
Flashlight : Scalable Link Prediction with Effective Decoders

Anonymous Author(s)

Anonymous Affiliation

Anonymous Email

Abstract

Link prediction (LP) has been recognized as an important task in graph learning with its broad practical applications. A typical application of LP is to retrieve the top scoring neighbors for a given source node, such as the friend recommendation. These services desire the high inference scalability to find the top scoring neighbors from many candidate nodes at low latencies. There are two popular decoders that the recent LP models mainly use to compute the edge scores from node embeddings: the **HadamardMLP** and **Dot Product** decoders. After theoretical and empirical analysis, we find that the HadamardMLP decoders are generally more effective for LP. However, HadamardMLP lacks the scalability for retrieving top scoring neighbors on large graphs, since to the best of our knowledge, there does not exist an algorithm to retrieve the top scoring neighbors for HadamardMLP decoders in sublinear complexity. To make HadamardMLP scalable, we propose the *Flashlight* algorithm to accelerate the top scoring neighbor retrievals for HadamardMLP: a sublinear algorithm that progressively applies approximate maximum inner product search (MIPS) techniques with adaptively adjusted query embeddings. Empirical results show that Flashlight improves the inference speed of LP by more than 100 times on the large OGBL-CITATION2 dataset without sacrificing effectiveness. Our work paves the way for large-scale LP applications with the effective HadamardMLP decoders by greatly accelerating their inference.

1 Introduction

The goal of link prediction (LP) is to predict the missing links in a graph [1]. LP is drawing increasing attention in the past decade due to its board practical applications [2]. For instance, LP can be used to recommend new friends on social media [3], and recommend attractive items to the costumers on E-commerce sites [4], so as to improve the user experience. During inference, these applications demand the LP methods to retrieve the top scoring neighbors for a source node at low latencies. This is especially challenging on large graphs because the LP methods need to search many candidate nodes to find the top scoring neighbors.

There are two main kinds of architecture followed by the recent LP models. The first uses an encoder, e.g., GCN [5], to obtain the node-level embeddings and uses a decoder, e.g., Dot Product, to get the edge scores between the paired nodes [6]. The second crops a subgraph for every edge and computes the edge score from the subgraph directly [7]. The inference speed of the second is much lower than the first, so we focus on the first kind of models to achieve fast inference on large graphs. In the last years, extensive research focuses on developing more expressive LP encoders [6, 8]. However, much less work pays attention to the essential impacts of the choice of decoders on LP's performance. In this work, we theoretically and empirically analyze two popular LP decoders: Dot Product and HadamardMLP (a MLP following the Hadamard Product), and find that the latter is generally more effective than the former.

In practical applications, we should not only consider the effectiveness of LP, but also inference efficiency. Many LP applications generally require fast retrieval of the top scoring neighbors for low-latency services [3, 9, 10]. For a Dot Product decoder, this retrieval can be approximated efficiently at the sublinear time complexity [11]. However, to the best of our knowledge, no such sublinear algorithms exist for the top scoring neighbor retrievals of the HadamardMLP decoders. This means

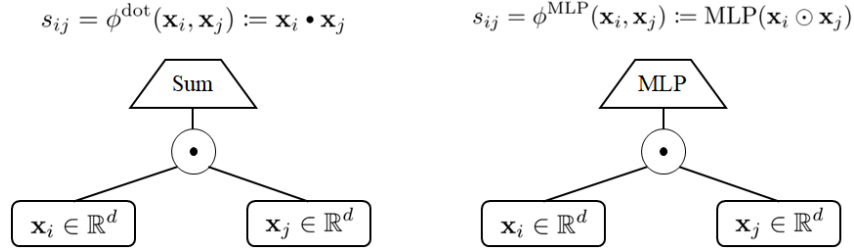


Figure 1: Two popular LP decoders: The Dot Product (left), equivalent to the element-wise summation following the Hadamard product, and the HadamardMLP decoder (right).

44 that for every source node, we have to iterate over all the nodes in the graph to compute the scores
 45 so as to find the top scoring neighbors for HadamardMLP, which is of linear complexity and cannot
 46 scale to large graphs.

47 To allow LP applications to enjoy the high effectiveness of HadamardMLP decoders while avoiding
 48 the poor inference scalability, we propose the scalable top scoring neighbor search algorithm named
 49 *Flashlight*. Our Flashlight progressively calls the well-developed approximate maximum inner
 50 product search (MIPS) techniques for a few iterations. At every iteration, we analyze the retrieved
 51 neighbors and adaptively adjust the query embedding for Flashlight to find the missed high scoring
 52 neighbors. Our Flashlight algorithm holds sublinear time complexity on finding top scoring neighbors
 53 for HadamardMLP decoders, allowing for fast and scalable inference. Empirical results show that
 54 Flashlight accelerates the inference of LP models by more than 100 times on the large OGBL-
 55 CITATION2 dataset without sacrificing the effectiveness. Overall, our work paves the way for the
 56 use of effective LP decoders in practical settings by greatly accelerating their inference.

57 2 Revisiting Link Prediction Decoders

58 In this section, we formalize the link prediction (LP) problem and the LP decoders. Typically, many
 59 LP models include an encoder that learns the node-level embeddings $\mathbf{x}_i, i \in \mathcal{V}$, where \mathcal{V} is the set of
 60 nodes, and an decoder $\phi : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ that combines the node-level embeddings of a pair of nodes:
 61 $\mathbf{x}_i, \mathbf{x}_j$ into a single score: s_{ij} . If s_{ij} is higher, the link between nodes i and j is more likely to exist.
 62 The state-of-the-art models generally use graph neural networks as the encoders [5, 6, 8, 12, 13].
 63 From here on, we mainly focus on the decoder ϕ .

64 2.1 Dot Product Decoder

65 The most common decoder of link prediction is the Dot Product [6, 8, 10]:

$$s_{ij} = \phi^{\text{dot}}(\mathbf{x}_i, \mathbf{x}_j) := \mathbf{x}_i \bullet \mathbf{x}_j, \quad (1)$$

66 where \bullet denotes the dot product.

67 Training a link prediction model with the Dot Product decoder encourages the embeddings of the
 68 connected nodes to be close to each other. Intuitively, the score s_{ij} can be thought as a measure of the
 69 squared Eulidean distance between the node embeddings $\mathbf{x}_i, \mathbf{x}_j$, as $\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 - 2\mathbf{x}_i \bullet \mathbf{x}_j +$
 70 $\|\mathbf{x}_j\|^2$, if the $\|\mathbf{x}_j\|$ is constant over the neighbors $j \in \mathcal{N}$, e.g., after normalization [14]. Because the
 71 node embeddings represent the semantic information of nodes, Dot Product assumes the homophily
 72 of graph topology, i.e., the semantically similar nodes are more likely to be connected.

73 2.2 HadamardMLP (MLP following Hadamard Product) Decoder

74 Multi layer perceptrons (MLPs) are known to be universal approximators that can approximate any
 75 continuous function on a compact set [15]. A MLP layer can be defined as a function $f : \mathbb{R}^{d_{\text{in}}} \rightarrow$
 76 $\mathbb{R}^{d_{\text{out}}}$:

$$f_{\mathbf{W}}(\mathbf{x}) = \text{ReLU}(\mathbf{W}\mathbf{x}) \quad (2)$$

77 which is parameterized by the learnable weight $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ (the bias, if exists, can be represented
 78 by an additional column in \mathbf{W} and an additional channel in the input \mathbf{x} with the value as 1). ReLU
 79 is the activation function. In a MLP, several layers of f are stacked, e.g., a 3-layer MLP can be
 80 formalized as $f_{\mathbf{W}_3}(f_{\mathbf{W}_2}(f_{\mathbf{W}_1}(\mathbf{x})))$.

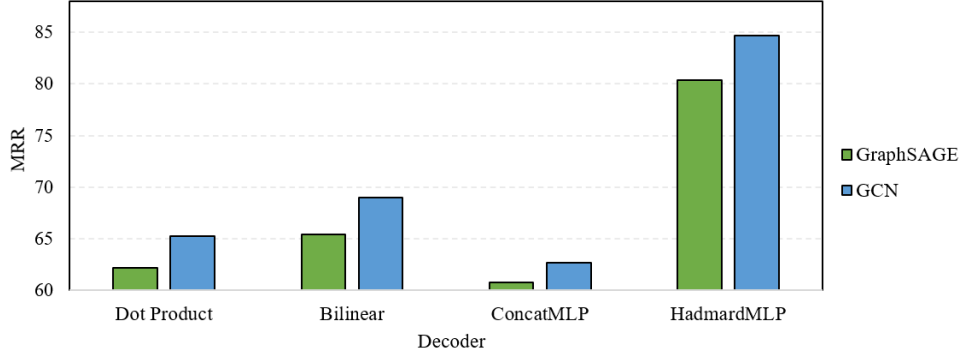


Figure 2: HadamardMLP achieves higher Mean Reciprocal Rank (MRR, higher is better) than other decoders on the OGBL-CITATION2 [16] dataset with the encoder as GraphSAGE [12] and GCN [5]. More empirical results and the detailed settings are in Sec. 5.3.

81 The state-of-the-art models widely use a MLP following the Hadamard Product between the paired
 82 nodes as the decoder (short as the HadamardMLP decoders) [6, 8, 10, 16]:

$$s_{ij} = \phi^{\text{MLP}}(\mathbf{x}_i, \mathbf{x}_j) := \text{MLP}(\mathbf{x}_i \odot \mathbf{x}_j) = \mathbf{w}_L^T (f_{\mathbf{w}_{L-1}}(\dots f_{\mathbf{w}_1}(\mathbf{x}_i \odot \mathbf{x}_j) \dots)), \quad (3)$$

83 where \odot denotes the Hadamard Product. Fig. 1 illustrates these two models the Dot Product and
 84 HadamardMLP decoders.

