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

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

2 Revisiting Link Prediction Decoders

In this section, we formalize the link prediction (LP) problem and the LP decoders. Typically, many LP models include an encoder that learns the node-level embeddings $x_i$, $i \in V$, where $V$ is the set of 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: $x_i, x_j$ into a single score: $s_{ij}$. If $s_{ij}$ is higher, the link between nodes $i$ and $j$ is more likely to exist. The state-of-the-art models generally use graph neural networks as the encoders [5, 6, 8, 12, 13]. From here on, we mainly focus on the decoder $\phi$.

2.1 Dot Product Decoder

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

$$s_{ij} = \phi^{\text{dot}}(x_i, x_j) := x_i \cdot x_j$$

where $\cdot$ denotes the dot product.

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

2.2 HadamardMLP (MLP following Hadamard Product) Decoder

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

$$f_W(x) = \text{ReLU}(Wx)$$

which is parameterized by the learnable weight $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ (the bias, if exists, can be represented by an additional column in $W$ and an additional channel in the input $x$ with the value as 1). ReLU is the activation function. In a MLP, several layers of $f$ are stacked, e.g., a 3-layer MLP can be formalized as $f_{W_3}(f_{W_2}(f_{W_1}(x)))$. 

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).
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.

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

$$s_{ij} = \phi_{\text{MLP}}(x_i, x_j) := \text{MLP}(x_i \odot x_j) = w^T_L (f_{W_{L-1}}(\ldots f_{W_1}(x_i \odot x_j) \ldots)),$$

(3)

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

2.3 Other Link Prediction Decoders

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

$$s_{ij} = h_i^T W h_j,$$

(4)

where $W$ is the learnable weight, and the MLPs following the concatenate decoder [6, 10] (short as ConcatMLP decoder):

$$s_{ij} = \text{MLP}(h_i \| h_j)$$

(5)

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

2.4 HadamardMLP is Generally More Effective than Other Decoders

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

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

Actually, our work is not the first to discuss the practical limitations of ConcatMLP decoders. For example, the existing work [10, 17, 18] has pointed out that In deep neural network designs, it is very
common to replace a ConcatMLP with a more specialized structure that has an inductive bias that represents the problem better, which is crucial for advancing the state of the art of deep learning, although in theory they can all be approximated by ConcatMLPs. The inefficiency of ConcatMLPs to capture dot and tensor products has been studied by [19] in the context of recommender systems. Similar to our analysis in Sec. 2 and Appendix A, [19] points out that it is hard for ConcatMLPs to approximate dot products and tensor products with ConcatMLPs empirically.

3 Scalability of Link Prediction Decoders

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

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

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

4 Flashlight: Scalable Link Prediction with Effective Decoders

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

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

The challenge of retrieving the neighbors of highest scores for HadamardMLP is rooted in the unawareness of which neurons are activated, since if we know which neurons are activated, the nonlinear HadamardMLP degrades to a linear model. On the \( i \)th MLP layer, we define the mask matrix \( M_{A,i} \in \mathbb{R}^{d_i \times d_i} \) to represent the set of activated neurons \( A \) as

\[
M_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } i \in A \\ 0, & \text{otherwise} \end{cases}
\]  

(6)

With \( M_{A,i} \), we reformulate the HadamardMLP decoder as:

\[
s_{ij} = \phi^{MLP}(x_i, x_j) = W_T^T M_{A,L} W_{L-1} \ldots M_{A,1} W_1 (x_i \odot x_j) = (W_T^T M_{A,1} \ldots W_{L-1}^T M_{A,L-1} W_L \odot x_i) \cdot x_j
\]

(7)

In the Equation (7), the final equation holds because the matrix computation on the left hand side of \( x_i \) is equal to a vector of the same dimension as \( x_i \). For the simplicity of expression, we denote it as left vector. In this way, transposing the matrix computation on the left hand side of \( x_i \) and multiply it
We name our algorithm as Flashlight because it works like a flashlight to progressively “illuminates” which neurons are activated. We use the initial iteration that assumes all the neurons are activated only reflect the general trend of the inner products, which correspond to the highest scores. Here, how to get the activated neurons \( A \) is an issue. Different node pairs activate different neurons \( A \). Initially, without knowing which neurons are activated, we first assume all the neurons are activated, i.e., we have the initial query embedding as:

\[
q[1] = \left( \prod_{i=1}^{L-1} W_i^T \right) w_L \odot x_i
\]  

