Neural Collaborative Filtering vs. Matrix Factorization Revisited

Steffen Rendle
srendle@google.com
Google Research
Mountain View, California

Walid Krichene
walidk@google.com
Google Research
Mountain View, California

Li Zhang
liqzhang@google.com
Google Research
Mountain View, California

John Anderson
janders@google.com
Google Research
Mountain View, California

ABSTRACT

Embedding based models have been the state of the art in collaborative filtering for over a decade. Traditionally, the dot product or higher order equivalents have been used to combine two or more embeddings, e.g., most notably in matrix factorization. In recent years, it was suggested to replace the dot product with a learned similarity e.g. using a multilayer perceptron (MLP). This approach is often referred to as *neural collaborative filtering* (NCF). In this work, we revisit the experiments of the NCF paper that popularized learned similarities using MLPs. First, we show that with a proper hyperparameter selection, a simple dot product substantially outperforms the proposed learned similarities. Second, while a MLP can in theory approximate any function, we show that it is non-trivial to learn a dot product with an MLP. Finally, we discuss practical issues that arise when applying MLP based similarities and show that MLPs are too costly to use for item recommendation in production environments while dot products allow to apply very efficient retrieval algorithms. We conclude that MLPs should be used with care as embedding combiner and that dot products might be a better default choice.

CCS CONCEPTS

• **Information systems → Recommender systems**; *Collaborative filtering*; • **Computing methodologies →** *Learning from implicit feedback*; Neural networks; Factorization methods.

KEYWORDS

Item Recommendation; Matrix Factorization; Neural Collaborative Filtering

ACM Reference Format:
Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20), September 22–26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3383313.3412488

1 INTRODUCTION

Embedding based models have been the state of the art in collaborative filtering for over a decade. A core operation of most of these embedding based models is to combine two or more embeddings. For example, combining a user embedding with an item embedding to obtain a single score that indicates the preference of the user for the item. This can be viewed as a *similarity function* in the embedding space. Traditionally, a dot product or higher order products have been used for the similarity. Recently, it has become popular to learn the similarity function with a neural network. Most commonly, a multilayer perceptron (MLP) is used for the network architecture (e.g. [19, 21, 28, 33, 38, 39]). This approach is often referred to as *neural collaborative filtering* (NCF) [17]. The rationale is that MLPs are general function approximators so that they should be strictly better than a fixed similarity function such as the dot product. This has made NCF the model of choice for comparison in many recommender studies (e.g. [19, 21, 28, 31, 38, 39]).

In this work, we study MLP versus dot product similarities in more detail. We start with revisiting the experiments of the NCF paper [17] that popularized the use of MLPs in recommender systems. We show that a carefully configured dot product baseline largely outperforms the MLP. At first glance, it looks surprising that the MLP, which is a universal function approximator, does not perform at least as well as the dot product. We investigate this issue in a second experiment and show empirically that learning a dot product with high accuracy for a decently large embedding dimension requires a large model capacity as well as many training data. Besides prediction quality, we also discuss the inference cost of dot product versus MLPs, where dot products have a large advantage due to the existence of efficient maximum inner product search algorithms. Finally, we discuss that dot product vs MLP is not a question of whether a deep neural network (DNN) is useful. In fact, many of the most competitive DNN models, such as transformers in natural language processing [10] or resnets for image classification [15], use a dot product similarity in their output layer.

To summarize, this paper argues that MLP-based similarities for combining embeddings should be used with care. While MLPs can approximate any continuous function, their inductive bias might not be well suited for a similarity measure. Unless the dataset is large or the embedding dimension is very small, a dot product is likely a better choice.

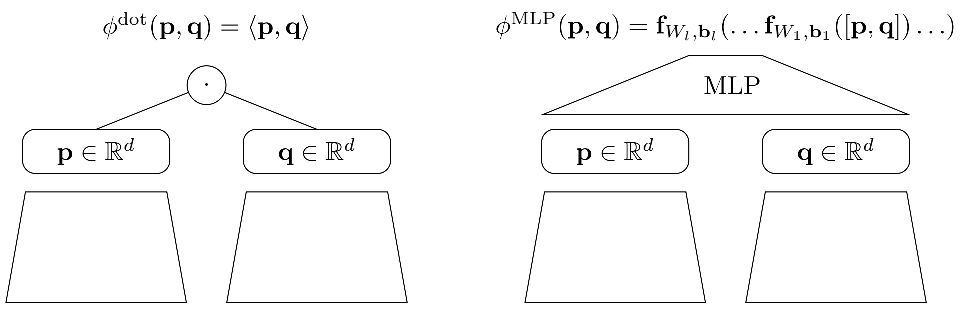
$\phi^{\text{dot}}(\mathbf{p},\mathbf{q}) = \langle \mathbf{p},\mathbf{q} \rangle$
$\cdot$
$\mathbf{p} \in \mathbb{R}^d$
$\mathbf{q} \in \mathbb{R}^d$
$\phi^{\text{MLP}}(\mathbf{p},\mathbf{q}) = \mathbf{f}_{W_l,\mathbf{b}_l}(\ldots \mathbf{f}_{W_1,\mathbf{b}_1}([\mathbf{p},\mathbf{q}])\ldots)$
MLP
$\mathbf{p} \in \mathbb{R}^d$
$\mathbf{q} \in \mathbb{R}^d$

Figure 1: A model with dot product similarity (left) and MLP-based learned similarity (right).

2 DEFINITIONS

In this section, we formalize the problem and review dot product (esp., matrix factorization) and learned similarity functions (esp., MLP and NeuMF). We denote matrices by upper case letters X, vectors by lowercase bold letters $\mathbf{x}$, scalars by lowercase letters x. A concatenation of two vectors $\mathbf{x}, \mathbf{z}$ is denoted by $[\mathbf{x}, \mathbf{z}]$.

Our paper studies functions $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that combine two d-dimensional embedding vectors $\mathbf{p} \in \mathbb{R}^d$ and $\mathbf{q} \in \mathbb{R}^d$ into a single score. For example $\mathbf{p}$ could be the embedding of a user, $\mathbf{q}$ the embedding of an item, and $\phi(\mathbf{p}, \mathbf{q})$ is the affinity of this user to the item.