85 2.3 Other Link Prediction Decoders

86 In principle, every function that takes two vectors as the input and outputs a scalar can act as the
 87 decoder. For example, there are bilinear dot product decoder (short as **Bilinear decoder**) [6]:

$$s_{ij} = \mathbf{h}_i^T \mathbf{W} \mathbf{h}_j, \quad (4)$$

88 where \mathbf{W} is the learnable weight, and the MLPs following the concatenate decoder [6, 10] (short as
 89 **ConcatMLP decoder**):

$$s_{ij} = \text{MLP}(\mathbf{h}_i \parallel \mathbf{h}_j) \quad (5)$$

90 , etc. These two decoders are used much less than Dot Product and HadamardMLP in the state-of-
 91 the-art LP models possibly due to their lower effectiveness [6, 8, 10, 16].

92 2.4 HadamardMLP is Generally More Effective than Other Decoders

93 Dot Product demands the homophily of graph data to effectively infer the link between nodes. In
 94 contrast, thanks to the universal approximation capability, MLP can approximate any continuous
 95 function, and thus does not demand the homophily of graph data for effective LP. This gap in the
 96 expressiveness accounts for the performance difference of these two decoders on many datasets (see
 97 Sec. 5.3). We additionally show in Appendix. A that using a HadamardMLP is easy to learn Dot
 98 Product, which also partially accounts for the better effectiveness of the HadamardMLP decoders
 99 over the Dot Product. Existing work also finds that the effectiveness of Bilinear and ConcatMLP is
 100 generally worse than the HadamardMLP or Dot Product decoder [6, 8, 10, 16]. We confirm these
 101 findings more rigorously in the empirical results in Fig. 2 and more complete in Sec. 5.3.

102 Although ConcatMLP is as expressive as HadamardMLP in theory because MLP is a universal
 103 function approximator [15], such an argument neglects the difficulty of learning the target function
 104 using ConcatMLP. The ConcatMLP decoder processes the concatenation of the paired embeddings
 105 instead of the Hadamard product of the paired embeddings. In contrast, HadamardMLP takes the
 106 Hadamard product of the paired embeddings as the input. The inductive bias of HadamardMLP
 107 enables HadamardMLP to decode the semantic connectivity of different embeddings more easily in
 108 practice. Indeed, specialized structures, e.g. Dot Product, HadamardMLP, convolutional, recurrent,
 109 and attention structures, are common in neural networks. There is probably no hope to replace them
 110 using a ConcatMLP though they should all be representable.

111 Actually, our work is not the first to discuss the practical limitations of ConcatMLP decoders. For
 112 example, the existing work [10, 17, 18] has pointed out that In deep neural network designs, it is very

113 common to replace a ConcatMLP with a more specialized structure that has an inductive bias that
 114 represents the problem better, which is crucial for advancing the state of the art of deep learning,
 115 although in theory they can all be approximated by ConcatMLPs. The inefficiency of ConcatMLPs
 116 to capture dot and tensor products has been studied by [19] in the context of recommender systems.
 117 Similar to our analysis in Sec. 2 and Appendix A, [19] points out that it is hard for ConcatMLPs to
 118 approximate dot products and tensor products with ConcatMLPs empirically.

119 3 Scalability of Link Prediction Decoders

120 Most academic studies focus on training runtime when discussing scalability. However, in industrial
 121 applications, the inference speed is often more important. The inference of many LP applications
 122 needs to retrieve the top scoring neighbors given a source node, e.g., recommending friends to a
 123 user for friend recommendation. Given a source node, if there are n nodes in the graph, then the
 124 inference time complexity is $\mathcal{O}(n)$ if the decoder needs to iterate over all the n nodes to compute
 125 the edge scores. For large scale applications, n is typically in the range of millions, or even larger.
 126 The empirical results show that the inference time of finding the top scoring neighbors for a source
 127 node is longer than one second for HadamardMLP on the OGBL-CITATION2 dataset of nearly three
 128 million nodes (see Sec. 5.5).

129 For a Dot Product decoder, the problem of finding the top scoring neighbors can be approximated
 130 efficiently. This is a well-studied problem, known as approximate maximum inner product search
 131 (MIPS) [20, 21] (see Sec. 6.2 for a comprehensive literature review). MIPS techniques allow Dot
 132 Product’ inference to be completed in a few milliseconds, even with millions of neighbors. There
 133 exists some work that tries to extend MIPS to the ConcatMLP [22, 23]. These methods hold strict
 134 assumptions on the models’ training and are not directly applicable to the HadamardMLP. To the best
 135 of our knowledge, no such sublinear techniques exist for the top scoring neighbor retrieval with the
 136 HadamardMLP [10], which is a complex nonlinear function.

137 To summarize, the HadamardMLP decoder is not scalable for the real time LP services on large graphs,
 138 while the Dot Product decoder allows fast retrieval using the well established MIPS techniques.

139 4 Flashlight: Scalable Link Prediction with Effective Decoders

140 Sec. 2 has shown that the HadamardMLP decoder enjoys higher effectiveness than the Dot Product
 141 decoder, which supports the superior performance of HadamardMLP on many LP benchmarks. On
 142 the other hand, Sec. 3 has shown that the HadamardMLP is not scalable for real time LP applications
 143 on large graphs, while Dot Product supports the fast inference using the well-established MIPS
 144 techniques. In this section, we aim to devise fast inference algorithms for HadamardMLP to enable
 145 scalable LP with effective decoders.

146 We try to exploit the advances in the well-developed MIPS techniques to accelerate the inference of
 147 HadamardMLP. Specifically, we divide the top scoring retrievals for HadamardMLP predictors into a
 148 sequence of MIPS. Our algorithm works in a progressive manner. The query embedding in every
 149 search is adaptively adjusted to find the high scoring neighbors missed in the last search.

150 The challenge of retrieving the neighbors of highest scores for HadamardMLP is rooted in the
 151 unawareness of which neurons are activated, since if we know which neurons are activated, the
 152 nonlinear HadamardMLP degrades to a linear model. On the l th MLP layer, we define the mask
 153 matrix $\mathbf{M}_{\mathcal{A},l} \in \mathbb{R}^{d_l \times d_l}$ to represent the set of activated neurons \mathcal{A} as

$$M_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } i \in \mathcal{A} \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

154 With $\mathbf{M}_{\mathcal{A},l}$, we reformulate the HadamardMLP decoder as:

$$\begin{aligned} s_{ij} &= \phi^{\text{MLP}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{w}_L^T \mathbf{M}_{\mathcal{A},L-1} \mathbf{W}_{L-1} \dots \mathbf{M}_{\mathcal{A},1} \mathbf{W}_1 (\mathbf{x}_i \odot \mathbf{x}_j) \\ &= (\mathbf{W}_1^T \mathbf{M}_{\mathcal{A},1} \dots \mathbf{W}_{L-1}^T \mathbf{M}_{\mathcal{A},L-1} \mathbf{w}_L \odot \mathbf{x}_i) \bullet \mathbf{x}_j \end{aligned} \quad (7)$$

155 In the Equation (7), the final equation holds because the matrix computation on the left hand side of
 156 x_i is equal to a vector of the same dimension as x_i . For the simplicity of expression, we denote it as
 157 left vector. In this way, transposing the matrix computation on the left hand side of x_i and multiply it

158 to x_i is equivalent to the dot product between the left vector and x_i , which leads to the final equation
 159 in Eq. (7). Because the vector $\mathbf{W}_1^T \mathbf{M}_{\mathcal{A},1} \dots \mathbf{W}_{L-1}^T \mathbf{M}_{\mathcal{A},L-1}^T \mathbf{w}_L$ is determined by the weights of
 160 MLP and the activated neurons \mathcal{A} , we term it as $\text{MLP}_{\mathcal{A}}(\cdot)$:

$$\text{MLP}_{\mathcal{A}}(\cdot) := \mathbf{W}_1^T \mathbf{M}_{\mathcal{A},1} \dots \mathbf{W}_{L-1}^T \mathbf{M}_{\mathcal{A},L-1}^T \mathbf{w}_L \quad (8)$$

161 Given the source node i , because the score s_{ij} is obtained by the dot product between
 162 $(\mathbf{W}_1^T \mathbf{M}_{\mathcal{A},1} \dots \mathbf{W}_{L-1}^T \mathbf{M}_{\mathcal{A},L-1}^T \mathbf{w}_L \odot \mathbf{x}_i)$ and the neighbor embedding \mathbf{x}_j , we term the former vector
 163 as the query embedding \mathbf{q} :

$$\mathbf{q} := \mathbf{W}_1^T \mathbf{M}_{\mathcal{A},1} \dots \mathbf{W}_{L-1}^T \mathbf{M}_{\mathcal{A},L-1}^T \mathbf{w}_L \odot \mathbf{x}_i = \text{MLP}_{\mathcal{A}}(\cdot) \odot \mathbf{x}_i \quad (9)$$

164 In this way, we can reformulate the output of decoder $\phi^{\text{MLP}}(\mathbf{x}_i, \mathbf{x}_j)$ as

$$s_{ij} = \phi^{\text{MLP}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q} \bullet \mathbf{x}_j. \quad (10)$$

165 In practice, we can use the \mathbf{q} as the query embedding in MIPS to retrieve the neighbors of highest
 166 inner products, which correspond to the highest scores. Here, how to get the activated neurons \mathcal{A} so
 167 as to obtain the query embedding \mathbf{q} is an issue. Different node pairs activate different neurons \mathcal{A} .
 168 Initially, without knowing which neurons are activated, we first assume all the neurons are activated,
 169 i.e., we have the initial query embedding as:

$$\mathbf{q}[1] = \left(\prod_{i=1}^{L-1} \mathbf{W}_i^T \right) \mathbf{w}_L \odot \mathbf{x}_i \quad (11)$$

