Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems

A.A. Ginart*, Maxim Naumov†, Dheevatsa Mudigere†, Jiyan Yang†, James Zou*

*Stanford University, Palo Alto, California, {tginart, jamesz}@stanford.edu
†Facebook, Inc. Menlo Park, California, {mnaumov, dheevatsa, chocjy}@facebook.com

Abstract—**Embedding representations power machine intelligence in many applications, including recommendation systems, but they are space intensive — potentially occupying hundreds of gigabytes in large-scale settings. To help manage this outsized memory consumption, we explore *mixed dimension embeddings*, an embedding layer architecture in which a particular embedding vector's dimension scales with its query frequency. Through theoretical analysis and systematic experiments, we demonstrate that using mixed dimensions can drastically reduce the memory usage, while maintaining and even improving the ML performance. Empirically, we show that the proposed mixed dimension layers improve accuracy by 0.1% using half as many parameters or maintain it using 16× fewer parameters for click-through rate prediction on the Criteo Kaggle dataset. They also train over 2× faster on a GPU.**

A full version of this paper is accessible at: https://arxiv.org/abs/1909.11810

I. INTRODUCTION

Embedding representations power state-of-the-art applications in diverse domains, including computer vision [1], [2], natural language processing [3]–[5], and recommendation systems [6]–[8]. It is standard practice to embed objects into $\mathbb{R}^d$ at a fixed uniform dimension (UD) d. When the embedding dimension d is too low, the downstream statistical performance suffers [9]. When d is high and the number of objects to represent is large, memory consumption becomes an issue. For example, in recommendation models, the embedding layer can make up more than 99.9% of the memory it takes to store the model, and in large-scale settings, it could consume hundreds of gigabytes or even terabytes [7], [10]. Therefore, finding innovative embedding representations that use fewer parameters while preserving statistical performance of the downstream model is an important challenge.

Object frequencies are often heavily skewed in real-world applications. For instance, for the full MovieLens dataset, the top 10% of users receive as many queries as the remaining 90% and the top 1% of items receive as many queries as the remaining 99%. To an even greater extent, on the Criteo Kaggle dataset the top 0.0003% of indices receive as many queries as the remaining $\sim$32 million. To leverage the heterogeneous object popularity in recommendation, we propose mixed dimension (MD) embedding layers, in which the dimension of a particular object's embedding scales with that object's popularity rather than remaining uniform. Our case studies and theoretical analysis demonstrate that MD embeddings work well because they do not underfit popular embeddings while wasting parameters on rare embeddings. Additionally, MD embeddings minimize popularity-weighted loss at test time by efficiently allocating parameters.

In Section 3, we introduce the proposed architecture for the embedding layer. In Section 4, we theoretically investigate MD embeddings. Our theoretical framework splits embedding-based recommendation systems into either the *data-limited regime* or the *memory-limited regime*, depending on the parameter budget

and sample size. We prove mathematical guarantees, which demonstrate that when the frequency of categorical values is sufficiently skewed, MD embeddings are both better at matrix recovery and incur lower reconstruction distortion than UD embeddings. Our method is faster to train while requiring far less tuning than other non-uniform embedding layers. In Section 5, we demonstrate that MD embeddings improve both parameter-efficiency and training time in click through rate (CTR) prediction tasks.

Summary of Contributions:

(1) We propose an MD embeddings layer for recommendation systems and provide a novel, mathematical method for sizing the dimension of features with variable popularity that is fast to train, easy to tune, and performs well empirically.

(2) With matrix completion and factorization models, we prove that with sufficient popularity skew, mixed dimension embeddings incur lower distortion when memory-limited and generalize better when data-limited.

(3) For the memory-limited regime we derive the *optimal* feature dimension. This dimension only depends on the feature's *popularity*, the parameter budget, and the *singular-value spectrum* of the pairwise interactions.

II. BACKGROUND & PROBLEM FORMULATION

We review the CTR prediction task here (more details in Appendix). Compared to canonical collaborative filtering (CF), CTR prediction tasks include additional context that can be incorporated to predict user-item interactions. These contextual features are expressed through sets of indices (categorical) and floating point values (continuous). These features can represent arbitrary details about the context of an on-click or personalization event. The i-th categorical feature can be represented by an index $x_i \in \{1, ..., n_i\}$ for $i = 1, ..., \kappa$. In addition to κ categorical features, we also have s scalar features, together producing a dense feature vector $\mathbf{x}' \in \mathbb{R}^s$. Thus, given some $(x_1, ..., x_\kappa, \mathbf{x}') \in ([n_1] \times ... \times [n_\kappa]) \times \mathbb{R}^s$, we would like to predict $y \in \{0, 1\}$, which denotes a click event in response to a particular personalized context.

We use state-of-the-art deep learning recommendation model (DLRM) [11] as an off-the-shelf deep model. Various deep CTR prediction models, including [6], [11]–[15], are powered by memory-intensive embedding layers that utterly dwarf the rest of the model. The trade-off between the size of the embedding layer and the statistical performance seems to be an unavoidable trade-off. Generally these deep models are trained via empirical risk minimization (ERM) and back-propagation. For a given model f_θ (parameterized by θ) the standard practice is to represent categorical features with some indexed embedding layer E. The ERM objective is then:

$\min_{\theta, E} \sum_{i \in \mathcal{D}} \ell\left(f_\theta(\mathbf{x}'_i, E[(x_1, ..., x_\kappa)_i]), y_i\right)$ where the sum is over all data points $\{(x_1, ..., x_\kappa, \mathbf{x}')_i, y_i\}$ in the dataset and the loss function ℓ is taken to be cross entropy for our purposes. Usually, each categorical feature has its own independent embedding matrix: $E[(x_0, ..., x_\kappa)_i] = (E^{(1)}[x_1], ..., E^{(\kappa)}[x_\kappa])$.

Related Works. Recent works have proposed similar but substantially different techniques for non-uniform embedding architectures, particularly for the natural language processing (NLP) domain [16], [17]. Neither of those methods would work out-of-the-box for CTR prediction because they ignore the inherit feature-level structure in CTR that is absent in NLP. We discuss key distinctions in more detail in Appendix.

Other approaches propose neural architecture search (NAS) for RecSys embedding layers is proposed in [18], where generic reinforcement learning algorithms are used to architect the embedding layer. In contrast to computationally expensive NAS, we show that the architecture search over non-uniform embedding layers can be distilled into tuning a *single* hyper-parameter and does not require the heavy-machinery of NAS. This simplification in model search is only possible due to our theoretical framework. Furthermore, in contrast to all previous works with non-uniform embeddings, we theoretically analyze our method. Moreover, past works do not empirically validate the speculated mechanisms by which their methods work.

III. MIXED DIMENSION EMBEDDING LAYER

The MD embedding layer architecture, $\bar{\mathbf{E}}$, consists of k blocks and is defined by $2k$ matrices: $\bar{\mathbf{E}} = (\bar{E}^{(1)}, ..., \bar{E}^{(k)}, P^{(1)}, ..., P^{(k)})$ with $\bar{E}^{(i)} \in \mathbb{R}^{n_i \times d_i}$ and $P^{(i)} \in \mathbb{R}^{d_i \times \bar{d}}$ for $i = 1, ..., k$. Together, $\bar{E}^{(i)}$ and $P^{(i)}$ form the i-th block. In total, there are $n = \sum_{i=1}^{k} n_i$ embedding vectors in the layer. We always treat embedding vectors as row vectors. The $\bar{E}^{(i)}$ can be interpreted as the MD embeddings, and the $P^{(i)}$ are projections that lift the embeddings into a *base dimension* $\bar{d}$ such that $\bar{d} \geq d_i$. The entire layer can be thought of as a single $n \times \bar{d}$ block matrix for which the i-th block is factored at dimension d_i.
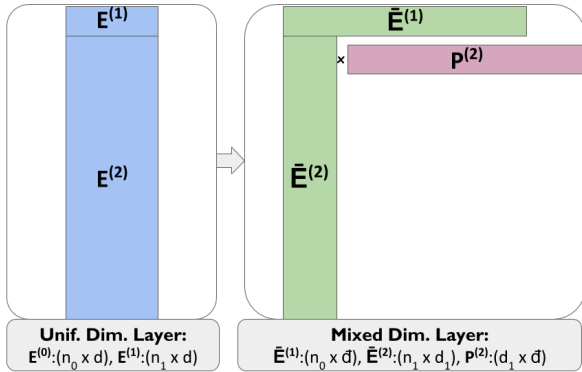

Figure 1: Matrix Architecture for UD and MD Embedding Layers.

Forward propagation for a MD embedding layer is performed by indexing an embedding vector and then projecting it. For example, compute $P^{(1)} \bar{E}_\ell^{(1)}$ for the ℓ-th vector in the first block. Downstream models based on a MD embedding layer

should be sized with respect to $\bar{d}$. If $d_i = \bar{d}$ for any block, the projection $P^{(i)}$ is not needed and may be replaced with an identity mapping. We illustrate this along with the general matrix architecture of a two block MD embedding layer in Fig. 1. The parameter budget (total area) consumed by UD and MD embedding layers is the same, but the parameters are allocated unevenly to different indices in the MD embeddings.

For MD embedding layers, there are two primary architectural decisions to make: (i) ***blocking:*** *how to block n total embedding indices into k blocks?* and (ii) ***sizing:*** *how to size the embedding dimensions* $\mathbf{d} = (d_1, ..., d_k)$? For large-scale CTR, κ is generally on the order of 10 to 100. Standard embedding layers allocate κ UD embedding matrices to these κ features. For MD layers, it is both simple and natural to inherit the block structure as delineated by the task itself. We let $k = \kappa$ and use the same number of MD embedding blocks as categorical features in the CTR prediction task. The MD layer satisfies $\bar{E}^{(i)} \in \mathbb{R}^{n_i \times d_i}$ for $i \in \{1, ..., \kappa\}$. The value range for each categorical feature defines the row counts n_i in the corresponding block of the MD layer. Any re-indexing can trivially be stored in a low-cost length k offset vector. For the Criteo dataset, there are $\kappa = 26$ distinct categorical features, so we produce a MD embedding layer with $k = 26$ blocks. To get meaningful and useful blocks, this blocking scheme depends on the fact that our task has a large number of contextual features k, with value ranges varying from order 10 to order 10 million. Thus, even if feature values are roughly uniformly popular within each feature, the large variation in value ranges leads to a significantly skewed popularity. In contrast to CTR prediction tasks, when using word embeddings in NLP, one cannot block the mixed layer by feature because this inherent structure is absent. Thus, one needs to resort to complex blocking and sizing schemes, such as those proposed in [16], [17]. Furthermore, we found that accuracy significantly drops if the layer is *not* blocked by feature. We hypothesize that embedding projections encode feature-level semantic information when blocking by feature. As for the question of *sizing* the embedding blocks, we defer discussion until after our theoretical analysis.

IV. THEORETICAL FRAMEWORK

As is standard, our theoretical analysis models CF and Rec-Sys tasks with matrix completion and factorization (additional references and all proofs are in Appendix).

Let $M \in \mathbb{R}^{n \times m}$, for $n \geq m$, be an unknown target matrix. Without loss of generality, we also assume M has a *block structure* such

$$M = \begin{bmatrix} M^{(11)} & \cdots & M^{(1k_W)} \\ \vdots & \ddots & \vdots \\ M^{(k_V 1)} & \cdots & M^{(k_V k_W)} \end{bmatrix}$$

that M is comprised of blocks $M^{(i,j)} \in \mathbb{R}^{n_i \times m_j}$ for $1 \leq i \leq k_V$ and $1 \leq j \leq k_W$. When indexing M, we use subscripts, as in M_{kl}, to denote the kl-th scalar entry in M, and superscripts in parenthesis, such as $M^{(i,j)}$, to denote the ij-th block of M (the comma is often omitted in the superscript). Let $\mathrm{rank}(M) = r$ and $\mathrm{rank}(M^{(ij)}) = r_{ij}$. Let $\Omega \subset [n] \times [m]$ denote a sample of indices. Our observations, denoted $\mathbf{\Omega}$, act as a training set: $\mathbf{\Omega} = \{(k, l, M_{kl}) : (k, l) \in \Omega\}$. We say the target matrix

M is *completed* or *recovered* if recovery algorithm $\mathcal{A}$ returns $\mathcal{A}(\mathbf{\Omega}) = M$. We are interested in the probability of recovery event: $\mathbf{Pr}[M = \mathcal{A}(\mathbf{\Omega})]$ for an algorithm $\mathcal{A}$ under a sampling model for Ω. Given both the block-wise structure of M and the MD embeddings, it is straightforward to apply MD. The goal is to train the layer $\bar{\mathbf{E}}$ to represent M with the block structure in $\bar{\mathbf{E}}$ inherited from M. We can train $\bar{\mathbf{E}}$ using stochastic gradient descent (SGD).

