WARPSPEED COMPUTATION OF OPTIMAL TRANSPORT, GRAPH DISTANCES, AND EMBEDDING ALIGNMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimal transport (OT) is a cornerstone of many machine learning tasks. The current best practice for computing OT is via entropy regularization and Sinkhorn iterations. This algorithm runs in quadratic time and requires calculating the full pairwise similarity matrix, which is prohibitively expensive for large sets of objects. To alleviate this limitation we propose to use a sparse approximation of the cost matrix based on locality sensitive hashing (LSH). Moreover, we fuse this sparse approximation with the Nyström method, resulting in the locally corrected Nyström method (LCN). These approximations enable the first log-linear time algorithms for entropy-regularized OT that work for complex real-world tasks using high-dimensional spaces with little to no loss in accuracy. We thoroughly demonstrate this by evaluating different Sinkhorn approximations both directly and as a component of two real-world models. Using approximate Sinkhorn for unsupervised word embedding alignment enables us to train the model full-batch in a fraction of the time while improving upon the original on average by 3.1 percentage points without any model changes. For graph distance regression we propose the graph transport network (GTN), which combines graph neural networks (GNNs) with enhanced Sinkhorn and outcompetes previous models by $52\,\%$. LCN-Sinkhorn enables GTN to achieve this while still scaling log-linearly in the number of nodes.

1 INTRODUCTION

Similarity functions such as optimal transport (OT) are central building blocks of modern machine learning (ML). They can be found in generative modelling (Arjovsky et al., 2017; Bousquet et al., 2017; Genevay et al., 2018), loss functions (Frogner et al., 2015), set matching (Wang et al., 2019), domain adaptation (Courty et al., 2017), kernels (Shawe-Taylor & Cristianini, 2004), attention mechanisms (Vaswani et al., 2017), semi-supervised learning (Zhu & Goldberg, 2009), or memory-augmented networks (Santoro et al., 2016). Motivated by this, many different methods for accelerating OT have been proposed in recent years (Indyk & Thaper, 2003; Papadakis et al., 2014; Backurs et al., 2020). However, most of these approaches are specialized methods that do not generalize to modern ML models, which rely on dynamically changing high-dimensional embeddings.

In this work we aim to fill this gap and make modern models significantly more scalable by proposing two fast and accurate approximations of entropy-regularized optimal transport – sparse Sinkhorn and LCN-Sinkhorn – along with the locally corrected Nyström (LCN) method. LCN is a general approximation approach for similarity matrices that fuses local (sparse) and global (low-rank) approximations, allowing us to simultaneously capture both kinds of behavior. Sparse Sinkhorn uses a sparse cost matrix to leverage the fact that in entropy-regularized OT (also known as the Sinkhorn distance) (Cuturi, 2013) often only each point's nearest neighbors influence the result. LCN-Sinkhorn extends this approach to less sparse use cases by leveraging LCN, thereby fusing sparse Sinkhorn and Nyström-Sinkhorn (Altschuler et al., 2019). Both sparse Sinkhorn and LCN-Sinkhorn run in time $\mathcal{O}(n \log n)$. Additionally, we derive analytic gradients for sparse Sinkhorn and LCN-Sinkhorn and improve the expressiveness of OT via multi-head optimal transport and learnable unbalanced OT.

We validate these approximations by showing that they are able to reproduce both the Sinkhorn distance and transport plan significantly better than previous methods across a wide range of regularization parameters and computational budgets (as e.g. demonstrated in Fig. 1). We then show the

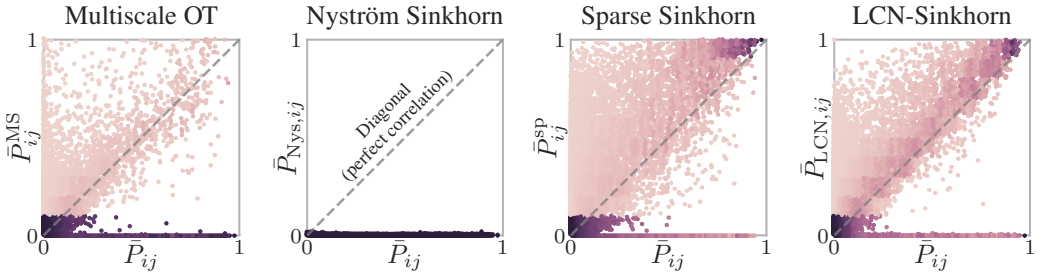

Figure 1: The proposed methods (sparse and LCN-Sinkhorn) show a clear correlation with the full Sinkhorn transport plan, as opposed to previous methods. Entries of approximations (y-axis) and full Sinkhorn (x-axis) for pre-aligned word embeddings (EN-DE). Color denotes sample density.

impact of these improvements by employing Sinkhorn approximations end-to-end in two high-impact machine learning tasks. First, we incorporate them into Wasserstein Procrustes for word embedding alignment (Grave et al., 2019). LCN-Sinkhorn improves upon the original method's accuracy by 3.1 percentage points using a third of the training time without *any* further model changes. Second, we develop the graph transport network (GTN), which combines graph neural networks (GNNs) with multi-head unbalanced OT, and evaluate it on graph distance regression. GTN with LCN-Sinkhorn is the first model that both overcomes the bottleneck of using a single embedding per graph and scales log-linearly in the number of nodes. In summary, our paper's main contributions are:

- Locally Corrected Nyström (LCN), a flexible, log-linear time approximation for pairwise similarity matrices, leveraging both local (sparse) and global (low-rank) approximations.
- Optimal transport with $\mathcal{O}(n \log n)$ runtime via sparse Sinkhorn and LCN-Sinkhorn, the first such approaches stable enough to substitute full entropy-regularized OT (a.k.a. Sinkhorn distance) in ML models that leverage high-dimensional spaces.
- Improvements to optimal transport for machine learning: Multi-head OT and learnable unbalanced OT via the bipartite matching matrix.
- The graph transport network (GTN), which combines a graph neural network (GNN) with multi-head unbalanced LCN-Sinkhorn. GTN both sets the state of the art on graph distance regression and still scales log-linearly in the number of nodes.

2 Related Work

Log-linear optimal transport. For an overview of optimal transport and its foundations see Peyré & Cuturi (2019). On grids and 2D surfaces OT can be solved using dynamical OT (Papadakis et al., 2014; Solomon et al., 2014), convolutions (Solomon et al., 2015), or embedding/hashing schemes (Indyk & Thaper, 2003; Andoni et al., 2008). In higher dimensions we can use tree-based algorithms (Backurs et al., 2020) or hashing schemes (Charikar, 2002), which are however limited to a previously fixed set of embeddings, on which only the mass distribution changes. For sets that change dynamically (e.g. during training) the currently most common method of achieving log-linear OT is a multiscale approach of entropy-regularized optimal transport (Schmitzer, 2019; Gerber & Maggioni, 2017). Tenetov et al. (2018) recently proposed using a low-rank approximation of the Sinkhorn similarity matrix obtained via a semidiscrete approximation of the Euclidean distance. Altschuler et al. (2019) improved upon this approach by using the Nyström method for the approximation. These approaches still struggle with high-dimensional real-world problems, as we will show in Sec. 7.

Sliced Wasserstein distance. Another approach to reduce the computational complexity of optimal transport (without entropy regularization) are sliced Wasserstein distances (Rabin et al., 2011). However, they require the L_2 distance as a cost function and are either unstable in convergence or prohibitively expensive for high-dimensional problems ($\mathcal{O}(nd^3)$) (Meng et al., 2019).

Fast Sinkhorn. Another line of work has been pursuing accelerating entropy-regularized OT without changing its computational complexity w.r.t. the number of discrete points. Original Sinkhorn requires $\mathcal{O}(1/\varepsilon^3)$ iterations, Greenkhorn reduced these to $\mathcal{O}(1/\varepsilon^2)$ (Altschuler et al., 2017) and Jambulapati et al. (2019) recently proposed an algorithm that reduces them to $\mathcal{O}(1/\varepsilon)$. Alaya et al.

(2019) proposed to reduce the size of the Sinkhorn problem by screening out neglectable components, which allows for approximation guarantees. Genevay et al. (2016) proposed using a stochastic optimization scheme instead of Sinkhorn iterations. Blondel et al. (2018) proposed alternative regularizations to obtain an OT problem that has comparable runtime to the Sinkhorn algorithm, while providing a sparse solution. This work is largely orthogonal to ours.

Embedding alignment. For an overview of cross-lingual word embedding models see Ruder et al. (2019). Unsupervised word embedding alignment was proposed by Conneau et al. (2018), with subsequent advances by Alvarez-Melis & Jaakkola (2018); Grave et al. (2019); Joulin et al. (2018).

Graph matching and distance learning. Most recent approaches for graph matching and graph distance learning either rely on a single fixed-dimensional graph embedding (Bai et al., 2019; Li et al., 2019), or only use attention or some other strongly simplified variant of optimal transport (Bai et al., 2019; Riba et al., 2018; Li et al., 2019). Others break permutation invariance and are thus ill-suited for this task (Ktena et al., 2017; Bai et al., 2018). So far only approaches using a single graph embedding allow faster than quadratic scaling in the number of nodes. Compared to the Sinkhorn-based image model concurrently proposed by Wang et al. (2019) GTN uses no CNN or cross-graph attention, but an enhanced GNN and embedding aggregation scheme. OT has recently been proposed for graph kernels (Maretic et al., 2019; Vayer et al., 2019), which can (to a limited extent) be used for graph matching, but not for distance learning.

3 Sparse Entropy-Regularized Optimal Transport

Entropy-regularized optimal transport. Optimal transport is concerned with the problem of finding the transport plan between two probability distributions with the lowest cost. For continuous distributions it is mostly known as the Wasserstein distance, while in the discrete case the term earth mover's distance (EMD) is more common. In this work we will focus on the latter and furthermore add entropy regularization, which enables fast computation and often even performs better than regular OT (Cuturi, 2013). Formally, given two categorical distributions modelled via the vectors $\boldsymbol{p} \in \mathbb{R}^n$ and $\boldsymbol{q} \in \mathbb{R}^m$ supported on two sets of points $X_\text{p} = \{\boldsymbol{x}_{\text{p}1}, \dots, \boldsymbol{x}_{\text{p}n}\}$ and $X_\text{q} = \{\boldsymbol{x}_{\text{q}1}, \dots, \boldsymbol{x}_{\text{q}m}\}$ in $\mathbb{R}^d$ and the cost function $c : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ giving rise to the cost matrix $\boldsymbol{C} \in \mathbb{R}^{n \times m}$ we aim to find the distance d_c^λ and the associated optimal transport plan $\bar{\boldsymbol{P}}$ (Cuturi, 2013)

$$d_c^\lambda = \min_{\boldsymbol{P}} \langle \boldsymbol{P}, \boldsymbol{C} \rangle_\text{F} - \lambda H(\boldsymbol{P}), \qquad \text{s.t. } \boldsymbol{P}\mathbf{1}_m = \boldsymbol{p},\ \boldsymbol{P}^T\mathbf{1}_n = \boldsymbol{q}, \tag{1}$$

with the entropy $H(\boldsymbol{P}) = -\sum_{i=1}^n \sum_{j=1}^m \boldsymbol{P}_{ij} \log \boldsymbol{P}_{ij}$. Note that d_c^λ includes the entropy and can thus be negative, while Cuturi (2013) originally used $d^{1/\lambda}_{\text{Cuturi},c} = \left\langle \bar{\boldsymbol{P}}, \boldsymbol{C} \right\rangle_\text{F}$ (Cuturi, 2013). The optimization problem can be solved by normalizing row i and column j of the matrix $\boldsymbol{K}$ with $\boldsymbol{K}_{ij} = e^{-\frac{\boldsymbol{C}_{ij}}{\lambda}}$ to $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$. The normalization is usually computed via the Sinkhorn algorithm, which alternatingly computes the normalization vectors

$$\boldsymbol{s} = \boldsymbol{p} \oslash (\boldsymbol{K}\boldsymbol{t}), \qquad \boldsymbol{t} = \boldsymbol{q} \oslash (\boldsymbol{K}^T\boldsymbol{s}) \tag{2}$$

until convergence, where $\oslash$ denotes elementwise division. The optimal transport plan is then obtained via $\bar{\boldsymbol{P}} = \operatorname{diag}(\bar{\boldsymbol{s}})\boldsymbol{K}\operatorname{diag}(\bar{\boldsymbol{t}})$, where $\bar{\boldsymbol{s}}$ and $\bar{\boldsymbol{t}}$ are the optimal normalization vectors. Computations are usually performed in log-space via the log-sum-exp trick to improve stability.