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

\[
A = A(\text{MLP}(\cdot), x_i \odot x_j).
\]  

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

\[
A[t] \leftarrow A(\text{MLP}(\cdot), x_i \odot x_{j^*}[t]), \quad \text{where } j^*[t] = \arg \max_{j \in N[t]} \text{MLP}(x_i \odot x_j).
\]  

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

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

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

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

The initial iteration that assumes all the neurons are activated only reflect the general trend of the output increasing of HadamardMLP. Then, based on the actual neuron activation condition of the found high scoring neighbors, we can find other high scoring neighbors that activate the close neurons by adaptively adjusting the query embeddings with the new assumed activated neurons. \( A \) is a 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 $\phi^{\text{MLP}}$ that outputs the logit $s_{ij}$ for the input $x_i \odot x_j$. The set of nodes $\mathcal{V}$. The node embedding set $\mathcal{X} = \{x_i | i \in \mathcal{V}\}$. A source node $i$. The number of iterations $T$. The number of neighbors to retrieve at every iteration: $\mathcal{N} = [N_1, N_2, \ldots, N_T]$.

**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 $A[0]$ as all the neurons in MLP.
3: for $t \leftarrow 1$ to $T$ do
4: Calculate the query embedding $q[t] \leftarrow x_i \odot \text{MLP}_{A[t \leftarrow 1]}()$.
5: $\mathcal{N}[t] \leftarrow N_t$ neighbors in $\mathcal{X}$ that maximizes the inner product with $q[t]$.
6: $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x_j | j \in \mathcal{N}[t]\}$.
7: $j^*[t] = \arg\max_{j \in \mathcal{N}[t]} \text{MLP}(x_i \odot x_j)$
8: $A[t] \leftarrow A(\text{MLP}(), x_i \odot x_{j^*[t]})$.
9: $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}[t]$.
10: return $\mathcal{N}$

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

The convergence of our Flashlight is guaranteed since it iterates over a limited number of times to find the maximum inner product neighbors with different queries, as shown in line 3 of Algorithm 1. We agree that more theoretical analysis on Flashlight can further justify the effectiveness of our Flashlight. We take this as an important future direction for exploration.

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

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

**Complexity Analysis.** Using MLP decoders to compute the LP probabilities of all the neighbors holds the complexity as $O(N)$, where $N$ is the number of nodes in the whole graph. Finding the top scoring neighbors from the exact probabilities of all the neighbors also holds the linear complexity $O(N)$. Overall, using MLP decoders to find the top scoring neighbors is of the time complexity $O(N)$. In contrast, our Flashlight progressively calls the MIPS techniques for a constant number of times invariant to the graph data, which leads to the sublinear complexity as same as MIPS. In
Flashlight: Scalable Link Prediction with Effective Decoders

Table 1: Statistics of datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>OGBL-DDI</th>
<th>OGBL-COLLAB</th>
<th>OGBL-PPA</th>
<th>OGBL-CITATION2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Nodes</td>
<td>4,267</td>
<td>235,868</td>
<td>576,289</td>
<td>2,927,963</td>
</tr>
<tr>
<td>#Edges</td>
<td>1,334,889</td>
<td>1,285,465</td>
<td>30,326,273</td>
<td>30,561,187</td>
</tr>
</tbody>
</table>

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
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 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 link prediction models need to make predictions on. 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.