Data-Limited & Memory-Limited Regimes. In contextual recommendation engines, there are two primary bottlenecks. In the data-limited regime, (when the number of samples is $o(nr \log n)$) the model does not have enough samples to accurately recover the preference matrix unless a popularity-based approach like MD embeddings is used. In the memory-limited regime (when the space constraint is $o(nr)$), the model has sufficient data to recover the preference matrix but not enough space for the parameters that comprise the embedding layer, which requires us to use fewer parameters than are naively required. We leave analysis of both regimes simultaneously for future work. Because large-scale CTR prediction systems can use up to order 10^9 contextual features and constantly generate data, they are usually in the memory-limited regime.

A. Generalization in the Data-Limited Regime

It is common practice to study a Bernoulli sampling model for Ω [19]–[23], where each entry is revealed independently with some small probability. Below, we extend Bernoulli sampling for the proposed block-wise structure such that sampling probabilities are constant within a block.

Definition: Block-wise Bernoulli sampling *Let* $\Pi \in \mathbb{R}^{k_W \times k_V}$ *be a probability matrix. Let N denote the expected number of observed indices in a training sample. Let $\mathbf{i}(\cdot)$ and $\mathbf{j}(\cdot)$ map each index of M to the index of the block to which it belongs. Each index (k, l) is independently sampled as follows:* $\mathbf{Pr}[(k, l) \in \Omega] = N\Pi_{\mathbf{i}(k),\mathbf{j}(l)}/(n_{\mathbf{i}(k)}m_{\mathbf{j}(l)})$

We use standard matrix completion assumptions. We show that when the training samples are sufficiently skewed, MD embeddings can recover many more matrices than UD embeddings. We use recovery of the matrix as a proxy for good generalization. For brevity, we focus on exact completion for matrices, but it is well-understood how to extend these results to approximate completion and for low-rank tensors (refer to Appendix).

We refer to any algorithm that ignores popularity as *popularity-agnostic* (formalized in Appendix). Under uniform popularity, popularity-agnostic algorithms need $\Theta(rn \log n)$ samples to recover the matrix [20]. We provide an asymptotic lower bound on the sample complexity for matrix recovery under popularity skew by *any* popularity-agnostic algorithm.

Theorem IV.1. *Let M be a target matrix following the block-wise Bernoulli sampling model under probability matrix Π. Let $\varepsilon = \min\{\min_i \frac{1}{n_i} \sum_j \Pi_{ij}, \min_j \frac{1}{m_j} \sum_i \Pi_{ij}\}$.*

(1) Suppose $N = o(\frac{r}{\varepsilon}n \log n)$. Then any algorithm that does not take popularity into account will have asymptotically zero probability of recovering M.

(2) Let C be some non-asymptotic constant independent of n. If $N \geq C(\max_{ij} \frac{r_{ij}}{\Pi_{ij}})n \log n$, then mixed dimension factorization with SGD recovers M with high probability.

Thm IV.1 states that with popularity-agnostic methods, completion of matrix M is bottlenecked by the row or column with lowest popularity. It is impossible to complete rare rows at the same rank as popular rows. When popularity skew is large, the $\frac{1}{\varepsilon}$ factor greatly increases the sample size necessary to complete M. In contrast, MD factorization gets a significant reprieve if low-popularity blocks are also treated as low-rank, implying $\max_{ij} \frac{r_{ij}}{\Pi_{ij}} \ll \frac{r}{\varepsilon}$.

Two block example. The theorems above are more easily interpreted for a special case block matrix consisting of two vertically stacked matrices $M = [M^{(1)}, M^{(2)}]^T$. Let us assume block-wise popularity sampling with $\Pi_1 = 1 - \epsilon$ and $\Pi_2 = \epsilon$ for small $0 < \epsilon < 1/2$, so that we can interpret $M^{(1)}$ as the popular and $M^{(2)}$ as the rare block. For illustrative purposes, assume that r_2 is a small constant and $r_1 \approx \frac{1}{\epsilon}$ is significantly larger. Then popularity-agnostic algorithms suffer a $\frac{1}{\epsilon^2}$ quadratic penalty in sample complexity due to popularity skew, whereas MD factorization only pays a $\frac{1}{\epsilon}$ factor because the rare block is completed at much lower rank.

B. Space Efficiency in the Memory-Limited Regime

To study memory-constrained deployment, we assume that we have sufficient data to complete the target matrix. We are instead constrained by a small parameter budget B. The goal is to optimally allocate parameters to embedding vectors such that we minimize the expected reconstruction over a non-uniformly sampled test set. Under mild assumptions this dimension allocation problem is a convex program and is amenable to closed-form solutions. We prove that the optimal dimension for a given embedding scales with that embedding's popularity in a manner that depends deeply on the spectral properties of the target matrix.

For most applications, popularity skew is present at test time as well as training time. In this setting, the natural loss metric to study is the *popularity-weighted mean square error (MSE)*.

Definition: Popularity-Weighted MSE *Let Π be a probability matrix. Let (k, l) be a test coordinate sampled according to Π. The popularity-weighted MSE is given by $L_\Pi(M, \hat{M}) = \mathbb{E}_{k,l}|M_{kl} - \hat{M}_{kl}|^2 = \sum_{i,j} \frac{1}{n_i m_j}\Pi_{ij}||M^{(ij)} - \hat{M}^{(ij)}||_F^2$.*

Let us now assume that the target matrix is given and that we are trying to optimally size MD embeddings layers, W and V, with respect to popularity-weighted MSE reconstruction. We assume to have a small parameter budget, so that we cannot factor the target matrix exactly. We formulate this optimization as a convex program with linear constraints (we treat the dimensions as continuous — this is a convex relaxation of a hard problem, see Appendix).

Convex program for optimizing mixed dimensions:

$$\min_{d_w, d_v} \left(\min_{W,V} L_\Pi(M, WV^T) \right) \text{ s.t. } \sum_i n_i(d_w)_i + \sum_j m_j(d_v)_j \leq B$$

where $d_w \in \mathbb{R}^{k_W}$ and $d_v \in \mathbb{R}^{k_V}$ denote the dimensions of the embedding blocks of W and V, respectively.

We can obtain a solution using first-order conditions and Lagrange multipliers [24]. Our analysis reveals that under mild assumptions, the optimal dimension to popularity scaling is the functional inverse of the spectral decay of the target matrix (see Thm. IV.2).

Theorem IV.2. *Let M be a block matrix with block-wise spectral (singular value) decay given by $\sigma_{ij}(\cdot)$. Then, the optimal embedding dimensions for MD layers W and V are given by: $(d_w^*)_i = \sum_j d_{ij}^*$, $(d_v^*)_j = \sum_i d_{ij}^*$, where $d_{ij}^* = \sigma_{ij}^{-1}\left(\sqrt{\lambda(n_i + m_j)(n_i m_j)\Pi_{ij}^{-1}}\right)$ and $\sum_{ij}(n_i + m_j)d_{ij}^* = B$.*

When we have a closed-form expression for the spectral decay of M, we can give a closed-form solution in terms of that expression. For illustrative purposes, we give the optimal dimensions for the simple two-block example under *power spectral decay*. A matrix with spectral norm ρ exhibits a singular value spectrum with power spectral decay $\beta > 0$ if the k-th singular values is given by: $\sigma(k) = \rho k^{-\beta}$. Based on the corollary below, the optimal dimension for an embedding vector scales with its popularity based on a fractional power law.

Corollary IV.2.1. *For a vertically stacked two-block matrix, with each block exhibiting a power spectral decay, then $d_1^* \propto (1-\epsilon)^{\frac{1}{2\beta}}$ and $d_2^* \propto \epsilon^{\frac{1}{2\beta}}$*

C. Large-scale CTR Prediction is Memory-Limited

Labeled training data is easy to acquire in most large-scale CTR prediction systems because one can directly observe user engagement (or lack thereof) with personalized content. The embedding layer's memory footprint ends up being the primary bottleneck. In this situation, the results of Thm IV.2 yield appropriate guidelines for the optimal dimension for each embedding vector. The unavoidable difficulty is that one needs to know the spectrum of the target matrix to know the optimal dimension. One solution is to train an enormous embedding table with an enormous data set, thereby obtaining the spectrum, and then factoring (or re-training from scratch) at the optimal size. However, this solution still requires enormous resource usage during training. Alternatively, we can still leverage the insight of our theoretical framework to efficiently find good mixed embedding dimensions. Most spectral decays, whether they are flat or steep, can be decently well fit by a power law (so much so that there is a large literature dedicated to *falsifying* power-laws even when the numerical fit appears reasonable [25]). Varying the temperature adequately captures the trend of the decay. By *a priori* assuming that the spectral decay is a power law, we only have to tune one hyper-parameter over a narrow range that requires only exploring a small number of values.

Power-law Sizing Scheme. Here we define *block-level probability* **p**. Let **p** be a k-dimensional probability vector (recall k is the # of blocks in the embedding layer) such that $\mathbf{p}_i$ is the average query probability over rows in the i-th block. When blocks exactly correspond to features, as in our CTR experiments, then $\mathbf{p}_i = \frac{1}{n_i}$ because each one row per block is queried per inference based on the value of each feature. More generally, under a block sampling structure Π, $\mathbf{p}_i = \sum_j \Pi_{ij}$.

Algorithm 1 Popularity-Based Dimension Sizing

Input: Baseline dimension $\bar{d}$ and fixed temperature $0 \le \alpha \le 1$
Input: Probability vector **p**
Output: Dimension assignment vector **d**
$\lambda \leftarrow \bar{d}\|\mathbf{p}\|_\infty^{-\alpha}$ ▷ Compute scalar scaling factor
$\mathbf{d} \leftarrow \lambda \mathbf{p}^\alpha$ ▷ Component-wise exponent

---

Alg. 1 shows code for the scheme. We use a temperature parameter $\alpha > 0$ to control the degree to which popularity influences the embedding dimension (as a simplification over using decay parameter β). As α increases, so does the popularity-based skew in the embedding dimension. At $\alpha = 0$, the embedding dimensions are all uniform. At $\alpha = 1$, the embedding dimensions are proportional to their popularity.

V. Experiments

We measure memory in terms of the number of 32-bit floating point parameters in the model. Since different embedding base dimensions imply different widths in first hidden layers of the downstream deep model, for fairness, our parameter counts include both the embedding layer and all model parameters (recall that the parameter count is overwhelmingly dominated by embeddings). We report statistical performance in terms of cross entropy loss. Accuracy is reported in Appendix (along with other experimental details). DLRM with uniform $d = 32$ embeddings obtains an accuracy of $\sim 79\%$, close to state-of-the-art for this dataset [11]. We sweep parameter budgets from a lower bound given by 1 parameter per embedding vector ($d = 1$) to an upper bound given by the memory limit of a 16GB GPU. The parameters are allocated to embeddings according to the α-parameterized rule proposed in Alg. 1.