Sparse Sinkhorn. The Sinkhorn algorithm is very fast compared to non-regularized EMD algorithms, which run in $\mathcal{O}(n^2 m \log n \log(n \max(\boldsymbol{C})))$ (Tarjan, 1997). However, its computational cost is still quadratic in time, i.e. $\mathcal{O}(nm)$, which is prohibitively expensive for large n and m. We propose to overcome this by observing that the matrix $\boldsymbol{K}$, and hence also $\bar{\boldsymbol{P}}$, is negligibly small everywhere except at each point's closest neighbors because of the exponential used in $\boldsymbol{K}$'s computation. We propose to leverage this by approximating $\boldsymbol{C}$, $\boldsymbol{K}$, and $\bar{\boldsymbol{P}}$ via the sparse matrices $\boldsymbol{C}^\text{sp}$, $\boldsymbol{K}^\text{sp}$, and $\bar{\boldsymbol{P}}^\text{sp}$. The resulting sparse Sinkhorn method scales log-linearly with the number of points, i.e. $\mathcal{O}(n \log n)$ for $n \approx m$ (see App. A and App. G for details). To construct $\boldsymbol{C}^\text{sp}$ we need to find each point's near neighbors, a task that can efficiently be solved via locality sensitive hashing (LSH) on $X_\text{p} \cup X_\text{q}$.

Locality sensitive hashing. LSH tries to filter data points by separating "near" from "far" points by putting them into different hash buckets. Points closer than a certain distance r_1 are put into the same bucket with probability at least p_1, while those beyond some distance $r_2 = c \cdot r_1$ with $c > 1$

are put into the same bucket with probability at most $p_2 \ll p_1$. There is a plethora of LSH methods for different cost functions (Wang et al., 2014; Shrivastava & Li, 2014), so we do not have to restrict our approach to a limited set of functions. In this work we focus on cross-polytope LSH (Andoni et al., 2015) and k-means LSH (Paulevé et al., 2010), depending on the cost function (see App. D). Unfortunately, LSH can fail when e.g. the cost between pairs is very similar (see App. B). However, we can alleviate such limitations by fusing the sparse approximation with the Nyström method.

4 Locally Corrected Nyström

Nyström method. The Nyström method is a popular way of approximating kernel matrices that provides performance guarantees for many important tasks (Williams & Seeger, 2001; Musco & Musco, 2017). It approximates a positive semi-definite (PSD) kernel matrix $\boldsymbol{K}$ via its low-rank decomposition $\boldsymbol{K}_{\text{Nys}} = \boldsymbol{U}\boldsymbol{A}^{-1}\boldsymbol{V}$. Since the optimal decomposition via SVD is too expensive to compute, Nyström instead chooses a set of l landmarks $L = \{\boldsymbol{x}_{\text{l}1}, \dots, \boldsymbol{x}_{\text{l}l}\}$ and obtains the matrices via $\boldsymbol{U}_{ij} = k(\boldsymbol{x}_{\text{p}i}, \boldsymbol{x}_{\text{l}j})$, $\boldsymbol{A}_{ij} = k(\boldsymbol{x}_{\text{l}i}, \boldsymbol{x}_{\text{l}j})$, and $\boldsymbol{V}_{ij} = k(\boldsymbol{x}_{\text{l}i}, \boldsymbol{x}_{\text{q}j})$, where $k(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is an arbitrary PSD kernel, e.g. $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = e^{-\frac{c(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\lambda}}$ for Sinkhorn. Common methods of choosing landmarks from $X_{\text{p}} \cup X_{\text{q}}$ are uniform and ridge leverage score (RLS) sampling. We instead focus on k-means Nyström and sampling via k-means++, which we found to be significantly faster than recursive RLS sampling (Zhang et al., 2008) and perform better than both uniform and RLS sampling (see App. D).

Sparse vs. Nyström. Since the Nyström method is a low-rank approximation it only accounts for the global structure of the kernel matrix $\boldsymbol{K}$ and not for the local structure around each point $\boldsymbol{x}$. As such, it is ill-suited for any moderately low entropy regularization parameter, where the transport matrix $\bar{\boldsymbol{P}}$ will mostly resemble a permuted identity matrix. Indeed, our experiments show that the Nyström approximation struggles in real-world OT tasks (see Sec. 7). Sparse Sinkhorn, on the other hand, cannot account for global structure and instead approximates all non-selected pairs as infinity. It will hence fail for any case that requires more than a handful of neighbors per point. These approximations are thus opposites of each other, and as such not competing but rather *complementary* approaches.

Locally corrected Nyström. Since we know that the entries in our sparse approximation are exact, fusing this matrix with the Nyström method is rather straightforward. For all non-zero values in the sparse approximation $\boldsymbol{K}^{\text{sp}}$ we first calculate the corresponding Nyström approximations, obtaining the sparse matrix $\boldsymbol{K}^{\text{sp}}_{\text{Nys}}$. To obtain the locally corrected Nyström (LCN) approximation we remove these entries from $\boldsymbol{K}_{\text{Nys}}$ and replace them with their exact values, i.e.

$$\boldsymbol{K}_{\text{LCN}} = \boldsymbol{K}_{\text{Nys}} + \boldsymbol{K}^{\text{sp}}_{\Delta} = \boldsymbol{K}_{\text{Nys}} - \boldsymbol{K}^{\text{sp}}_{\text{Nys}} + \boldsymbol{K}^{\text{sp}}. \tag{3}$$

LCN-Sinkhorn. For entropy-regularized optimal transport with LCN we never fully instantiate $\boldsymbol{K}_{\text{LCN}}$. Instead, we only save the decomposition and directly use these parts for the matrix-vector product in Eq. (2) via $\boldsymbol{K}_{\text{LCN}}\boldsymbol{t} = \boldsymbol{U}(\boldsymbol{A}^{-1}\boldsymbol{V}\boldsymbol{t}) + \boldsymbol{K}^{\text{sp}}_{\Delta}\boldsymbol{t}$, similarly to Altschuler et al. (2019). We furthermore precompute $\boldsymbol{W} = \boldsymbol{A}^{-1}\boldsymbol{V}$ to avoid unnecessary operations. As result, we obtain the decomposition of $\bar{\boldsymbol{P}}_{\text{LCN}} = \bar{\boldsymbol{P}}_{\text{Nys}} + \bar{\boldsymbol{P}}^{\text{sp}}_{\Delta} = \bar{\boldsymbol{P}}_U \bar{\boldsymbol{P}}_W + \bar{\boldsymbol{P}}^{\text{sp}} - \bar{\boldsymbol{P}}^{\text{sp}}_{\text{Nys}}$ and the approximate distance (using Lemma A from Altschuler et al. (2019))

$$d^{\lambda}_{\text{LCN},c} = \lambda \left(\boldsymbol{s}^T \bar{\boldsymbol{P}}_U \bar{\boldsymbol{P}}_W \boldsymbol{1}_m + \boldsymbol{1}_n^T \bar{\boldsymbol{P}}_U \bar{\boldsymbol{P}}_W \boldsymbol{t} + \boldsymbol{s}^T \bar{\boldsymbol{P}}^{\text{sp}}_{\Delta} \boldsymbol{1}_m + \boldsymbol{1}_n^T \bar{\boldsymbol{P}}^{\text{sp}}_{\Delta} \boldsymbol{t} \right). \tag{4}$$

This approximation scales log-linearly with the dataset size (see App. A and App. G for details). LCN allows smoothly moving from Nyström-Sinkhorn to sparse Sinkhorn by varying the number of neighbors e and landmarks l. This allows us to freely choose the optimal "operating point" depending on the underlying problem and regularization parameter. Note that $d^{\lambda}_{\text{LCN},c}$ and $\bar{\boldsymbol{P}}_{\text{LCN}}$ are symmetric and permutation invariant (and equivariant, respectively) w.r.t. the sets X_{p} and X_{q}. We discuss the limitations of LCN-Sinkhorn in App. B.

Backpropagation. Most modern ML models are trained using backpropagation, making efficient gradient computation almost as important as the algorithm itself. Usually we want to learn embeddings which act as point sets X_{p} and X_{q} and therefore need gradients w.r.t. the cost matrix $\boldsymbol{C}$. We can either estimate these gradients via automatic differentiation of the unrolled Sinkhorn iterations or via an analytic equation obtained by assuming exactness of the solution. Depending on the problem at hand, either the automatic or the analytic estimator will lead to faster overall convergence (Ablin et al., 2020). LCN-Sinkhorn works flawlessly with automatic backpropagation since it only relies on basic linear algebra (except for choosing Nyström landmarks and LSH neighbors, for which we use

a simple straight-through estimator (Bengio et al., 2013)). To enable analytic backpropagation we provide analytic gradients in Proposition 1, derived via Danskin's theorem and the implicit function theorem (see App. C). Note that both methods allow efficient backpropagation with linear runtime.

Proposition 1. *The derivatives of the distances d_c^λ and $d_{LCN,c}^\lambda$ (Eqs.* (1) *and* (4)*) and the optimal transport plan $\bar{\boldsymbol{P}}$ w.r.t. the (decomposed) cost matrix $\boldsymbol{C}$ in entropy-regularized OT and LCN-Sinkhorn are*

$$\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}} = \bar{\boldsymbol{P}}, \qquad \frac{\partial \bar{\boldsymbol{P}}_{ij}}{\partial \boldsymbol{C}_{kl}} = -\frac{1}{\lambda}\bar{\boldsymbol{P}}_{ij}\delta_{ik}\delta_{jl}, \tag{5}$$

$$\frac{\partial d_{\text{LCN},c}^\lambda}{\partial \boldsymbol{U}} = -\lambda \bar{\boldsymbol{s}}(\boldsymbol{W}\bar{\boldsymbol{t}})^T, \quad \frac{\partial d_{\text{LCN},c}^\lambda}{\partial \boldsymbol{W}} = -\lambda(\bar{\boldsymbol{s}}^T\boldsymbol{U})^T\bar{\boldsymbol{t}}^T, \quad \frac{\partial d_{\text{LCN},c}^\lambda}{\partial \log \boldsymbol{K}^{\text{sp}}} = -\lambda \bar{\boldsymbol{P}}^{\text{sp}}, \quad \frac{\partial d_{\text{LCN},c}^\lambda}{\partial \log \boldsymbol{K}_{\text{Nys}}^{\text{sp}}} = -\lambda \bar{\boldsymbol{P}}_{\text{Nys}}^{\text{sp}}, \tag{6}$$

$$\begin{gathered}\frac{\partial \bar{\boldsymbol{P}}_{U,ij}}{\partial \boldsymbol{U}_{kl}} = \delta_{ik}\delta_{jl}\boldsymbol{s}_i, \qquad \frac{\partial \bar{\boldsymbol{P}}_{W,ij}}{\partial \boldsymbol{U}_{kl}} = \bar{\boldsymbol{P}}_{U,ik}^\dagger \boldsymbol{s}_k \bar{\boldsymbol{P}}_{W,lj}, \qquad \frac{\partial \bar{\boldsymbol{P}}_{U,ij}}{\partial \boldsymbol{W}_{kl}} = \bar{\boldsymbol{P}}_{U,ik}\boldsymbol{t}_l \bar{\boldsymbol{P}}_{W,lj}^\dagger, \\ \frac{\partial \bar{\boldsymbol{P}}_{W,ij}}{\partial \boldsymbol{W}_{kl}} = \delta_{ik}\delta_{jl}\boldsymbol{t}_j, \qquad \frac{\partial \bar{\boldsymbol{P}}_{ij}^{\text{sp}}}{\partial \log \boldsymbol{K}_{kl}^{\text{sp}}} = \bar{\boldsymbol{P}}_{ij}^{\text{sp}}\delta_{ik}\delta_{jl}, \qquad \frac{\partial \bar{\boldsymbol{P}}_{\text{Nys},ij}^{\text{sp}}}{\partial \log \boldsymbol{K}_{\text{Nys},kl}^{\text{sp}}} = \bar{\boldsymbol{P}}_{\text{Nys},ij}^{\text{sp}}\delta_{ik}\delta_{jl},\end{gathered} \tag{7}$$

with δ_{ij} denoting the Kronecker delta and † the Moore-Penrose pseudoinverse. Using these decompositions we can backpropagate through LCN-Sinkhorn in time $\mathcal{O}(Nl^2 + l^3)$.