The embeddings $\mathbf{p}$ and $\mathbf{q}$ can be model parameters such as in matrix factorization, but they can also be functions of other features, for example the user embedding $\mathbf{p}$ could be the output of a deep neural network taking user features as input. From here on, we focus mainly on the similarity function ϕ but in Section 6.1 we will discuss the embeddings in more detail.

Dot Product. The most common combination of two embeddings is the dot product.

$$\phi^{\text{dot}}(\mathbf{p}, \mathbf{q}) := \langle \mathbf{p}, \mathbf{q} \rangle = \mathbf{p}^T \mathbf{q} = \sum_{f=1}^{d} p_f q_f. \tag{1}$$

If $\mathbf{p}$ and $\mathbf{q}$ are free model parameters, then this is equivalent to matrix factorization. A common trick is to add explicit biases:

$$\phi^{\text{dot}}(\mathbf{p}, \mathbf{q}) := b + p_1 + q_1 + \langle \mathbf{p}_{[2,\ldots,d]}, \mathbf{q}_{[2,\ldots,d]} \rangle. \tag{2}$$

This modification does not add expressiveness but has been found to be useful in many studies, likely because its inductive bias is better suited to the problem [23, 32].

Learned Similarity. Multi layer perceptrons (MLPs) are known to be universal approximators that can approximate any continuous function on a compact set as long as the MLP has enough hidden states [7]. One layer of a multi layer perceptron can be defined as a function $\mathbf{f} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$:

$$\mathbf{f}_{W,\mathbf{b}}(\mathbf{x}) = \boldsymbol{\sigma}(W\,\mathbf{x} + \mathbf{b}), \quad \boldsymbol{\sigma}(\mathbf{z}) = [\sigma(z_1), \ldots, \sigma(z_{\text{out}})], \tag{3}$$

which is parameterized by $W \in \mathbb{R}^{\text{in}\times\text{out}}$, $\mathbf{b} \in \mathbb{R}^{\text{out}}$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. In a multilayer perceptron (MLP), several layers of $\mathbf{f}$ are stacked, e.g., for a 3-layer MLP, $\mathbf{f}_{W_3,\mathbf{b}_3}(\mathbf{f}_{W_2,\mathbf{b}_2}(\mathbf{f}_{W_1,\mathbf{b}_1}(\mathbf{x})))$.

He et al. [17] propose to replace the dot product with learned similarity functions for collaborative filtering. They suggest to concatenate the two embeddings, $\mathbf{p}$ and $\mathbf{q}$, and apply an MLP:

$$\phi^{\text{MLP}}(\mathbf{p}, \mathbf{q}) := \mathbf{f}_{W_l,\mathbf{b}_l}(\ldots \mathbf{f}_{W_1,\mathbf{b}_1}([\mathbf{p}, \mathbf{q}])\ldots). \tag{4}$$

They further suggest a variation that combines the MLP with a weighted dot product model and name it *neural matrix factorization* (NeuMF):

$$\phi^{\text{NeuMF}}(\mathbf{p}, \mathbf{q}) := \phi^{\text{MLP}}\left(\mathbf{p}_{[1,\ldots,j]}, \mathbf{q}_{[1\ldots j]}\right) \tag{5}$$

$$+ \phi^{\text{GMF}}\left(\mathbf{p}_{[j+1\ldots d]}, \mathbf{q}_{[j+1\ldots d]}\right), \tag{6}$$

where GMF is a 'generalized' matrix factorization model:

$$\phi^{\text{GMF}}(\mathbf{p}, \mathbf{q}) := \sigma\left(\mathbf{w}^T(\mathbf{p} \odot \mathbf{q})\right) = \sigma\left(\sum_{f=1}^{d} w_f p_f q_f\right). \tag{7}$$

with learned weights $\mathbf{w} \in \mathbb{R}^d$. For NeuMF, they recommend to use one part of the embedding (here the first j entries) in the MLP and the remaining $d - j$ entries with the GMF.

Fig. 1 illustrates two models with dot product and MLP-based similarity.

3 REVISITING NCF EXPERIMENTS

In this section, we revisit the experiments of the NCF paper [17] that popularized the use of MLPs as embedding combiners in recommender systems. We show that a simple dot product yields better results.

3.1 Experimental setup

The NCF paper [17] evaluates on an item retrieval task on two datasets: a binarized version of Movielens 1M [14] and a dataset from Pinterest [13]. Both are implicit feedback datasets, i.e. they contain only binary positive tuples between a user and an item. For each user, the last item is held out and used as the test set, the remaining items of the user are placed into the training set. For evaluation, each recommender ranks, for each user, a set of 101 items consisting of the withheld test item together with 100 random items. For each user, the position at which the withheld item is ranked by the recommender is recorded, then two metrics are measured: (1) Hit Ratio (i.e. Recall) among the top 10 ranked items – which in this case is 1 if the withheld item is in the top 10 or 0 otherwise. (2) NDCG among the top 10 ranked items – which