MD embeddings with $\alpha = 0.3$ produce a learning curve on par to that of $d = 32$ UD embeddings using a total parameter count equivalent to $d = 2$ UD (Fig. 2a), yielding a $16\times$ reduction in parameters. We can see that using MD embedding layers improves the memory-performance frontier at each parameter budget (Fig. 2b). The optimal temperature (α) is dependent on the parameter budget, with higher temperatures leading to lower loss for smaller budgets. Embeddings with $\alpha = 0.4$ obtain performance on par with uniform embeddings using $16\times$ fewer parameters. Embeddings with $\alpha = 0.2$ modestly outperform UD embeddings by an accuracy of 0.1% using half as many parameters. For RecSys, small improvements in accuracy are still important when volume is sufficiently large. The std. dev. estimates indicate that this gain is significant. MD embeddings not only improve prediction quality for a given parameter budget, but also for a given training time. MD embeddings can train $> 2\times$ faster than UD embeddings at a given test loss (Fig. 2c). This is possible because for a given loss, the MD layer uses far fewer parameters than a UD layer. The faster training we observe is likely due to superior bandwidth efficiency and caching on the GPU, enabled by the reduced memory footprint. We run all of our models on a single GPU with identical hyperparameters (including batch size) across all α (more details in Appendix).

Case Study: Collaborative Filtering. Because context-free CF only includes two features, users and items, it is easier to
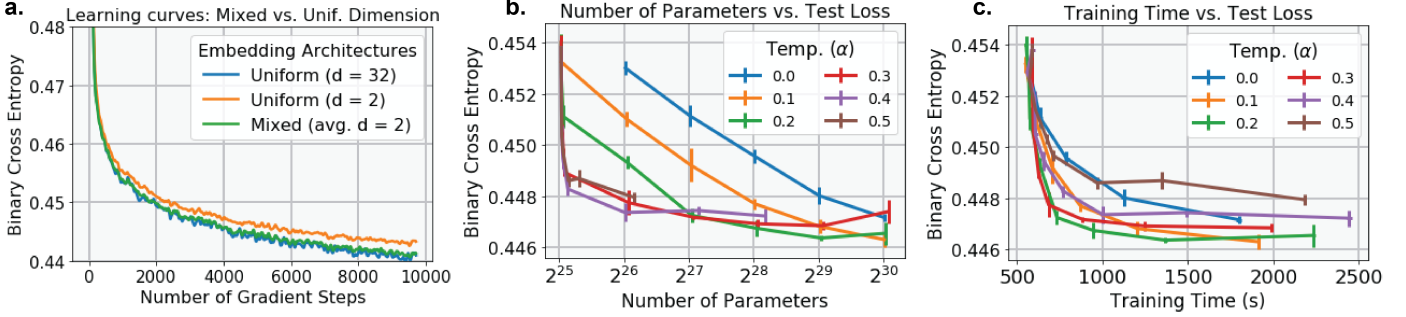
Figure 2: CTR prediction results for MD embeddings on Criteo dataset using DLRM. Implementation is available as part of an open-source project on GitHub: `facebookresearch/dlrm`. Fig. 2a (left): Learning curves for selected emb. arch. Fig. 2b (center): Loss vs. # param. for varying α. Fig 2c (right): Train time vs. loss for varying α
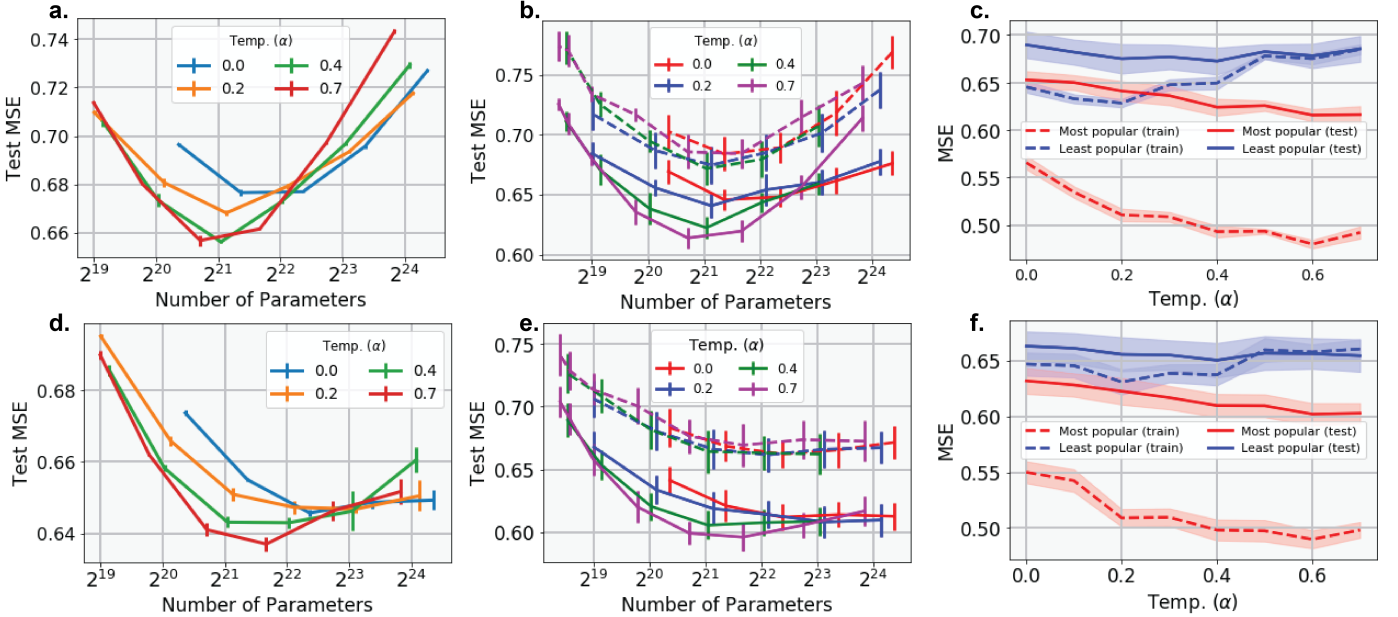

Figure 3: Matrix Factorization (top row; a-c) and NCF (bottom row; d-f) for collaborative filtering on the MovieLens dataset. Fig. 3a & 3d (right): MSE vs. # params for varying α. Fig. 3b & 3e (center): MSE vs. # params for varying α. Dashed lines correspond to test samples that contain the one-third least popular items. Solid lines correspond to test samples that contain the one-third most popular items. Fig. 3c & 3f (left): Generalization for popular (red) and rare (blue) items. Dashed lines correspond to training loss and solid lines correspond to test loss.

gain insight into the effects of MD embeddings. Matrix factorization (MF) is a typical algorithm for CF where the matrix factors are embeddings. We also include Neural Collaborative Filtering (NCF) models [26] to ensure that trends still hold when non-linear layers are introduced. Because we only have two features for this case study (users and items) we slightly modify our blocking approach. We sort users and items by empirical popularity, and uniformly partition them into block matrices. Then we can apply mixed dimensions within the users and items based on partitions (more details in Appendix).

MD embeddings substantially outperform UD embeddings, especially at low parameter budgets (Figs. 3a & 3d). For Figs. 3b & 3e, we partition *item* embeddings (since they often the focal point of recommendation) into the one-third most and least popular items (by empirical training frequency). These results show that non-zero temperature (α) improves test loss on popular items and performs on par with or slightly better than UD embeddings on rare items. We report generalization curves for the popular and rare items in Figs 3c and 3f. We fix a parameter budget (2^{21}), vary the temperature (α), and plot training and test loss for both popular and rare item

partitions. Allocating more parameters to popular items by increasing temperature (α) decreases both training and test loss. On the other hand, allocating more parameters to rare items by decreasing temperature (α) only decreases training loss but not test loss, indicating that the additional parameters on the rare embeddings are wasteful. Uniform parameter allocation, agnostic to popularity, is inefficient. A majority of parameters are severely underutilized on rare embeddings, whereas popular embeddings could still benefit from increased representational capacity.

VI. CONCLUSION

We show that MD embeddings greatly improve both parameter efficiency and training time. Through our case study, we demonstrate systematic and compelling empirical evidence that MD embeddings work by improving capacity for learning popular embeddings without compromising rare embeddings. Our theoretical framework is the first to mathematically explain how this occurs, and our experiments are the first to validate the phenomena.

REFERENCES

[1] Björn Barz and Joachim Denzler. Hierarchy-based image embeddings for semantic image retrieval. In *Proc. IEEE Winter Conf. Applications of Computer Vision*, pages 638–647. IEEE, 2019.

[2] Mariya I. Vasileva, Bryan A. Plummer, Krishna Dusad, Shreya Rajpal, Ranjitha Kumar, and David Forsyth. Learning type-aware embeddings for fashion compatibility. In *Proc. European Conf. Computer Vision*, 2018.

[3] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[4] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proc. 27th Int. Conf. Computational Linguistics*, pages 1638–1649, 2018.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proc. 1st Workshop Deep Learning Recommender Systems*, pages 7–10. ACM, 2016.

[7] Jongsoo Park et al. Deep learning inference in Facebook data centers: Characterization, performance optimizations and hardware implications. *CoRR*, abs/1811.09886, 2018.

[8] Xiaorui Wu, Hong Xu, Honglin Zhang, Huaming Chen, and Jian Wang. Saec: Similarity-aware embedding compression in recommendation systems. *CoRR*, abs/1903.00103, 2019.

[9] Zi Yin and Yuanyuan Shen. On the dimensionality of word embedding. In *Proc. Advances in Neural Information Processing Systems*, 2018.

[10] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. Practice on long sequential user behavior modeling for click-through rate prediction. *CoRR*, abs/1905.09248, 2019.

[11] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.

[12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. DeepFM: a factorization-machine based neural network for ctr prediction. *CoRR*, abs/1703.04247, 2017.

[13] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xDeepFM: Combining explicit and implicit feature interactions for recommender systems. In *Proc. 24th Int. Conf. Knowledge Discovery & Data Mining*, pages 1754–1763. ACM, 2018.

[14] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. *CoRR*, abs/1809.03672, 2018.

[15] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proc. 24th Int. Conf. Knowledge Discovery & Data Mining*, pages 1059–1068. ACM, 2018.

[16] Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In *Advances in Neural Information Processing Systems*, 2018.

[17] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*, 2018.

[18] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. Neural input search for large scale recommendation models. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2387–2397, 2020.

[19] Emmanuel J Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

[20] Emmanuel J Candes and Yaniv Plan. Matrix completion with noise. *Proc. IEEE*, 98(6):925–936, 2010.

[21] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 2009.

[22] Emmanuel J Candes and Yaniv Plan. Tight oracle inequalities for low-rank matrix recovery from a minimal number of noisy random measurements. *IEEE Transactions on Information Theory*, 57(4):2342–2359, 2011.

[23] Ruoyu Sun and Zhi-Quan Luo. Guaranteed matrix completion via non-convex factorization. *IEEE Transactions on Information Theory*, 62(11):6535–6579, 2016.

[24] David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*, volume 2. Springer.

[25] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[26] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proc. 26th Int. Conf. World Wide Web*, pages 173–182, 2017.

[27] Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *The Journal of Machine Learning Research*, 16:3367–3402, 2015.

[28] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[29] Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. *CoRR*, abs/1905.01395, 2019.

[30] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. *CoRR*, abs/1907.06902, 2019.

[31] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, pages 3203–3209, 2017.

[32] Jicong Fan and Jieyu Cheng. Matrix completion by deep matrix factorization. *Neural Networks*, 98:34–41, 2018.

[33] Moritz Hardt. Understanding alternating minimization for matrix completion. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 651–660. IEEE, 2014.

[34] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proc. 45th annual ACM symposium on Theory of computing*. ACM, 2013.

[35] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, pages 7411–7422, 2019.

[36] Sahand Negahban and Martin J Wainwright. Restricted strong convexity and weighted matrix completion: Optimal bounds with noise. *Journal of Machine Learning Research*, 13:1665–1697, 2012.

[37] Raghu Meka, Prateek Jain, and Inderjit S Dhillon. Matrix completion from power-law distributed samples. In *Advances in neural information processing systems*, 2009.