5 Enhanced Optimal Transport

Multi-head OT. The Sinkhorn algorithm can be viewed as alternatingly applying row-wise and column-wise softmax with temperature $T = \lambda$ to the cost matrix $\boldsymbol{C}$. When we simplify this process by only applying row-wise softmax we arrive at the attention mechanism used in many modern models (Vaswani et al., 2017). Considering this close connection between entropy-regularized OT and attention we can expect that using OT with multiple heads leads to significant performance improvements, as it does in language models (Devlin et al., 2019). Using K heads means that we calculate OT in parallel for K separate sets of embeddings representing the same pair of objects and obtain a set of distances $\boldsymbol{d}_c^\lambda \in \mathbb{R}^K$. We can then transform these distances to a final distance prediction using e.g. a multilayer perceptron (MLP).

Learnable unbalanced OT. Unbalanced OT allows handling cases where $\|\boldsymbol{p}\|_1 \neq \|\boldsymbol{q}\|_1$ or when we want to soften the requirements $\boldsymbol{P}\boldsymbol{1}_m = \boldsymbol{p}$ and $\boldsymbol{P}^T\boldsymbol{1}_n = \boldsymbol{q}$ (Peyré & Cuturi, 2019). Previous methods did so by swapping these requirements with a uniform divergence loss term on $\boldsymbol{p}$ and $\boldsymbol{q}$ (Frogner et al., 2015; Chizat et al., 2018). These approaches *uniformly* penalize deviations from balanced OT and therefore cannot adapt to only ignore parts of the distribution. We propose to alleviate this limitation by swapping the cost matrix $\boldsymbol{C}$ with the bipartite matching (BP) matrix (Riesen & Bunke, 2009)

$$\boldsymbol{C}_{\text{BP}} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{C}^{(\text{p},\varepsilon)} \\ \boldsymbol{C}^{(\varepsilon,\text{q})} & \boldsymbol{C}^{(\varepsilon,\varepsilon)} \end{bmatrix}, \quad \boldsymbol{C}_{ij}^{(\text{p},\varepsilon)} = \begin{cases} c_{i,\varepsilon} & i = j \\ \infty & i \neq j \end{cases}, \quad \boldsymbol{C}_{ij}^{(\varepsilon,\text{q})} = \begin{cases} c_{\varepsilon,j} & i = j \\ \infty & i \neq j \end{cases}, \quad \boldsymbol{C}_{ij}^{(\varepsilon,\varepsilon)} = 0, \tag{8}$$

and adaptively computing the costs $c_{i,\varepsilon}$, $c_{\varepsilon,j}$ and $c_{\varepsilon,\varepsilon}$ based on the input sets X_p and X_q. Deleting mass is especially useful in the common case where not every point in X_p or X_q has a corresponding point in the opposite set. Using the BP matrix adds minor computational overhead since we only need to save the diagonals $\boldsymbol{c}_{\text{p},\varepsilon}$ and $\boldsymbol{c}_{\varepsilon,\text{q}}$ of $\boldsymbol{C}_{\text{p},\varepsilon}$ and $\boldsymbol{C}_{\varepsilon,\text{q}}$. We can then include the additional parts of $\boldsymbol{C}_{\text{BP}}$ in the Sinkhorn algorithm (Eq. (2)) via

$$\boldsymbol{K}_{\text{BP}}\boldsymbol{t} = \begin{bmatrix} \boldsymbol{K}\hat{\boldsymbol{t}} + \boldsymbol{c}_{\text{p},\varepsilon} \odot \check{\boldsymbol{t}} \\ \boldsymbol{c}_{\varepsilon,\text{q}} \odot \hat{\boldsymbol{t}} + \boldsymbol{1}_n^T\check{\boldsymbol{t}} \end{bmatrix}, \qquad \boldsymbol{K}_{\text{BP}}^T\boldsymbol{s} = \begin{bmatrix} \boldsymbol{K}^T\hat{\boldsymbol{s}} + \boldsymbol{c}_{\varepsilon,\text{q}} \odot \check{\boldsymbol{s}} \\ \boldsymbol{c}_{\text{p},\varepsilon} \odot \hat{\boldsymbol{s}} + \boldsymbol{1}_m^T\check{\boldsymbol{s}} \end{bmatrix}, \tag{9}$$

where $\hat{\boldsymbol{t}}$ denotes the upper and $\check{\boldsymbol{t}}$ the lower part of the vector $\boldsymbol{t}$. To calculate d_c^λ we can decompose the transport plan $\boldsymbol{P}_{\text{BP}}$ in the same way as $\boldsymbol{C}_{\text{BP}}$, with a single scalar for $\boldsymbol{P}_{\varepsilon,\varepsilon}$. In our implementation we obtain the deletion cost via $c_{i,\varepsilon} = \|\boldsymbol{\alpha} \odot \boldsymbol{x}_{\text{p}i}\|_2$, with a learnable vector $\boldsymbol{\alpha} \in \mathbb{R}^d$.

Table 1: Mean and standard deviation (in parentheses) of relative Sinkhorn distance error, IoU of top $0.1\,\%$ and correlation coefficient (PCC) of OT plan entries across 5 runs. Sparse Sinkhorn and LCN-Sinkhorn consistently achieve the best approximation in all 3 measures.

	EN-DE			EN-ES			EN-FR			EN-RU		
	Rel. err. d_c^λ	PCC	IoU	Rel. err. d_c^λ	PCC	IoU	Rel. err. d_c^λ	PCC	IoU	Rel. err. d_c^λ	PCC	IoU
Multiscale OT	0.634(11)	0.308(14)	0.123(5)	0.645(14)	0.321(6)	0.125(12)	0.660(17)	0.330(9)	0.121(7)	0.667(16)	0.281(19)	0.125(9)
Nyström Skh.	1.183(5)	0.077(1)	0.045(5)	1.175(18)	0.068(1)	0.048(6)	1.172(13)	0.070(3)	0.052(4)	1.228(18)	0.091(2)	0.047(6)
Sparse Skh.	**0.233(2)**	0.552(4)	0.102(1)	**0.217(1)**	0.623(4)	0.102(1)	**0.220(1)**	0.608(5)	0.104(2)	**0.272(2)**	0.446(8)	0.090(1)
LCN-Sinkhorn	0.406(15)	**0.673(12)**	**0.197(7)**	0.368(12)	**0.736(3)**	**0.201(3)**	0.342(5)	**0.725(4)**	**0.209(3)**	0.465(10)	**0.623(5)**	**0.210(4)**

6 Graph Transport Network

Graph distance learning. The ability to predict similarities or distances between graph-structured objects is useful across a wide range of applications. It can e.g. be used to predict the reaction rate between molecules (Houston et al., 2019), search for similar images (Johnson et al., 2015), similar molecules for drug discovery (Birchall et al., 2006), or similar code for vulnerability detection (Li et al., 2019). We propose the graph transport network (GTN) to evaluate approximate Sinkhorn and enhanced optimal transport and advance the state of the art on this task.

Graph transport network. GTN first uses a Siamese graph neural network (GNN) to embed two graphs independently as *sets* of node embeddings. These embeddings are then matched using enhanced entropy-regularized optimal transport. Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with node set $\mathcal{V}$ and edge set $\mathcal{E}$, node attributes $\boldsymbol{x}_i \in \mathbb{R}^{H_\text{x}}$ and (optional) edge attributes $\boldsymbol{e}_{i,j} \in \mathbb{R}^{H_\text{e}}$, with $i, j \in \mathcal{V}$, we update the node embeddings in each GNN layer via

$$\boldsymbol{h}_{\text{self},i}^{(l)} = \sigma(\boldsymbol{W}_{\text{node}}^{(l)} \boldsymbol{h}_i^{(l-1)} + \boldsymbol{b}^{(l)}), \tag{10}$$

$$\boldsymbol{h}_i^{(l)} = \boldsymbol{h}_{\text{self},i}^{(l)} + \sum_{j \in \mathcal{N}_i} \eta_{i,j}^{(l)} \boldsymbol{h}_{\text{self},j}^{(l)} \mathbf{W}_{\text{edge}} \boldsymbol{e}_{i,j}, \tag{11}$$

with $\mathcal{N}_i$ denoting the neighborhood of node i, $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i$, $\boldsymbol{h}_i^{(l)} \in \mathbb{R}^{H_\text{N}}$ for $l \geq 1$, the bilinear layer $\mathbf{W}_{\text{edge}} \in \mathbb{R}^{H_\text{N} \times H_\text{N} \times H_\text{e}}$, and the degree normalization $\eta_{i,j}^{(1)} = 1$ and $\eta_{i,j}^{(l)} = 1/\sqrt{\deg_i \deg_j}$ for $l > 1$. This choice of $\eta_{i,j}$ allows our model to handle highly skewed degree distributions while still being able to represent node degrees. We found the choice of non-linearity σ not to be critical and chose a LeakyReLU. We do not use the bilinear layer $\mathbf{W}_{\text{edge}} \boldsymbol{e}_{i,j}$ if there are no edge attributes. We aggregate each layer's node embeddings to obtain the final embedding of node i

$$\boldsymbol{h}_i^{\text{final}} = [\boldsymbol{h}_{\text{self},i}^{(1)} \parallel \boldsymbol{h}_i^{(1)} \parallel \boldsymbol{h}_i^{(2)} \parallel \ldots \parallel \boldsymbol{h}_i^{(L)}]. \tag{12}$$

Having obtained the embedding sets H_1^{final} and H_2^{final} of both graphs we use the L_2 distance as a cost function and calculate the Sinkhorn distance with multi-head unbalanced OT, as described in Sec. 5. If we have a single head we obtain the embeddings for matching via $\boldsymbol{h}_i^{(0)} = \text{MLP}(\boldsymbol{h}_i^{\text{final}})$ and obtain the final prediction via $d = d_c^\lambda w_{\text{out}} + b_{\text{out}}$, with learnable w_{out} and b_{out}. Otherwise we flip this order, using a set of linear layers $\boldsymbol{h}_i^{(\text{k})} = \boldsymbol{W}^{(k)} \boldsymbol{h}_i^{\text{final}}$ for head k and obtaining the final prediction via $d = \text{MLP}(\boldsymbol{d}_c^\lambda)$. All weights in GTN are trained end-to-end via backpropagation. For small graphs we use the full Sinkhorn distance and scale to large graphs by leveraging LCN-Sinkhorn. GTN is more expressive than models that use a single fixed-size embedding for the entire graph (Ying et al., 2018) but still scales log-linearly in the number of nodes, as opposed to previous approaches that scale quadratically (Li et al., 2019; Fey et al., 2020). Note that GTN inherently performs graph matching and can therefore also be applied to this task.

7 Experiments

Approximating Sinkhorn. We start by directly investigating different Sinkhorn approximations. To do so we compute entropy-regularized OT on pairs of $10\,000$ word embeddings, which we preprocess with Wasserstein Procrustes alignment in order to obtain both close and distant neighbors. For a fair comparison we let every method use the same total number of 40 neighbors and landmarks (LCN uses 20 each) and set the regularization to $\lambda = 0.05$ (as in Grave et al. (2019)) We measure transport plan approximation quality by (a) calculating the Pearson correlation coefficient (PCC) between all

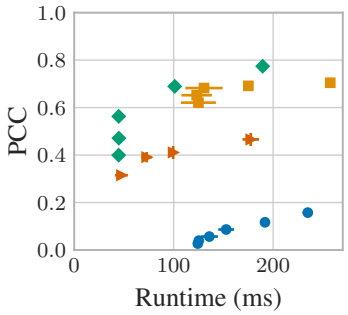

Figure 2: Tradeoff between OT plan approximation (via PCC) and runtime. Sparse Sinkhorn offers the best tradeoff, with LCN-Sinkhorn trailing closely behind.