**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.

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Dot Product</th>
<th>Bilinear</th>
<th>ConcatMLP</th>
<th>HadamardMLP</th>
<th>HadamardMLP w/ Flashlight</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OGBL-DDI</strong></td>
<td>13.8 ± 1.8</td>
<td>16.1 ± 1.2</td>
<td>12.9 ± 1.4</td>
<td><strong>37.1 ± 5.1</strong></td>
<td><strong>37.1 ± 5.1</strong></td>
</tr>
<tr>
<td>GCN [5]</td>
<td>36.5 ± 2.6</td>
<td>39.4 ± 1.7</td>
<td>34.2 ± 1.9</td>
<td><strong>53.9 ± 4.7</strong></td>
<td><strong>53.9 ± 4.7</strong></td>
</tr>
<tr>
<td>GraphSAGE [12]</td>
<td>11.6 ± 1.9</td>
<td>13.8 ± 1.6</td>
<td>10.8 ± 1.7</td>
<td><strong>23.3 ± 2.1</strong></td>
<td><strong>23.3 ± 2.1</strong></td>
</tr>
<tr>
<td>Node2Vec [26]</td>
<td>27.7 ± 1.1</td>
<td>31.5 ± 1.0</td>
<td>27.2 ± 0.8</td>
<td><strong>48.9 ± 0.5</strong></td>
<td><strong>48.9 ± 0.5</strong></td>
</tr>
</tbody>
</table>

| **OGBL-COLLAB**  | 42.9 ± 0.7  | 43.2 ± 0.9 | 42.3 ± 1.0 | **44.8 ± 1.1** | **44.8 ± 1.1**            |
| GCN [5]          | 37.3 ± 0.9  | 41.5 ± 0.8 | 37.0 ± 0.7 | **48.1 ± 0.8** | **48.1 ± 0.8**            |
| GraphSAGE [12]   | 27.7 ± 1.1  | 31.5 ± 1.0 | 27.2 ± 0.8 | **48.9 ± 0.5** | **48.9 ± 0.5**            |
| Node2Vec [26]    | 5.1 ± 0.4   | 5.8 ± 0.5  | 6.2 ± 0.6  | **18.7 ± 1.3** | **18.7 ± 1.3**            |

| **OGBL-PPA**     | 3.2 ± 0.3   | 6.5 ± 0.7  | 5.8 ± 0.4  | **16.6 ± 2.4** | **16.6 ± 2.4**            |
| GCN [5]          | 4.2 ± 0.5   | 7.8 ± 0.6  | 8.3 ± 0.4  | **22.3 ± 0.8** | **22.3 ± 0.8**            |
| GraphSAGE [12]   | 65.3 ± 0.4  | 69.0 ± 0.8 | 62.7 ± 0.3 | **84.7 ± 0.2** | **84.7 ± 0.2**            |
| Node2Vec [26]    | 62.2 ± 0.7  | 65.4 ± 0.9 | 60.8 ± 0.6 | **80.4 ± 0.1** | **80.4 ± 0.1**            |

| **OGBL-CITATION2** | 52.7 ± 0.8  | 54.1 ± 0.6 | 51.4 ± 0.5 | **61.4 ± 0.1** | **61.4 ± 0.1**           |
| GCN [5]           | 62.2 ± 0.7  | 65.4 ± 0.9 | 60.8 ± 0.6 | **80.4 ± 0.1** | **80.4 ± 0.1**           |
| 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**           |
to adapt to different neighbors’ diverse information specialized by the complex graph topology. We follow the reviewers’ comments to discuss this phenomenon among the baseline methods to improve our paper.

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

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

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

5.4 The Flashlight Algorithm Effectively Finds the Top Scoring Neighbors

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

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

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

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

5.5 Inference Efficiency of Link Prediction with Our Flashlight Algorithm

We use the throughputs to evaluate the inference speed of neighbor retrieval of different methods. The throughput is defined as how many source nodes that a method can serve to retrieve the top 100 scoring neighbors per second. Except for the LP models that follow the encoder and decoder
architecture, e.g., GraphSAGE [12], GCN [5], and PLNLP [6], there are some subgraph based LP models, e.g., SUREL [7] and SEAL [27]. The common issue of the subgraph based models is the poor efficiency: they have to crop a separate subgraph for every node pair to calculate the LP probability on the node pair. In this sense, the node embeddings cannot be shared on the LP calculation for different node pairs. This leads to the much lower inference speed of the subgraph based LP models than the encoder-decoder LP models. We compare the inference efficiency of different methods on the OGBL-CITATION2 dataset in Fig. 4, where we present the inference speed of different methods when achieving the 100% Recall for the top 100 scoring neighbors.