[38] Guangcan Liu, Qingshan Liu, and Xiaotong Yuan. A new theory for matrix completion. In *Advances in Neural Information Processing Systems*, 2017.

[39] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[40] Fabián P Lousame and Eduardo Sánchez. A taxonomy of collaborative-based recommender systems. In *Web Personalization in Intelligent Environments*, pages 81–117. Springer, 2009.

[41] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. *IEEE transactions on information theory*, 56(6):2980–2998, 2010.

[42] Sriram Vishwanath. Information theoretic bounds for low-rank matrix completion. In *2010 IEEE Int. Symposium on Information Theory*, pages 1508–1512. IEEE, 2010.

[43] Benjamin Recht. A simpler approach to matrix completion. *Journal of Machine Learning Research*, 12(Dec):3413–3430, 2011.

[44] Martin Andrews. Compressing word embeddings. *CoRR*, abs/1511.06397, 2015.

[45] Prasad Bhavana, Vikas Kumar, and Vineet Padmanabhan. Block based singular value decomposition approach to matrix factorization for recommender systems. *Pattern Recognition Letters*, abs/1907.07410, 2019.

[46] Prasanna Sattigeri and Jayaraman J. Thiagarajan. Sparsifying word representations for deep unordered sentence modeling. In *Proc. 1st Workshop on Representation Learning for NLP*, pages 206–214.

[47] Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. Sparse word embeddings using *l*1 regularized online learning. In *Proc. 25th Int. Joint Conf. Artificial Intelligence*, 2016.

[48] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Proc. Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[49] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *CoRR*, abs/1803.03635, 2018.

[50] Maxim Naumov, Utku Diril, Jongsoo Park, Benjamin Ray, Jedrzej Jablonski, and Andrew Tulloch. On periodic functions as regularizers for quantization of neural networks. *CoRR*, abs/1811.09862, 2018.

[51] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. Value-aware quantization for training and inference of neural networks. In *Proc. European Conf. Computer Vision*, pages 580–595, 2018.

[52] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H Chi. Learning multi-granular quantized embeddings for large-vocab categorical features in recommender systems. In *Companion Proceedings of the Web Conference 2020*, 2020.

[53] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. *CoRR*, abs/1711.01068, 2017.

[54] Julien Tissier, Amaury Habrard, and Christophe Gravier. Near-lossless binarization of word embeddings. *CoRR*, abs/1803.09065, 2018.

[55] Joan Serrà and Alexandros Karatzoglou. Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 279–287, 2017.

[56] Jiaxi Tang and Ke Wang. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2289–2298, 2018.

[57] Josh Attenberg, Kilian Weinberger, Anirban Dasgupta, Alex Smola, and Martin Zinkevich. Collaborative email-spam filtering with the hashing trick. In *Proc. 6th Conf. Email and Anti-Spam*, 2009.

[58] Jinyang Gao, Beng Chin Ooi, Yanyan Shen, and Wang-Chien Lee. Cuckoo feature hashing: Dynamic weight sharing for sparse analytics. In *IJCAI*, pages 2135–2141, 2018.

[59] Alexandros Karatzoglou, Alex Smola, and Markus Weimer. Collaborative filtering on a budget. In *Proc. 13th Int. Conf. Artificial Intelligence and Statistics*, pages 389–396, 2010.

[60] Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan V. Oseledets. Tensorized embedding layers for efficient model compression. *CoRR*, abs/1901.10787, 2019.

[61] Kaiyu Shi and Kai Yu. Structured word embedding for low memory neural network language model. In *Proc. Interspeech*, 2018.

[62] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[63] Nvidia. Tesla V100 GPU architecture white paper. 2017.

[64] F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5, 2015.

[65] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. In *Proc. Int. Conf. Learning Representations*, 2018.

[66] Florian Strub, Romaric Gaudel, and Jérémie Mary. Hybrid recommender system based on autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 11–16, 2016.

[67] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. 13th Int. Conf. Artificial Intelligence and Statistics*, pages 249–256, 2010.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[69] Jerzy K Baksalary, Peter Semrl, and George PH Styan. A note on rank additivity and range additivity. *Linear algebra and its applications*, 237:489–498, 1996.

[70] Yongge Tian. Rank equalities for block matrices and their moore-penrose inverses. *Houston journal of mathematics*, 30(2):483–510, 2004.

[71] George Matsaglia and George PH Styan. Equalities and inequalities for ranks of matrices. *Linear and multilinear Algebra*, 2(3):269–292, 1974.

[72] Shouyuan Chen, Michael R Lyu, Irwin King, and Zenglin Xu. Exact and stable recovery of pairwise interaction tensors. In *Advances in neural information processing systems*, pages 1691–1699, 2013.

[73] Hongyang Zhang, Vatsal Sharan, Moses Charikar, and Yingyu Liang. Recovery guarantees for quadratic tensors with sparse observations. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

[74] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[75] Brian Dawkins. Siobhan's problem: the coupon collector revisited. *The American Statistician*, 45(1):76–82, 1991.

[76] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[77] Ivan Markovsky. Structured low-rank approximation and its applications. *Automatica*, 44(4):891–909, 2008.

[78] Richard Courant and Fritz John. *Introduction to calculus and analysis I*. Springer Science & Business Media, 2012.

APPENDIX

The appendix is organized as follows. We begin with an extended discussion, including additional related works and background references in A. We include supplementary experiments and reproducibility details concerning our empirical work in B. Finally, we include mathematical details including assumptions, theorems, and proofs in C.

SUPPLEMENTARY & EXTENDED DISCUSSION

A. Representation Learning

Embedding-based approaches, such as matrix factorization (MF) [27]–[29] or neural collaborative filtering (NCF) [26], [30]–[32], are among the most computationally efficient solutions to CF and matrix completion. The simplest model for canonical CF is MF, which treats embeddings as factorizing the target matrix directly and comes with nice theoretical guarantees. Direct optimization over embedding layers is a non-convex problem. However, due to specific problem structure, many simple algorithms, including first-order methods, have been proven to find global optima (under mild assumptions) [23], [33], [34]. Neural collaborative filtering (NCF) models, which make use of non-linear neural processing, are more flexible options. While NCF models often lack the theoretical guarantees offered by MF, they usually perform mildly better on real-world datasets. In CTR prediction, it is common to use embedding layers to represent categorical features and have various neural layers to process the various embeddings.

B. Regularization

We note that in many instances, embeddings for collaborative filtering tasks are usually trained with some type of norm-based regularization for the embedding vectors. While this particular form of regularization works well for small-scale embeddings, it is non-trivial to scale it for large-scale embeddings. For many standard loss functions, only the embeddings queried on the forward pass have non-zero gradients on the backward pass.

Using sparse updates for the embedding tables is essentially mandatory for efficient training at large scale. Thus, contrary to popular belief in academic circles, large-scale embeddings are often trained without norm-based regularization which are incompatible with sparse updates. This is because the gradient update when using a regularization term should back-propagate to every embedding vector in the layer, rather than just those queried in the forward pass. Because this technique is not feasible in large-scale CTR prediction, we explicitly do not use embedding regularization in our collaborative filtering task. Furthermore, we note that from the perspective of popularity skew, that embedding norm regularization actually implicitly penalizes rare embeddings more than popular ones, since a larger fraction of training updates only contain norm penalty signal for rare embeddings than popular ones. This is an interesting connection that could be explored in future work, but it does not achieve the stated goal of parameter reduction.

C. Non-uniform Embeddings

Recent works have proposed similar but distinct non-uniform embedding architectures, particularly for the natural language processing (NLP) domain [16], [17]. As mentioned in the main text, there are substantial differences between those methods and ours. We emphasize that neither of those methods would work out-of-the-box for CTR prediction. Critically, in contrast to NLP, CTR embeddings encode categorical values for individual features, and thus come with feature-level structure that should not be ignored when architecting the embedding layer. In NLP, embeddings represent words and can be thought of a single large bag of values — in contrast to representing the various categorical values a particular feature can take on. We discover that ignoring this feature-level structure in the embedding layer adversely affects performance, dropping accuracy by $> 1\%$. For this reason, the sorted blocking technique introduced in [17] is not effective in CTR prediction. Additionally, embedding layers in NLP are pre-trained from unsupervised language corpus, unlike in RecSys, which means that clustering and factoring the embeddings as in [16] prior to training is not feasible in CTR prediction.

Furthermore, in contrast to previous works, we theoretically analyze our method. Moreover, past methods do not even empirically validate the speculated mechanisms by which their methods work. For example, in [17], authors claim their proposed architecture, "reduces the capacity for less frequent words with the benefit of reducing overfitting to rare word." While the proposed method works well on benchmarks, the claim that the method reduces overfitting is not supported. As shown in [9], [35], embedding overfitting depends critically on the training algorithm and even the model atop the embeddings. In fact, when training is properly tuned, embeddings are quite resilient to overfitting, which means the claim made in [17] is far from self-evident. It is more accurate to view rare embeddings as wasting parameters rather than overfitting them.

Non-uniform and deterministic sampling have been discussed in the matrix completion literature [36]–[38], but only in so far as how to correct for popularity so as to improve statistical recovery performance, or build theoretical guarantees for completion under deterministic or non-uniform sampling.

D. Collaborative Filtering & Matrix Completion

CF tasks come in many variations and have a large mass scientific literature, with good reviews of classical algorithms and approaches provided in [39], [40]. Related to CF is the simplified matrix completion problem [19]–[21], [23], [27], [41]–[43].

E. Memory-Efficient Embeddings

The techniques to decrease the memory consumed by embedding tables can be roughly split into two high-level classes: (i) compression algorithms and (ii) compressed architectures. Simple *offline* compression algorithms include post-training quantization, pruning or low-rank SVD [44]–[47]. *Online* compression algorithms, including quantization-aware training, gradual pruning, and periodic regularization, are somewhat less popular in practice because of the additional complications they add to already intricate deep model training methods. [48]–[52]. Model distillation techniques [53]–[56] are another form of compression and can have online and offline variants. On the other hand, compressed architectures have the advantage of not only reducing memory requirements for inference time, but also at training time. This is the approach followed by hashing-based and tensor factorization methods [57]–[61].

F. Forward Propagation with Mixed Dimension Embeddings

Forward propagation for a MD embedding layer can be summarized in Alg. 2. The steps involved in this algorithm are differentiable, therefore we can perform backward propagation through this layer and update matrices $\bar{E}^{(i)}$ and $P^{(i)}$ accordingly. We note that Alg. 2 may be generalized to support multi-hot lookups, where embedding vectors corresponding to z query indices are fetched and reduced by a differentiable operator, such as add, multiply or concatenation.

Algorithm 2 Forward Propagation

Input: Index $x \in [n]$
Output: Embedding vector $\mathbf{e}_x$
$i \leftarrow 0$ and $t \leftarrow 0$
while $t + n_i < x$ **do**   ▷ Find offset t and sub-block i
    $t \leftarrow t + n_i$
    $i \leftarrow i + 1$
end while
$\mathbf{e}_x \leftarrow \bar{E}^{(i)}[x - t]P^{(i)}$ ▷ Construct an embedding vector

---

G. Popularity Histograms

The key idea of the MD embeddings is to address the skew in the popularity of objects appearing in a dataset. To illustrate it we present popularity histograms of accesses across all features for the Criteo Kaggle dataset in Fig. 4a and individually for users and items features for the MovieLens dataset in Fig. 4c and 4b, respectively.
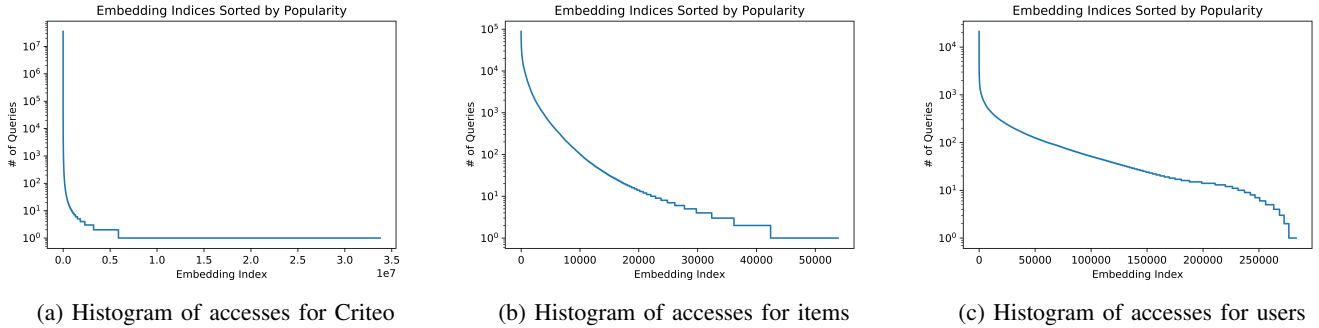
(a) Histogram of accesses for Criteo   (b) Histogram of accesses for items   (c) Histogram of accesses for users

Figure 4: Popularity skew in real-world datasets.