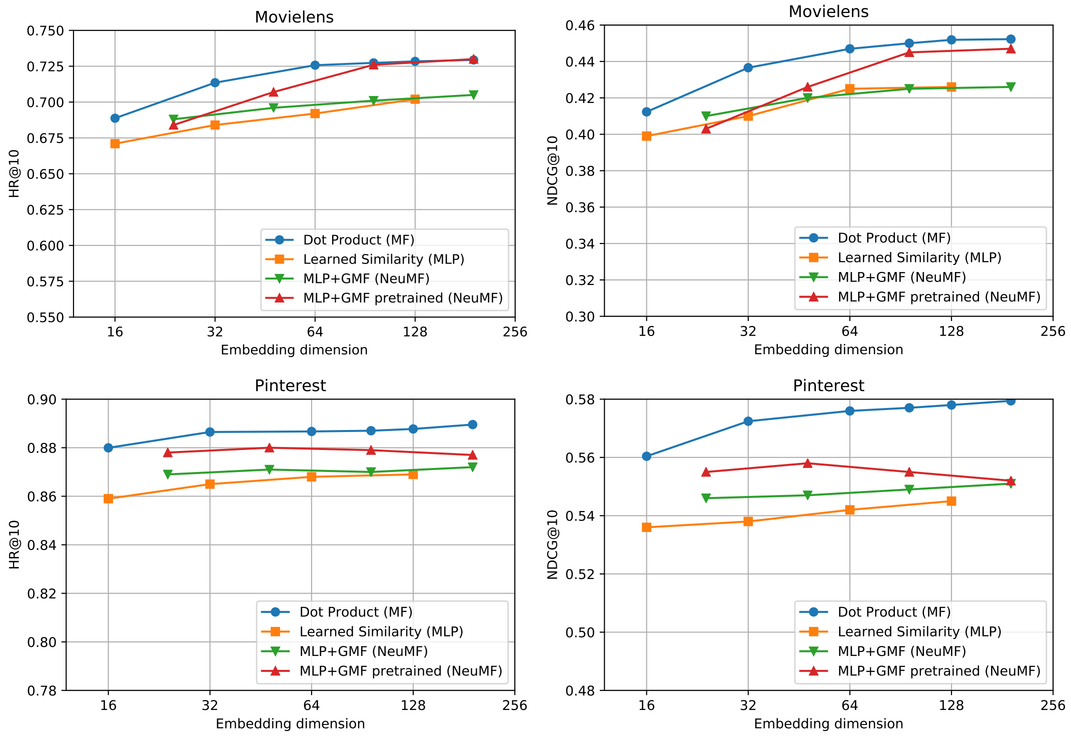

Figure 2: Comparison of learned similarities (MLP, NeuMF) to a dot product: The results for MLP and NeuMF are from [17]. The dot product substantially outperforms the learned similarity measures. Only the pretrained NeuMF is competitive, on one dataset, and for large embedding dimension.

in this case is $1/log(r + 1)$ where r is the rank of the withheld item. The average metric over all users is reported. The authors have published the dataset splits and the evaluation code. This allows us to evaluate on exactly the same setting and to compare our results directly with the ones reported in [17].

3.2 Models, loss and training algorithm

We compare three models: MLP-learned similarity models introduced in [17], which use ϕ^{MLP} and ϕ^{NeuMF} respectively, and a simple matrix factorization baseline which uses ϕ^{dot} from Eq. (2). The only difference between these models is the similarity function. In particular, the embeddings $\mathbf{p}, \mathbf{q}$ are free parameters in all models. We train the matrix factorization baseline by minimizing a logistic loss with L2 regularization, using stochastic gradient descent (with no batching, no momentum or other variations) with negative sampling, as in the original paper [17][1]. More precisely, for each training example (consisting of a user and a positive item), we sample m negative items, uniformly at random. Finally, we vary the embedding dimension $d \in \{16, 32, 64, 96, 128, 192\}$. Additional details about the setup can be found in Appendix A.

[1]It is possible that a different loss or a different sampling strategy could lead to an even better performance of our method. However, we wanted to use the same loss and sampling strategy for all competing methods to ensure that this is a meaningful comparison, which will allow us to attribute differences in quality to the choice of similarity functions.

3.3 Results

The results are reported in Fig. 2. Contrary to the findings of the NCF paper, the simple matrix factorization model exhibits the best quality over all evaluation metrics, and all embedding dimensions but one.

3.3.1 Matrix Factorization vs MLP. Our main interest is to investigate if the MLP-learned similarity is superior to a simple dot product. As can be seen in Fig. 2, the dot product substantially outperforms MLP on all datasets, evaluation metrics and embedding dimensions. With a properly set up matrix factorization model, the experiments do not show any evidence that a MLP is superior. In addition to a lower prediction quality, MLP-learned similarity suffers from other disadvantages compared to dot-product: the model has more model parameters (see Section 4.1), and is more expensive to serve (see Section 5).

3.3.2 Matrix Factorization vs NeuMF. The NCF paper [17] also proposes a combined model where the similarity function is a sum of dot-product and MLP, as in Eq. (5) – this is called NeuMF[2]. The green curve in Fig. 2 shows the performance of this combined model. One can observe only a minor improvement over MLP and overall a much worse quality than MF. The experiments do not support the

[2]Following [17], the NeuMF uses 2/3rds of the embeddings for the MLP and 1/3rd for the MF. See the discussion about "predictive factors" in Section A.3 for details.

Table 1: Comparison from [8] of NeuMF with various baselines and our results. The best results are highlighted in bold, the second best result is underlined.

Method	Movielens		Pinterest		Result
	HR@10	NDCG@10	HR@10	NDCG@10	from
Popularity	0.4535	0.2543	0.2740	0.1409	[8]
SLIM [25, 30]	<u>0.7162</u>	<u>0.4468</u>	0.8679	<u>0.5601</u>	[8]
iALS [20]	0.7111	0.4383	0.8762	0.5590	[8]
NeuMF (MLP+GMF) [17]	0.7093	0.4349	<u>0.8777</u>	0.5576	[8]
Matrix Factorization	**0.7294**	**0.4523**	**0.8895**	**0.5794**	Fig. 2

claim in [17] that a dot product model can be enhanced by feeding some part of its embeddings through an MLP.

A second variant of NeuMF was proposed in [17], that first trains MLP and MF models separately, then fine tunes the combined model. This can be viewed as a form of ensembling. The red curve shows this variant, which performs better than training the combined model directly (in green), but performs worse than the MF baseline overall, except on one datapoint (HR on Movielens with embedding dimension $d = 192$). Once again, the results do not support the claim that a learned similarity using a MLP is superior to a dot product. The experiment only indicates that ensembling two models can be helpful, a fact that has been observed for a variety of applications, and it is possible that ensembling with different models may yield a similar improvement. The fact remains that using a simple dot product outperforms this ensemble.

3.3.3 On the performance of GMF. Other variants of matrix factorization were considered in [17]. In particular, the GMF model uses a weighted dot product ϕ^{GMF} as described in Eq. (7). Except for the weights in the dot product, this model is very similar to the MF baseline we trained, in particular, both models use the same loss and negative sampling method. Nevertheless, the GMF results reported in [17] are much worse than our MF results. This discrepancy may seem surprising at first glance. We can see two reasons for this difference. First, properly setting up and tuning baseline methods can be difficult in general, as argued in [34], and the reported results may be improved by a more careful setup.

