We thank the reviewers for their constructive feedback and address their comments below.

(R1, R2) Larger memory budgets. In this paper, we focus on the models with low memory budgets. Such models are needed for mobile devices, embedded systems, etc. However, as requested by the reviewers, we perform an additional comparison with a budget of 32 parameters per datapoint. The results from Table 1 (bottom) demonstrate that the advantage of PRODIGE over vectorial counterparts persists in this operating point as well.

(R3) Why does PRODIGE become deterministic? This property is induced by our sparsity regularizer (2) that minimizes the L_0 norm of the adjacency matrix. This regularizer was originally proposed in [1], which explains why the solutions are deterministic. Empirically, we also observe that edge probabilities converge to 0 or 1.

(R2,R3) On independence of edges: Yes, in our model edge indicators are independent random variables. However, the training algorithm jointly optimizes all edges in the graph. Therefore, PRODIGE can learn arbitrary graph by setting the probability of kept edges to 1 and the rest to 0. Thus, the independence assumption does not reduce the expressive power of our model.

(R3) Sampling efficiency. The actual distance between two datapoints only depends on the existence of edges along the shortest path and the absence of edges that would have led to an even shorter path if they were kept. There are typically only a few such edges (e.g. about 15 for MNIST10K). We leverage this fact in lines 120-123 to propose an objective (4) for more efficient optimization.

(R2) PRODIGE vs Isomap. Isomap constructs knn- or ϵ-graph, then computes shortest paths in that graph and approximates these paths with Multi-Dimensional Scaling in Euclidean space. Hence, Isomap produces vectorial embeddings, while PRODIGE does not. Moreover, Isomap is specific to manifold learning and can only be applied to task 4.1. We compare PRODIGE with Isomap (scikit-learn implementation), see Table 1 (top), which demonstrates the advantage of our approach. Furthermore, PRODIGE is a general method that works for a variety of tasks (e.g. 4.2, 4.3). If accepted, we will include a more detailed comparison of the two methods with explanation.

(R2) SVD/ALS baselines in task 4.2 We also tried several neural collaborative filtering methods, but neither of them was competitive with SVD and ALS. In fact, a recent study [2] demonstrates that these simple baselines perform remarkably well for recommendation task on a variety of datasets, including Pinterest.

(R3) The importance of initialization. We investigate this issue in task 4.2, see lines 237-242 and Table 2 (original paper). While the initialization is important, a completely random initialization still performs on par with most baselines.

(R2,R3) Robustness and sensitivity to hyperparameters. We verify the robustness of our training procedure by running several experiments with different random initializations and different initial numbers of neighbors. Figure 1 shows the learning curves of PRODIGE under various conditions. While these results exhibit some noise from Stochastic Gradient Descent, the overall training procedure is stable and robust.

Method	MSE	Params
PRODIGE	**0.03356**	76.5k
MDS	0.05584	80k
Poincare MDS	0.04839	80k
Isomap	0.28242	80k
PRODIGE	**0.01031**	292k
MDS	0.01297	320k
Poincare-MDS	0.01178	320k

Table 1: Additional evaluations for task 4.1 performed on the **GLOVE10K** data

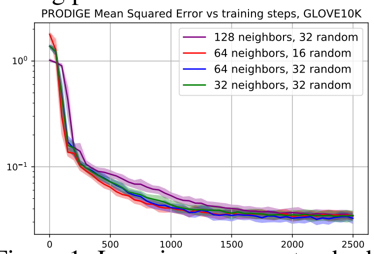

Figure 1: Learning curves, standard deviation over 10 runs shown in pale

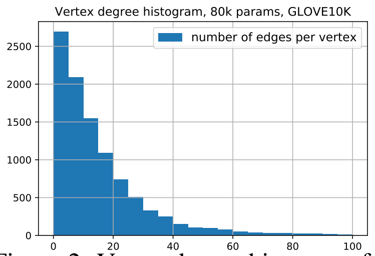

Figure 2: Vertex degree histogram for **GLOVE10K** with 8 params/vertex

(R2) Reconstructing known graphs with PRODIGE. For this experiment, we generate connected Erdős–Rényi graphs with 10-25 vertices with edge probability $p = 0.25$ and edge weights sampled from uniform $U(0, 1)$ distribution. We then train PRODIGE to reconstruct these graphs. Out of 100 random graphs, in 91 cases PRODIGE was able to reconstruct all edges that affect shortest any paths and all 100 runs the resulting graph had MSE below 10^{-3}. We agree that this experiment is a useful sanity check and we will add it in the final version, if accepted.

(R3) Metric learning details: See lines 245-247. **(R2, R3) Tuning λ :** We tune λ to fit into the memory budget and compare methods with the same budget. **(R2) Do we need to know dependency structures a priori?** No, PRODIGE learns dependency structure from data. **(R2) Paths between all node pairs:** This can be performed efficiently with an algorithm[3]. **(R2) Increasing K in 4.3:** We observe that for the same memory budget, larger K result in almost equivalent performance. **(R2) Using known hubs as anchors in 4.3:** We tried that, however using K most frequent tokens as anchors results in similar performance. We opted for synthetic anchors for generality; **(R2) Demonstrate scale-free:** We observed the scale-free property, see Figure 2. Will add this in the final version, if accepted.

[1] Christos Louizos et al. Learning sparse neural networks through L_0 regularization. In *ICLR, 2018.*

[2] Ferrari Dacrema et al. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *RecSys, 2019.*

[3] Sebastian Knopp et al. Computing many-to-many shortest paths using highway hierarchies. In *ALENEX, 2007.*