170 This initial design can reflect the general trends of increasing the edge scores on LP, without restricting
 171 which neurons are activated. We use $\mathbf{q}[1]$ as the query embedding to retrieve the highest inner product
 172 neighbors as $\mathcal{N}[1]$ in the first iteration. Then, given the retrieved neighbors in the t th iteration as
 173 $\mathcal{N}[t]$, we analyze the $\mathcal{N}[t]$ and adaptively adjust the query embedding $\mathbf{q}[t+1]$ that we use in the
 174 next iteration to find more high scoring neighbors. Specifically, we operate the feed-forward to MLP
 175 for $\mathcal{N}(t)$. We define the function $A(\cdot, \cdot)$ that returns the set of activated neurons for a MLP (the first
 176 input) with the input $\mathbf{x}_i \odot \mathbf{x}_j$ (the second input). Then we can use it to extract \mathcal{A} as:

$$\mathcal{A} = A(\text{MLP}(\cdot), \mathbf{x}_i \odot \mathbf{x}_j). \quad (12)$$

177 Then, we obtain the set of activated neurons of the highest scored neighbor at the t th iteration as:

$$\mathcal{A}[t] \leftarrow A(\text{MLP}(\cdot), \mathbf{x}_i \odot \mathbf{x}_{j^*[t]}), \text{ where } j^*[t] = \arg \max_{j \in \mathcal{N}[t]} \text{MLP}(\mathbf{x}_i \odot \mathbf{x}_j). \quad (13)$$


178 This implies that the neighbors activating $\mathcal{A}[t]$ can obtain the high edge scores. Then, if we take $\mathcal{A}[1]$
 179 as the set of neurons that we activate at the next query, we could find more high scoring neighbors. In
 180 this way, we set the neurons that we assume to activate in the next iteration as $\mathcal{A}[t]$. We repeat the
 181 above iterations until enough neighbors are retrieved. The algorithm is summarized in Alg. 1.

182 We name our algorithm as Flashlight because it works like a flashlight to progressively “illuminates”
 183 the semantic space to find the high scoring neighbors. The query embeddings are like the lights sent
 184 from the flashlight. And our process of adjusting the query embeddings is just like progressively
 185 adjusting the “lights” from the “flashlight” by checking the “objects” found in the last “illumination”.

186 In the experiments, we find that our Flashlight algorithm is effective to find the top scoring neighbors
 187 from the massive candidate neighbors. For example, in Fig. 3, our Flashlight is able to find the top
 188 100 scoring neighbors from nearly three million candidates by retrieving only 200 neighbors in the
 189 large OGBL-CITATION2 graph dataset for the HadamardMLP decoders.

190 **Whether HadamardMLP degraded to a linear model, when we know which neurons are activated,**
 191 **depends on what activation functions are used. Since the link prediction decoders use ReLU [24]**
 192 **as the activation functions in the HadamardMLP decoder, the HadamardMLP degrades to a linear**
 193 **function when what neurons (with ReLU functions) are activated are known.**

194 **The initial iteration that assumes all the neurons are activated only reflect the general trend of the**
 195 **output increasing of HadamardMLP. Then, based on the actual neuron activation condition of the**
 196 **found high scoring neighbors, we can find other high scoring neighbors that activate the close neurons**
 197 **by adaptively adjusting the query embeddings with the new assumed activated neurons. \mathcal{A} is a**
 198 **learning based module or operator, it is just a tracker that check the neurons inside the HadamardMLP**

Algorithm 1 *Flashlight* : progressively “illuminates” the semantic space to retrieve the high scoring neighbors for the LP HadamardMLP decoders.

Input: A trained HadamardMLP decoder ϕ^{MLP} that outputs the logit s_{ij} for the input $\mathbf{x}_i \odot \mathbf{x}_j$. The set of nodes \mathcal{V} . The node embedding set $\mathcal{X} = \{\mathbf{x}_i | i \in \mathcal{V}\}$. A source node i . The number of iterations T . The number of neighbors to retrieve at every iteration: $\mathbf{N} = [N_1, N_2, \dots, N_T]$.

Output: The recommended neighbors \mathcal{N} for the source node i .

- 1: Initialize the set of retrieved recommended neighbors $\mathcal{N} \leftarrow \emptyset$
 - 2: Initialize the set of activated neurons as $\mathcal{A}[0]$ as all the neurons in MLP.
 - 3: **for** $t \leftarrow 1$ to T **do**
 - 4: Calculate the query embedding $\mathbf{q}[t] \leftarrow \mathbf{x}_i \odot \text{MLP}_{\mathcal{A}[t-1]}(\cdot)$.
 - 5: $\mathcal{N}[t] \leftarrow N_t$ neighbors in \mathcal{X} that maximizes the inner product with $\mathbf{q}[t]$.
 - 6: $\mathcal{X} \leftarrow \mathcal{X} \setminus \{\mathbf{x}_j | j \in \mathcal{N}[t]\}$.
 - 7: $j^*[t] = \arg \max_{j \in \mathcal{N}[t]} \text{MLP}(\mathbf{x}_i \odot \mathbf{x}_j)$
 - 8: $\mathcal{A}[t] \leftarrow A(\text{MLP}(\cdot), \mathbf{x}_i \odot \mathbf{x}_{j^*[t]})$.
 - 9: $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}[t]$.
 - 10: **return** \mathcal{N}
-

199 for a specific input. In other words, if the output of a neuron (the input of the following ReLU
200 function) is positive, that neuron will be recorded by A and denoted as being activated. Therefore, it
201 needs not to be trained and can be used directly during inference.

202 The convergence of our Flashlight is guaranteed since it iterates over a limited number of times to
203 find the maximum inner product neighbors with different queries, as shown in line 3 of Algorithm 1.
204 We agree that more theoretical analysis on Flashlight, such as in what cases it can find all the top
205 k -scoring neighbors for HadamardMLP, can further justify the effectiveness of our Flashlight. We
206 take this as an important future direction for exploration.

207 We agree with the reviewers that the theoretical justification surrounding the Flashlight helps to
208 further improve the contribution of our work. Therefore, in Section 3 and 4, we analyze Flashlight
209 and find that our Flashlight can significantly reduce the inference time complexity of HadamardMLP
210 to the sublinear level and thus enhance the applicability and scalability of the state-of-the-art link
211 prediction decoder HadamardMLP by a large margin. In Section 4, we analyzed in theory how to
212 approximate the outputs of HadamardMLP through a sequence of maximum inner product search. In
213 section 2, we analyzed the gap in the expressiveness between HadamardMLP and other decoders. In
214 addition, in Appendix A, we analyze how difficult it is for HadamardMLP to learn a dot product. The
215 analytic results show that the inductive bias of HadamardMLP enables it to learn a dot product easily
216 and outperform other link prediction decoders.

217 Both the inner product maximization search in every Flashlight iteration and the top scoring neighbors
218 from HadamardMLP outputs are to maximize the sum of different paths in the MLP across neurons
219 from the input layer to the output layer. The only difference is that for the former all the paths
220 contribute to the output while for the later only those paths with all the neurons activated contribute
221 to the final result. No matter whether activated or not, every path and every neuron’s monotonicity
222 is not changed. In this sense, in the initial iteration, our Flashlight’s maximizing the inner product
223 is to encourage the values from all paths to be larger, which reflect the general increasing trend of
224 the HadamardMLP. In the later iterations, based on the activation patterns found on the top scoring
225 neighbors, our Flashlight adaptively adjusts the query embedding to find more top scoring neighbors
226 accurately. We agree that more theoretical analysis on Flashlight can further justify the effectiveness
227 of our Flashlight. We take this as an important future direction for exploration and emphasize it in
228 our paper.

229 **Complexity Analysis.** Using MLP decoders to compute the LP probabilities of all the neighbors
230 holds the complexity as $\mathcal{O}(N)$, where N is the number of nodes in the whole graph. Finding the top
231 scoring neighbors from the exact probabilities of all the neighbors also holds the linear complexity
232 $\mathcal{O}(N)$. Overall, using MLP decoders to find the top scoring neighbors is of the time complexity
233 $\mathcal{O}(N)$. In contrast, our Flashlight progressively calls the MIPS techniques for a constant number
234 of times invariant to the graph data, which leads to the sublinear complexity as same as MIPS. In

Table 1: Statistics of datasets.

Dataset	OGBL-DDI	OGBL-COLLAB	OGBL-PPA	OGBL-CITATION2
#Nodes	4,267	235,868	576,289	2,927,963
#Edges	1,334,889	1,285,465	30,326,273	30,561,187

conclusion, our Flashlight improves the scalability and applicability of HadamardMLP decoders by reducing their inference time complexity from linear to sublinear time.

5 Experiments

In this section, we first compare the effectiveness of different LP decoders. We find that the HadamardMLP decoders generally perform better than other decoders. Then, we implement our Flashlight algorithm with LP models to show that Flashlight effectively retrieves the top scoring neighbors for the HadamardMLP decoders. As a result, the inference efficiency and scalability of HadamardMLP decoders are improved significantly by our work.

In Table 2, we report the performance of LP methods with different decoders: Dot Product, Bilinear, ConcatMLP, HadamardMLP, HadamardMLP with Flashlight, as denoted in different column names. In Fig. 3, 4, 5, we report the experimental results with the decoder HadamardMLP and HadamardMLP with Flashlight, which hold much better effectiveness on link prediction than other decoders, as discussed in Sec. 6.3.