Second, ϕ^{GMF} introduces new model parameters – the vector $\mathbf{w}$ in Eq. (7). While this appears to be an innocuous generalization of the dot product similarity, it can have negative effects. For example, L2 regularization of the embeddings ($\mathbf{p}$ and $\mathbf{q}$) is meaningless unless $\mathbf{w}$ is regularized as well. More precisely, suppose the loss function is of the form

$$L(P, Q, \mathbf{w}, \lambda) = \ell\left(\left\{\phi^{\text{GMF}}_{\mathbf{w}}(\mathbf{p}, \mathbf{q}) : \mathbf{p} \in \text{Rows}(P),\ \mathbf{q} \in \text{Rows}(Q)\right\}\right) + \lambda\left(\|P\|_F^2 + \|Q\|_F^2\right)$$

where P, Q are embedding matrices, the first term of the loss ℓ depends on the pairwise similarities (i.e. the model output), and the second term is a regularization term, where P, Q are regularized but $\mathbf{w}$ is not. Observe that if we scale the model parameters as $P/a, Q/a, a^2\mathbf{w}$ for some positive scalar a, then the model output is unchanged (given the expression of ϕ^{GMF}), and we have

$$L(P, Q, \mathbf{w}, \lambda) = L\left(\frac{1}{a}P, \frac{1}{a}Q, a^2\mathbf{w}, a^2\lambda\right). \tag{8}$$

It follows that minimizing L with a given λ is equivalent to minimizing L with any other $\tilde{\lambda}$ up to the change of variable $(P/a, Q/a, a^2\mathbf{w})$ with $a = \sqrt{\frac{\tilde{\lambda}}{\lambda}}$, a change of variable which leaves the model output unchanged. The solution is therefore unaffected by regularization. A second consequence is that unless $\lambda = 0$, minimizing the loss L will likely result in embedding matrices P, Q of vanishing norm and a vector of weights $\mathbf{w}$ of diverging norm, leading to numerical instability.

The GMF results in [17] support that the model is indeed not properly regularized because its results do not improve with a higher embedding dimension – unlike in our experiments.

Finally, we observe that GMF does not improve model expressivity compared to a simple dot product, since the weights $\mathbf{w}$ can simply be absorbed into the embedding matrices P and Q. This is another indicator that adding parameters to a simple model is not always a good idea and has to be done carefully.

3.4 Further comparison

As reported in the meta study of [8], the results for NeuMF and MLP in [17] were cherry-picked in the following sense: the metrics are reported for the best iteration *selected on the test set.* The NeuMF and MLP numbers we report in Fig. 2 are from the original paper and likely over-estimate the actual test performance of those methods. On the other hand, our MF results in Fig. 2 are not cherry picked, because we select all hyperparameters including the stopping iteration on a validation set – see Appendix A for details. The fact that our baseline MF outperforms the MLP-learned similarity despite the cherry-picking in the latter strengthens our conclusions.

In this section, we give an additional comparison using non cherry-picked results produced by [8]. Table 1 includes their results together with our matrix factorization (same as in Fig. 2), with embedding dimension $d = 192$. The results confirm that the simple matrix factorization model substantially outperforms NeuMF on all metrics and datasets. Our results provide further evidence to the conclusion of [8] that simple, well-known baselines outperform NCF. Note that matrix factorization was also one of the baselines in [8] (the iALS method in Table 1), but our experiment shows a much larger margin than was obtained in [8]. The difference might be caused by a suboptimal choice of hyperparameters.

3.5 Discussion

Following the arguments in [34], it is possible that the studies in [17] and [8] did not properly set up MLP and NeuMF, and that these results could be further improved. It is also possible that the performance of these models is different on other datasets. Nevertheless, at this point, the revised experiments from [17] provide no evidence supporting the claim that a MLP-learned similarity is superior to a dot product. This negative result also holds for NeuMF where a GMF is added to the MLP. And it also holds for the pretrained version of NeuMF. Our study treats MLP and NeuMF favorably: (1) we report the results for MLP and NeuMF that were obtained by the original authors, avoiding any bias in improperly running their methods. (2) These cited numbers for MLP and NeuMF are likely too optimistic as they were obtained through cherry picking as identified by [8].

4 LEARNING A DOT PRODUCT WITH MLP IS HARD

An MLP is a universal function approximator: any smooth function on a compact set can be approximated with a large enough MLP [3, 7, 18]. It is tempting to argue that this makes the MLP a more powerful embedding combiner and it should thus perform at least as well or better than a dot product. However, such an argument neglects the difficulty of learning the target function using MLPs: the larger class of functions also implies more parameters needed for representing the function. Hence it would require more data to learn the function and may encounter difficulty in actually learning the desired target function. Indeed, specialized structures, e.g. convolutional, recurrent, and attention structures, are common in neural networks. There is probably no hope to replace them using an MLP though they should all be representable. However, is this also true for the simple "structure" of the dot product? Similar problems turn out to be actively studied subjects in machine learning theory [1, 2, 11, 26]. To our knowledge, the best theoretical bound for learning the dot product, a degree two polynomial, requires $O(d^4/\epsilon^2)$ steps for an error bound of ϵ [2]. While the theory gives only a sufficient condition, it does hint that the difficulty scales polynomially with dimension d and $1/\epsilon$. This motivates us to investigate the question empirically.

4.1 Experimental setup

We set up a synthetic learning task[3] where given two embeddings $\mathbf{p}, \mathbf{q} \in \mathbb{R}^d$ and a label $y(\mathbf{p}, \mathbf{q})$, we want to learn a function $\hat{y} : \mathbb{R}^{2d} \to \mathbb{R}$ that approximates y with $\hat{y}(\mathbf{p}, \mathbf{q})$. We draw the embeddings $\mathbf{p}, \mathbf{q}$ from $\mathcal{N}(0, \sigma^2_{\text{emb}} I)$ and set the true label as $y(\mathbf{p}, \mathbf{q}) = \langle \mathbf{p}, \mathbf{q} \rangle + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2_{\text{label}})$ models the label noise. From this process we create three datasets each consisting of tuples $(\mathbf{p}, \mathbf{q}, y)$. One of the datasets is used for training and the remaining two for testing. For the training and first test dataset, we first sample M different user embeddings and N different item embeddings, i.e., there are two fixed embedding matrices $P \in \mathbb{R}^{M \times d}$ and $Q \in \mathbb{R}^{N \times d}$. Then we uniformly sample (without replacement) 100 M user-item combinations and put 90% into the training set and 10% into the test set. We create a second test set that consists of *fresh* embeddings that did not appear in the training or test set, i.e., we sample the embeddings for every case from $\mathcal{N}(0, \sigma^2_{\text{emb}} I)$ instead of picking them from P and Q. The motivation for this setup is to investigate if the learned similarity function generalizes to embeddings that were not seen during training.

