Large Scale Graph Learning from Smooth Signals

Vassilis Kalofolias*
Signal Processing Laboratory 2 (LTS2 - EPFL)
Ecole Polytechnique Fédérale de Lausanne
Station 11, 1015 Lausanne, Switzerland

Nathanaël Perraudin*
Swiss Data Science Center (SDSC - ETHZ)
Eidgenössische Technische Hochschule Zürich
Universitätstrasse 25, 8006 Zürich, Switzerland

Abstract

Graphs are a prevalent tool in data science, as they model the inherent structure of the data. They have been used successfully in unsupervised and semi-supervised learning. Typically they are constructed either by connecting nearest samples, or by learning them from data, solving an optimization problem. While graph learning does achieve a better quality, it also comes with a higher computational cost. In particular, the current state-of-the-art model cost is $\mathcal{O}\left(n^2\right)$ for n samples. In this paper, we show how to scale it, obtaining an approximation with leading cost of $\mathcal{O}\left(n \log(n)\right)$, with quality that approaches the exact graph learning model. Our algorithm uses known approximate nearest neighbor techniques to reduce the number of variables, and automatically selects the correct parameters of the model, requiring a single intuitive input: the desired edge density.

1 Introduction

Graphs are an invaluable tool in data science, as they can capture complex structures inherent in seemingly irregular high-dimensional data. While classical applications of graphs include data embedding, manifold learning, clustering and semi-supervised learning [34, 2, 29], they were later used for regularizing various machine learning models, for example for classification, sparse coding, matrix completion, or PCA [32, 33, 16, 28].

More recently, the ability of graphs to capture the structure of seemingly unstructured data, drew the attention of the deep learning community. While convolutional neural networks (CNNs) were highly successful for learning image representations, it was not obvious how to generalize them for high dimensional irregular domains, were standard convolution is not applicable. Graphs gave the solution to bridge the gap between irregular data and CNNs through the generalization of convolutions on graph structures [6, 17, 22, 20]. While it is clear that the graph quality is important in such applications [10, 6], the question of how to optimally construct a graph remains an open problem.

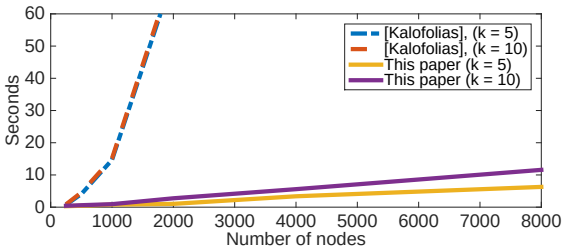

Figure 1: Time needed to learn a graph between different words using a word2vec representation. By k we denote the average number of edges per node.

The quality of the graph arguably plays an important role in the success of such methods. The first applications mostly used weighted k-nearest neighbors graphs (k-NN) [34, 2, 29], but the last few years more sophisticated methods of *learning* graphs from data were proposed. Today, *graph learning*, or *network inference*, has become an important problem itself [30, 5, 12, 19, 11, 7, 15].

Unfortunately, graph learning is computationally too costly for large-scale applications that often need graphs between millions of samples. The current state of the art learning models for weighted undirected graphs [7, 15] cost $\mathcal{O}\left(n^2\right)$ per iteration for n nodes, while previous solutions are even more expensive. Furthermore, they come with parameters that control sparsity, and choosing the correct ones adds an extra burden making them prohibitive for applications with more than a few thousands of nodes.

*Part of this work is in the thesis of the first author [14]. Part of this work was done while the second author was at LTS2.

Large-scale applications can only resort to approximate nearest neighbor methods (A-NN), e.g. [25, 21], that run with a leading cost of $\mathcal{O}(n \log(n))$. This is remarkably low if we consider that computing a simple k-NN graph, or even the pairwise distance matrix between all samples costs $\mathcal{O}(n^2)$. However, the quality of A-NN should be expected to be slightly worse than the quality of k-NN, that is already not as good as if we would learn the graph from data.

In this paper, we propose the first scalable graph learning method, with the same leading cost as A-NN, and with quality that approaches state-of-the-art graph learning. Our method leverages A-NN graphs to effectively reduce the number of variables, and the state-of-the-art graph learning model by Kalofolias [15] in order to achieve the best of both worlds: low cost and good quality. In Figure 1 we illustrate the advantage of our solution compared to the current state-of-the-art. Note that while the standard model costs the same regardless of the graph density k, our solution benefits from the desired graph sparsity to reduce computation.