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

Taking a further step, we comprehensively evaluate 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-PPI (right) datasets. We take the HadamardMLP decoder with the Random Sampling as the baseline for comparison. For example, on the OGBL-CITATION2 dataset, when achieving the 100% Recall for the top 100 scoring neighbors, the HadamardMLP with our Flashlight can serve more than 200 source nodes per second, while the HadamardMLP with the random sampling can only serve less than 1 node per second. Overall, our Flashlight achieves much better inference speed and effectiveness tradeoff than the HadamardMLP with random sampling.
We take the random sampling as a baseline method to reduce the search space for HadmardMLP because it is easy to understand and can act as a default choice when reducing the search space. We have taken a stronger baseline DotMax for comparison. We believe that the DotMax is a good baseline for comparison to distinguish the effectiveness on inference acceleration of our Flashlight.

5.6 Ablation Study

We analyze the sensitivity of Flashlight to the hyper-parameter: the number of iterations, and the 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

<table>
<thead>
<tr>
<th>Number of Iterations</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieving 100 neighbors per iteration</td>
<td>58.7%</td>
<td>91.5%</td>
<td>99.2%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Retrieving 200 neighbors per iteration</td>
<td>64.3%</td>
<td>94.2%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Retrieving 300 neighbors per iteration</td>
<td>66.2%</td>
<td>95.7%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Retrieving 400 neighbors per iteration</td>
<td>67.6%</td>
<td>98.9%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

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

5.7 Comparison with More Baselines and Datasets

We agree with the reviewers that comparing our Flashlight method with more baselines can better justify the effectiveness of Flashlight. We also agree that finding the neighbors that maximize the dot product between the target node and neighbors’ embeddings to reduce the search space for the HadmardMLP is a good baseline to compare with. For the simplicity of expression, we term this baseline as DotMax. We conducted the experiments on finding the top scoring neighbors for HadmardMLP with DotMax. We found that DotMax needs to retrieve much more neighbors than our Flashlight to achieve the same Recall as Flashlight on finding the top 100 scoring neighbors. For example, on the OGBL-CITATION2 dataset, which is the largest dataset among the used data holding nearly 3 million nodes as shown in Table 1, we follow the experimental settings as introduced 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

<table>
<thead>
<tr>
<th>Number of Neighbors Retrieved by DotMax</th>
<th>20000</th>
<th>40000</th>
<th>60000</th>
<th>80000</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall on Finding the Top 100 Scoring Neighbors</td>
<td>28.3%</td>
<td>42.5%</td>
<td>51.1%</td>
<td>63.4%</td>
<td>71.2%</td>
</tr>
</tbody>
</table>

the same experimental setting, when retrieving only 100 neighbors, our Flashlight can cover more than 80% of the top 100 scoring neighbors for HadmardMLP (as shown in Fig. 3). When retrieving more than 200 neighbors, Flashlight achieves the recall of 100%. Overall, our proposed Flashlight can reduce the search space for the HadmardMLP decoder much more effectively than the DotMax. The reason is that Dot Product demands the homophily of graph data to effectively infer the link between nodes. In contrast, thanks to the universal approximation capability, MLP can approximate any continuous function, and thus does not demand the homophily of graph data for effective LP. This constraint makes the Dot Product and HadamardMLP prefer different patterns of nodes’ semantic
embeddings on computing the link prediction scores. As a result, using DotMax to reduce the search space cannot effectively cover the top scoring neighbors for HadamardMLP with a small number of retrieved neighbors.