(a) Effect of num. of equiparititons on CF with MD   (b) Effect of num. of equiparititons on CF with UD
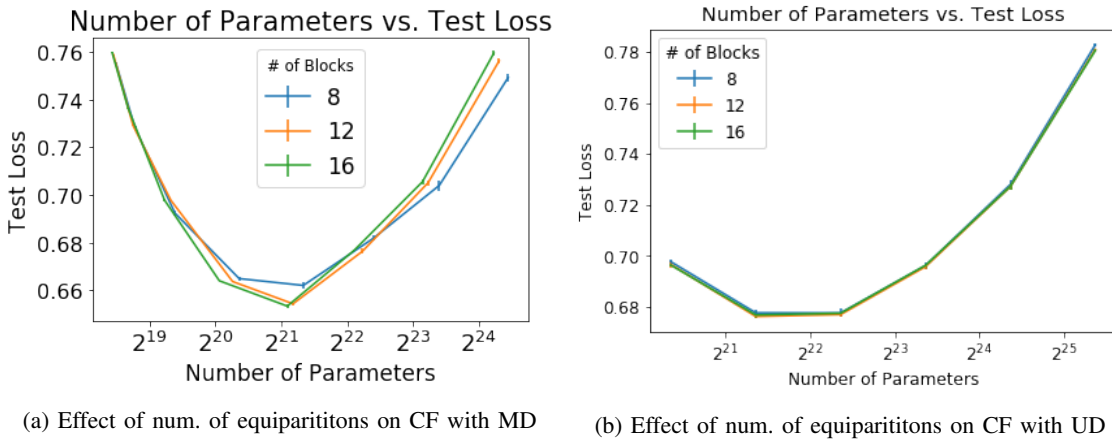
Figure 5: Effect of num. of equiparititons on CF

EXPERIMENTAL DETAILS & SUPPLEMENTARY EXPERIMENTS

In this section we provide detailed protocols for the experiments in Section 5 of the main text. All code is implemented using the Pytorch [62] library. All algorithms are run as single GPU jobs on Nvidia V100s [63]. All confidence intervals reported are standard deviations obtained from 5 replicates per experiment with different random seeds. In Table I we summarize the datasets and models used.

H. Collaborative Filtering Case Study

Recall that CF tasks require a different blocking scheme than the one presented in the main text because we only have 2 categorical features. These features have corresponding embedding matrices $W \in \mathbb{R}^{n \times d}$ and $V \in \mathbb{R}^{m \times d}$, for users and items, respectively. To size the MD embedding layer we apply MDs within individual embedding matrices by partitioning them. We block W and V separately. First, we sort and re-index the rows based on empirical row-wise frequency: $i < i' \implies f_i \geq f_{i'}$, where f_i is the frequency that a user or item appears in the training data[1]. Then, we partition each embedding matrix into k blocks such that the sum of frequencies in each block is equal.

[1]Sorting and indexing can be done quickly on a single node as well as in the distributed settings.

For each of the k blocks, the total popularity (i.e. probability that a random query will include that row) for each block is constant (see Alg. 3). Then, for a given frequency $\mathbf{f}$ the k-equipartition is unique and is simple to compute. In our experiments, we saw that setting k anywhere in the $[8, 16]$ range is sufficient to observe the effects induced by MDs, with diminishing effect beyond these thresholds (Fig. 5).

Algorithm 3 Partitioning Scheme for CF

Input: Desired number of blocks k
Input: Row-wise frequencies vector $\bar{\mathbf{f}}$
Output: Offsets vector $\mathbf{t}$
$\mathbf{f} \leftarrow \text{sort}(\bar{\mathbf{f}})$ ▷ Sort by row-wise frequency
Find offsets t_i such that $\sum_{l=t_i}^{t_{i+1}} f_l = \sum_{l=t_j}^{t_{j+1}} f_l$ for $\forall i, j$

In our experiments we use the full 27M MovieLens dataset [64]. We train at learning rate 10^{-2}, found in the same manner as for CTR prediction. For consistency with CTR prediction, we also used the Amsgrad optimizer [65]. We train for 100 epochs, taking the model with the lowest validation loss. We early terminate if the model does not improve after 5 epochs in a row. We use a batch size of 2^{15} in order to speed-up training. We did not observe significant differences between this and smaller batch sizes. Our reported performance, in terms of MSE, are comparable to those elsewhere reported in the literature [66].

Dataset	Tasks	# Samples	# Categories	Models Used
MovieLens	CF	27M	330K	MF, NCF
Criteo Kaggle	CTR	40M	32M	DLRM

Table I: Datasets, tasks and models

For initialization we use the uniform Xavier initialization (for consistency with CTR prediction). Also, for the NCF model, we use a 3-layer MLP with hidden layer dimensions $128-128-32$. We used default LeakyReLU activations.

For the item embedding partitioning (in Fig. 3), we partitioned the item embeddings by empirical popularity mass. This means that the top third item embeddings represent to order 10^3 items, whereas the bottom third item embeddings represent order 10^5 items. Thus, the top third and bottom third partitions have the same empirical popularity mass, not the same number of items.

I. CTR Prediction

From the perspective of this work, using MD embedding layers on real CTR prediction data with modern deep recommendation models is an important experiment that shows how MD embedding layers might scale to real-world recommendation engines.
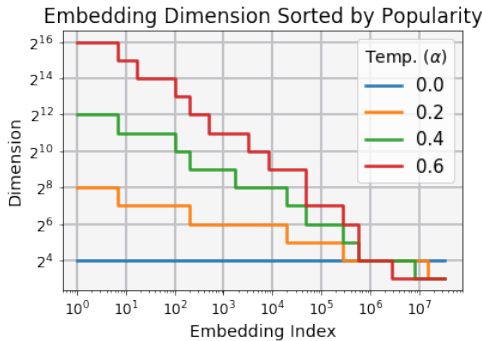

Figure 6: Embedding parameters for 26 categorical features allocated at different temperatures (α) for the same parameter budget on the Criteo dataset. Higher temperatures results in higher dimensions for popular embeddings and $\alpha = 0$ is uniform dimensions. See Section 4 and Alg.1 for more details concerning the assignment scheme. The dimensions are rounded to powers of 2.

In our experiments we use state-of-the-art DLRM [11] and the Criteo Kaggle dataset. We determined the learning rates, optimizer, batch size, and initializations scheme by doing a simple grid search sweep on *uniform* dimension embeddings of dimension 32. *Ultimately, we used Amsgrad with a learning rate of 10^{-3}, a batch size of 2^{12}, and a uniform Xavier initialization for all CTR experiments reported in the main text.* As is customary in CTR prediction tasks, we train for only one epoch [11]. Examples of parameter allocation based on Alg. 1 can be found in Fig. 6.

For learning rates, we tried $\lambda \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. 10^{-3} was best. We tried Amsgrad, Adam, Adagrad, and SGD optimizers. Amsgrad was best.

For batch size, we tried powers of 2 from 2^5 to 2^{12}. The batch size hardly affected the memory footprint on the GPU, since the
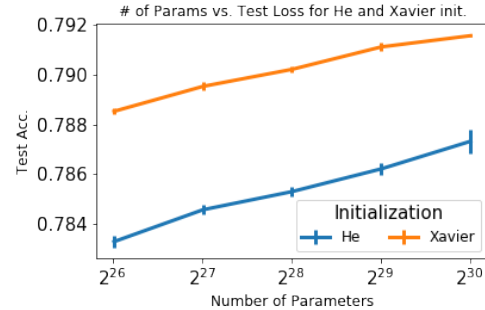

Figure 7: Xavier and He (fan-out) initializations for DLRM with UD embeddings at various parameter budgets
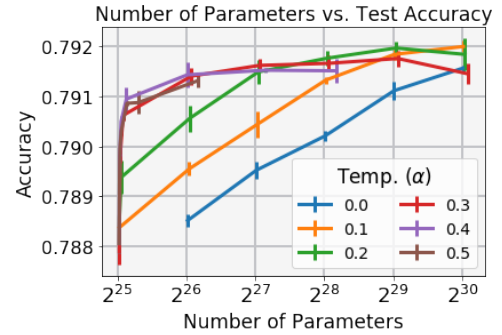

Figure 8: Accuracy on CTR Prediction for varying temp. (α)
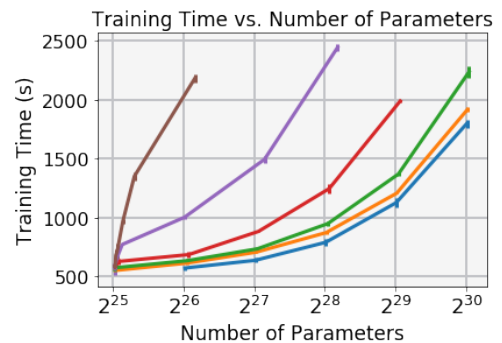

Figure 9: Training time vs. parameter counts for varying temperature (legend same as Fig. 8)

majority of memory is spent on the embedding layer, which is only sparsely queried on each forward pass in the batch. We also found that performance was largely invariant in batch size, so we selected batch size of 2^{12}.

For initialization schemes, we tried uniform Xavier [67], uniform He (fan-in) and uniform He (fan-out) [68]. We initialized all neural network parameters, including embedding matrices, according to the same scheme. We found that He fan-in resulted in severe training instability. Xavier outperformed He fan-out by

a considerable margin (Fig. 7).

We also report the accuracy for our CTR prediction experiments (Fig. 8). The curves are like a reflection of the cross entropy loss in the main text.

Finally, we report training time vs. parameter counts for varying temperatures (Fig. 9).

J. On the Range of Temperatures (α)

Recall that we use a power-law inspired embedding dimension sizing scheme. In the main text, we emphasize that α should be in the range $(0, 1)$. In principle, one could pick an $\alpha > 1$, but since it is natural to assume that there is diminishing returns to the embedding dimension of a feature, it should follow that such a choice is poor. An $\alpha > 1/2$ would imply a sub-linear spectral decay which is rarely the case in embeddings learned form real-world data. This coincides with out experiments, where the best α were actually below $1/2$.

THEORETICAL DETAILS

We proceed to present proofs of theorems as well as additional results that did not fit into the main text of the paper. First, we describe the details of the mathematical optimization procedure used in the proofs. Then, we discuss the extension of our method from the matrix case to the tensor case. Finally, we enumerate the details/assumptions and give the proofs for our theorems.