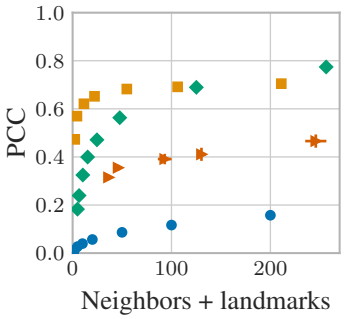

Figure 3: Tradeoff between OT plan approximation and number of neighbors/landmarks. LCN-Sinkhorn achieves the best approximation for low budgets.

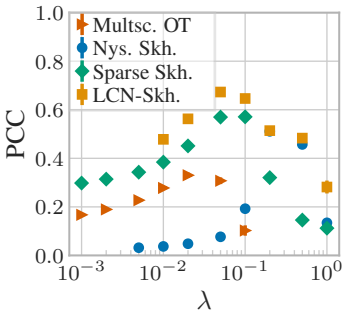

Figure 4: OT plan approximation quality for varying entropy regularization λ. Sparse Sinkhorn performs best for low and LCN-Sinkhorn for moderate λ.

Table 2: Accuracy and standard deviation across 5 runs for unsupervised word embedding alignment with Wasserstein Procrustes. LCN-Sinkhorn improves upon the original by 3.1 pp. before and 2.0 pp. after iterative CSLS refinement. *Migrated and re-run on GPU via PyTorch

	Time (s)	EN-ES	ES-EN	EN-FR	FR-EN	EN-DE	DE-EN	EN-RU	RU-EN	Avg.
Original*	268	79.2(2)	78.8(2.8)	81.0(3)	79.4(9)	71.7(2)	65.7(3.4)	36.3(1.1)	51.1(1.1)	67.9
Full Sinkhorn	402	81.1()	**82.0()**	81.2()	81.3()	**74.1()**	70.7()	37.3()	53.5()	70.1
Multiscale OT	88.2	23.6(31.4)	74.7(3.3)	26.9(31.7)	6.3(4.4)	35.8(10.4)	47.0(20.5)	0.0()	0.2(1)	26.8
Nyström Skh.	102	64.4(1.0)	59.3(1.2)	64.1(1.6)	56.8(4.0)	54.1(6)	47.1(3.5)	14.1(1.2)	22.5(2.4)	47.8
Sparse Skh.	49.2	80.2(2)	81.7(4)	80.9(3)	80.1(2)	72.1(6)	65.1(1.7)	35.5(6)	51.5(4)	68.4
LCN-Sinkhorn	86.8	**81.8(2)**	**81.3(1.8)**	**82.0(4)**	**82.1(3)**	73.6(2)	**71.3(9)**	**41.0(8)**	**55.1(1.4)**	**71.0**
Original* + ref.	268+81	83.0(3)	**82.0(2.5)**	**83.8(1)**	83.0(4)	**77.3(3)**	69.7(4.3)	46.2(1.0)	54.0(1.1)	72.4
LCN-Skh. + ref.	86.8+81	**83.5(2)**	**83.1(1.3)**	**83.8(2)**	**83.6(1)**	**77.2(3)**	**72.8(7)**	**51.8(2.6)**	**59.2(1.9)**	**74.4**

entries in the approximated plan and the true $\bar{\boldsymbol{P}}$ and (b) comparing the sets of $0.1\,\%$ largest entries in the approximated and true $\bar{\boldsymbol{P}}$ using the Jaccard similarity (intersection over union, IoU).
Table 1 shows that both sparse Sinkhorn and LCN-Sinkhorn obtain distances that are significantly closer to the true d_c^λ than Multiscale OT and Nyström-Sinkhorn. Furthermore, the transport plan computed by sparse Sinkhorn and LCN-Sinkhorn show both a PCC and IoU that are around twice as high as Multiscale OT, while Nyström-Sinkhorn exhibits almost no correlation. LCN-Sinkhorn performs especially well in this regard. This is also evident in Fig. 1, which shows how the $10^4 \times 10^4$ approximated OT plan entries compared to the true Sinkhorn values. Fig. 2 shows that sparse Sinkhorn offers the best trade-off between runtime and OT plan quality. However, LCN-Sinkhorn performs better for use cases with constrained memory (few neighbors/landmarks), as shown in Fig. 3. The number of neighbors and landmarks directly determines memory usage and is linearly proportional to the runtime (see App. G). Fig. 9 shows that sparse Sinkhorn performs best for low regularizations, where LCN-Sinkhorn fails due to the Nyström part going out of bounds. Nyström Sinkhorn performs best at rather high values and LCN-Sinkhorn always performs better than both (as long as it can be calculated). Interestingly, all approximations seem to fail at high λ. We defer the analogous discussion on the distance approximation to App. H. All approximations scale linearly both in the number of neighbors/landmarks and dataset size, as shown in App. G. Overall, we see that sparse Sinkhorn and LCN-Sinkhorn perform significantly better than previous approximations. However, do these improvements also translate to overall performance on downstream tasks?

Embedding alignment. Embedding alignment is the task of finding the orthogonal matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ that best aligns the vectors from two different embedding spaces, which is e.g. useful for unsupervised word translation. We use the experimental setup established by Conneau et al. (2018) by migrating Grave et al. (2019)'s implementation to PyTorch. The only change we make is using the full set of 20 000 word embeddings and training for 300 steps, while reducing the learning rate by half every 100 steps. We do not change *any* other hyperparameters and do not use unbalanced OT. After training we match pairs via cross-domain similarity local scaling (CSLS) (Conneau et al., 2018). We use 10 Sinkhorn iterations, 40 neighbors for sparse Sinkhorn, and 20 neighbors and landmarks for LCN-Sinkhorn (for details see App. D). We allow both multiscale OT and Nyström Sinkhorn to use

Table 3: RMSE for GED regression across 3 runs and the targets' standard deviation σ. GTN outperforms previous models by $52\,\%$.

	Linux	AIDS30	Pref. att.
σ	0.184	16.2	48.3
SiamMPNN	$0.090(7)$	$13.8(3)$	$12.1(6)$
SimGNN	0.039	$4.5(3)$	$8.3(1.4)$
GMN	$0.021(4)$	$10.3(6)$	$7.8(3)$
GTN, 1 head	$0.022(1)$	$3.7(1)$	$4.5(3)$
8 OT heads	$\mathbf{0.011()}$	$\mathbf{3.2(1)}$	$\mathbf{3.6(2)}$
Balanced OT	$0.034(1)$	$15.3(1)$	$27.4(9)$

Table 4: RMSE for graph distance regression across 3 runs. Using LCN-Sinkhorn with GTN increases the error by only $11\,\%$ and allows log-linear scaling.

	GED		PM $[10^{-2}]$
	AIDS30	Pref. att.	Pref. att. 200
σ	16.2	48.3	10.2
Full Sinkhorn	$3.7(1)$	$4.5(3)$	$1.27(6)$
Nyström Skh.	$3.6(3)$	$6.1(8)$	$2.43(7)$
Multiscale OT	$11.2(3)$	$24.5(2.8)$	$6.71(44)$
Sparse Skh.	$44.0(24.8)$	$40.7(6.6)$	$7.57(1.09)$
LCN-Skh.	$4.0(1)$	$5.2(5)$	$1.41(15)$

as many landmarks and neighbors as can fit into GPU memory and finetune both methods.
Table 2 shows that using full entropy-regularized OT yields a significant improvement in accuracy on this task compared to the original stochastic training scheme. In most cases LCN-Sinkhorn even outperforms the *full* version, while running 4.6x faster than full Sinkhorn and 3.1x faster than the original scheme. Sparse Sinkhorn runs 1.8x faster than LCN-Sinkhorn but cannot match its accuracy. LCN-Sinkhorn still outcompetes the original method after refining the embeddings with iterative local CSLS (Conneau et al., 2018). Both multiscale OT and Nyström Sinkhorn fail at this task, despite their larger computational budget. This shows that the improvements achieved by sparse Sinkhorn and LCN-Sinkhorn have an even larger impact in practice.

Graph distance regression. The graph edit distance (GED) is useful for various tasks, such as image retrieval (Xiao et al., 2008) or fingerprint matching (Neuhaus & Bunke, 2004), but its computation is NP-complete (Bunke & Shearer, 1998). Therefore, to use it on larger graphs we need to learn an approximation. We use the Linux dataset by Bai et al. (2019) and generate 2 new datasets by computing the exact GED using the method by Lerouge et al. (2017) on small graphs (≤ 30 nodes) from the AIDS dataset (Riesen & Bunke, 2008) and a set of preferential attachment graphs. We compare GTN to 3 state-of-the-art baselines: SiameseMPNN (Riba et al., 2018), SimGNN (Bai et al., 2019), and the Graph Matching Network (GMN) (Li et al., 2019). We tune the hyperparameters of all baselines and GTN on the validation set via a grid search. For more details see App. D to F.
We first test both GTN and the proposed OT enhancements. Table 3 shows that GTN improves upon competing models by $34\,\%$ with a single head and by $52\,\%$ with 8 OT heads. These improvements break down when using regular balanced OT, showing the importance of learnable unbalanced OT.
Having established GTN as a state-of-the-art model we next ask whether we can sustain its performance when using approximate OT. To test this we additionally generate a set of larger graphs with around 200 nodes and use the Pyramid matching (PM) kernel (Nikolentzos et al., 2017) as the prediction target, since these graphs are too large to compute the GED. See App. F for hyperparameter details. Table 4 shows that both sparse Sinkhorn and the multiscale method using 4 (expected) neighbors fail at this task, demonstrating that the low-rank approximation in LCN has a crucial stabilizing effect during training. Nyström Sinkhorn with 4 landmarks performs surprisingly well on the AIDS30 dataset, suggesting an overall low-rank structure with Nyström acting as regularization. However, it does not perform as well on the other two datasets. Using LCN-Sinkhorn with 2 neighbors and landmarks works well on all three datasets, with an RMSE increased by only $11\,\%$ compared to full GTN. App. G furthermore shows that GTN with LCN-Sinkhorn indeed scales linearly in the number of nodes across multiple orders of magnitude. This model thus allows to perform graph matching and distance learning on graphs that are considered large even for simple node-level tasks ($20\,000$ nodes).

8 Conclusion

Locality sensitive hashing (LSH) and the novel locally corrected Nyström (LCN) method enable fast and accurate approximations of entropy-regularized OT with log-linear runtime: Sparse Sinkhorn and LCN-Sinkhorn. Sinkhorn's performance for ML models can be further improved with multi-head OT and learnable unbalanced OT. The graph transport network (GTN) is one example for such a model. It sets the new state of the art for graph distance learning while still scaling log-linearly with graph size. These contributions enable new applications and models that are faster and more accurate, since they can sidestep workarounds such as pooling.

REFERENCES

Pierre Ablin, Gabriel Peyré, and Thomas Moreau. Super-efficiency of automatic differentiation for functions defined as a minimum. In *ICML*, 2020.

Mokhtar Z. Alaya, Maxime Berar, Gilles Gasso, and Alain Rakotomamonjy. Screening Sinkhorn Algorithm for Regularized Optimal Transport. In *NeurIPS*, 2019.

Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *NeurIPS*, 2017.

Jason Altschuler, Francis Bach, Alessandro Rudi, and Jonathan Niles-Weed. Massively scalable Sinkhorn distances via the Nyström method. In *NeurIPS*, 2019.

David Alvarez-Melis and Tommi S. Jaakkola. Gromov-Wasserstein Alignment of Word Embedding Spaces. In *EMNLP*, 2018.

Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *ACM-SIAM symposium on Discrete algorithms (SODA)*, 2008.

Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. In *NeurIPS*, 2015.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In *ICML*, 2017.

Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable Nearest Neighbor Search for Optimal Transport. In *ICML*, 2020.

Yunsheng Bai, Hao Ding, Yizhou Sun, and Wei Wang. Convolutional Set Matching for Graph Similarity. In *NeurIPS-W*, 2018.

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In *WSDM*, 2019.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR*, 1308.3432, 2013.

Christian Berg, Jens Peter Reus Christensen, and Paul Ressel. *Harmonic Analysis on Semigroups*. Number 100 in Graduate Texts in Mathematics. 1984.

Kristian Birchall, Valerie J. Gillet, Gavin Harper, and Stephen D. Pickett. Training Similarity Measures for Specific Activities: Application to Reduced Graphs. *Journal of Chemical Information and Modeling*, 46(2):577–586, 2006.

Mathieu Blondel, Vivien Seguy, and Antoine Rolet. Smooth and Sparse Optimal Transport. In *AISTATS*, 2018.

Olivier Bousquet, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. From optimal transport to generative modeling: the VEGAN cookbook. *arXiv*, 1705.07642, 2017.

Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3):255–259, 1998.

Moses Charikar. Similarity estimation techniques from rounding algorithms. In *ACM symposium on Theory of computing (STOC)*, 2002.

Lénaïc Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard. Scaling algorithms for unbalanced optimal transport problems. *Mathematics of Computation*, 87(314):2563–2609, 2018.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. In *ICLR*, 2018.

Nicolas Courty, Rémi Flamary, Amaury Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. In *NeurIPS*, 2017.

Marco Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In *NeurIPS*, 2013.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*, 2019.

Matthias Fey and Jan E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR-W*, 2019.

Matthias Fey, Jan Eric Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. Deep Graph Matching Consensus. In *ICLR*, 2020.

Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, and Tomaso A. Poggio. Learning with a Wasserstein Loss. In *NeurIPS*, 2015.

Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis R. Bach. Stochastic Optimization for Large-scale Optimal Transport. In *NeurIPS*, 2016.

Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning Generative Models with Sinkhorn Divergences. In *AISTATS*, 2018.

Samuel Gerber and Mauro Maggioni. Multiscale Strategies for Computing Optimal Transport. *J. Mach. Learn. Res.*, 18:72:1–72:32, 2017.

Edouard Grave, Armand Joulin, and Quentin Berthet. Unsupervised Alignment of Embeddings with Wasserstein Procrustes. In *AISTATS*, 2019.

Paul L. Houston, Apurba Nandi, and Joel M. Bowman. A Machine Learning Approach for Prediction of Rate Constants. *The Journal of Physical Chemistry Letters*, 10(17):5250–5258, 2019.

Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *ICCV-W*, 2003.

Arun Jambulapati, Aaron Sidford, and Kevin Tian. A Direct tilde{O}(1/epsilon) Iteration Parallel Algorithm for Optimal Transport. In *NeurIPS*, 2019.

Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Image retrieval using scene graphs. In *CVPR*, 2015.

Armand Joulin, Piotr Bojanowski, Tomas Mikolov, Hervé Jégou, and Edouard Grave. Loss in Translation: Learning Bilingual Word Mapping with a Retrieval Criterion. In *EMNLP*, 2018.

Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Rueckert. Distance Metric Learning Using Graph Convolutional Networks: Application to Functional Brain Networks. In *MICCAI*, 2017.

Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognit.*, 72: 254–265, 2017.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *ICML*, 2019.

Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. GOT: An Optimal Transport framework for Graph comparison. In *NeurIPS*, 2019.

Cheng Meng, Yuan Ke, Jingyi Zhang, Mengrui Zhang, Wenxuan Zhong, and Ping Ma. Large-scale optimal transport map estimation using projection pursuit. In *NeurIPS*, 2019.

Cameron Musco and Christopher Musco. Recursive Sampling for the Nystrom Method. In *NeurIPS*, 2017.

Michel Neuhaus and Horst Bunke. An Error-Tolerant Approximate Matching Algorithm for Attributed Planar Graphs and Its Application to Fingerprint Classification. In *Structural, Syntactic, and Statistical Pattern Recognition*, 2004.

Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching Node Embeddings for Graph Similarity. In *AAAI*, 2017.

David Nistér and Henrik Stewénius. Scalable Recognition with a Vocabulary Tree. In *CVPR*, 2006.

Nicolas Papadakis, Gabriel Peyré, and Édouard Oudet. Optimal Transport with Proximal Splitting. *SIAM J. Imaging Sciences*, 7(1):212–238, 2014.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, 2019.

Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognit. Lett.*, 31(11):1348–1358, 2010.

Gabriel Peyré and Marco Cuturi. Computational Optimal Transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein Barycenter and Its Application to Texture Mixing. In *Scale Space and Variational Methods in Computer Vision (SSVM)*, 2011.

Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning Graph Distances with Message Passing Neural Networks. In *ICPR*, 2018.

Kaspar Riesen and Horst Bunke. IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In *Structural, Syntactic, and Statistical Pattern Recognition*, 2008.

Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27(7):950–959, 2009.

Sebastian Ruder, Ivan Vulic, and Anders Søgaard. A Survey of Cross-lingual Word Embedding Models. *J. Artif. Intell. Res.*, 65:569–631, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In *ICML*, 2016.

Bernhard Schmitzer. Stabilized Sparse Scaling Algorithms for Entropy Regularized Transport Problems. *SIAM Journal on Scientific Computing*, 41(3):A1443–A1481, 2019.

John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. 2004.

Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *NeurIPS*, 2014.

Justin Solomon, Raif M. Rustamov, Leonidas J. Guibas, and Adrian Butscher. Earth mover's distances on discrete surfaces. *ACM Trans. Graph.*, 33(4):67:1–67:12, 2014.

Justin Solomon, Fernando de Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas J. Guibas. Convolutional wasserstein distances: efficient optimal transportation on geometric domains. *ACM Trans. Graph.*, 34(4):66:1–66:11, 2015.

Robert E. Tarjan. Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. *Mathematical Programming*, 78(2):169–177, 1997.

Evgeny Tenetov, Gershon Wolansky, and Ron Kimmel. Fast Entropic Regularized Optimal Transport Using Semidiscrete Cost Approximation. *SIAM J. Sci. Comput.*, 40(5):A3400–A3422, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *NeurIPS*, 2017.

Titouan Vayer, Nicolas Courty, Romain Tavenard, Laetitia Chapel, and Rémi Flamary. Optimal Transport for structured data with application on graphs. In *ICML*, 2019.

Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for Similarity Search: A Survey. *CoRR*, 1408.2927, 2014.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning Combinatorial Embedding Networks for Deep Graph Matching. In *ICCV*, 2019.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström Method to Speed Up Kernel Machines. In *NeurIPS*, 2001.

Bing Xiao, Xinbo Gao, Dacheng Tao, and Xuelong Li. HMM-based graph edit distance for image indexing. *Int. J. Imaging Systems and Technology*, 18(2-3):209–218, 2008.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*, 2018.

Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved Nyström low-rank approximation and error analysis. In *ICML*, 2008.

Xiaojin Zhu and Andrew B. Goldberg. Introduction to Semi-Supervised Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130, 2009.

A COMPLEXITY ANALYSIS

Sparse Sinkhorn. Sparse Sinkhorn first obtains neighbor pairs via LSH and then computes the cost matrix entries $\boldsymbol{C}_{ij}$ only for these pairs, effectively assuming it to be infinity everywhere else. Calculating the hash buckets for all points for a given B, r, and b hash buckets per function scales as $\mathcal{O}((n+m)dBbr)$ for the hash functions we consider. As expected, for the tasks and hash functions we investigated we obtain approximately m/b^r and n/b^r neighbors, with b^r hash buckets per band. Using this we can fix the number of neighbors to a small, constant β in expectation with $b^r = \min(n,m)/\beta$. We thus obtain a sparse cost matrix $\boldsymbol{C}^{\text{sp}}$ with $\mathcal{O}(\max(n,m)\beta)$ non-infinite values and can calculate $\boldsymbol{s}$ and $\boldsymbol{t}$ in linear time $\mathcal{O}(N_{\text{sink}} \max(n,m)\beta)$, where $N_{\text{sink}} \propto \mathcal{O}(1/\varepsilon^3)$ (Altschuler et al., 2017) denotes the number of Sinkhorn iterations, with distance error ε. Calculating the hash buckets with $r = \frac{\log \min(n,m) - \log \beta}{\log b}$ takes $\mathcal{O}((n+m)dBb(\log \min(n,m) - \log \beta)/\log b)$. Since B, b, and β are small, we obtain roughly log-linear scaling with the number of points overall, i.e. $\mathcal{O}(n \log n)$ for $n \approx m$.

LCN-Sinkhorn. Both choosing landmarks via k-means++ sampling and via k-means with a fixed number of iterations have the same runtime complexity of $\mathcal{O}((n+m)ld)$. Precomputing $\boldsymbol{W}$ can be done in time $\mathcal{O}(nl^2 + l^3)$. The low-rank part of updating the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ can be computed in $\mathcal{O}(nl + l^2 + lm)$, with l chosen constant, i.e. independently of n and m. Since sparse Sinkhorn with LSH has a log-linear runtime we again obtain log-linear overall runtime for LCN-Sinkhorn.

B LIMITATIONS

Sparse Sinkhorn. Using a sparse approximation for $\boldsymbol{K}$ works well in the common case when the regularization parameter λ is low and the cost function varies enough between data pairs, such that the transport plan $\boldsymbol{P}$ resembles a sparse matrix. However, it can fail if the cost between pairs is very similar or the regularization is very high, if the dataset contains many hubs, i.e. points with a large number of neighbors, or if the distributions $\boldsymbol{p}$ or $\boldsymbol{q}$ are spread very unevenly. Furthermore, sparse Sinkhorn can be too unstable to train a model from scratch, since randomly initialized embeddings often have no close neighbors (see Sec. 7). LCN-Sinkhorn largely alleviates these limitations.

LCN-Sinkhorn. Since we cannot calculate the full cost matrix, LCN-Sinkhorn cannot provide accuracy guarantees in general. Highly concentrated distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ might have adverse effects on LCN-Sinkhorn. However, we can compensate for these by sampling landmarks or neighbors proportional to each point's probability mass. We therefore do not expect LCN-Sinkhorn to break down in this scenario. If the regularization parameter is low or the cost function varies greatly, we sometimes observed stability issues (over- and underflows) with the Nyström approximation because of the inverse $\boldsymbol{A}^{-1}$, which cannot be calculated in log-space. Due to its linearity the Nyström method furthermore sometimes approximates similarities as negative values, which leads to a failure if the result of the matrix product in Eq. (2) becomes negative. In these extreme cases we also observed catastrophic elimination caused by the correction $\boldsymbol{K}^{\text{sp}}_{\Delta}$. Since this essentially means that optimal transport will be very local, we recommend using sparse Sinkhorn in these scenarios. This again demonstrates the complementarity of the sparse approximation and Nyström: In cases where one fails we can often resort to the other.