5.1 Datasets

We evaluate the link prediction on Open Graph Benchmark (OGB) data [25]. We use four OGB datasets with different graph types, including OGBL-DDI, OGBL-COLLAB, OGBL-CITATION2, and OGBL-PPA. OGBL-DDI is a homogeneous, unweighted, undirected graph, representing the drug-drug interaction network. Each node represents a drug. Edges represent interactions between drugs. OGBL-COLLAB is an undirected graph, representing a subset of the collaboration network between authors indexed by MAG. Each node represents an author and edges indicate the collaboration between authors. All nodes come with 128-dimensional features. OGBL-CITATION2 is a directed graph, representing the citation network between a subset of papers extracted from MAG. Each node is a paper with 128-dimensional word2vec features. OGBL-PPA is an undirected, unweighted graph. Nodes represent proteins from 58 different species, and edges indicate biologically meaningful associations between proteins. The statistics of these datasets is presented in Table. 1.

5.2 Hyper-parameter Settings

For all experiments in this section, we report the average and standard deviation over ten runs with different random seeds. The results are reported on the the best model selected using validation data. We set hyper-parameters of the used techniques and considered baseline methods, e.g., the batch size, the number of hidden units, the optimizer, and the learning rate as suggested by their authors. We use the recent MIPS method ScaNN [21] in the implementation of our Flashlight. For the hyper-parameters of our Flashlight, we have found in the experiments that the performance of Flashlight is robust to the change of hyper-parameters in a board range. Therefore, we simply set the number of iterations of our Flashlight as $T = 3$ and the number of retrieved neighbors constant as 200 per iteration by default. We run all experiments on a machine with 80 Intel(R) Xeon(R) E5-2698 v4 @ 2.20GHz CPUs, and a single NVIDIA V100 GPU with 16GB RAM.

5.3 Effectiveness of Link Prediction Decoders

We follow the standard benchmark settings of OGB datasets to evaluate the effectiveness of LP with different decoders. The benchmark setting of OGBL-DDI is to predict drug-drug interactions given information on already known drug-drug interactions. The performance is evaluated by Hits@20: each true drug interaction is ranked among a set of approximately 100,000 randomly-sampled negative drug interactions, and count the ratio of positive edges that are ranked at 20-place or above. The task of OGBL-COLLAB is to predict the future author collaboration relationships given the past collaborations. Evaluation metric is Hits50, where each true collaboration is ranked among a set

Table 2: The test effectiveness comparison of LP decoders on four OGB datasets (DDI, COLLAB, PPA, and CITATION2) [16]. We report the results of the standard metrics averaged over 10 runs following the existing work [6, 16]. HadamardMLP is more effective than other decoders. Flashlight effectively retrieves the top scoring neighbors for HadamardMLP and keep its exact outputs.

Decoder	Dot Product	Bilinear	ConcatMLP	HadamardMLP	HadamardMLP w/ Flashlight
OGBL-DDI					
GCN [5]	13.8 ± 1.8	16.1 ± 1.2	12.9 ± 1.4	37.1 ± 5.1	37.1 ± 5.1
GraphSAGE [12]	36.5 ± 2.6	39.4 ± 1.7	34.2 ± 1.9	53.9 ± 4.7	53.9 ± 4.7
Node2Vec [26]	11.6 ± 1.9	13.8 ± 1.6	10.8 ± 1.7	23.3 ± 2.1	23.3 ± 2.1
OGBL-COLLAB					
GCN [5]	42.9 ± 0.7	43.2 ± 0.9	42.3 ± 1.0	44.8 ± 1.1	44.8 ± 1.1
GraphSAGE [12]	37.3 ± 0.9	41.5 ± 0.8	37.0 ± 0.7	48.1 ± 0.8	48.1 ± 0.8
Node2Vec [26]	27.7 ± 1.1	31.5 ± 1.0	27.2 ± 0.8	48.9 ± 0.5	48.9 ± 0.5
OGBL-PPA					
GCN [5]	5.1 ± 0.4	5.8 ± 0.5	6.2 ± 0.6	18.7 ± 1.3	18.7 ± 1.3
GraphSAGE [12]	3.2 ± 0.3	6.5 ± 0.7	5.8 ± 0.4	16.6 ± 2.4	16.6 ± 2.4
Node2Vec [26]	4.2 ± 0.5	7.8 ± 0.6	8.3 ± 0.4	22.3 ± 0.8	22.3 ± 0.8
OGBL-CITATION2					
GCN [5]	65.3 ± 0.4	69.0 ± 0.8	62.7 ± 0.3	84.7 ± 0.2	84.7 ± 0.2
GraphSAGE [12]	62.2 ± 0.7	65.4 ± 0.9	60.8 ± 0.6	80.4 ± 0.1	80.4 ± 0.1
Node2Vec [26]	52.7 ± 0.8	54.1 ± 0.6	51.4 ± 0.5	61.4 ± 0.1	61.4 ± 0.1

of 100,000 randomly-sampled negative collaborations. The task of OGBL-PPA is to predict new association edges given the training edges. Evaluation metric is Hits@100, where each positive edge is ranked among 3,000,000 randomly-sampled negative edges. The task of OGBL-CITATION2 is predict missing citation given existing citations. The evaluation metric is Mean Reciprocal Rank (MRR), where the reciprocal rank of the true reference among 1,000 sampled negative candidates is calculated for each source nodes, and then the average is taken over all source nodes.

We implement different decoders as introduced in Sec. 2, including the Dot Product, Bilinear, ConcatMLP, and the HadamardMLP decoders, over the LP encoders, including GCN [5], GraphSAGE [12], and Node2Vec [26], to compare the effects of different decoders on the LP effectiveness. We present the results on the OGBL-DDI, OGBL-COLLAB, OGBL-PPA, and OGBL-CITATION2 datasets in Table. 2. We observe that the HadamardMLP decoder outperforms other decoders on all encoders and datasets. Our Flashlight algorithm can effectively retrieve the top scoring neighbors for the HadamardMLP decoder and keep the exact LP probabilities of HadamardMLPs’ output, which leads to the same results of the HadamardMLP decoder with and without Flashlight.

Our Flashlight is to reduce the search space of HadamardMLP to improve the inference efficiency. For the top scoring neighbors, the final exact link prediction scores and orders are still determined by HadamardMLP without being influenced by our Flashlight. Our Flashlight is able to accurately retrieve the candidate neighbors that include the top scoring ones for the HadamardMLP decoder. This is why in Table 2, the HadamardMLP with and without our Flashlight exhibit the same performance. In Table 2, the training and inference time mainly vary by how many links that the link prediction models need to make predictions on. It is defined the settings of different benchmarks. For example, in the CITATION2 dataset, the model needs to predict the 1,000 negative neighbor candidates for every positive link. These settings are defined in the official OGBL benchmarks and we follow them to conduct the experiments in Table 2. For example, the training time on the per epoch is around 4 minutes on the OGBL-CITATION2 dataset, in our experiments running on a machine with 80 Intel(R) Xeon(R) E5-2698 v4 @ 2.20GHz CPUs, and a single NVIDIA V100 GPU with 16GB RAM. Empirically, we observe that the training and inference time grows linearly by the number of links that the model needs to predict on.

We agree with the reviewers that the different performance gaps between the baseline methods are worth investigation. A major difference between GraphSAGE and GCN is that the former samples partial neighbors for message passing during training. And a significant difference between DDI and other datasets is its small number of nodes and higher average degrees per node. When aggregating the features from massive neighbors in DDI, it is challenging for GCN to aggregate the most valuable information from massive neighbors. In contrast, GraphSAGE is able to aggregate information from partial neighbors at different training steps, which act as a kind of data augmentation for GraphSAGE

314 to adapt to different neighbors’ diverse information specialized by the complex graph topology. We
 315 follow the reviewers’ comments to discuss this phenomenon among the baseline methods to improve
 316 our paper.

317 In our experiments, we strictly follow the official settings of OGBL leaderboards to guarantee a fair
 318 comparison. Therefore, our results are as same as those reported in the leaderboards. In the OGBL
 319 leaderboards, the decoders that are used with the method GCN, GraphSAGE, and Node2Vec in the
 320 OGBL leaderboards are all the HadmardMLP, as shown in the official implementations given by the
 321 OGBL group .

322 In Table 2, different columns refer to the performances of different decoders. Therefore, to compare
 323 the results in Table 2 with those in the OGBL leaderboard of the same encoders, please refer to the
 324 column of HadmardMLP, the fifth column in Table 2. The performance in Table 2 is as same as those
 325 in the OGBL leaderboard for the same encoders since we strictly follow the official settings of OGBL
 326 leaderboards to guarantee a fair comparison.

327 Note that the benchmark settings of these datasets sample a small portion of negative edges for the test
 328 evaluation, which is not challenging enough to evaluate the scalability of LP decoders on retrieving the
 329 top scoring neighbors from massive candidates in practice.

330 5.4 The Flashlight Algorithm Effectively Finds the Top Scoring Neighbors

331 To evaluate the effectiveness of our Flashlight on retrieving the top scoring neighbors for the
 332 HadamardMLP decoder, we propose a more challenging test setting for the OGB LP datasets.
 333 Given a source node, we takes its top 100 scoring neighbors of the HadamardMLP decoder as the
 334 ground-truth for retrievals. We set the task as retrieving k neighbors for a source node that can match
 335 the ground-truth neighbors as much as possible. We formally define the metric as $\text{Recall}@k$, which is
 336 the portion of the ground-truth neighbors being in the top k neighbors retrieved by different methods.