K. Block-wise MD Factorization

The first point to address is the specifics of the SGD-variant used to solve the MD factorization. We adopt the particular variation of the SGD algorithm for matrix factorization proposed in [23] and assume that block structure is known or has been estimated. We show that under rank additivity assumption it can be applied to factor the blocks of the matrix M independently and yet construct a rank-r approximation for it. Note that in this scenario the projections do not need to be free parameters (see Alg. 4).

a) Two block example: We illustrate block-wise MD factorization on $2n \times m$ two block rank-r matrix

$$M = \begin{bmatrix} M^{(1)} \\ M^{(2)} \end{bmatrix} = \begin{bmatrix} W^{(1)}V^{(1)^T} \\ W^{(2)}V^{(2)^T} \end{bmatrix} = \begin{bmatrix} W^{(1)}P_W^{(1)} \\ W^{(2)}P_W^{(2)} \end{bmatrix} \begin{bmatrix} V^{(1)^T} \\ V^{(2)^T} \end{bmatrix} \text{ with}$$

$W^{(1)} \in \mathbb{R}^{n \times r_1}, V^{(1)} \in \mathbb{R}^{m \times r_1}, W^{(2)} \in \mathbb{R}^{n \times r_2}, V^{(2)} \in \mathbb{R}^{m \times r_2}$ obtained by Alg. 4, projections $P_W^{(1)} = [I, 0] \in \mathbb{R}^{r_1 \times r}$, $P_W^{(2)} = [0, I] \in \mathbb{R}^{r_2 \times r}$ defined by construction and I being an identity matrix.

b) Four block example: We extend the example to $2n \times 2m$ four block rank-r matrix below

$$M = \begin{bmatrix} M^{(11)} M^{(12)} \\ M^{(21)} M^{(22)} \end{bmatrix} = \begin{bmatrix} \bar{W}^{(1)}P_W^{(1)} \\ \bar{W}^{(2)}P_W^{(2)} \end{bmatrix} \begin{bmatrix} (\bar{V}^{(1)}P_V^{(1)})^T \\ (\bar{V}^{(2)}P_V^{(2)})^T \end{bmatrix}$$

where rank-r_{ij} block $M^{(ij)} = W^{(ij)}V^{(ij)^T}$ factors and

$$\bar{W}^{(1)} = [W^{(11)}, W^{(12)}] \in \mathbb{R}^{n \times r_{11} + r_{12}}$$

$$\bar{W}^{(2)} = [W^{(21)}, W^{(22)}] \in \mathbb{R}^{n \times r_{21} + r_{22}}$$

$$\bar{V}^{(1)} = [V^{(11)}, \quad V^{(21)}] \in \mathbb{R}^{n \times r_{11} + r_{21}}$$

Algorithm 4 Block-wise Mixed Dim. Factorization

Input: Partially masked target block matrix $\mathcal{P}_\Omega(M)$.
Input: The blocks $M^{(ij)}$ with block-wise rank r_{ij}.
Output: MD embeddings $\bar{W}, \bar{V}$.
$W^{(i,j)}, V^{(i,j)} \leftarrow \mathbf{SGD}(\mathcal{P}_\Omega(M^{(ij)}))$ ▷ Factor each block
for $1 \le i \le k_W$ **do**
  $\bar{W}^{(i)} \leftarrow [W^{(i,1)}, ..., W^{(i,k_V)}] \in \mathbb{R}^{n_i \times d_w^i}$ ▷ Assemble $\bar{W}$
  $d_w^i \leftarrow \sum_{j=1}^{k_V} r_{ij}$ ▷ Construct projection P
  $s_w^i \leftarrow \sum_{l=1}^{i-1} \sum_{j=1}^{k_V} r_{lj}$
  $t_w^i \leftarrow \sum_{l=i+1}^{k_W} \sum_{j=1}^{k_V} r_{lj}$
  $P_W^{(i)} \leftarrow [0_{d_w^i \times s_w^i}, I_{d_w^i \times d_w^i}, 0_{d_w^i \times t_w^i}] \in \mathbb{R}^{d_w^i \times r}$
end for
for $1 \le j \le k_V$ **do**
  $\bar{V}^{(j)} \leftarrow [V^{(1,j)}, ..., V^{(k_W, j)}] \in \mathbb{R}^{m_j \times d_v^j}$ ▷ Assemble $\bar{V}$
  $d_v^j \leftarrow \sum_{i=1}^{k_W} r_{ij}$ ▷ Construct projection P
  $s_{ij} \leftarrow \sum_{l=1}^{i-1} r_{li} + \sum_{l=1}^{j-1} r_{il}$
  $t_{ij} \leftarrow \sum_{l=j+1}^{k_V} r_{il} + \sum_{l=i+1}^{k_W} r_{lj}$

$$P_V^{(j)} \leftarrow \begin{bmatrix} 0_{r_{1j} \times s_{1j}}, I_{r_{1j} \times r_{1j}}, 0_{r_{1j} \times t_{1j}} \\ \vdots \\ 0_{r_{ij} \times s_{ij}}, I_{r_{ij} \times r_{ij}}, 0_{r_{ij} \times t_{ij}} \\ \vdots \\ 0_{r_{k_W j} \times s_{k_W j}}, I_{r_{k_W j} \times r_{k_W j}}, 0_{r_{k_W j} \times t_{k_W j}} \end{bmatrix} \in \mathbb{R}^{d_v^j \times r}$$

end for
$\bar{W} \leftarrow (\bar{W}^{(1)}, ..., \bar{W}^{(k_W)}, P_W^{(1)}, ..., P_W^{(k_W)})$
$\bar{V} \leftarrow (\bar{V}^{(1)}, ..., \bar{V}^{(k_V)}, P_V^{(1)}, ..., P_V^{(k_V)})$

$$\bar{V}^{(2)} = [V^{(12)}, \quad V^{(22)}] \in \mathbb{R}^{n \times r_{12} + r_{22}}$$

were obtained by Alg. 4, while projections

$$P_W^{(1)} = \begin{bmatrix} I, 0, 0, 0 \\ 0, I, 0, 0 \end{bmatrix} \in \mathbb{R}^{r_{11} + r_{12} \times r}$$

$$P_W^{(2)} = \begin{bmatrix} 0, 0, I, 0 \\ 0, 0, 0, I \end{bmatrix} \in \mathbb{R}^{r_{21} + r_{22} \times r}$$

$$P_V^{(1)} = \begin{bmatrix} I, 0, 0, 0 \\ 0, 0, I, 0 \end{bmatrix} \in \mathbb{R}^{r_{11} + r_{21} \times r}$$

$$P_V^{(2)} = \begin{bmatrix} 0, I, 0, 0 \\ 0, 0, 0, I \end{bmatrix} \in \mathbb{R}^{r_{12} + r_{22} \times r}$$

are defined by construction. Note that expanded terms

$$\bar{W}^{(1)}P_W^{(1)} = [W^{(11)}, W^{(12)}, 0, 0]$$

$$\bar{W}^{(2)}P_W^{(2)} = [0, 0, W^{(12)}, W^{(22)}]$$

$$\bar{V}^{(1)}P_V^{(1)} = [V^{(11)}, 0, V^{(21)}, 0]$$

$$\bar{V}^{(2)}P_V^{(2)} = [0, V^{(12)}, 0, V^{(22)}]$$

Intuitively, in a case with more blocks, the projections generalize this pattern of "sliding" the block elements of W and "interleaving" the block elements of V as defined in Alg. 4.

All of the proofs in this work rely on this particular variant of SGD (a slight departure from the practical solver used in the experiments).

L. Block-wise Rank Additivity

Implicitly, we have assume a notion of *rank additivity* over the block structure of M. Our notion of rank additivity used above is slightly less general than the one in [69] but is sufficient for our purposes.

Definition A.1. Rank Additive *Block matrix M is **rank additive** if $r = \sum_{i=1}^{k_V} \sum_{j=1}^{k_W} r_{ij}$, where $r = \texttt{rank}(M)$ and $r_{ij} = \texttt{rank}(M^{(ij)})$.*

Of course, rank additivity is a mild assumption when the ranks $r_{ij} \ll m$ and the number of blocks is asymptotically constant. In fact, it holds with high probability for standard random matrix models.

Let us show when the assumption of block-wise rank additivity holds for target matrix M. We begin by restating a relevant lemma [70], [71].

Lemma A.1. *Let $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times k}, C \in \mathbb{R}^{l \times n}$ and $D \in \mathbb{R}^{l \times k}$, while $\mathcal{R}_{\mathcal{M}} = \texttt{range}(M)$ and $r_M = \texttt{rank}(M)$. Then, $\texttt{rank}(\begin{bmatrix} A & B \\ C & D \end{bmatrix}) = r_A + r_B + r_C + r_D$ iff $\mathcal{R}_A \cap \mathcal{R}_B = \mathcal{R}_C \cap \mathcal{R}_D = \mathcal{R}_{A^T} \cap \mathcal{R}_{C^T} = \mathcal{R}_{B^T} \cap \mathcal{R}_{D^T} = \{0\}$.*

We generalize the above lemma in the proposition below.

Prop. A.2. *A block matrix M is rank additive if $\mathcal{R}_{M^{(ij)}} \cap \mathcal{R}_{M^{(ij')}} = \{0\}$ for all $1 \leq j \leq k_V$ and any $j \neq j'$*
$\mathcal{R}_{M^{(ij)T}} \cap \mathcal{R}_{M^{(i'j)T}} = \{0\}$ for all $1 \leq i \leq k_W$ and any $i \neq i'$

Proof. Let $M^{(i)}$ be the i-th block-row of M. If for all $j \neq j'$, the $\texttt{range}(M^{(ij)}) \cap \texttt{range}(M^{(ij')}) = \{0\}$, then we directly obtain Eqn. 1:

$$\mathbf{dim}(\texttt{range}(M^{(i)})) = \sum_j \mathbf{dim}(\texttt{range}(M^{(ij)}))$$

Thus, each block-row is rank additive under the assumptions. With some minor additional effort, we can re-apply the above reasoning on the transpose: $M^T = [M^{(1)T}, ..., M^{(k_V)T}]$, treating the whole matrix as a block-row with $M^{(i)T}$ as the constituent blocks.

Note that we have that $\texttt{range}(M^{(i)T}) = \bigoplus_{j=1}^{k_V} \texttt{range}(M^{(ij)T})$ since $M^{(i)T}$ is a concatenation: $M^{(i)T} = [M^{(i0)T}, ..., M^{(ik_V)T}]$.

Thus we have for $i \neq i'$:
$\texttt{range}(M^{(i)T}) \cap \texttt{range}(M^{(i')T}) =$

$$\left(\bigoplus_{j=1}^{k_V} \texttt{range}(M^{(ij)T}) \right) \bigcap \left(\bigoplus_{j=1}^{k_V} \texttt{range}(M^{(i'j)T}) \right) \subset$$

$$\bigoplus_{j=1}^{k_V} \left(\texttt{range}(M^{(ij)T}) \cap \texttt{range}(M^{(i'j)T}) \right) = \bigoplus_j (\{0\}) = \{0\}$$

This implies

$$\{0\} \subset \texttt{range}(M^{(i)T}) \cap \texttt{range}(M^{(i')T})$$

and thus

$$\texttt{range}(M^{(i)T}) \cap \texttt{range}(M^{(i')T}) = \{0\}$$

from which we can conclude Eqn. 2:

$$\mathbf{dim}(\texttt{range}(M^T)) = \sum_i \mathbf{dim}(\texttt{range}(M^{(i)T}))$$

To conclude the proof, we have
$\texttt{rank}(M) = \texttt{rank}(M^T)$
$= \mathbf{dim}(\texttt{range}(M^T))$
$= \sum_i \mathbf{dim}(\texttt{range}(M^{(i)T}))$ by Eqn. 2
$= \sum_i \texttt{rank}(M^{(i)T})$
$= \sum_i \texttt{rank}(M^{(i)})$
$= \sum_i \mathbf{dim}(\texttt{range}(M^{(i)}))$
$= \sum_i \sum_j \mathbf{dim}(\texttt{range}(M^{(ij)}))$ by Eqn. 1
$= \sum_i \sum_j \texttt{rank}(M^{(ij)})$

$\square$

In other words, as long as the column and row spaces of these block matrices only intersect at the origin, rank additivity is attained. Of course, in a high-dimensional ambient space, randomly selected low-dimensional subspaces will not intersect beyond the origin from which it follows that rank additivity is in general a mild assumption that generally holds in practice.

M. Data-Limited Regime

We cover additional details about for the data-limited, as well as provide proofs for the associated theorems in this section.