C PROOF OF PROPOSITION 1

Theorem 1 (Danskin's theorem)**.** *Consider a continuous function $\phi : \mathbb{R}^k \times Z \to \mathbb{R}$, with the compact set $Z \subset \mathbb{R}^j$. If $\phi(\boldsymbol{x}, \boldsymbol{z})$ is convex in $\boldsymbol{x}$ for every $\boldsymbol{z} \in Z$ and $\phi(\boldsymbol{x}, \boldsymbol{z})$ has a unique maximizer $\bar{\boldsymbol{z}}$, the derivative of*

$$f(\boldsymbol{x}) = \max_{\boldsymbol{z} \in Z} \phi(\boldsymbol{x}, \boldsymbol{z}) \tag{13}$$

is given by the derivative at the maximizer, i.e.

$$\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial \phi(\boldsymbol{x}, \bar{\boldsymbol{z}})}{\partial \boldsymbol{x}}. \tag{14}$$

We start by deriving the derivatives of the distances. To show that the Sinkhorn distance fulfills the conditions for Danskin's theorem we first identify $\boldsymbol{x} = \boldsymbol{C}$, $\boldsymbol{z} = \boldsymbol{P}$, and $\phi(\boldsymbol{C}, \boldsymbol{P}) = -\langle \boldsymbol{P}, \boldsymbol{C} \rangle_{\text{F}} +$

$\lambda H(\boldsymbol{P})$. We next observe that the restrictions $\boldsymbol{P}\mathbf{1}_m = \boldsymbol{p}$ and $\boldsymbol{P}^T\mathbf{1}_n = \boldsymbol{q}$ define a compact, convex set for $\boldsymbol{P}$. Furthermore, ϕ is a continuous function and linear in $\boldsymbol{C}$, i.e. both convex and concave for any finite $\boldsymbol{P}$. Finally, $\phi(\boldsymbol{C}, \boldsymbol{P})$ is concave in $\boldsymbol{P}$ since $\langle \boldsymbol{P}, \boldsymbol{C}\rangle_{\mathrm{F}}$ is linear and $\lambda H(\boldsymbol{P})$ is concave. Therefore the maximizer $\bar{\boldsymbol{P}}$ is unique and Danskin's theorem applies to the Sinkhorn distance. Using

$$\frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{U}_{kl}} = \frac{\partial}{\partial \boldsymbol{U}_{kl}}\left(-\lambda \log(\sum_a \boldsymbol{U}_{ia}\boldsymbol{W}_{aj})\right) = -\lambda\delta_{ik}\frac{\boldsymbol{W}_{lj}}{\sum_a \boldsymbol{U}_{ia}\boldsymbol{W}_{aj}} = -\lambda\delta_{ik}\frac{\boldsymbol{W}_{lj}}{\boldsymbol{K}_{\text{Nys},ij}}, \tag{15}$$

$$\frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{W}_{kl}} = \frac{\partial}{\partial \boldsymbol{W}_{kl}}\left(-\lambda \log(\sum_a \boldsymbol{U}_{ia}\boldsymbol{W}_{aj})\right) = -\lambda\delta_{jl}\frac{\boldsymbol{U}_{ik}}{\sum_a \boldsymbol{U}_{ia}\boldsymbol{W}_{aj}} = -\lambda\delta_{jl}\frac{\boldsymbol{U}_{ik}}{\boldsymbol{K}_{\text{Nys},ij}}, \tag{16}$$

$$\frac{\bar{\boldsymbol{P}}_{\text{Nys},ij}}{\boldsymbol{K}_{\text{Nys},ij}} = \frac{\sum_b \bar{\boldsymbol{P}}_{U,ib}\bar{\boldsymbol{P}}_{W,bj}}{\sum_a \boldsymbol{U}_{ia}\boldsymbol{W}_{aj}} = \frac{\bar{\boldsymbol{s}}_i\bar{\boldsymbol{t}}_j\sum_b \boldsymbol{U}_{ib}\boldsymbol{W}_{bj}}{\sum_a \boldsymbol{U}_{ia}\boldsymbol{W}_{aj}} = \bar{\boldsymbol{s}}_i\bar{\boldsymbol{t}}_j \tag{17}$$

we can calculate the derivative w.r.t. the cost matrix as

$$\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}} = -\frac{\partial}{\partial \boldsymbol{C}}\left(-\left\langle \bar{\boldsymbol{P}}, \boldsymbol{C}\right\rangle_{\mathrm{F}} + \lambda H(\bar{\boldsymbol{P}})\right) = \bar{\boldsymbol{P}}, \tag{18}$$

$$\begin{aligned}\frac{\partial d^\lambda_{\text{LCN},c}}{\partial \boldsymbol{U}_{kl}} &= \sum_{i,j}\frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{U}_{kl}}\frac{\partial d^\lambda_{\text{LCN},c}}{\partial \boldsymbol{C}_{\text{Nys},ij}} = -\lambda\sum_{i,j}\delta_{ik}\boldsymbol{W}_{lj}\frac{\bar{\boldsymbol{P}}_{\text{Nys},ij}}{\boldsymbol{K}_{\text{Nys},ij}} \\ &= -\lambda\sum_{i,j}\delta_{ik}\boldsymbol{W}_{lj}\bar{\boldsymbol{s}}_i\bar{\boldsymbol{t}}_j = -\lambda\bar{\boldsymbol{s}}_k\sum_j \boldsymbol{W}_{lj}\bar{\boldsymbol{t}}_j = \left(-\lambda\bar{\boldsymbol{s}}(\boldsymbol{W}\bar{\boldsymbol{t}})^T\right)_{kl},\end{aligned} \tag{19}$$

$$\begin{aligned}\frac{\partial d^\lambda_{\text{LCN},c}}{\partial \boldsymbol{W}_{kl}} &= \sum_{i,j}\frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{W}_{kl}}\frac{\partial d^\lambda_{\text{LCN},c}}{\partial \boldsymbol{C}_{\text{Nys},ij}} = -\lambda\sum_{i,j}\delta_{jl}\boldsymbol{U}_{ik}\frac{\bar{\boldsymbol{P}}_{\text{Nys},ij}}{\boldsymbol{K}_{\text{Nys},ij}} \\ &= -\lambda\sum_{i,j}\delta_{jl}\boldsymbol{U}_{ik}\bar{\boldsymbol{s}}_i\bar{\boldsymbol{t}}_j = -\lambda\left(\sum_i \bar{\boldsymbol{s}}_i\boldsymbol{U}_{ik}\right)\bar{\boldsymbol{t}}_l = \left(-\lambda(\bar{\boldsymbol{s}}^T\boldsymbol{U})^T\bar{\boldsymbol{t}}^T\right)_{kl},\end{aligned} \tag{20}$$

and $\frac{\partial d^\lambda_{\text{LCN},c}}{\partial \log \boldsymbol{K}^{\text{sp}}}$ and $\frac{\partial d^\lambda_{\text{LCN},c}}{\partial \log \boldsymbol{K}^{\text{sp}}_{\text{Nys}}}$ directly follow from $\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}}$.

Theorem 2 (Implicit function theorem)**.** *Let $f : \mathbb{R}^{n'} \times \mathbb{R}^{m'} \to \mathbb{R}^{m'}$ be a continuously differentiable function with $f(\boldsymbol{a}, \boldsymbol{b}) = \mathbf{0}$. If its Jacobian matrix $J_{f_i(\boldsymbol{x},\boldsymbol{y}),\boldsymbol{y}_j} = \frac{\partial f_i}{\partial \boldsymbol{y}_j}(\boldsymbol{a}, \boldsymbol{b})$ is invertible, then there exists an open set $\boldsymbol{a} \in U \subset \mathbb{R}^{n'}$ on which there exists a unique continuously differentiable function $g : U \to \mathbb{R}^{m'}$ with $g(\boldsymbol{a}) = \boldsymbol{b}$ and $\forall \boldsymbol{x} \in U : f(\boldsymbol{x}, g(\boldsymbol{x})) = 0$. Moreover,*

$$\frac{\partial g_i}{\partial \boldsymbol{x}_j}(\boldsymbol{x}) = -\left(J_{f(\boldsymbol{x},\boldsymbol{y}),\boldsymbol{y}}(\boldsymbol{x}, g(\boldsymbol{x}))\right)^{-1}_{i,:}\left(\frac{\partial}{\partial \boldsymbol{x}}f(\boldsymbol{x}, g(\boldsymbol{x}))\right)_{:,j} \tag{21}$$

Next we derive the transport plan derivatives. To apply the implicit function theorem we identify $\boldsymbol{x} = \boldsymbol{C}$ and $\boldsymbol{y} = \boldsymbol{P}$ as flattened matrices. We will index these flat matrices via index pairs to simplify interpretation. We furthermore identify

$$f(\boldsymbol{C}, \boldsymbol{P}) = \frac{\partial}{\partial \bar{\boldsymbol{P}}}\left(\left\langle \bar{\boldsymbol{P}}, \boldsymbol{C}\right\rangle_{\mathrm{F}} - \lambda H(\bar{\boldsymbol{P}})\right) = \boldsymbol{C} + \lambda(\log \bar{\boldsymbol{P}} + 1). \tag{22}$$

The minimizer $\bar{\boldsymbol{P}}$ cannot lie on the boundary of $\boldsymbol{P}$'s valid region since $\lim_{p\to 0}\frac{\partial}{\partial p}p\log p = \lim_{p\to 0}\log p + 1 = -\infty$ and therefore $\bar{\boldsymbol{P}}_{ij} > 0$. Hence, $f(\boldsymbol{C}, \bar{\boldsymbol{P}}) = 0$ with $\bar{\boldsymbol{P}}(\boldsymbol{C}) = \arg\min_{\boldsymbol{P}} \langle \boldsymbol{P}, \boldsymbol{C}\rangle_{\mathrm{F}} - \lambda H(\boldsymbol{P})$ and we find that $g(\boldsymbol{x}) = \bar{\boldsymbol{P}}(\boldsymbol{C})$. We furthermore obtain

$$J_{f_{ij}(\boldsymbol{C},\boldsymbol{P}),\boldsymbol{P}_{ab}}(\boldsymbol{C}, \bar{\boldsymbol{P}}(\boldsymbol{C})) = \frac{\partial}{\partial \bar{\boldsymbol{P}}_{ab}}\left(\boldsymbol{C}_{ij} + \lambda(\log \bar{\boldsymbol{P}}_{ij} + 1)\right) = \delta_{ia}\delta_{jb}\lambda/\bar{\boldsymbol{P}}_{ij}, \tag{23}$$

$$\frac{\partial}{\partial \boldsymbol{C}_{kl}}f_{ab}(\boldsymbol{C}, \bar{\boldsymbol{P}}(\boldsymbol{C})) = \frac{\partial}{\partial \boldsymbol{C}_{kl}}\left(\boldsymbol{C}_{ab} + \lambda(\log \bar{\boldsymbol{P}}_{ab} + 1)\right) = \delta_{ak}\delta_{bl}. \tag{24}$$

$J_{f_{ij}(\boldsymbol{C},\boldsymbol{P}),\boldsymbol{P}_{ab}}(\boldsymbol{C},\bar{\boldsymbol{P}}(\boldsymbol{C}))$ is hence a diagonal matrix and invertible since $\lambda/\bar{\boldsymbol{P}}_{ij} > 0$. We can thus use the implicit function theorem and obtain

$$\begin{aligned}\frac{\partial \bar{\boldsymbol{P}}_{ij}}{\partial \boldsymbol{C}_{kl}} &= -\sum_{a,b} \left(J_{f_{ij}(\boldsymbol{C},\boldsymbol{P}),\boldsymbol{P}_{ab}}(\boldsymbol{C},\bar{\boldsymbol{P}}(\boldsymbol{C}))\right)^{-1} \frac{\partial}{\partial \boldsymbol{C}_{kl}} f_{ab}(\boldsymbol{C},\bar{\boldsymbol{P}}(\boldsymbol{C})) \\ &= -\sum_{a,b} \delta_{ia}\delta_{jb}\frac{1}{\lambda}\bar{\boldsymbol{P}}_{ij}\delta_{ak}\delta_{bl} = -\frac{1}{\lambda}\bar{\boldsymbol{P}}_{ij}\delta_{ik}\delta_{jl}.\end{aligned} \tag{25}$$