337 We sample 1000 nodes as the source nodes from the OGBL-DDI and OGBL-
 338 CITATION2 datasets respectively for evaluation. We evaluate the effectiveness of our
 339 Flashlight algorithm by checking whether it can find the top scoring neighbors for every
 340 source node. We set the number of Flashlight iterations as 10 and the number of re-
 341 trieved neighbors per iteration as 50. We present the $\text{Recall}@k$ for k from 1 to 500
 342 averaged over all the source nodes in Fig. 3. The “oracle” curve represents the performance
 343 of a optimum searcher, of which the retrieved top k neighbors are exactly the top
 344 k scoring neighbors of HadamardMLP.
 345

346 When $k = 100$, the 100 neighbors retrieved by our Flashlight can cover more than 80%
 347 ground-truth neighbors. When $k \geq 200$, the recall reaches 100%. As a comparison,
 348 if we randomly sample the candidate neighbors for retrievals, the $\text{Recall}@k$ grows linearly with k
 349 and is less than 1×10^{-4} for $k = 100$ on the OGBL-CITATION2 dataset. The curves of Flashlight is
 350 close the optimum curve of the “oracle”. These results demonstrate the highly effectiveness of our
 351 Flashlight on finding the top scoring neighbors.

352 Given the large OGBL-Citation2 dataset and smaller DDI dataset, our Flashlight exhibits similar
 353 $\text{Recall}@k$ performance given different numbers k of retrieved neighbors. This implies that our
 354 Flashlight can accurately find the top scoring neighbors for both small and large graphs.

363 5.5 Inference Efficiency of Link Prediction with Our Flashlight Algorithm

364 We use the throughputs to evaluate the inference speed of neighbor retrieval of different methods.
 365 The throughput is defined as how many source nodes that a method can serve to retrieve the top
 366 100 scoring neighbors per second. Except for the LP models that follow the encoder and decoder

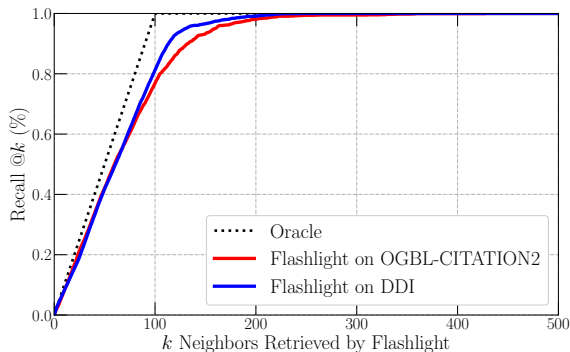


Figure 3: $\text{Recall}@k$ is the fraction of the 100 top scoring neighbors of HadamardMLP ranked in the top k neighbors retrieved by Flashlight. We report $\text{Recall}@k$ averaged over all the source nodes on OGBL-CITATION2 and OGBL-DDI.

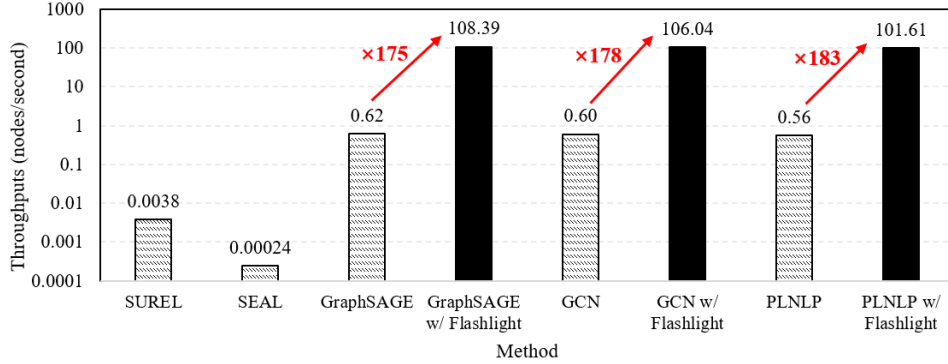


Figure 4: The inference speed of different LP methods on the OGBL-CITATION2 dataset. The y-axis (throughputs) is in the logarithmic scale.

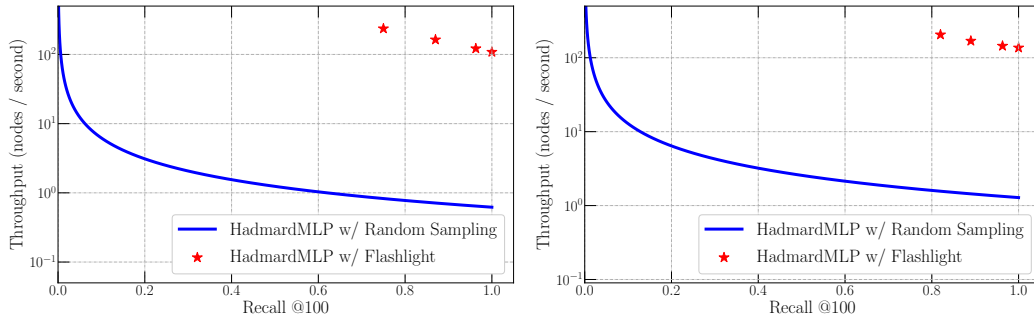


Figure 5: The tradeoff between the inference speed (y-axis) and the effectiveness of finding the top scoring neighbors (x-axis) on the OGBL-CITATION2 (left) and OGBL-PPA (right) datasets.

367 architectures, e.g., GraphSAGE [12], GCN [5], and PLNLP [6], there are some subgraph based LP
 368 models, e.g., SUREL [7] and SEAL [27]. The common issue of the subgraph based models is the poor
 369 efficiency: they have to crop a separate subgraph for every node pair to calculate the LP probability
 370 on the node pair. In this sense, the node embeddings cannot be shared on the LP calculation for
 371 different node pairs. This leads to the much lower inference speed of the subgraph based LP models
 372 than the encoder-decoder LP models. We compare the inference efficiency of different methods on
 373 the OGBL-CITATION2 dataset in Fig. 4, where we present the inference speed of different methods
 374 when achieving the 100% Recall for the top 100 scoring neighbors.

375 We observe that our Flashlight significantly accelerate the inference speed of LP models GraphSAGE
 376 [12], GCN [5], and PLNLP [6] with the HadamardMLP decoders by more than 100 times. This gap
 377 will be even larger for the datasets of larger scales, because the inference with our Flashlight holds the
 378 sublinear time complexity while the HadamardMLP decoders holds the linear complexity. Note that
 379 the y-axis is in logarithmic scale. The subgraph based methods SUREL [7] and SEAL [27] hold the
 380 inference speed of throughputs lower than 1×10^{-2} and 1×10^{-3} respectively, which is not applicable to
 381 the practical services that require the low latency of milliseconds.

382 Taking a further step, we comprehensively evaluate the tradeoff between the inference speed and the
 383 effectiveness of finding the top scoring neighbors. Taking GraphSAGE as the encoder, we present
 384 the tradeoff curves between the throughputs and the Recall on retrieving the top 100 neighbors for
 385 the OGBL-CITATION2 and OGBL-PPA datasets in Fig. 5. In comparison with our Flashlight, we
 386 take the HadamardMLP decoder with the Random Sampling as the baseline for comparison. For
 387 example, on the OGBL-CITATION2 dataset, when achieving the Recall as more than 80%, the
 388 HadamardMLP with our Flashlight can serve more than 200 source nodes per second, while the
 389 HadamardMLP with the random sampling can only serve less than 1 node per second. Overall, our
 390 Flashlight achieves much better inference speed and effectiveness tradeoff than the HadamardMLP
 391 with random sampling.

392 We take the random sampling as a baseline method to reduce the search space for HadmardMLP
 393 because it is easy to understand and can act as a default choice when reducing the search space.
 394 Wwe have taken a stronger baseline DotMax for comparison. We believe that the DotMax is a good
 395 baseline for comparison to distinguish the effectiveness on inference acceleration of our Flashlight.

396 5.6 Ablation Study

397 We analyze the sensitivity of Flashlight to the hyper-parameter: the number of iterations, and the
 398 number of retrieved neighbors per iteration. The recall result on retrieving the top 100 scoring
 neighbors for the OGBL-CITATION2 dataset is presented in the following table: We alter number

Number of Iterations	1	2	3	4	5
Retrieving 100 neighbors per iteration	58.7%	91.5%	99.2%	100.0%	100.0%
Retrieving 200 neighbors per iteration	64.3%	94.2%	100.0%	100.0%	100.0%
Retrieving 300 neighbors per iteration	66.2%	95.7%	100.0%	100.0%	100.0%
Retrieving 400 neighbors per iteration	67.6%	98.9%	100.0%	100.0%	100.0%

399 the number of iterations among {1, 2, 3, 4, 5} and number of retrieved neighbors per iteration
 400 among {100, 200, 300, 400}. For the result corresponding to Flashlight using the all-one mask
 401 neuron activation, please refer to the first column. The performance of Flashlight is relatively smooth
 402 when parameters are within certain ranges. However, extremely small values of the number of
 403 iterations and the number of retrieved neighbors per iteration result in poor performances. A too
 404 small number of iterations make the Flashlight unable to adaptively adjust its querying embedding
 405 on finding neighbors, while a too small number of retrieved neighbors per iteration make Flashlight
 406 unable to retrieve enough neighbors to cover the top 100 scoring neighbors, i.e., the ground-truth
 407 ones. Empirically, increasing the number of iterations for Flashlight can boost the performance
 408 more fast than increasing the number of retrieved neighbors per iteration. The reason is that with
 409 more iterations, Flashlight can find the neuron activation patterns of the high scoring neighbors more
 410 effectively and thus be able to adaptively adjust the query embeddings. Moreover, only a poorly set
 411 hyper-parameter does not lead to significant performance degradation, which demonstrates that our
 412 Flashlight framework is able to find the high scoring neighbors among massive candidates on large
 413 graphs. We follow the reviewers’ suggestions to add these additional experimental results to improve
 414 our paper.
 415