1) Extension to Tensor Completion: As mentioned before, matrix completion is a common model for CF tasks. We assume the reader is familiar with this literature. We introduce the more general tensor completion problem as well. Tensor completion generalizes to contextual CF tasks and subsumes matrix completion as a special case. We review this here, following a setting similar to [72], [73]. For tensor completion, the goal is recovering $T \in \mathbb{R}^{n_1 \times ... \times n_\kappa}$ where $T_{x_1,...,x_\kappa} \in \{0,1\}$ denotes if given context $(x_3, ..., x_\kappa)$, user x_1 engages with item x_2. We assume T has a low *pairwise interaction rank* r, meaning T can be factored into κ matrices, $\mathcal{M}^{(i)} \in \mathbb{R}^{n_i \times r}$ for $i \in [\kappa]$ as follows:

$$T_{x_1,...,x_\kappa} = \sum_{(i,j) \in [\kappa] \times [\kappa]} \langle \mathcal{M}_{x_i}^{(i)}, \mathcal{M}_{x_j}^{(j)} \rangle$$

.

Under the assumed pairwise interaction rank r for T, we can factor T into κ matrices, $\mathcal{M}^{(1)}, ..., \mathcal{M}^{(1)}$. we can adapt our model to the tensor case by exploiting the block structure and treating M as a block pairwise interaction matrix rather than a preference matrix. Let $k_V = k_W = \kappa$ and let each block represent an interaction matrix: $M^{(ij)} = \mathcal{M}^{(i)}(\mathcal{M}^{(j)})^T$. Hence, M is symmetric and with this simple construction, the factors of T are represented as the blocks of M. The remaining distinction is that in the tensor case, the algorithm only observe sums of elements selected from the blocks of M instead of observing the entries of M directly. This minor distinction is addressed in both [72] and [73] and with appropriate care to details, the two observation structures are largely equivalent. For brevity, we discuss only the matrix case here, while keeping in mind this natural extension to the tensor case. When thinking about the categorical features in CTR prediction, this construction is precisely the one we use to block by features, as described in Section 3.

2) Assumptions: Beyond the rank addivity assumption, we also implicitly assume a classical assumption on *incoherence*.

The notion of *incoherence* is central to matrix completion [19]–[23], [41]. Throughout this work, we implicitly assume that M is μ-incoherent. Note that asymptotic notation occults this. For many standard random models for M, μ scales like $\sqrt{r \log n}$ [41], but here, we simply take μ-incoherence as an assumption on M. Note that all matrices are incoherent for some $\mu \in [1, \frac{\max\{n,m\}}{r}]$ [23].

Definition A.2. Incoherence. *Let $M = USV^T$ be the singular-value decomposition for a matrix rank-r matrix $M \in \mathbb{R}^{n \times m}$. Matrix M is μ-incoherent if for all $1 \leq i \leq n, ||U_i||_2^2 \leq \frac{\mu r}{n}$ and for all $1 \leq j \leq m, ||V_j||_2^2 \leq \frac{\mu r}{n}$.*

a) Low-sample Sub-matrix: We denote M_ε as the *low sample sub-matrix* of blocks corresponding to minimum marginal sampling rate ε. Concretely, $M_\varepsilon = [M^{(i_\varepsilon 1)}, ..., M^{(i_\varepsilon k_V)}] \in \mathbb{R}^{n_{i_\varepsilon} \times m}$ if $\varepsilon_W \leq \varepsilon_V$ and $M_\varepsilon = [M^{(1 j_\varepsilon)}, ..., M^{(k_W j_\varepsilon)}] \in \mathbb{R}^{n \times m_{j_\varepsilon}}$ otherwise. For convenience, we also define $\tilde{n}_\varepsilon = n_{i_\varepsilon}$ if $\varepsilon_W \leq \varepsilon_V$ and $\tilde{n}_\varepsilon = m_{j_\varepsilon}$ otherwise. We refer to $\tilde{n}_\varepsilon$ as the **size** of the low-sample sub-matrix (since the other dimension is inherited from the size of M itself).

3) Popularity-agnostic algorithms: (including UD matrix factorization) are those that can be seen as empirically matching at the observed indices at a given rank constraint, or any relaxation thereof, without taking advantage of popularity. MD factorization imposes additional popularity-based constraints. These additional constraints become essential to completion when the popularity is significantly skewed.

Definition A.3. Popularity-Agnostic Algorithm. *Let $f(\theta)$ be some arbitrary but fixed model with parameters θ that outputs an attempted reconstruction $\hat{M}$ of matrix M. For a given rank r^* let $\mathcal{S}$ be any set of matrices such that for $\hat{S} = \{\hat{M} | rank(\hat{M}) = r^*\}$ we have $\hat{S} \subseteq \mathcal{S}$. An algorithm $\mathcal{A}$ is popularity-agnositc if it outputs the solution to an optimization characterized by a Lagrangian of the form $\mathcal{L}(\theta, \lambda) = ||M - \hat{M}||_\Omega^2 + \lambda \mathbf{1}[\hat{M} \notin \mathcal{S}]$ where indicator function $\mathbf{1}[x] = 1$ if x=True and 0 otherwise.*

4) Popularity-Agnostic Completion Bounds: It is standard to impose a low-rank structure in the context of matrix completion. We are interested in understanding how and when popularity-based structure can improve recovery. While, UD embeddings impose a low-rank structure, at a given rank, we can interpret our MD embeddings as imposing additional popularity-based constraints on the matrix reconstruction. While our MD embeddings maintain a particular rank, they do so with less parameters, thereby imposing an additional popularity-based restriction on the space of admissible solution matrices.

a) Non-asymptotic Upper Bound: We first give a simple lower bound on the sample complexity for popularity-agnostic algorithm. The bound is straightforward, based on the fact that without additional problem structure, in order to complete a matrix at rank r, you need at least r observations, even on the least popular row or column. The global reconstruction efforts will always be thwarted by the least popular user or item. The

bound below is non-asymptotic, holding for any problem size. The theorem below implies that popularity-agnostic algorithms pay steep recovery penalties depending on the least likely row/column to sample. If you want to exactly recover a matrix, you can only do as well as your most difficult row/column.

Theorem A.3. *Fix some $0 < \delta < \frac{1}{2}$. Let ε be the minimum marginal sampling rate and let $\tilde{n}_\varepsilon$ be the size of the low-sample sub-matrix. Suppose number of samples $N \leq \frac{r}{\varepsilon}(1-\delta)$. Then, no popularity-agnostic algorithm can recover M with probability greater than $\exp(-\frac{r \tilde{n}_\varepsilon \delta^2}{3(1-\delta)})$.*

Proof. Let ψ be any one of the $\tilde{n}_\varepsilon$ vectors in the low-probability sub-matrix M_ε (ψ is of length n or m, depending on if M_ε is a block-wise row or column). Let X_ψ be a random variable denoting the number of observations in corresponding to ψ under block-wise Bernoulli sampling. Since X_ψ is the sum of independent Bernoulli variables, we have $\mathbb{E}[X_\psi] = N\varepsilon$ by linearity of expectation. Furthermore, we require at least r observations for each row and column in M in order to achieve exact recovery at rank r. In order to see this directly, assume that an oracle completes all the embeddings except the row or column in question. Then, each observations defines immediately removes only one degree of freedom, since it defines the inner product with a known vector. It will be impossible to complete the final row or column with less than r observations because popularity-agnostic algorithms provide no further constraints beyond a low-rank structure. Given that we need r observations per vector, we can see that $\mathbb{E}[X_\psi] = N\varepsilon < r$. This implies if $N < r/\varepsilon$, we can use the Chernoff tail bound [74] to bound $\mathbf{Pr}[X_\psi \geq r]$ from above. We take $1/2 < \delta < 1$ such that $N \leq \delta r/\varepsilon$. By application of the Chernoff bound, we have $\mathbf{Pr}[X_\psi \geq r] \leq \exp(-\frac{r}{3}\frac{\delta^2}{1-\delta})$. To complete the proof, notice that our argument extends to each of the $\tilde{n}_\varepsilon$ vectors in M_ε independently. Since all of these vectors require r observations in order to complete of M_ε, we obtain the final probability by computing a product over the probability that each of $\tilde{n}_\varepsilon$ vectors obtains at least r observations. $\square$

b) Asymptotic Upper Bound: We also provide a stronger asymptotic lower bound for exact completion, based on the results of [19], [41]. This lower bound assumes that the matrix size n increases while keeping the sampling rate constant. It includes an additional $O(\log n)$ factor, due to the well-known *coupon collector* effect [75].

Since M is still a block matrix, we assume that asymptotically, each individual block becomes large, while Π is held constant. More concretely, we assume each block scales at the same rate as the entire matrix: $n_{ij} = \Theta(n)$ for all i, j and $m_{ij} = \Theta(n)$ for all i, j. In principle, we could also support an asymptotic number of blocks as well, as long as the number of blocks grows slowly enough compared to size of each block. Other numerical constants, such as the condition number and incoherence, are taken to be non-asymptotic. Note that we do not require the block additivity assumption for this to hold.

Theorem A.4. *Let M be a target matix following the block-wise Bernoulli sampling model. Let ε be the minimum marginal*

sampling rate. Suppose $N = o(\frac{r}{\varepsilon} n \log n)$. Then any popularity-agnostic algorithm will have arbitrarily small probability of recovery, asymptotically.

Proof. Order $\Theta(nr \log n)$ observations are necessary for exact completion at a given probability in the asymptotic setting [19], [41]. This is because $\Theta(nr \log n)$ observations are necessary to obtain r observations per row and column. Each vector in the low-sample sub-matrix also requires r observations. Since the number of samples in the low-sample sub-matrix concentrates around $N\varepsilon$, this number must be order $\Theta(rn \log n)$ in order to have a chance of reconstruction. □

It is instructive to understand why UD embeddings fail to recover rank-r matrix M under popularity skew. For argument's sake, let d denote a potential uniform embedding dimension. Suppose we have $\Theta(rn \log n)$ samples and we set UD to r: $d \leftarrow r$. When sampling is skewed, $M^{(2)}$ will be too sparsely covered to reveal r degrees of freedom, since it only generates ϵ fraction of the observations. Thus, the r-dimensional embeddings would over-fit the under-constrained $M^{(2)}$ block as a result. Alternatively, if we set $d \leftarrow \epsilon r$, as so to match the sample size over sub-matrix $M^{(2)}$, then our ϵr-dimensional embeddings would be unable to fit the larger training sample over the $M^{(1)}$ block. Namely, we would now have too many samples coming from a rank-r matrix, resulting in an over-constrained problem that is infeasible with ϵr-dimensional embeddings. By using MD embeddings, we can avoid this problem by simultaneously fitting popular and rare blocks.

5) Completion Guarantees for Mixed Dimension Embeddings: In [23] it was shown that various non-convex optimization algorithms, including SGD, could exactly complete the unknown matrix, under the Bernoulli sampling model. For convenience, the theorem is reproduced below. The details of the SGD implementation, such as step sizes and other parameters, can be found in [23].

Theorem A.5. *(Sun and Luo, 2016) Let $\alpha = \frac{n}{m} \geq 1$, κ be the condition number of M and μ be the incoherence of M. If (expected) number of samples $N \geq C_0 nr\kappa^2 \alpha(\max\{\mu \log n, \sqrt{\alpha}\mu^2 r^6 \kappa^4\})$ then SGD completes M with probability greater than $1 - \frac{2}{n^4}$.*

We can use Thm A.5 and Alg. 4 to construct a guarantee for mixed-dimension block-wise factorization, as follows.