One of our key contributions is to provide *a method to automatically select the parameters of the model* by Kalofolias [15] given a desired graph sparsity level. Like in k-NN, the user can choose the number of neighbors k, *without performing grid search* over two parameters. Using our scheme, we can learn a 1-million-nodes graph with a desired sparsity level on a desktop in half an hour, with a simple Matlab implementation.

2 Graph Learning from Smooth Signals

A widely used assumption for data residing on graphs is that values change smoothly across adjacent nodes. The smoothness of a set of vectors $x_1, \dots, x_n \in \mathbb{R}^d$ on a given weighted undirected graph is usually quantified by the *Dirichlet energy* [1]

$$\frac{1}{2} \sum_{i,j} W_{ij} \|x_i - x_j\|^2 = \operatorname{tr}\left(X^\top L X\right), \tag{1}$$

where $W_{ij} \in \mathbb{R}_+$ denotes the weight of the edge between nodes i and j, $L = D - W$ is the graph Laplacian, and $D_{ii} = \sum_j W_{ij}$ is the diagonal weighted degree matrix.

Regularization using the Dirichlet energy has been used extensively in machine learning, to enhance for example image processing [9, 33], non-negative matrix factorization [4, 3], matrix completion [16], or principal component analysis (PCA) [13, 27, 28].

In these methods the Dirichlet energy is minimized w.r.t. matrix X given the Laplacian L. On the other hand, we can learn a graph under the assumption that X is smooth on it, by minimizing the same energy w.r.t. L, when X is given.

The first works for graph learning focused on learning the weights of a fixed k-nearest neighbor pattern [30], learning a binary pattern [12] or the whole adjacency matrix [5]. The idea of minimizing Dirichlet energy for graph learning is more recent [19, 11, 7, 15]. In the last work, Kalofolias [15] proposed a unified model for learning a graph from smooth signals, that reads as follows:

$$\min_{W \in \mathcal{W}} \|W \circ Z\|_{1,1} + f(W). \tag{2}$$

Here, $Z_{ij} = \|x_i - x_j\|^2$, $\circ$ denotes the Hadamard product, and the first term is equal to $\operatorname{tr}\left(X^\top L X\right)$. The optimization is over the set $\mathcal{W}$ of valid adjacency matrices (non-negative, symmetric, with zero diagonal).

The role of matrix function $f(W)$ is to prevent W from obtaining a trivial zero value, control sparsity, and impose further structure, depending on the data and the application. Kalofolias obtained state-of-the-art results using

$$f(W) = -\alpha \mathbf{1}^\top \log(W\mathbf{1}) + \frac{\beta}{2}\|W\|_F^2, \tag{3}$$

where $\mathbf{1} = [1, \dots 1]^\top$. We will call this the *log model*. The previous state of the art was proposed by Hu et al. [11] and by Dong et al. [7], using

$$f(W) = \alpha\|W\mathbf{1}\|^2 + \alpha\|W\|_F^2 + \mathbb{1}\left\{\|W\|_{1,1} = n\right\}, \tag{4}$$

where $\mathbb{1}\{\text{condition}\} = 0$ if condition holds, ∞ otherwise. In the sequel we call this the ℓ_2 *model*. Since $W\mathbf{1}$ is the node degrees' vector, the *log* model (3) prevents the formation of disconnected nodes due to the logarithmic barrier, while the ℓ_2 model (4) controls sparsity by penalizing large degrees due to the first term. The choice therefore depends on the data and application in question.

3 Constrained edge pattern

In traditional graph learning, all $\binom{n}{2}$ possible edges between n nodes are considered, that results in a cost of at least $\mathcal{O}(n^2)$ computations per iteration. Often, however, we need graphs with a roughly fixed number of edges per node, like in k-NN graphs. It is natural to then ask ourselves whether the cost of graph learning can be reduced, reflecting the final desired graph sparsity.