416 5.7 Comparison with More Baselines and Datasets

417 We agree with the reviewers that comparing our Flashlight method with more baselines can better
 418 justify the effectiveness of Flashlight. We also agree that finding the neighbors that maximize the
 419 dot product between the target node and neighbors’ embeddings to reduce the search space for
 420 the HadmardMLP is a good baseline to compare with. For the simplicity of expression, we term
 421 this baseline as DotMax. We conducted the experiments on finding the top scoring neighbors for
 422 HadmardMLP with DotMax. We found that DotMax needs to retrieve much more neighbors than
 423 our Flashlight to achieve the same Recall as Flashlight on finding the top 100 scoring neighbors.
 424 For example, on the OGBL-CITATION2 dataset, which is the largest dataset among the used data
 425 holding nearly 3 million nodes as shown in Table 1, we follow the experimental settings as introduced
 426 in Sec. 6.4 to test DotMax. DotMax achieves the following Recall on finding the top 100 scoring
 neighbors for HadmardMLP with different numbers of retrieved neighbors: In comparison, under

Number of Neighbors Retrieved by DotMax	20000	40000	60000	80000	100000
Recall on Finding the Top 100 Scoring Neighbors	28.3%	42.5%	51.1%	63.4%	71.2%

427 the same experimental setting, when retrieving only 100 neighbors, our Flashlight can cover more
 428 than 80% of the top 100 scoring neighbors for HadmardMLP (as shown in Fig. 3). When retrieving
 429 more than 200 neighbors, Flashlight achieves the recall of 100%. Overall, our proposed Flashlight
 430 can reduce the search space for the HadmardMLP decoder much more effectively than the DotMax.
 431 The reason is that Dot Product demands the homophily of graph data to effectively infer the link
 432 between nodes. In contrast, thanks to the universal approximation capability, MLP can approximate
 433 any continuous function, and thus does not demand the homophily of graph data for effective LP. This
 434 constraint makes the Dot Product and HadamardMLP prefer different patterns of nodes’ semantic
 435

436 embeddings on computing the link prediction scores. As a result, using DotMax to reduce the search
 437 space cannot effectively cover the top scoring neighbors for HadamardMLP with a small number of
 438 retrieved neighbors.

439 Overall, our Flashlight improves the link prediction inference speed of HadamardMLP much more
 440 significantly than DotMax. For example, under the experimental setting introduced in Sec. 6.5. The
 441 inference speed in terms of throughputs of HadamardMLP (nodes/second), of HadamardMLP with
 442 DotMax, and of HadamardMLP with Flashlight on OGBL- CITATION2, when the Recall on retrieving
 the top 100 neighbors on the OGBL-CITATION2 dataset is 100%, are as following respectively: The

Method	HadamardMLP	HadamardMLP w/ DotMax	HadamardMLP w/ Flashlight
Inference Throughputs (nodes/second)	0.62	6.13	108.45

443 experiments are run on on a machine with 80 Intel(R) Xeon(R) E5-2698 v4 @ 2.20GHz CPUs, and a
 444 single NVIDIA V100 GPU with 16GB RAM. Overall, our Flashlight achieves much higher inference
 445 acceleration than the DotMax for the HadamardMLP decoder.
 446

447 Our Flashlight algorithm holds sublinear time complexity on finding top scoring neighbors for
 448 HadamardMLP decoders, allowing for fast and scalable inference. Empirical results show that
 449 Flashlight accelerates the inference of LP models by more than 100 times on the large OGBL-
 450 CITATION2 dataset without sacrificing the effectiveness. Overall, our work paves the way for the
 451 use of effective LP decoders in practical settings by greatly accelerating their inference.

452 We take the newly constructed OGBL dataset OGBL leaderboard OGBL-VESSEL as an additional
 453 dataset for evaluation. OGBL-VESSEL holds 3,538,495 nodes and 5,345,897 edges. The OGBL-
 454 VESSEL dataset is an undirected, unweighted spatial graph of the whole mouse brain [28]. To
 455 generate it, the authors developed a graph extraction pipeline, where nodes represent bifurcation
 456 points, and edges represent the vessels. The node features are 3-dimensional, representing the spatial
 457 (x, y, z) coordinates of the nodes in Allen Brain atlas reference space. The OGBL-VESSEL graph
 458 aims to inspire researchers in the neuroscience domain to adapt graph-structure representations
 459 for their research. For machine learning researchers, this dataset raises challenging graph learning
 460 research questions in terms of incorporating biological priors into learning algorithms, or in scaling
 461 these algorithms to handle sparse, spatial graphs with millions of nodes and edges.

462 Following the experimental setting in Sec. 6.5, we test the inference speed in terms of throughputs of
 463 HadamardMLP (nodes/second), of HadamardMLP with DotMax, and of HadamardMLP with Flashlight
 464 on OGBL-VESSEL, when the Recall on retrieving the top 100 neighbors on the OGBL-VESSEL
 dataset is 100%. The results are presented as follows: The experiments are run on on a machine with

Method	HadamardMLP	HadamardMLP w/ DotMax	HadamardMLP w/ Flashlight
Inference Throughputs (nodes/second)	0.49	4.11	91.52

465 80 Intel(R) Xeon(R) E5-2698 v4 @ 2.20GHz CPUs, and a single NVIDIA V100 GPU with 16GB
 466 RAM. Overall, our Flashlight achieves much higher inference acceleration than the DotMax for the
 467 HadamardMLP decoder on the OGBL-VESSEL dataset.
 468

469 We agree that the evaluation on more complex datasets can further justify the applicability of our
 470 method. Therefore, we additionally test our Flashlight method on the heterogenous OGB dataset:
 471 OGBN-MAG. OGBN-MAG is a heterogeneous network composed of a subset of the Microsoft
 472 Academic Graph (MAG). It contains 736,389 papers and 1,134,649 authors as well as the links
 473 between the above two kinds of nodes: an author “writes” a paper. Each paper is associated with a
 474 128-dimensional word2vec feature vector.

475 We follow the experimental settings in Sec. 5.4 to conducted the experiments on finding the top
 476 scoring authors for every paper for GCN with HadamardMLP. We found that our Flashlight exhibits
 477 consistently high effectiveness on finding the top scoring neighbors on the OGBN-MAG. Specifically,
 478 our Flashlight achieves the following Recall on finding the top 100 scoring neighbors for HadamardMLP
 479 with different numbers of retrieved neighbors:

480 We found that DotMax needs to retrieve much more neighbors than our Flashlight to achieve the
 481 same Recall as Flashlight on finding the top 100 scoring neighbors. For example, on the OGBL-
 482 CITATION2 dataset, which is the largest dataset among the used data holding nearly 3 million nodes

483 as shown in Table 1, we follow the experimental settings as introduced in Sec. 6.4 to test DotMax.
 484 DotMax achieves the following Recall on finding the top 100 scoring neighbors for HadamardMLP
 with different numbers of retrieved neighbors:

Number of Neighbors Retrieved by Flashlight	50	100	150	200	250
Recall on Finding the Top 100 Scoring Neighbors	42.8%	77.9%	91.6%	97.1%	99.2%

485

486 In contrast, the baseline method DotMax (as shown in our response to Q1 of reviewer LgUa), DotMax
 487 needs to retrieve much more neighbors than our Flashlight to achieve the same Recall as Flashlight
 488 on finding the top 100 scoring neighbors:

Number of Neighbors Retrieved by DotMax	20000	40000	60000	80000	100000
Recall on Finding the Top 100 Scoring Neighbors	22.1%	39.3%	46.4%	55.2%	60.1%

489 We observe that, under the same experimental setting, when retrieving only 150 neighbors, our
 490 Flashlight can cover more than 90% of the top 100 scoring neighbors for HadamardMLP, while
 491 DotMax needs to retrieve more than 100000 neighbors to achieve the recall of only around 60%. Dot
 492 Product demands the homophily of graph data to effectively infer the link between nodes, which
 493 makes it hard to effectively work on the heterogeneous graphs. In contrast, our Flashlight effectively
 494 retrieve the high scoring neighbors for HadamardMLP, which does not rely on the homophily of
 495 graph data for effective link prediction.

496 Our Flashlight algorithm holds sublinear time complexity on finding top scoring neighbors for
 497 HadamardMLP decoders, allowing for fast and scalable inference. Empirical results show that
 498 Flashlight accelerates the inference of LP models by more than 100 times on the large OGBL-
 499 CITATION2 and OGBL-VESSEL datasets without sacrificing the effectiveness. Overall, our work
 500 paves the way for the use of effective LP decoders in practical settings by greatly accelerating their
 501 inference.

502 6 Related Work

503 6.1 Link Prediction Models

504 Existing LP models can be categorized into three families: heuristic feature based [3, 9, 29–31],
 505 latent embedding based [12, 26, 32–35], and neural network based ones. The neural network-based
 506 link prediction models are mainly developed in recent years, which explore non-linear deep structural
 507 features with neural layers. Variational graph auto-encoders [13] predict links by encoding graph with
 508 graph convolutional layer [5]. Another two state-of-the-art neural models WLNLM [36] and SEAL
 509 [37] use graph labeling algorithm to transfer union neighborhood of two nodes (enclosing subgraph)
 510 as meaningful matrix and employ convolutional neural layer or a novel graph neural layer DGCNN
 511 [38] for encoding. More recently, [6, 8] summarized the architectures LP models, and formally define
 512 the encoders and decoders.