We train the MLP on the training dataset and evaluate it on both test datasets. For the architecture of the MLP, we follow the suggestions in the NCF paper: we use an input layer of size $2d$ consisting of the concatenation of the two embeddings, and 3 hidden layers with sizes $[4\,h, 2\,h, h]$ where h is a parameter, and use the ReLU as the activation function. The NCF paper suggests to use $h = d/2$, we also experiment with $h = d$ and $h = 2d$. For $h = d$, the number of model parameters are about $18\,d^2$, so for example for d=8: 1,152 or d=64: 73,728 or for d=256: 1,179,648. For optimization, we also follow the NCF paper and choose the Adam optimizer.

As evaluation metric, we compute the RMSE between the predicted similarity of the MLP and the true similarity y. We also measure the RMSE of a trivial model that predicts always 0 (=average rating in our dataset). In our setup, this RMSE is equal in expectation to $\sqrt{\text{Var}(y)} = \sqrt{\sigma^2_{\text{label}} + d\,\sigma^4_{\text{emb}}}$. Secondly, we measure the RMSE of the dot product model, i.e., $\hat{y}(\mathbf{p}, \mathbf{q}) = \langle \mathbf{p}, \mathbf{q} \rangle$. This RMSE is equal in expectation to σ_{label}. We report the approximation error, i.e., the difference between the RMSE of the dot product model and the MLP. Each experiment is repeated 5 times and we report the mean.

We want to choose the experimental parameters σ_{label} and σ_{emb} such that the approximation error gives some indication what values are acceptable. To do this we choose values that are related to well-studied rating prediction tasks. In the Netflix prize, the best models have RMSEs of about 0.85 [22] – for Movielens 10M, the best models have about 0.75 [34]. For these datasets, it is likely that the label noise is close to these values, thus we choose the label noise $\sigma_{label} = 0.85$. For the Netflix prize, the trivial model that predicts always the average rating has an RMSE of 1.13. Thus we set $\sigma^2_{emb} = \sqrt{\frac{1.13^2 - 0.85^2}{d}}$. With this setup, the trivial model in our experiment has the same RMSE as the trivial model on Netflix. By aligning both the trivial model and the noise to the Netflix prize, absolute differences in our experiment give some indication of the scale of acceptable errors. In both Netflix and ML 10M, a difference in RMSE of 0.01 is considered very large. For example, for the Netflix prize it took the community about a year[4] to lower the RMSE from 0.8712 to 0.8616. Similarly, for Movielens 10M, it took about 4 years to lower the RMSE from 0.7815 to 0.7634. Much smaller differences have been published. For example many published increments on Movielens 10M are about 0.001 [34]. We will use these thresholds of 0.01 and 0.001 as an indication whether the approximation errors are acceptable in our experiments. While this is not a perfect comparison, we hope that it can serve as a reasonable indicator.

4.2 Results

Figure 3 shows the approximation error of the MLP for different choices of embedding dimensions and as a function of training data. The figure suggests that with enough training data and wide enough hidden layers, an MLP can approximate a dot product. This holds for embeddings that have been seen in the training data as well

[3]The code is available at https://github.com/google-research/google-research/tree/master/dot_vs_learned_similarity.

[4]https://www.netflixprize.com/leaderboard_quiz.html

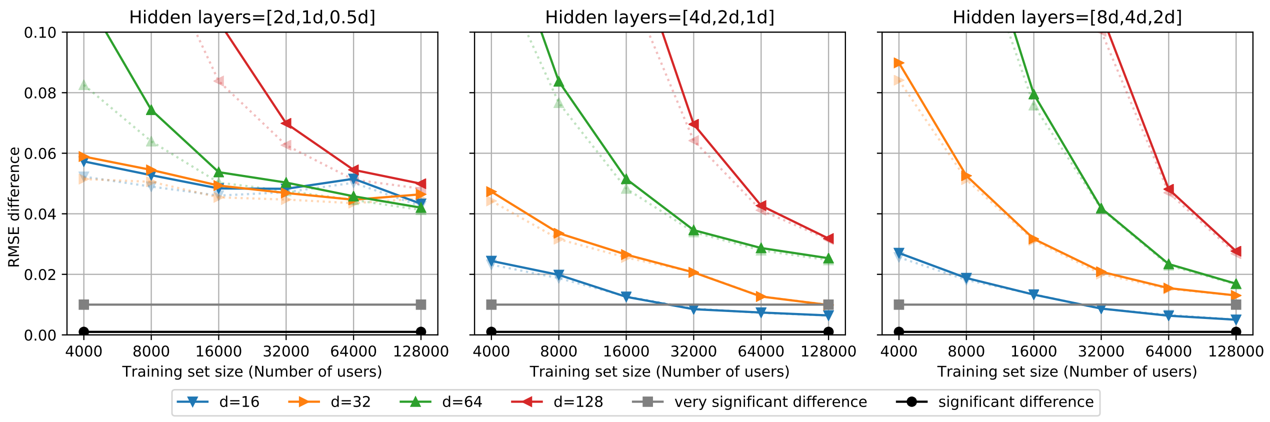

Figure 3: How well a MLP can learn a dot product over embeddings of dimension d. The ground truth is generated from a dot product of Gaussian embeddings plus Gaussian label noise. The graphs show the difference between the RMSE of the dot product and the RMSE of the learned similarity measure; the solid line measures the difference on the fresh set, the dotted on the test set. Noise and scale have been chosen such that 0.01 could indicate a very significant difference and 0.001 a significant difference.

as for fresh embeddings. However, consistent with the theory, the number of samples needed scales polynomially with the increasing dimensions and reduced error. Anecdotally, we observe the number of samples needed is about $O(d/\epsilon)^\alpha$ for $1 \leq \alpha \leq 2$. The experiments clearly indicate that it becomes increasingly difficult for an MLP to fit the dot product function with increasing dimensions. In all cases, the approximation error is well above what is considered a large difference for problems with comparable scale. For example, for the moderate $d = 128$, with 128000 users, the error is still above 0.02, much higher than the *very significant* difference of 0.01.