Overall, our Flashlight improves the link prediction inference speed of HadamardMLP much more significantly than DotMax. For example, under the experimental setting introduced in Sec. 6.5. The inference speed in terms of throughputs of HadamardMLP (nodes/second), of HadamardMLP with 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

<table>
<thead>
<tr>
<th>Method</th>
<th>HadamardMLP</th>
<th>HadamardMLP w/ DotMax</th>
<th>HadamardMLP w/ Flashlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference Throughputs (nodes/second)</td>
<td>0.62</td>
<td>6.13</td>
<td>108.45</td>
</tr>
</tbody>
</table>

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

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

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

Following the experimental setting in Sec. 6.5, we test the inference speed in terms of throughputs of HadamardMLP (nodes/second), of HadamardMLP with DotMax, and of HadamardMLP with Flashlight 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 a machine with

<table>
<thead>
<tr>
<th>Method</th>
<th>HadamardMLP</th>
<th>HadamardMLP w/ DotMax</th>
<th>HadamardMLP w/ Flashlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference Throughputs (nodes/second)</td>
<td>0.49</td>
<td>4.11</td>
<td>91.52</td>
</tr>
</tbody>
</table>

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

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

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

We found that DotMax needs to retrieve much more neighbors than our Flashlight to achieve the same Recall as Flashlight on finding the top 100 scoring neighbors. For example, on the OGBL-CITATION2 dataset, which is the largest dataset among the used data holding nearly 3 million nodes.
as shown in Table 1, we follow the experimental settings as introduced 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:

<table>
<thead>
<tr>
<th>Number of Neighbors Retrieved by Flashlight</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>250</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall on Finding the Top 100 Scoring Neighbors</td>
<td>42.8%</td>
<td>77.9%</td>
<td>91.6%</td>
<td>97.1%</td>
<td>99.2%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Number of Neighbors Retrieved by DotMax</th>
<th>20000</th>
<th>40000</th>
<th>60000</th>
<th>80000</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall on Finding the Top 100 Scoring Neighbors</td>
<td>22.1%</td>
<td>39.3%</td>
<td>46.4%</td>
<td>55.2%</td>
<td>60.1%</td>
</tr>
</tbody>
</table>

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

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

6 Related Work

6.1 Link Prediction Models

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

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

6.2 Maximum Inner Product Search

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

With the explosive growth of datasets’ scale and the inevitable curse of dimensionality, MIPS is essential to offer the scalable services. However, the HadamardMLP decoders are nonlinear and there do not exist the well studied sublinear complexity algorithms to find the top scoring neighbors for
HadamardMLP [10]. In this work, we utilize the well studied approximate MIPS techniques with the adaptively adjusted query embeddings to find the top scoring neighbors for the MLP decoders in a progressive manner. Our method supports the plug-and-play use during inference and significantly accelerates the LP inference with the effective MLP decoders.

7 Conclusion

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

References


[38] Qiang Huang, Guihong Ma, Jianlin Feng, Qiong Fang, and Anthony KH Tung. Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1561–1570, 2018.


We train the MLP on the training set and evaluate it on the test set. For the architecture of the MLP, we set up a synthetic learning task where given two embeddings \(x_i, x_j \in \mathbb{R}^d\) and a label \(y\), we want to obtain a MLP function that approximates the \(y = \langle x_i, x_j \rangle\) with the inputs \(x_i, x_j \in \mathbb{R}^d\). For this experiment, we create the datasets including the embedding matrix as \(E \in \mathbb{R}^{10^4 \times d}\). We draw every row in \(E\) from \(N(0, I)\) independently. Then, we uniformly sample (without replacement) \(10^4\) and \(S\) embedding pair combinations from \(E\) to form the test and training sets (no overlap) respectively.

We train the MLP on the training and evaluate it on the test set. For the architecture of the MLP, we keep it simple: we follow the existing work \([6, 8]\) to set the number of layers as 2 and the number of parameters...
hidden units as same as the input embeddings: \( d \). For the optimizer, we also follow the existing work [6, 8] to choose the Adam optimizer.

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

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

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

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

\[
s_{ij} = \phi_{\text{MLP}}(\mathbf{x}_i, \mathbf{x}_j) = \text{ReLU}(1 \cdot (\mathbf{x}_i \odot \mathbf{x}_j) + \text{ReLU}(-1 \cdot (\mathbf{x}_i \odot \mathbf{x}_j)) = 1 \cdot (\mathbf{x}_i \odot \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j, \quad (14)
\]
which is equivalent to the Dot Product decoder. From this result, we find that any MLP decoder with the careful initialization is equivalent to the Dot Product decoder and thus can learn the Dot Product easily.