513 Different from the previous work, we focus on analyzing the effectiveness of different LP decoders
 514 and improving the scalability of the effective LP decoders. In practice, we find that the Hadamard
 515 decoders exhibit superior effectiveness but poor scalability for inference. Our work significantly
 516 accelerates the inference of HadamardMLP decoders to make the effective LP scalable.

517 6.2 Maximum Inner Product Search

518 Finding the top scoring neighbors for the Dot Product decoder at the sublinear time complexity is a
 519 well studied research problem, known as the approximate maximum inner product search (MIPS).
 520 There are several approaches to MIPS: sampling based [11, 39, 40], LSH-based [41–44], graph based
 521 [45–47], and quantization approaches [20, 21]. MIPS is a fundamental building block in various
 522 application domains [48–53], such as information retrieval [54, 55], pattern recognition [56, 57], data
 523 mining [58, 59], machine learning [60, 61], and recommendation systems [62, 63].

524 With the explosive growth of datasets’ scale and the inevitable curse of dimensionality, MIPS is
 525 essential to offer the scalable services. However, the HadamardMLP decoders are nonlinear and there
 526 do not exist the well studied sublinear complexity algorithms to find the top scoring neighbors for

527 HadamardMLP [10]. In this work, we utilize the well studied approximate MIPS techniques with the
 528 adaptively adjusted query embeddings to find the top scoring neighbors for the MLP decoders in a
 529 progressive manner. Our method supports the plug-and-play use during inference and significantly
 530 accelerates the LP inference with the effective MLP decoders.

531 7 Conclusion

532 Our theoretical and empirical analysis suggests that the HadamardMLP decoders are a better default
 533 choice than the Dot Product in terms of LP effectiveness. Because there does not exist a well-
 534 developed sublinear complexity top scoring neighbor searching algorithm for HadamardMLP, the
 535 HadamardMLP decoders are not scalable and cannot support the fast inference on large graphs. To
 536 resolve this issue, we propose the Flashlight algorithm to accelerate the inference of LP models with
 537 HadamardMLP decoders. Flashlight progressively operates the well-studied MIPS techniques for a
 538 few iterations. We adaptively adjust the query embeddings at every iteration to find more high scoring
 539 neighbors. Empirical results show that our Flashlight accelerates the inference of LP models by more
 540 than 100 times on the large OGBL-CITATION2 graph. Overall, our work paves the way for the use
 541 of strong LP decoders in practical settings by greatly accelerating their inference.

542 References

- 543 [1] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011. 1
- 544 [2] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33, 2016. 1
- 545 [3] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3): 211–230, 2003. 1, 13
- 546 [4] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. 1
- 547 [5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 2, 3, 8, 10, 13
- 548 [6] Zhitao Wang, Yong Zhou, Litao Hong, Yuanhang Zou, and Hanjing Su. Pairwise learning for neural link prediction. *arXiv preprint arXiv:2112.02936*, 2021. 1, 2, 3, 8, 10, 13, 17, 18
- 549 [7] Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. Algorithm and system co-design for efficient subgraph-based graph representation learning. *arXiv preprint arXiv:2202.13538*, 2022. 1, 10
- 550 [8] Chuxiong Sun and Guoshi Wu. Adaptive graph diffusion networks with hop-wise attention. *arXiv preprint arXiv:2012.15024*, 2020. 1, 2, 3, 13, 17, 18
- 551 [9] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009. 1, 13
- 552 [10] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Fourteenth ACM conference on recommender systems*, pages 240–248, 2020. 1, 2, 3, 4, 14, 17, 18
- 553 [11] Rui Liu, Tianyi Wu, and Barzan Mozafari. A bandit approach to maximum inner product search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4376–4383, 2019. 1, 13
- 554 [12] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 2, 3, 8, 10, 13
- 555 [13] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016. 2, 13
- 556 [14] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018. 2
- 557
- 558
- 559
- 560
- 561
- 562
- 563
- 564
- 565
- 566
- 567
- 568
- 569
- 570
- 571
- 572
- 573
- 574
- 575

- 576 [15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of*
577 *control, signals and systems*, 2(4):303–314, 1989. 2, 3
- 578 [16] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-
579 lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*,
580 2021. 3, 8
- 581 [17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez,
582 Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information*
583 *processing systems*, 30, 2017. 3
- 584 [18] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang
585 Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine
586 translation system: Bridging the gap between human and machine translation. *arXiv preprint*
587 *arXiv:1609.08144*, 2016. 3
- 588 [19] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent
589 cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh*
590 *ACM International Conference on Web Search and Data Mining*, pages 46–54, 2018. 4
- 591 [20] Xinyan Dai, Xiao Yan, Kelvin KW Ng, Jiu Liu, and James Cheng. Norm-explicit quantization:
592 Improving vector quantization for maximum inner product search. In *Proceedings of the AAAI*
593 *Conference on Artificial Intelligence*, volume 34, pages 51–58, 2020. 4, 13
- 594 [21] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv
595 Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International*
596 *Conference on Machine Learning*, pages 3887–3896. PMLR, 2020. 4, 7, 13
- 597 [22] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. Fast item ranking under neural network
598 based measures. In *Proceedings of the 13th International Conference on Web Search and Data*
599 *Mining*, pages 591–599, 2020. 4
- 600 [23] Rihan Chen, Bin Liu, Han Zhu, Yaoxuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang,
601 Yunlong Xu, Hongbo Deng, et al. Approximate nearest neighbor search under neural similarity
602 metric for large-scale recommendation. *arXiv preprint arXiv:2202.10226*, 2022. 4
- 603 [24] Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu
604 activation. *Advances in neural information processing systems*, 30, 2017. 5
- 605 [25] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele
606 Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs.
607 *Advances in neural information processing systems*, 33:22118–22133, 2020. 7
- 608 [26] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In
609 *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and*
610 *data mining*, pages 855–864, 2016. 8, 13
- 611 [27] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using
612 graph neural networks for multi-node representation learning. *Advances in Neural Information*
613 *Processing Systems*, 34:9061–9073, 2021. 10
- 614 [28] Johannes C Paetzold, Julian McGinnis, Suprosanna Shit, Ivan Ezhov, Paul Büschl, Chinmay
615 Prabhakar, Anjany Sekuboyina, Mihail Todorov, Georgios Kaissis, Ali Ertürk, et al. Whole brain
616 vessel graphs: A dataset and benchmark for graph learning and neuroscience. In *Thirty-fifth*
617 *Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round*
618 *2)*, 2021. 12
- 619 [29] Gobinda G Chowdhury. *Introduction to modern information retrieval*. Facet publishing, 2010.
620 13
- 621 [30] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In
622 *Proceedings of the twelfth international conference on Information and knowledge management*,
623 pages 556–559, 2003.
- 624 [31] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceed-*
625 *ings of the eighth ACM SIGKDD international conference on Knowledge discovery and data*
626 *mining*, pages 538–543, 2002. 13
- 627 [32] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Joint*
628 *european conference on machine learning and knowledge discovery in databases*, pages 437–
629 452. Springer, 2011. 13

- 630 [33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social repre-
631 sentations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge*
632 *discovery and data mining*, pages 701–710, 2014.
- 633 [34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-
634 scale information network embedding. In *Proceedings of the 24th international conference on*
635 *world wide web*, pages 1067–1077, 2015.
- 636 [35] Zhitao Wang, Chengyao Chen, and Wenjie Li. Predictive network representation learning for
637 link prediction. In *Proceedings of the 40th international ACM SIGIR conference on research*
638 *and development in information retrieval*, pages 969–972, 2017. 13
- 639 [36] Muhan Zhang and Yixin Chen. Weisfeiler-lehman neural machine for link prediction. In
640 *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and*
641 *data mining*, pages 575–583, 2017. 13
- 642 [37] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in*
643 *neural information processing systems*, 31, 2018. 13
- 644 [38] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning
645 architecture for graph classification. In *Proceedings of the AAAI conference on artificial*
646 *intelligence*, volume 32, 2018. 13
- 647 [39] Edith Cohen and David D Lewis. Approximating matrix multiplication for pattern recognition
648 tasks. *Journal of Algorithms*, 30(2):211–252, 1999. 13
- 649 [40] Hsiang-Fu Yu, Cho-Jui Hsieh, Qi Lei, and Inderjit S Dhillon. A greedy approach for budgeted
650 maximum inner product search. *Advances in neural information processing systems*, 30, 2017.
651 13
- 652 [41] Qiang Huang, Guihong Ma, Jianlin Feng, Qiong Fang, and Anthony KH Tung. Accurate
653 and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In
654 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &*
655 *Data Mining*, pages 1561–1570, 2018. 13
- 656 [42] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product
657 search. In *International Conference on Machine Learning*, pages 1926–1934. PMLR, 2015.
- 658 [43] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner
659 product search (mips). *Advances in neural information processing systems*, 27, 2014.
- 660 [44] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. Norm-ranging lsh for
661 maximum inner product search. *Advances in Neural Information Processing Systems*, 31, 2018.
662 13
- 663 [45] Jie Liu, Xiao Yan, Xinyan Dai, Zhirong Li, James Cheng, and Ming-Chang Yang. Understanding
664 and improving proximity graph based maximum inner product search. In *Proceedings of the*
665 *AAAI Conference on Artificial Intelligence*, volume 34, pages 139–146, 2020. 13
- 666 [46] Stanislav Morozov and Artem Babenko. Non-metric similarity graphs for maximum inner
667 product search. *Advances in Neural Information Processing Systems*, 31, 2018.
- 668 [47] Zhixin Zhou, Shulong Tan, Zhaozhuo Xu, and Ping Li. Möbius transformation for fast inner
669 product search on graph. *Advances in Neural Information Processing Systems*, 32, 2019. 13
- 670 [48] Kazuo Aoyama, Kazumi Saito, Hiroshi Sawada, and Naonori Ueda. Fast approximate similarity
671 search based on degree-reduced neighborhood graphs. In *Proceedings of the 17th ACM SIGKDD*
672 *international conference on Knowledge discovery and data mining*, pages 1055–1063, 2011. 13
- 673 [49] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. Hd-index: Pushing the
674 scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *arXiv*
675 *preprint arXiv:1804.06829*, 2018.
- 676 [50] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search
677 with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*, 2017.
- 678 [51] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search
679 using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and*
680 *machine intelligence*, 42(4):824–836, 2018.
- 681 [52] Philipp M Riegger. Literature survey on nearest neighbor search and search in graphs. 2010.