This experiment shows the difficulty of using an MLP to approximate the dot product, even when explicitly trained to do so. Hence, if the dot product performs well on a given task, there could be a significant price to pay for an MLP to approximate it. We hope this can explain, at least partially, why the dot product model outperforms the MLP model in the experiments of Section 3.3.

5 APPLICABILITY OF DOT PRODUCT MODELS

Most academic studies focus on training runtime when discussing applicability. However, in industrial applications, the serving runtime is often more important, in particular when the recommendations cannot be precomputed offline but need to be computed at the time of the user's request. This is the case for most context-aware recommenders in which the recommendation depends on contextual features that are only available at query time. For instance, consider a sequential recommender that recommends items to a user based on the previously selected L items. Here the top scoring items cannot be precomputed for all possible combinations of L items. Instead the recommender would need to retrieve the highest scoring items from the whole item catalogue with a latency of a few milliseconds after the user's request. Such real time retrieval is a common application in real world recommender systems [6].

Computing a dot product similarity takes $O(d)$ time while computing an MLP-learned similarity takes $O(d^2)$ time. If there are n items to score, then the total costs are $O(dn)$ (for dot) vs $O(d^2n)$ (for MLP). For large scale applications, n is typically in the range of millions and d is in the hundreds, and while dot has a lower complexity, both are impractical for retrieval applications that require latencies of a few milliseconds. However, for a dot product, the problem of finding the top scoring items can be approximated efficiently. Indeed, given the user embedding $\mathbf{p}$, the problem is to find items i that maximize $\langle \mathbf{p}, \mathbf{q}_i \rangle$. This is a well-studied problem, known as *approximate nearest neighbor search* [27] or *maximum inner product search* [35]. Efficient sublinear time algorithms exist that makes dot product retrieval feasible in typically a few milliseconds, even with millions of items n [6]. To the best of our knowlegde, no such sublinear techniques exist for nearest neighbor retrieval with MLPs.

To summarize, MLP similarity is not applicable for real time top-N recommenders, while the dot product allows fast retrieval using well established nearest neighbor search algorithms.

6 RELATED WORK

6.1 Dot products at the Output Layer of DNNs

At first glance it might appear that our work questions the use of neural networks in recommender systems. This is not the case, and as we will discuss now, many of the most competitive neural networks use a dot product for the output but not an MLP. Consider the general multiclass classification task where $(\mathbf{x}, y)$ is a labeled training example with input $\mathbf{x}$ and label $y \in \{1, \ldots, n\}$. A common approach is to define a DNN $\mathbf{f}$ that maps the input $\mathbf{x}$ to a representation (or embedding) $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^d$. At the final stage, this representation is combined with the class labels to produce a vector of scores. Commonly, this is done by multiplying the input representation $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^d$ with a class matrix $Q \in \mathbb{R}^{n \times d}$ to obtain a scalar score for each of the n classes. This vector is then used in the loss function, for example as logits in a softmax cross entropy with the label y. This falls exactly under the family of models discussed in this paper, where $\mathbf{p} = \mathbf{f}(\mathbf{x}) \in \mathbb{R}^d$ and the classes are the items. In fact, the model as described above is a dot product model because at the output $Q\,\mathbf{f}(\mathbf{x}) = Q\,\mathbf{p} = [\langle \mathbf{p}, \mathbf{q}_i \rangle]_{i=1}^n$ which means each input-label or user-item combination is a dot product between an input (or user) embedding and label (or item) embedding. This dot product combination of input and class representation is commonly used in sophisticated DNNs for image classification [15, 24] and for natural language processing [4, 10, 29]. This makes our findings that a dot product is a powerful embedding combiner well aligned with the broader DNN community where it is common to apply a dot product at the output for multiclass classification.

6.2 MLPs at the Output Layer of DNNs

NeuMF is very closely related to the previously proposed *neural network matrix factorization* [12]. Neural network matrix factorization also uses a combination of an MLP plus extra embeddings with an explicit dot product like structure as in GMF. A follow up paper [16] proposes to replace the MLP in NCF by an outerproduct and pass this matrix through a convolutional neural network. Finding the dot product with this technique is trivial because the sum of the diagonal in the outerproduct is the dot product. Unfortunately, while written by the same authors as the NCF paper, it evaluates on different data, so our results in Section 3.3 cannot be compared to their numbers. A recent study [9] investigates convolutional structures at the output and finds no improvements in predictive quality over a dot product. Besides prediction quality, a convolutional structure at the output suffers from the same applicability issues as the MLP (see Section 5).

6.3 Specialized Structures inside a DNN

In DNN modeling it is very common to replace an MLP by a more specialized structure that has an inductive bias that represents the problem better. For example, in image classification structures such as convolutional neural networks are very popular because they represent the spatial structure of the input data. In recurrent neural networks, such parameter sharing is very important too. Another example are attention models, e.g. in Neural Machine Translation [37] and in the Transformer model [36], that contain a matrix product inside the neural network for combining multiple inputs – they can be regarded as the dot product model for combining "internal" embeddings too. All these specialized structures are crucial for advancing the state of the art of deep learning, although in theory they can all be approximated by MLPs.

The inefficiency of MLPs to capture dot and tensor products has been studied by [5] in the context of recommender systems. Here the authors examine how to add context to recurrent neural networks. Similar to our work and Section 4.1, [5] points out that MLPs do not model multiplications and it investigates approximating dot products and tensor products with MLPs empirically. Their study focuses on the model size required to learn a tensor product for embeddings of dimension $d = 1$ and $d = 2$, where the number of distinct embeddings is 100 per mode and the training error is measured.

6.4 Experimental Issues in Recommender Systems

The meta study [8] points out issues with evaluation in recommender system research. Their experiments also cover the NCF paper. They show that well studied baselines can get comparable results to (a reproducible value of) NeuMF (see Section 3.4). The goal of our study and [8] is different. While [8] covers a broad set of methods and publications, we are investigating the specific issue of learned similarity functions in more detail. Our work provides apples to apples comparisons of dot product vs MLP, stronger results (outperforming the original NCF results), and a thorough investigation of the reasons and consequences.