In fact, the original problem (2) for the log model (3) can be solved efficiently when a constrained set $\mathcal{E}^{\text{allowed}} \subseteq \{(i,j) : i < j\}$ of allowed edges is known a priori. In that case, it suffices to solve the modified

problem

$$\underset{W \in \widetilde{\mathcal{W}}}{\text{minimize}} \ \|W \circ Z\|_{1,1} - \alpha \mathbf{1}^\top \log(W\mathbf{1}) + \frac{\beta}{2}\|W\|_F^2, \quad (5)$$

where we optimize in the constrained set of adjacency matrices $W \in \widetilde{\mathcal{W}}$. After reducing the set of edges to $\mathcal{E}^{\text{allowed}}$, it suffices to solve the modified problem (5) Following Kalofolias [15], we can rewrite the problem as

$$\underset{\widetilde{w}}{\text{minimize}} \ f_1(\widetilde{w}) + f_2(K\widetilde{w}) + f_3(\widetilde{w}), \quad (6)$$

with

$$\begin{aligned} f_1(\widetilde{w}) &= \mathbb{1}\{\tilde{w} \geq 0\} + 2\tilde{w}^\top \tilde{z}, \\ f_2(v) &= -\alpha \mathbf{1}^\top \log(v), \\ f_3(\widetilde{w}) &= \beta \|\widetilde{w}\|^2, \text{ with } \zeta = 2\beta, \end{aligned}$$

where ζ is the Lipschitz constant of f_3. Note that we gather all free parameters of the adjacency matrix $\widetilde{W} \in \widetilde{\mathcal{W}}_m$ in a vector $\widetilde{w} \in \widetilde{\mathcal{W}}_v$ of size only $|\mathcal{E}^{\text{allowed}}|$, that is, the number of allowed edges, each counted only once. Accordingly, in $\tilde{z} = z(\mathcal{E}^{\text{allowed}})$ we only keep the corresponding pairwise distances from matrix Z. The linear operator $K = \widetilde{S} = S(:, \mathcal{E}^{\text{allowed}})$ is also modified, keeping only the columns corresponding to the edges in $\mathcal{E}^{\text{allowed}}$.

In this form, the problem can be solved by the primal dual techniques by Komodakis and Pesquet [18]. The cost of the dual step, operating on the dual variable v (degrees vector) remains $\mathcal{O}(n)$. However, the cost of the primal step, as well as the cost of applying the modified operator $\widetilde{S}$ in order to exchange between the primal and dual spaces is $\mathcal{O}\left(\mathcal{E}^{\text{allowed}}\right)$ instead of $\mathcal{O}\left(n^2\right)$ of the initial algorithm 1 by Kalofolias [15], reducing the overall complexity.

In some cases, a pattern of allowed edges $\mathcal{E}^{\text{allowed}}$ can be induced by constraints of the model, for example sensor networks only assume connections between geographically nearby sensors. In most applications, however, a constrained set is not known beforehand, and we need to approximate the edge support of the final learned graph in order to reduce the number of variables. To this end, we propose using approximate nearest neighbors graphs to obtain a good approximation. While computing a k-NN graph needs $\mathcal{O}\left(n^2 d\right)$ computations, *approximate nearest neighbors* (A-NN) algorithms [23, 24, 25, 21] offer a good compromise between accuracy and speed. Specifically, A-NN methods scale gracefully with the number of nodes n, the fastest ones having an overall complexity of $\mathcal{O}(n \log(n) d)$ for d-dimensional data.

When approximating the support of the final edges of a graph, we prefer to have false positives than false negatives. We thus start with an initial support with a larger cardinality than that of the desired final graph, and let the weight learning step automatically select which edges to set to zero. Precisely, we select a set $\mathcal{E}^{\text{allowed}}$ with cardinality $|\mathcal{E}^{\text{allowed}}| = \mathcal{O}(nkr)$, where k is the desired number of neighbors per node and r a small multiplicative factor. By setting the sparsity parameters correctly, the graph learning step will only keep the final $\mathcal{O}(nk)$ edges, setting the less important or wrong edges to zero. The bigger the factor r, the more freedom the learning algorithm has to select the right edges. If in the end of the graph learning it occurs that many nodes still have all allowed edges set to positive values, it is an indication that we should have given a more generous set of allowed edges.

3.1 Overall complexity

Asymptotically, the cost of learning a kr-A-NN graph is $\mathcal{O}(n \log(n) d)$ for a graph of n nodes and data with dimensionality d, while additionally learning the edge weights costs $\mathcal{O}(krn)$ per iteration. The overall complexity is therefore $\mathcal{O}(n \log(n) d) + \mathcal{O}(nkrI)$ for I iterations. For large n, the dominating cost is asymptotically the one of computing the A-NN and not the cost of learning the weights on the reduced set.

4 Automatic parameter selection

A major problem of models (3) and (4) is the choice of meaningful parameters α, β, as performing grid search increases significantly the computational complexity. In this section we show how this burden can be completely avoided for model (3). We do this in two steps. First, we show that the sparsity depends effectively on a single parameter, and second, we propose a method to set this parameter from the desired connectivity. Our method is based on predicting exactly the number of edges of any given node for a given parameter value, if we relax the symmetricity constraint of W.