To extend this result to LCN-OT we use

$$\frac{\partial \bar{\boldsymbol{P}}_{U,ij}}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} = \frac{\partial}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} \sum_k \bar{\boldsymbol{P}}_{\text{Nys},ik}\bar{\boldsymbol{P}}^{\dagger}_{W,kj} = \delta_{ia}\bar{\boldsymbol{P}}^{\dagger}_{W,bj}, \tag{26}$$

$$\frac{\partial \bar{\boldsymbol{P}}_{W,ij}}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} = \frac{\partial}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} \sum_k \bar{\boldsymbol{P}}^{\dagger}_{U,ik}\bar{\boldsymbol{P}}_{\text{Nys},kj} = \delta_{jb}\bar{\boldsymbol{P}}^{\dagger}_{U,ia} \tag{27}$$

and the chain rule to obtain

$$\begin{aligned}\frac{\partial \bar{\boldsymbol{P}}_{U,ij}}{\partial \boldsymbol{U}_{kl}} &= \sum_{a,b,c,d} \frac{\partial \boldsymbol{C}_{\text{Nys},cd}}{\partial \boldsymbol{U}_{kl}} \frac{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}}{\partial \boldsymbol{C}_{\text{Nys},cd}} \frac{\partial \bar{\boldsymbol{P}}_{U,ij}}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} \\ &= \sum_{a,b,c,d} \left(-\lambda\delta_{ck}\frac{\boldsymbol{W}_{ld}}{\boldsymbol{K}_{\text{Nys},cd}}\right)\left(-\frac{1}{\lambda}\bar{\boldsymbol{P}}_{\text{Nys},ab}\delta_{ac}\delta_{bd}\right)\left(\delta_{ia}\bar{\boldsymbol{P}}^{\dagger}_{W,bj}\right) \\ &= \sum_b \boldsymbol{W}_{lb}\frac{\bar{\boldsymbol{P}}_{\text{Nys},ib}}{\boldsymbol{K}_{\text{Nys},ib}}\bar{\boldsymbol{P}}^{\dagger}_{W,bj}\delta_{ik} = \delta_{ik}\sum_b \boldsymbol{s}_i\boldsymbol{W}_{lb}\boldsymbol{t}_b\bar{\boldsymbol{P}}^{\dagger}_{W,bj} = \delta_{ik}\boldsymbol{s}_i\sum_b \bar{\boldsymbol{P}}_{W,lb}\bar{\boldsymbol{P}}^{\dagger}_{W,bj} \\ &= \delta_{ik}\delta_{jl}\boldsymbol{s}_i,\end{aligned} \tag{28}$$

$$\begin{aligned}\frac{\partial \bar{\boldsymbol{P}}_{W,ij}}{\partial \boldsymbol{U}_{kl}} &= \sum_{a,b,c,d} \frac{\partial \boldsymbol{C}_{\text{Nys},cd}}{\partial \boldsymbol{U}_{kl}} \frac{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}}{\partial \boldsymbol{C}_{\text{Nys},cd}} \frac{\partial \bar{\boldsymbol{P}}_{W,ij}}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} \\ &= \sum_{a,b,c,d} \left(-\lambda\delta_{ck}\frac{\boldsymbol{W}_{ld}}{\boldsymbol{K}_{\text{Nys},cd}}\right)\left(-\frac{1}{\lambda}\bar{\boldsymbol{P}}_{\text{Nys},ab}\delta_{ac}\delta_{bd}\right)\left(\delta_{jb}\bar{\boldsymbol{P}}^{\dagger}_{U,ia}\right) \\ &= \boldsymbol{W}_{lj}\frac{\bar{\boldsymbol{P}}_{\text{Nys},kj}}{\boldsymbol{K}_{\text{Nys},kj}}\bar{\boldsymbol{P}}^{\dagger}_{U,ik} = \boldsymbol{W}_{lj}\boldsymbol{t}_j\bar{\boldsymbol{P}}^{\dagger}_{U,ik}\boldsymbol{s}_k = \bar{\boldsymbol{P}}^{\dagger}_{U,ik}\boldsymbol{s}_k\bar{\boldsymbol{P}}_{W,lj},\end{aligned} \tag{29}$$

$$\begin{aligned}\frac{\partial \bar{\boldsymbol{P}}_{U,ij}}{\partial \boldsymbol{W}_{kl}} &= \sum_{a,b,c,d} \frac{\partial \boldsymbol{C}_{\text{Nys},cd}}{\partial \boldsymbol{W}_{kl}} \frac{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}}{\partial \boldsymbol{C}_{\text{Nys},cd}} \frac{\partial \bar{\boldsymbol{P}}_{U,ij}}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} \\ &= \sum_{a,b,c,d} \left(-\lambda\delta_{dl}\frac{\boldsymbol{U}_{ck}}{\boldsymbol{K}_{\text{Nys},cd}}\right)\left(-\frac{1}{\lambda}\bar{\boldsymbol{P}}_{\text{Nys},ab}\delta_{ac}\delta_{bd}\right)\left(\delta_{ia}\bar{\boldsymbol{P}}^{\dagger}_{W,bj}\right) \\ &= \boldsymbol{U}_{ik}\frac{\bar{\boldsymbol{P}}_{\text{Nys},il}}{\boldsymbol{K}_{\text{Nys},il}}\bar{\boldsymbol{P}}^{\dagger}_{W,lj} = \boldsymbol{s}_i\boldsymbol{U}_{ik}\boldsymbol{t}_l\bar{\boldsymbol{P}}^{\dagger}_{W,lj} = \bar{\boldsymbol{P}}_{U,ik}\boldsymbol{t}_l\bar{\boldsymbol{P}}^{\dagger}_{W,lj},\end{aligned} \tag{30}$$

$$\begin{aligned}\frac{\partial \bar{\boldsymbol{P}}_{W,ij}}{\partial \boldsymbol{W}_{kl}} &= \sum_{a,b,c,d} \frac{\partial \boldsymbol{C}_{\text{Nys},cd}}{\partial \boldsymbol{W}_{kl}} \frac{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}}{\partial \boldsymbol{C}_{\text{Nys},cd}} \frac{\partial \bar{\boldsymbol{P}}_{W,ij}}{\partial \bar{\boldsymbol{P}}_{\text{Nys},ab}} \\ &= \sum_{a,b,c,d} \left(-\lambda\delta_{dl}\frac{\boldsymbol{U}_{ck}}{\boldsymbol{K}_{\text{Nys},cd}}\right)\left(-\frac{1}{\lambda}\bar{\boldsymbol{P}}_{\text{Nys},ab}\delta_{ac}\delta_{bd}\right)\left(\delta_{jb}\bar{\boldsymbol{P}}^{\dagger}_{U,ia}\right) \\ &= \sum_a \boldsymbol{U}_{ak}\frac{\bar{\boldsymbol{P}}_{\text{Nys},aj}}{\boldsymbol{K}_{\text{Nys},aj}}\bar{\boldsymbol{P}}^{\dagger}_{U,ia}\delta_{jl} = \delta_{jl}\sum_a \boldsymbol{s}_a\boldsymbol{U}_{ak}\boldsymbol{t}_j\bar{\boldsymbol{P}}^{\dagger}_{U,ia} = \delta_{jl}\boldsymbol{t}_j\sum_a \bar{\boldsymbol{P}}^{\dagger}_{U,ia}\bar{\boldsymbol{P}}_{U,ak} \\ &= \delta_{ik}\delta_{jl}\boldsymbol{t}_j.\end{aligned} \tag{31}$$

$\frac{\partial \bar{\boldsymbol{P}}^{\text{sp}}}{\partial \log \boldsymbol{K}^{\text{sp}}}$ and $\frac{\partial \bar{\boldsymbol{P}}^{\text{sp}}_{\text{Nys}}}{\partial \log \boldsymbol{K}^{\text{sp}}_{\text{Nys}}}$ directly follow from $\frac{\partial \bar{\boldsymbol{P}}}{\partial \boldsymbol{C}}$.

We can calculate the pseudoinverse $\bar{\boldsymbol{P}}^{\dagger}_U = (\bar{\boldsymbol{P}}_U^T\bar{\boldsymbol{P}}_U)^{-1}\bar{\boldsymbol{P}}_U$ and $\bar{\boldsymbol{P}}^{\dagger}_W = \bar{\boldsymbol{P}}_W^T(\bar{\boldsymbol{P}}_W\bar{\boldsymbol{P}}_W^T)^{-1}$ in time $\mathcal{O}(nl^2 + l^3)$ since $\bar{\boldsymbol{P}}_U \in \mathbb{R}^{n\times l}$ and $\bar{\boldsymbol{P}}_W \in \mathbb{R}^{l\times n}$. We do not fully instantiate the matrices required for

Table 5: Graph dataset statistics.

	Data	Distance	Distance (test set) Mean	Distance (test set) Std. dev.	Graphs train/val/test	Avg. nodes per graph	Avg. edges per graph	Node types	Edge types
AIDS30	Molecules	GED	50.5	16.2	144/48/48	20.6	44.6	53	4
Pref. att.	Init. attract.	GED	106.7	48.3	144/48/48	20.6	75.4	6	4
Pref. att. 200	Init. attract.	PM	0.400	0.102	144/48/48	199.3	938.8	6	-
Pref. att. 2k	Init. attract.	PM	0.359	0.163	144/48/48	2045.6	11330	6	-
Pref. att. 20k	Init. attract.	PM	0.363	0.151	144/48/48	20441	90412	6	-

backpropagation but instead save their decompositions, similar to the transport plan $\bar{\boldsymbol{P}}_{\text{Nys}} = \bar{\boldsymbol{P}}_U \bar{\boldsymbol{P}}_W$. We can then compute backpropagation in time $\mathcal{O}(nl^2)$ by applying the sums over i and j in the right order. We thus obtain $\mathcal{O}(nl^2 + l^3)$ overall runtime for backpropagation. □

D Choosing LSH Neighbors and Nyström Landmarks

We focus on two LSH methods for obtaining near neighbors. Cross-polytope LSH (Andoni et al., 2015) uses a random projection matrix $\boldsymbol{R} \in \mathbb{R}^{d \times b/2}$ with the number of hash buckets b, and then decides on the hash bucket via $h(\boldsymbol{x}) = \arg\max([\boldsymbol{x}^T \boldsymbol{R} \,\|\, -\boldsymbol{x}^T \boldsymbol{R}])$, where $\|$ denotes concatenation. K-means LSH computes k-means and uses the clusters as hash buckets.

We further improve the sampling probabilities of cross-polytope LSH via the AND-OR construction. In this scheme we calculate $B \cdot r$ hash functions, divided into B sets (hash bands) of r hash functions each. A pair of points is considered as neighbors if any hash band matches completely. K-means LSH does not work well with the AND-OR construction since its samples are highly correlated. For large datasets we use hierarchical k-means instead (Paulevé et al., 2010; Nistér & Stewénius, 2006).

Since the graph transport network (GTN) uses the L_2 distance between embeddings we use (hierarchical) k-means LSH and k-means Nyström in both sparse OT and LCN-OT. For embedding alignment we use cross-polytope LSH for sparse OT since similarities are measured via the dot product. For LCN-OT we found that using k-means LSH works better with Nyström using k-means++ sampling than cross-polytope LSH. This is most likely due to a better alignment between LSH samples and Nyström. We convert the cosine similarity to a distance via $d_{\cos} = \sqrt{1 - \frac{\boldsymbol{x}_{\text{p}}^T \boldsymbol{x}_{\text{q}}}{\|\boldsymbol{x}_{\text{p}}\|_2 \|\boldsymbol{x}_{\text{q}}\|_2}}$ (Berg et al., 1984) to use k-means with dot product similarity. Note that this is actually based on cosine similarity, not the dot product. Due to the balanced nature of OT we found this more sensible than maximum inner product search (MIPS). For both experiments we also experimented with uniform and recursive RLS sampling but found that the above mentioned methods work better.

E Implementational Details

Our implementation runs in batches on a GPU via PyTorch (Paszke et al., 2019) and PyTorch Scatter (Fey & Lenssen, 2019). To avoid over- and underflows we use log-stabilization throughout, i.e. we save all values in log-space and compute all matrix-vector products and additions via the log-sum-exp trick. Since the matrix $\boldsymbol{A}$ is small we compute its inverse using double precision to improve stability. Surprisingly, we did not observe any benefit from using the Cholesky decomposition or not calculating $\boldsymbol{A}^{-1}$ and instead solving the equation $\boldsymbol{B} = \boldsymbol{A}\boldsymbol{X}$ for $\boldsymbol{X}$.

We use 3 layers and an embedding size $H_{\text{N}} = 32$ for GTN. The MLPs use a single hidden layer, biases and LeakyReLU non-linearities. The single-head MLP uses an output size of $H_{\text{N, match}} = H_{\text{N}}$ and a hidden embedding size of $4H_{\text{N}}$, i.e. the same as the concatenated node embedding, and the multi-head MLP uses a hidden embedding size of H_{N}. To stabilize initial training we scale the node embeddings by $\frac{\bar{d}}{\bar{N}\sqrt{H_{\text{N, match}}}}$ directly before calculating OT. $\bar{d}$ denotes the average graph distance in the training set, $\bar{N}$ the average number of nodes per graph, and $H_{\text{N, match}}$ the matching embedding size, i.e. 32 for single-head and 128 for multi-head OT.