7 CONCLUSION

Our findings indicate that a dot product might be a better default choice for combining embeddings than learned similarities using MLP or NeuMF. Shifting the focus in the recommender system research community from learned similarities to dot products might have several positive effects: (1) The research becomes more relevant for the industry because models are applicable (see Section 5). (2) Dot product similarity simplifies modeling and learning (no pretraining, no need for large datasets) which facilitates both experimentation and understanding. (3) Better alignment with other research areas such as natural language processing or image models where the dot product is commonly used.

Finally, our experiments give further evidence that running machine learning methods properly is difficult [34] and one-off studies are prone to drawing wrong conclusions. Introducing shared benchmarks might help to better identify improvements.

REFERENCES

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. 2019. A convergence theory for deep learning via over-parameterization. In *Proceedings of the 36th International Conference on Machine Learning*. 242–252.

[2] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. 2014. Learning Polynomials with Neural Networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. JMLR.org, II–1908–II–1916.

[3] Andrew R Barron. 1993. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory* 39, 3 (1993), 930–945.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[5] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H. Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. Association for Computing Machinery, New York, NY, USA, 46–54. https://doi.org/10.1145/3159652.3159727

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[7] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[8] Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. 2019. A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research. arXiv:cs.IR/1911.07698

[9] Maurizio Ferrari Dacrema, Federico Parroni, Paolo Cremonesi, and Dietmar Jannach. 2020. Critically Examining the Claimed Value of Convolutions over User-Item Embedding Maps for Recommender Systems. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020), October 19–23, 2020, Virtual Event, Ireland.* https://doi.org/10.1145/3340531.3411901

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:cs.CL/1810.04805

[11] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. 2019. Gradient Descent Finds Global Minima of Deep Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*. 1675–1685.

[12] Gintare Karolina Dziugaite and Daniel M. Roy. 2015. Neural Network Matrix Factorization. arXiv:cs.LG/1511.06443

[13] X. Geng, H. Zhang, J. Bian, and T. Chua. 2015. Learning Image and User Features for Recommendation in Social Networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 4274–4282.

[14] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article Article 19 (Dec. 2015), 19 pages. https://doi.org/10.1145/2827872

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Jun 2016). https://doi.org/10.1109/cvpr.2016.90

[16] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. 2018. Outer Product-based Neural Collaborative Filtering. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 2227–2233. https://doi.org/10.24963/ijcai.2018/308

[17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 173–182. https://doi.org/10.1145/3038912.3052569

[18] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[19] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S. Yu. 2018. Leveraging Meta-Path Based Context for Top- N Recommendation with A Neural Co-Attention Model. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1531–1540. https://doi.org/10.1145/3219819.3219965

[20] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. 263–272.

[21] I. M. A. Jawarneh, P. Bellavista, A. Corradi, L. Foschini, R. Montanari, J. Berrocal, and J. M. Murillo. 2020. A Pre-Filtering Approach for Incorporating Contextual Information Into Deep Learning Based Recommender Systems. *IEEE Access* 8 (2020), 40485–40498.

[22] Yehuda Koren. 2009. The BellKor Solution to the Netflix Grand Prize.

[23] Yehuda Koren and Robert Bell. 2011. *Advances in Collaborative Filtering*. Springer US, Boston, MA, 145–186. https://doi.org/10.1007/978-0-387-85820-3_5

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[25] Mark Levy and Kris Jack. 2013. Efficient top-n recommendation by linear regression. In *RecSys Large Scale Recommender Systems Workshop*.

[26] Yuanzhi Li and Yang Yuan. 2017. Convergence Analysis of Two-layer Neural Networks with ReLU Activation. In *Advances in Neural Information Processing Systems*. 597–607.

[27] Ting Liu, Andrew W. Moore, Alexander Gray, and Ke Yang. 2004. An Investigation of Practical Approximate Nearest Neighbor Algorithms. In *Proceedings of the 17th International Conference on Neural Information Processing Systems (NIPS'04)*. MIT Press, Cambridge, MA, USA, 825–832.

[28] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Atsushi Ike, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Tsuguchika Tabaru, Carole-Jean Wu, Lingjie Xu, Masafumi Yamazaki, Cliff Young, and Matei Zaharia. 2019. MLPerf Training Benchmark. arXiv:cs.LG/1910.01500

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[30] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*. IEEE, 497–506.

[31] Wei Niu, James Caverlee, and Haokai Lu. 2018. Neural Personalized Ranking for Image Recommendation. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. Association for Computing Machinery, New York, NY, USA, 423–431. https://doi.org/10.1145/3159652.3159728

[32] Arkadiusz Paterek. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, Vol. 2007. 5–8.

[33] Jiarui Qin, Kan Ren, Yuchen Fang, Weinan Zhang, and Yong Yu. 2020. Sequential Recommendation with Dual Side Neighbor-Based Collaborative Relation Modeling. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM '20)*. Association for Computing Machinery, New York, NY, USA, 465–473. https://doi.org/10.1145/3336191.3371842

[34] Steffen Rendle, Li Zhang, and Yehuda Koren. 2019. On the Difficulty of Evaluating Baselines: A Study on Recommender Systems. *CoRR* abs/1905.01395 (2019). arXiv:1905.01395 http://arxiv.org/abs/1905.01395

[35] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 2321–2329.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[37] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[38] Hamed Zamani and W. Bruce Croft. 2020. Learning a Joint Search and Recommendation Model from User-Item Interactions. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM '20)*. Association for Computing Machinery, New York, NY, USA, 717–725. https://doi.org/10.1145/3336191.3371818

[39] Xing Zhao, Ziwei Zhu, Yin Zhang, and James Caverlee. 2020. Improving the Estimation of Tail Ratings in Recommender System with Multi-Latent Representations. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM '20)*. Association for Computing Machinery, New York, NY, USA, 762–770. https://doi.org/10.1145/3336191.3371810

A EXPERIMENTS FROM NCF PAPER

This section provides details about our setup for establishing a dot product baseline for the experiments of the NCF paper (Section 3.3). The code and datasets of NCF were provided by its authors[5]. We provide code for our implementation of matrix factorization and the script to generate the tuning split[6].