- 682 [53] Wenhui Zhou, Chunfeng Yuan, Rong Gu, and Yihua Huang. Large scale nearest neighbors
683 search based on neighborhood graph. In *2013 International Conference on Advanced Cloud
684 and Big Data*, pages 181–186. IEEE, 2013. 13
- 685 [54] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom,
686 Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, et al. Query by image and video
687 content: The qbic system. *computer*, 28(9):23–32, 1995. 13
- 688 [55] Chun Jiang Zhu, Tan Zhu, Haining Li, Jinbo Bi, and Minghu Song. Accelerating large-scale
689 molecular similarity search through exploiting high performance computing. In *2019 IEEE
690 International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 330–333. IEEE,
691 2019. 13
- 692 [56] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on
693 information theory*, 13(1):21–27, 1967. 13
- 694 [57] Atsutake Kosuge and Takashi Oshima. An object-pose estimation acceleration technique for
695 picking robot applications by using graph-reusing k-nn search. In *2019 First International
696 Conference on Graph Computing (GC)*, pages 68–74. IEEE, 2019. 13
- 697 [58] Qiang Huang, Jianlin Feng, Qiong Fang, Wilfred Ng, and Wei Wang. Query-aware locality-
698 sensitive hashing scheme for l_p norm. *The VLDB Journal*, 26(5):683–708, 2017. 13
- 699 [59] Masajiro Iwasaki. Pruned bi-directed k-nearest neighbor graph for proximity search. In
700 *International Conference on Similarity Search and Applications*, pages 20–33. Springer, 2016.
701 13
- 702 [60] Yuan Cao, Heng Qi, Wenrui Zhou, Jien Kato, Keqiu Li, Xiulong Liu, and Jie Gui. Binary
703 hashing for approximate nearest neighbor search on big data: A survey. *IEEE Access*, 6:
704 2039–2054, 2017. 13
- 705 [61] Scott Cost and Steven Salzberg. A weighted nearest neighbor algorithm for learning with
706 symbolic features. *Machine learning*, 10(1):57–78, 1993. 13
- 707 [62] Yitong Meng, Xinyan Dai, Xiao Yan, James Cheng, Weiwen Liu, Jun Guo, Benben Liao,
708 and Guangyong Chen. Pmd: An optimal transportation-based user distance for recommender
709 systems. In *European Conference on Information Retrieval*, pages 272–280. Springer, 2020. 13
- 710 [63] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative
711 filtering recommendation algorithms. In *Proceedings of the 10th international conference on
712 World Wide Web*, pages 285–295, 2001. 13
- 713 [64] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via
714 over-parameterization. In *International Conference on Machine Learning*, pages 242–252.
715 PMLR, 2019. 17
- 716 [65] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. Learning polynomials with
717 neural networks. In *International conference on machine learning*, pages 1908–1916. PMLR,
718 2014. 17

719 A Learning a Dot Product decoder with a HadamardMLP decoder is Easy

720 Before we have discussed the limitations of the Dot Product decoder. An interesting questions is
721 whether the HadamardMLP decoder can replace the Dot Product decoder by approximating it. If the
722 MLP decoder can learn a dot product easily, it is safe to use MLP decoder instead of the dot product
723 ones in most cases. There are similar problems actively studied in machine learning. Existing work
724 imply that the difficulty scales polynomial with dimensionality d and $1/\epsilon$ in theory [10, 64, 65]. This
725 motivates us to investigate the question empirically.

726 We set up a synthetic learning task where given two embeddings $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ and a label $\mathbf{x}_i \bullet \mathbf{x}_j$, we
727 want to obtain a MLP function that approximates the $\mathbf{x}_i \bullet \mathbf{x}_j$ with the inputs $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$. For this
728 experiment, we create the datasets including the embedding matrix as $\mathbf{E} \in \mathbb{R}^{10^6 \times d}$. We draw every
729 row in \mathbf{E} from $\mathcal{N}(0, \mathbf{I})$ independently. Then, we uniformly sample (without replacement) 10^4 and S
730 embedding pair combinations from \mathbf{E} to form the test and training sets (no overlap) respectively.

731 We train the MLP on the training and evaluate it on the test set. For the architecture of the MLP, we
732 keep it simple: we follow the existing work [6, 8] to set the number of layers as 2 and the number of

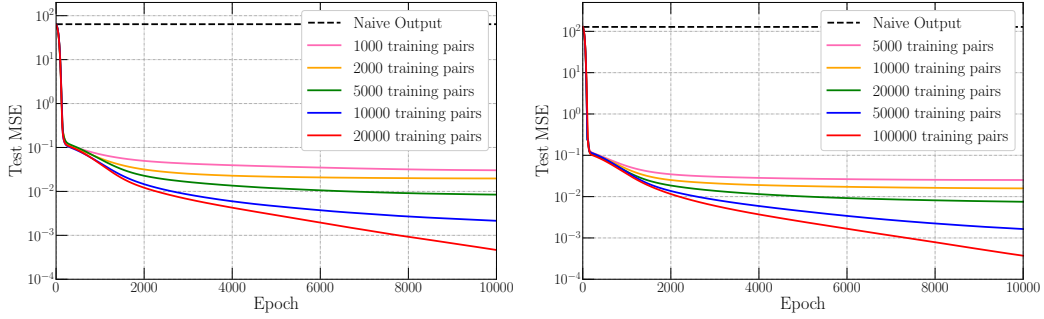


Figure 6: A MLP decoder can learn a Dot Product decoder well with enough training data. The left and right figures shows the MSE differences (y-axis) per epoch (x-axis) between the outputs of dot product and the MLP decoders given different training sizes with the input embedding dimensionality as $d = 64$ and $d = 128$ respectively. The naive output denotes the outputs of zeros.

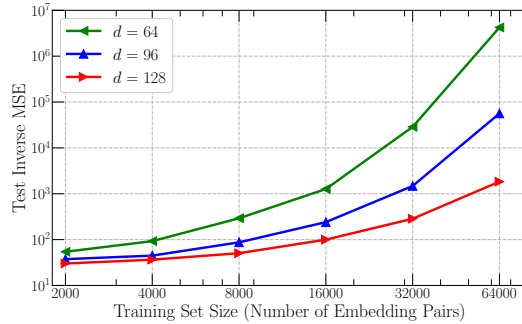


Figure 7: Test inverse MSE differences between the outputs of Dot Product and MLP decoders after convergence (y-axis) versus the training set size (x-axis).

733 hidden units as same as the input embeddings: d . For the optimizer, we also follow the existing work
 734 [6, 8] to choose the Adam optimizer.

735 As for evaluation metrics, we compute the MSE (Mean Squared Error) differences between the
 736 predicted score of the MLP and the dot product decoders. We measure the MSE of a naive model that
 737 predicts always 0 (the average rating). Every experiment is repeated 5 times and we report the mean.

738 Fig. 6 shows the approximation errors on the MLP per epoch given different number of training
 739 pairs and dimensions. The figure suggests that an MLP can easily approximate the dot product with
 740 enough training data. Consistent with the theory, the number of samples needed scales polynomially
 741 with the increasing dimensions and reduced errors. Anecdotaly, we observe the number of needed
 742 training samples is about $\mathcal{O}(d^\alpha/\epsilon^\beta)$ for $\alpha \approx 2, \beta \ll 1$ (see Fig. 7). In all cases, the MSE errors of
 743 the MLP decoder are negligible compared with the naive output.

744 This experiment shows that an MLP can easily approximate the dot product with enough training
 745 data. We hope this can explain, at least partially, why the MLP decoder generally performs better
 746 than the dot product.

747 Our conclusion seems to be distinct to to the existing work [10], which claims that the ConcatMLP
 748 is hard to learn a Dot Product. Actually, our conclusion is not conflicted with that in [10]. This
 749 ConcatMLP decoder processes the concatenation of the paired embeddings instead of the Hadamard
 750 product of the paired embeddings as the HadamardMLP. The HadamardMLP holds the inductive bias
 751 similar to the Dot Product, which makes the former easily learns the latter. Actually, we show that
 752 a simple two-layer MLP with only two hidden units is equivalent to the Dot Product with specific
 753 weights. We assign the first layer weights for two hidden units as $\mathbf{1}$ and $-\mathbf{1}$ and the second layer
 754 weights as ones. Then, we have its output as:

$$s_{ij} = \phi^{\text{MLP}}(\mathbf{x}_i, \mathbf{x}_j) = \text{ReLU}(\mathbf{1} \cdot (\mathbf{x}_i \odot \mathbf{x}_j)) + \text{ReLU}(-\mathbf{1} \cdot (\mathbf{x}_i \odot \mathbf{x}_j)) = \mathbf{1} \cdot (\mathbf{x}_i \odot \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j, \quad (14)$$

755 which is equivalent to the Dot Product decoder. From this result, we find that any MLP decoder with
756 the careful initialization is equivalent to the Dot Product decoder and thus can learn the Dot Product
757 easily.