Corollary A.5.1. *Let M be a target matrix following the block-wise Bernoulli sampling model. Let C_0 be a universal constant, $\hat{n}_{ij} = \min\{n_i, m_j\}$ and $N_{ij}^* = C_0 \Pi_{ij}^{-1} \hat{n}_{ij} r_{ij} \kappa_{ij}^2 \alpha_{ij}(\max\{\mu_{ij} \log \hat{n}_{ij}, \sqrt{\alpha_{ij}}\mu_{ij}^2 r_{ij}^6 \kappa_{ij}^4\})$ where κ_{ij} and μ_{ij} is the condition number and the incoherence of the ij-th block of M and $\alpha_{ij} = \frac{\max\{n_{ij}, m_{ij}\}}{\min\{n_{ij}, m_{ij}\}} \geq 1$. If $N \geq \max_{ij} N_{ij}^*$, then block-wise MD factorization completes rank additive block matrix M with probability greater than $1 - \sum_{ij} \frac{2}{\hat{n}_{ij}^4}$.*

Proof. Recall the construction used in Alg. 4. First, we complete each block individually. We apply Thm A.5 to each

block independently to guarantee its completion at rank r_{ij} with probability at least $1 - \frac{2}{n_{ij}^4}$. We then use the block-wise embeddings to construct MD embeddings $\bar{W}, \bar{V}$ as described Alg. 4. If $W^{(ij)}V^{(ij)T} = M^{(ij)}$, for all i, j, then $\bar{W}\bar{V}^T = M$. Thus, we need only a union bound on the failure probabilities from Thm A.5 to complete the proof. □

Note that Corollary A.5.1 implies Thm IV.1 as it is a non-asymptotic version. Namely, recall that $n_{ij} = \Theta(n)$ for all i, j. Furthermore, letting $C = C_0 \max_{ij} (\Pi_{ij}^{-1} r_{ij} \kappa_{ij}^2 \alpha_{ij} \mu_{ij})$ recovers Corollary IV.1.

N. Memory-Limited Regime

Now, we turn our attention to the allocation implications of non-uniformly sampled test sets. To abstract away training, we assume an oracle reveals the target matrix. Recall that our challenge is a small parameter budget — our embeddings can only use B parameters. The question is what dimensions d_w and d_v each embedding block should get to minimize our popularity-weighted reconstruction loss (under MSE)? Before proceeding, we pause to define some useful matrices from the block-wise MD factorization (Alg. 4).

Definition A.4. *In the context of **block-wise MD factorization** (Alg. 4) we refer to the matrices $(W^{(ij)}, V^{(ij)})$ as the ij-th **block-wise embeddings**. We refer to matrix $\bar{W}^{(i)}$ as the i-th **row embedding block** and matrix $\bar{V}^{(j)}$ as the j-th **column embedding block**.*

Note that, generally speaking, embedding tables (W, V), naturally inherit an *embedding block* structure based on the block structure of M. For example, standard UD embeddings partition such that the top block-wise row $M^{(1)} = [M^{(11)}, ..., M^{(1k_W)}]$ only depends on $W^{(1)}$. Thus, embedding blocks exist independent of the usage of block-wise MD factorization. On the other hand, *block-wise embeddings* $(W^{(ij)}, V^{(ij)})$, are a distinct byproduct of block-wise MD factorization.

1) Optimization over Embedding Dimensions: We assume the block structure and block-wise probability matrix is given — the variables over which the optimization takes place are 1) the dimensions of the embedding blocks, (d_w, d_v) such that $W^{(l)} \in \mathbb{R}^{n_l \times (d_w)_l}$ and $V^{(l)} \in \mathbb{R}^{n_l \times (d_v)_l}$ and 2) the embedding blocks themselves $W^{(i)}$ for $1 \leq i \leq k_W$ and $V^{(j)}$ for $1 \leq j \leq k_V$. Note that when the embedding block dimensions are uniform, such that $(d_w)_i = (d_v)_j$ for all i and j, this is equivalent to direct optimization over embedding matrices W, V (i.e. matrix factorization). Recall that L_Π is the popularity-weighted MSE. When d_w and d_v are treated as integers, this optimization is NP-Hard in general, since even integral feasibility under linear constraints is known to be NP-Hard [76]. Instead, we study a continuous relaxation that results in a convex program. The resultant convex program is far simpler, and yields a closed-form solution given the spectrum of the target matrix. We proceed to define another quantity of interest for our discussion, a *spectral (singular value) decay*. In order to save space in the main text, we do not introduce the spectral decay rule g but we imply it when referring to the

spectrum directly. After the upcoming definition, we restate and prove Thm. IV.2 from the main text.

Definition A.5. *A **spectral decay** is mapping from $[0, r]$ to $\mathbb{R}^+$ that describes the singular value scree plot for a matrix. Let σ_k be the k-th singular value of a matrix and $\sigma_k \geq \sigma_{k+1}$ for $k = 1, ..., r$. For any singular value spectrum we associate a spectral decay rule, a piece-wise step-function and its functional inverse, as $g(x) = \sigma_k$ for $k - 1 \leq x < k$ and $g^{-1}(x) = k$ for $\sigma_k < x \leq \sigma_{k+1}$, respectively.*

Theorem A.6. *The optimal block-wise embedding dimensions for the convex relaxation of the variable dimension embedding optimization under a parameter budget are given by $d_{ij}^* = g_{ij}^{-1}\left(\sqrt{\lambda(n_i + m_j)(n_i m_j)\Pi_{ij}^{-1}}\right)$ where g_{ij}^{-1} is the functional inverse of the spectral decay of block $M^{(ij)}$.*

Proof. The optimization is formulated as

$$\min_{d_w, d_v} \min_{W, V} L_\Pi(M, WV^T)$$

$$\text{s.t. } \sum_i n_i(d_w)_i + \sum_j m_j(d_v)_j \leq B$$

Under relaxation, we treat this a continuous optimization. Let (k, l) be a test coordinate sampled according to Π. If rank additivity holds, we can equivalently write

$$= \min_d \min_{W, V} \mathbb{E}_{(k,l) \in [n] \times [m]} |M_{kl} - W_k V_l^T|^2$$

$$\text{st } \sum_{ij} (n_i + m_j)d_{ij} \leq B$$

where M_{kl} is the kl-th element of M, $(d_w)_i = \sum_j d_{ij}$ and $(d_v)_j = \sum_i d_{ij}$. The d_w, d_v refer to the embedding block dimensions, whereas the d_{ij} refer to the block-wise embedding dimensions (Definition C.11). We may ignore the parameters in the projections since they are not free parameters (and also contribute a negligible amount of parameters to the total count). Under Bernoulli sampling model, our popularity distribution yields. As shorthand notation, let $\mathfrak{B} := \sum_{ij}(n_i + m_j)d_{ij}$.

$$= \min_d \min_{W, V} \sum_{ij} \frac{\Pi_{ij}}{n_i m_j} ||M^{(ij)} - W^{(ij)}V^{(ij)T}||_F^2 \text{ st } \mathfrak{B} \leq B$$

Since the constraints remain the same, we omit them for the time being. Letting $\sigma_k^{(ij)}$ be the singular values of block $M^{(ij)}$ and using the low-rank approximation theorem [77] we obtain

$$= \min_d \sum_{ij} \frac{\Pi_{ij}}{n_i m_j} \sum_{k=d_{ij}+1}^{r_{ij}} (\sigma_k^{(ij)})^2 \text{ st } \mathfrak{B} \leq B$$

Letting g_{ij} be the spectral decay rule for each block and noticing that by construction $\sum_{k=0}^r \sigma_k = \int_0^r g(k)dk$ we obtain

$$= \min_d \sum_{ij} \frac{\Pi_{ij}}{n_i m_j} \left(\int_0^{r_{ij}} g_{ij}^2(k)dk - \int_0^{d_{ij}} g_{ij}^2(k)dk \right) \text{ st } \mathfrak{B} \leq B$$

$$= \min_d \sum_{ij} \frac{\Pi_{ij}}{n_i m_j} \left(||M^{(ij)}||_F^2 - \int_0^{d_{ij}} g_{ij}^2(k)dk \right) \text{ st } \mathfrak{B} \leq B$$

Observe that the objective is convex. To see this, note that each g is decreasing since the spectral decay is decreasing. Thus, g^2 is also decreasing. The negative integral of a decreasing function is convex. Finally, since the objective is a sum of functions that are convex along one variable and constant along the rest, the entire optimization is convex (and well-posed under the linear constraint, which is guaranteed to be active). Thus we can solve with the optimization with first-order conditions [24]. The corresponding Lagrangian can be written as

$$\mathcal{L} = \sum_{ij} \frac{\Pi_{ij}}{n_i m_j} \left(||M^{(ij)}||_F^2 - \int_0^{d_{ij}} g_{ij}^2(k)dk \right)$$

$$+ \lambda \left(-B + \sum_{ij} (n_i + m_j)d_{ij} \right)$$

Note that $M^{(ij)}$ does not depend on d_{ij}. Also, note that we can use the fundamental theorem of calculus $\frac{\partial}{\partial x} \int_0^x g(t)dt = g(x)$ [78]. Then, using Lagrange multipliers [24] we can write

$$\frac{\partial}{\partial d_{ij}} L_\Pi(M, WV^T) = -\frac{\Pi_{ij}}{n_i m_j} g_{ij}^2(d_{ij}) + \lambda(n_i + m_j)$$

Finally, using first order conditions $\nabla_{d_{ij}} = [\frac{\partial}{\partial d_{ij}}] = 0$ we obtain: $g_{ij}^2(d_{ij}) = \lambda(n_i + m_j)(n_i m_j)\Pi_{ij}^{-1}$. Solving for d_{ij} by taking the functional inverse of g_{ij} completes the proof. We conclude:

$$d_{ij}^* = g_{ij}^{-1}\left(\sqrt{\lambda(n_i + m_j)(n_i m_j)\Pi_{ij}^{-1}} \right)$$

$\square$

For specific spectral decay rules, we may give closed-form solutions, as done in the main text for power law decay. We can also analyze the performance gap between uniform and MD embeddings with respect to the optimization objective.

Corollary A.6.1. *The performance gap compared to UD embeddings is*

$$\sum_{ij} \Pi_{ij} (\mathbf{1}\{d_{ij}^* > \frac{B}{n+m}\}(\sum_{k=\frac{B}{n+m}}^{d_{ij}^*} (\sigma_k^{(ij)})^2)$$

$$-\mathbf{1}\{d_{ij}^* < \frac{B}{n+m}\}(\sum_{k=d_{ij}^*}^{\frac{B}{n+m}} (\sigma_k^{(ij)})^2))$$

Proof. Follows directly from plugging optimal d^* into the objective. $\square$

We can explain the intuition for Corollary A.6.1 as follows. The first term counts the spectral mass gained back by allocating more parameters to frequent embeddings. $B/(n+m)$ is the

embedding dimension under a uniform parameter allotment. When d_{ij}^* is greater than this, we are increasing the dimension which enables that embedding to recover more of the spectrum. This occurs when Π_{ij} is large. On the other hand, the trade-off is that lower-dimension embeddings recover less of the spectrum when Π_{ij} is small, which is the penalty incurred by the second term.

Corollary A.6.2. *When M exhibits a block-wise power spectral decay, this becomes:*

$$d_{ij}^* = \lambda \zeta_{ij} \Pi_{ij}^{\frac{1}{2\beta}}$$

where $\zeta_{ij} = \left(\frac{(n_i + m_j)(n_i m_j)}{\mu} \right)^{\frac{-1}{2\beta}}$ *and* $\lambda = \left(\frac{B}{\sum_{ij}(n_i + m_j)\zeta_{ij}} \right)^{-2\beta}$

Proof. Follows directly by substituting power spectral decay rule for $\sigma(\cdot)$. $\qquad\square$

2) Approximation Gap for Convex Relaxation: Note that we can bound the approximation gap of the proposed relaxation by simply rounding down each d_{ij} to the nearest integer, which ensures the feasibility of the assignment. The absolute approximation error is then less than $\sum_{ij} \Pi_{ij} \cdot g^2(d_{ij})$. For most applications, this quantity is small for a good MD assignment, since either the probability term, Π_{ij} is small, or the the spectrum at d_{ij} is small. For example, in typical use cases, the embedding dimensions may be on the order of $10 - 100$ – rounding down to the nearest integer would thus represent a loss of $10 - 1\%$ of the spectral mass.