A.1 Model and Optimization

We implemented a matrix factorization with bias (see Eq. 2). The parameters of this model are the embeddings $P \in \mathbb{R}^{M \times d}$ for M users and $Q \in \mathbb{R}^{N \times d}$ for N items. Following the NCF paper, for training we cast the implicit data, which contains only positive observations, into a binary two class classification problem and sample m negative items for each tuple in the implicit data. In each epoch a new set of negatives is drawn – the sampling distribution is uniform. We minimize the binary logistic loss with L2 regularization. For each training example (u, i, y) where $y \in \{0, 1\}$ is the binary label, the regularized loss is

$$l(u, i, y) = -y \, \ln \sigma(\phi(\mathbf{p}_u, \mathbf{q}_i)) - (1 - y) \, \ln(1 - \sigma(\phi(\mathbf{p}_u, \mathbf{q}_i))) \\ + \lambda \, \|\mathbf{p}_u\|^2 + \lambda \, \|\mathbf{q}_i\|^2$$

with the regularization constant $\lambda \in \mathbb{R}^+$. The loss is optimized with stochastic gradient descent with learning rate η, with the update rules:

$$\begin{aligned} p_{u,1} &\leftarrow p_{u,1} - \eta[(\sigma(\phi(\mathbf{p}_u, \mathbf{q}_i)) - y) + \lambda p_{u,1}] \\ q_{i,1} &\leftarrow q_{i,1} - \eta[(\sigma(\phi(\mathbf{p}_u, \mathbf{q}_i)) - y) + \lambda q_{i,1}] \\ \mathbf{p}_{u,[2,\ldots,d]} &\leftarrow \mathbf{p}_{u,[2,\ldots,d]} - \eta[(\sigma(\phi(\mathbf{p}_u, \mathbf{q}_i)) - y) \, \mathbf{q}_{i,[2,\ldots,d]} + \lambda \mathbf{p}_{u,[2,\ldots,d]}] \\ \mathbf{q}_{i,[2,\ldots,d]} &\leftarrow \mathbf{q}_{i,[2,\ldots,d]} - \eta[(\sigma(\phi(\mathbf{p}_u, \mathbf{q}_i)) - y) \, \mathbf{p}_{u,[2,\ldots,d]} + \lambda \mathbf{q}_{i,[2,\ldots,d]}] \end{aligned}$$

[5]https://github.com/hexiangnan/neural_collaborative_filtering

[6]https://github.com/google-research/google-research/tree/master/dot_vs_learned_similarity

The embeddings are initialized from a normal distribution. This configuration shares the same loss, regularization, negative sampling approach, and initialization procedure with MLP and NeuMF as proposed in [17].

The hyperparameters of the dot product model are: embedding dimension d, regularization λ, learning rate η, number of negative samples m, number of training epochs, standard deviation for initialization. Analogously to the NCF paper, we report results for $d \in \{16, 32, 64, 96, 128, 192\}$.

A.2 Hyperparameter Tuning

We create a tuning dataset that follows the same splitting protocol as the final training/test split. In particular, we remove the last feedback from each user from the training set and place it in a test set for tuning and keep the remaining training cases in a training set for tuning. We then train models on the training set for tuning and evaluate the model on the test set for tuning. We choose all hyperparameters including the number of training epochs on this tuning set. Note that both the training set for tuning and test set for tuning contain no information about the final test set.

From our past experience with matrix factorization models, if the other hyperparameters are chosen properly, then the larger the embedding dimension the better the quality – our experiments Figure 2 confirm this. For the other hyperparameters: learning rate and number of training epochs influence the convergence curves. Usually, the lower the learning rate, the better the quality but also the more epochs are needed. We set a computational budget of up to 256 epochs and search for the learning rate within this setting. In the first hyperparameter pass, we search a coarse grid of learning rates $\eta \in \{0.001, 0.003, 0.01\}$ and number of negatives $m = \{4, 8, 16\}$ while fixing the regularization to $\lambda = 0$. Then we did a search for regularization in $\{0.001, 0.003, 0.01\}$ around the promising candidates. To speed up the search, these first coarse passes were done with 128 epochs and a fixed dimension of $d = 64$ (Movielens) and $d = 128$ (Pinterest). We did further refinements around the most promising values of learning rate, number of negatives and regularization using $d = 128$ and 256 epochs. Throughout the experiments we initialize embeddings from a Gaussian distribution with standard deviation of 0.1; we tested some variation of the standard deviation but did not see much effect.

The final hyperparameters for Movielens are: learning rate $\eta = 0.002$, number of negatives $m = 8$, regularization $\lambda = 0.005$, number of epochs 256. For Pinterest: learning rate $\eta = 0.007$, number of negative samples $m = 10$, regularization $\lambda = 0.01$, number of epochs 256.

After hyperparameter selection, we trained on the full dataset with these hyperparameters and evaluated according to the protocol in [17]. We repeated the final training and evaluation 8 times and report the mean of the metrics.

A.3 MLP and NeuMF Results

We report the results for MLP and NeuMF from the original NCF paper [17]. As we share the same evaluation protocol and splits, the numbers are comparable. We report the results for NeuMF from Table 2 in [17] and the results for MLP from Tables 3,4 in [17] using the 'MLP-3' setting.

It should be noted that in [17], the tables and plots use "predictive factor" instead of embedding dimension. The predictive factor is defined as the size of the last hidden layer of the MLP, and as described in [17], for the 3-layer MLP a predictive factor of k operates on two input embeddings, each of dimension $d = 2\,k$. For the NeuMF model, a predictive factor of k operates on embeddings of dimension $d = 3\,k$ because it consists of an independent MLP with predictive factor of k (embedding size $d = 2\,k$) and a GMF with embedding size $d = k$. This definition of *predictive factor* is arbitrary, in fact it can be made arbitrarily small by adding layers to the MLP without changing anything else in the model. We think it is more meaningful to compare models with a fixed embedding dimension. In particular, we want to investigate the prediction quality of an MLP or a dot product over two embeddings of the same size d. We recast all results from the NCF paper in terms of embedding dimension, by multiplying the predictive factor by 3 for NeuMF results and by 2 for MLP results. This allows us to do an apples to apples comparison of different similarity functions over an embedding space of dimension d.