as a trivia on how changing the size of single math operations like matrix multiplication or addition would change time spent by CPU to compute it.

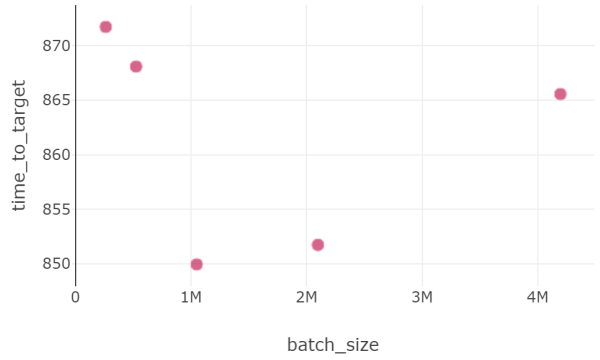

Figure 7. Time taken to train with regard to batch size

We didn't unfortunately have time to run it multiple time, but in our opinion it looks like a noise – on CPU size of operations doesn't change time taken by fixed number of CPU cores.

4. Conclusion

In this work, we analyzed the performance of the neural network architectures for collaborative filtering in the context of training time, speedup and efficiency. First of all we talked about the general problem of recommendation systems and the difference between the explicit and implicit feedback that is used for developing such a system. We then granularly described the model architecture of NCF pointing out the features of this model in comparison to traditional collaborative filtering models. We mentioned the data that we have used to train the model and gave an overview on two other datasets that can be used. Due to time constraints and requirement of the course this work is based on already existed code and primary focuses on evaluating the system. For this purpose we described the infrastructure of the underlying system the code was deployed on and defined metrics for the evaluation of the system as well as the metrics for performance comparison.

References

[1] https://en.wikipedia.org/wiki/collaborative_filtering.

[2] https://github.com/nvidia/deeplearningexamples/.

[3] https://labs.criteo.com/2015/03/criteo-releases-its-new-dataset/.

[4] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), Dec. 2015.

[5] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. *CoRR*, abs/1708.05031, 2017.

[6] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558, 2016.

[7] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008.

[8] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering, 2018.

[9] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G. Wei, and C. Wu. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

[10] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[12] Nitish Srivastava and Russ R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.

[13] M. Zaharia, Andrew Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, Siddharth Murching, Tomas Nykodym, P. Ogilvie, Mani Parkhe, F. Xie, and Corey Zumar. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41:39–45, 2018.

[14] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. Discrete collaborative filtering. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 325–334, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Hanwang Zhang, Yang Yang, Huanbo Luan, Shuicheng Yang, and Tat-Seng Chua. Start from scratch: Towards automatically identifying, modeling, and naming visual attributes. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 187–196, 2014.