Table 6: Hyperparameters for the Linux dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	32	5×10^{-4}	-
GMN	1×10^{-4}	20	3	64	0	-
GTN, 1 head	0.01	1000	3	32	1×10^{-6}	1.0
8 OT heads	0.01	1000	3	32	1×10^{-6}	1.0
Balanced OT	0.01	1000	3	32	1×10^{-6}	2.0

Table 7: Hyperparameters for the AIDS dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	32	5×10^{-4}	-
SimGNN	1×10^{-3}	1	3	32	0.01	-
GMN	1×10^{-2}	128	3	32	0	-
GTN, 1 head	0.01	100	3	32	5×10^{-3}	0.1
8 OT heads	0.01	100	3	32	5×10^{-3}	0.075
Balanced OT	0.01	100	3	32	5×10^{-3}	0.1
Nyström	0.015	100	3	32	5×10^{-3}	0.2
Multiscale	0.015	100	3	32	5×10^{-3}	0.2
Sparse OT	0.015	100	3	32	5×10^{-3}	0.2
LCN-OT	0.015	100	3	32	5×10^{-3}	0.2

F GRAPH DATASET GENERATION AND EXPERIMENTAL DETAILS

The dataset statistics are summarized in Table 5. Each dataset contains the distances between all graph pairs in each split, i.e. $10\,296$ and 1128 distances for preferential attachment. The AIDS dataset was generated by randomly sampling graphs with at most 30 nodes from the original AIDS dataset (Riesen & Bunke, 2008). Since not all node types are present in the training set and our choice of GED is permutation-invariant w.r.t. types, we permuted the node types so that there are no previously unseen types in the validation and test sets. For the preferential attachment datasets we first generated 12, 4, and 4 undirected "seed" graphs (for train, val, and test) via the initial attractiveness model with randomly chosen parameters: 1 to 5 initial nodes, initial attractiveness of 0 to 4 and $1/2N$ and $3/2N$ total nodes, where N is the average number of nodes (20, 200, 2000, and $20\,000$). We then randomly label every node (and edge) in these graphs uniformly. To obtain the remaining graphs we edit the "seed" graphs between $N/40$ and $N/20$ times by randomly adding, type editing, or removing nodes and edges. Editing nodes and edges is 4x and adding/deleting edges 3x as likely as adding/deleting nodes. Most of these numbers were chosen arbitrarily, aiming to achieve a somewhat reasonable dataset and process. We found that the process of first generating seed graphs and subsequently editing these is crucial for obtaining meaningfully structured data to learn from. For the GED we choose an edit cost of 1 for changing a node or edge type and 2 for adding or deleting a node or an edge.

Table 8: Hyperparameters for the preferential attachment GED dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	64	1×10^{-3}	-
SimGNN	1×10^{-3}	4	3	32	0	-
GMN	1×10^{-4}	20	3	64	0	-
GTN, 1 head	0.01	100	3	32	5×10^{-4}	0.2
8 OT heads	0.01	100	3	32	5×10^{-3}	0.075
Balanced OT	0.01	100	3	32	5×10^{-4}	0.2
Nyström	0.02	100	3	32	5×10^{-5}	0.2
Multiscale	0.02	100	3	32	5×10^{-5}	0.2
Sparse OT	0.02	100	3	32	5×10^{-5}	0.2
LCN-OT	0.02	100	3	32	5×10^{-5}	0.2

Table 9: Runtimes (ms) of Sinkhorn approximations for EN-DE embeddings at different dataset sizes. Full Sinkhorn scales quadratically, while all approximationes scale at most linearly with the size. Sparse approximations are 2-4x faster than low-rank approximations. Note that similarity computation time (Sim.) primarily depends on the LSH/Nyström method, not the OT approximation.

	$N = 10000$		$N = 20000$		$N = 50000$	
	Sim.	OT	Sim.	OT	Sim.	OT
Full Sinkhorn	8	2950	29	11 760	OOM	OOM
Multiscale OT	90	48	193	61	521	126
Nyström Skh.	29	135	41	281	79	683
Sparse Skh.	42	46	84	68	220	137
LCN-Sinkhorn	101	116	242	205	642	624

We represent node and edge types as one-hot vectors. We train all models except SiamMPNN (which uses SGD) and GTN on Linux with the Adam optimizer and mean squared error (MSE) loss for up to 300 epochs and reduce the learning rate by a factor of 10 every 100 steps. On Linux we train for up to 1000 epochs and reduce the learning rate by a factor of 2 every 100 steps. We use the parameters from the best epoch based on the validation set. We choose hyperparameters for all models using multiple steps of grid search on the validation set, see Tables 6 to 8 for the final values. We use the originally published result of SimGNN on Linux and thus don't provide its hyperparameters. GTN uses 500 Sinkhorn iterations. We obtain the final entropy regularization parameter from λ_{base} via $\lambda = \lambda_{\text{base}} \frac{\bar{d}}{\bar{N}} \frac{1}{\log N}$, where $\bar{d}$ denotes the average graph distance and $\bar{N}$ the average number of nodes per graph in the training set. The factor $\bar{d}/\bar{N}$ serves to estimate the embedding distance scale and $1/\log N$ counteracts the entropy scaling with $N \log N$. Note that the entropy regularization parameter was small, but always far from 0, which shows that entropy regularization actually has a positive effect on learning. On the pref. att. 200 dataset we use no L_2 regularization, $\lambda_{\text{base}} = 0.5$, and a batch size of 200. For pref. att. 2k we use $\lambda_{\text{base}} = 2$ and a batch size of 20 for full Sinkhorn and 100 for LCN-OT. For pref. att. 20k we use $\lambda_{\text{base}} = 50$ and a batch size of 4. λ_{base} scales with graph size due to normalization of the PM kernel.

For LCN-OT we use roughly 10 neighbors for LSH (20 k-means clusters) and 10 k-means landmarks for Nyström on pref. att. 200. We double these numbers for pure Nyström Sinkhorn, sparse OT, and multiscale OT. For pref. att. 2k we use around 15 neighbors ($10 \cdot 20$ hierarchical clusters) and 15 landmarks and for pref. att. 20k we use roughly 30 neighbors ($10 \cdot 10 \cdot 10$ hierarchical clusters) and 20 landmarks. The number of neighbors for the 20k dataset is higher and strongly varies per iteration due to the unbalanced nature of hierarchical k-means. This increase in neighbors and landmarks and PyTorch's missing support for ragged tensors largely explains LCN-OT's deviation from perfectly linear runtime scaling.

We perform all runtime measurements on a compute node using one Nvidia GeForce GTX 1080 Ti, two Intel Xeon E5-2630 v4, and 256GB RAM.

G RUNTIMES

Table 9 compares the runtime of the full Sinkhorn distance with different approximation methods using 40 neighbors/landmarks. We separate the similarity matrix computation from the optimal transport computation (Sinkhorn iterations), since the former primarily depends on the LSH and Nyström methods we choose. We observe a 2-4x speed difference between sparse (multiscale OT and sparse Sinkhorn) and low-rank approximations (Nyström Sinkhorn and LCN-Sinkhorn). In Fig. 5 we observe that this runtime gap stays constant independent of the number of neighbors/landmarks, i.e. the relative difference decreases as we increase the number of neighbors/landmarks. This gap could either be due to details in low-level CUDA implementations and hardware or the fact that low-rank approximations require 2x as many multiplications for the same number of neighbors/landmarks. In either case, both Table 9 and Fig. 5 show that the runtimes of all approximations scale linearly both in the dataset size and the number of neighbors and landmarks, while full Sinkhorn scales quadratically.

We furthermore investigate whether GTN with approximate Sinkhorn indeed scales log-linearly with the graph size by generating preferential attachment graphs with 200, 2000, and 20 000 nodes

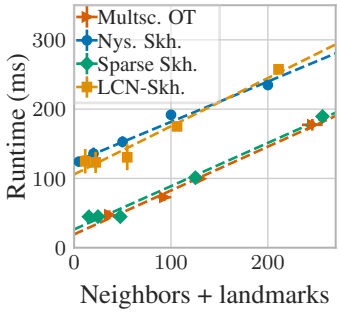

Figure 5: Runtime scales linearly with the number of neighbors/landmarks for all Sinkhorn approximation methods.

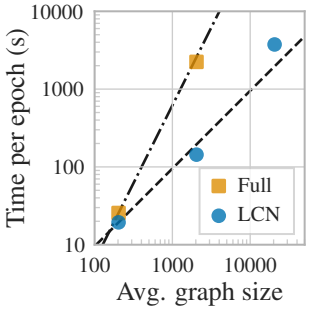

Figure 6: Log-log runtime per epoch for GTN with full Sinkhorn and LCN-Sinkhorn. LCN-Sinkhorn scales almost linearly with graph size while sustaining similar accuracy.

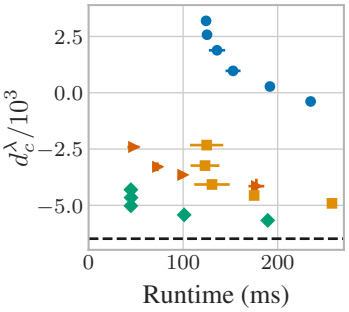

Figure 7: Sinkhorn distance approximation at different runtimes (varied via the number of neighbors/landmarks). The dashed line denotes the true Sinkhorn distance. Sparse Sinkhorn consistently performs best.

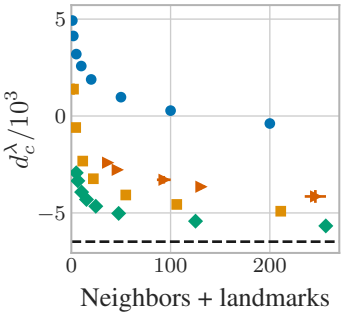

Figure 8: Sinkhorn distance approximation for varying computational budget. The dashed line denotes the true Sinkhorn distance. Sparse Sinkhorn consistently performs best, with LCN-Sinkhorn coming in second.

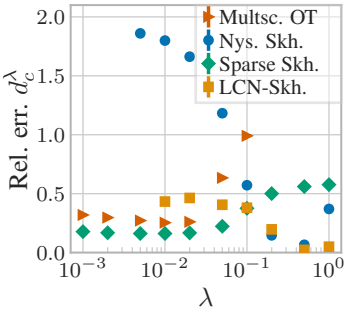

Figure 9: Sinkhorn distance approximation for varying entropy regularization λ at constant computational budget. Sparse Sinkhorn performs best for low λ and LCN-Sinkhorn for high λ.

($\pm 50\,\%$). We use the Pyramid matching (PM) kernel (Nikolentzos et al., 2017) as prediction target. Fig. 6 shows that the runtime of LCN-Sinkhorn scales almost linearly (dashed line) and regular full Sinkhorn quadraticly (dash-dotted line) with the number of nodes, despite both achieving similar accuracy and LCN using slightly more neighbors and landmarks on larger graphs to sustain good accuracy. Full Sinkhorn went out of memory for the largest graphs.

H Distance Approximation

Figs. 7 and 8 show that for the chosen $\lambda = 0.05$ sparse Sinkhorn offers the best trade-off between computational budget and distance approximation, with LCN-Sinkhorn and multiscale OT coming in second. Note that d_c^λ can be negative due to the entropy offset. This picture changes as we increase the regularization. For higher regularizations LCN-Sinkhorn is the most precise at constant computational budget (number of neighbors/landmarks), as shown in Fig. 9. Note that the crossover points in this figure roughly coincide with those in Fig. 4. Keep in mind that in most cases the OT plan is more important than the raw distance approximation, since it determines the training gradient and tasks like embedding alignment don't use the distance at all. This becomes evident in the fact that sparse Sinkhorn achieves a better distance approximation than LCN-Sinkhorn but performs worse in both downstream tasks investigated in Sec. 7.