4.1 Reduction to a single optimization parameter

In [15, Proposition 2], it is argued that model (3) effectively has only one parameter changing the shape of the edges, the second changing the magnitude. We re-formulate this claim as follows.

Proposition 1. *Let $W^*(Z, \alpha, \beta)$ denote the solution of model (3) for input distances Z and parameters $\alpha, \beta > 0$. Then the same solution can be obtained with fixed parameters $\alpha = 1$ and $\beta = 1$, by multiplying the input distances by $\theta = \frac{1}{\sqrt{\alpha\beta}}$ and the resulting edges by*

$\delta = \sqrt{\frac{\alpha}{\beta}}$:

$$W^*(Z, \alpha, \beta) = \sqrt{\frac{\alpha}{\beta}} W^*\left(\frac{1}{\sqrt{\alpha\beta}} Z, 1, 1\right) = \delta W^*(\theta Z, 1, 1). \tag{7}$$

Proof. *Apply [15, Prop. 2], with* $\gamma = \sqrt{\frac{\alpha}{\beta}}$ *and divide all operands by the same constant* $\sqrt{\alpha\beta}$.

Proposition 1 shows that the two parameter spaces (α, β) and (θ, δ) are equivalent. While the first one is convenient to define (3), the second one makes the sparsity analysis and the application of the model simpler. In words, all graphs that can be learned by model (3) can be equivalently computed by multiplying the initial distances Z by $\theta = \frac{1}{\sqrt{\alpha\beta}}$, using them to learn a graph with fixed parameters $\alpha = \beta = 1$, and multiplying all resulting edges by the same constant $\sqrt{\alpha\beta}$.

This property allows us to tune a single parameter θ that controls the sparsity of the solution instead of α and β. The larger θ is, the greater the pairwise distances between nodes and therefore the sparser the edges, as connections between distant nodes are penalized. This result is also important for graph-based applications that are multiplicative scale invariant. In the latter, multiplying all edges by the same constant does not change the functionality of the graph. In other cases, we want to explicitly normalize the graph to a specific size, for example setting $\|W\|_{1,1} = n$ as in [11], or making sure that $\lambda_{\max} = 1$. In these cases, the user needs only search for the value of θ that will obtain the desired sparsity and then multiply the graph with the appropriate constant.

4.2 Setting the remaining regularization parameter

The last step for automatizing parameter selection is to find a relation between θ and the desired sparsity (the average number of neighbors per node). We first analyze the sparsity level with respect to θ for each node independently. Once the independent problems are well characterized, we propose an empirical solution to obtain a global value of θ providing approximately the desired sparsity level.

4.2.1 Sparsity analysis for one node

In order to analyze the sparsity of the graphs obtained by the model (3), we take one step back and drop the symmetricity constraint. The problem becomes separable and we can focus on only one node. Keeping only one column w of matrix W, we arrive to the simpler optimization problem

$$\min_{w \in \mathbb{R}_+^n} \quad \theta w^\top z - \log(w^\top \mathbf{1}) + \frac{1}{2}\|w\|_2^2. \tag{8}$$

The above problem also has only one parameter θ that controls sparsity, so that larger values of θ yield sparser solutions w^*. Furthermore, *it enjoys an analytic solution if we sort the elements of* z, as we prove with the next theorem.

Theorem 1. *Suppose that the input vector* z *is sorted in ascending order. Then the solution of problem* (8) *has the form*

$$w^* = \max(0, \lambda^* - \theta z) = [\lambda^* - \theta z_{\mathcal{I}}; \mathbf{0}], \tag{9}$$

with

$$\lambda^* = \frac{\theta b_k + \sqrt{\theta^2 b_k^2 + 4k}}{2k}.$$

The set $\mathcal{I} = \{1, \ldots, k\}$ *corresponds to the indices of the* k *smallest distances* z_i *and* b_k *is the cumulative sum of the smallest* k *distances in* z, $b_k = \sum_{i=1}^k z_i$.

We provide the proof of Theorem 1 after presenting certain intermediate results. In order to solve Problem (8) we first introduce a slack variable l for the inequality constraint, so that the KKT optimality conditions are

$$\theta z - \frac{1}{w^\top \mathbf{1}} + w - l = 0, \tag{10}$$

$$w \geq 0, \tag{11}$$

$$l \geq 0, \tag{12}$$

$$l_i w_i = 0, \forall i. \tag{13}$$

the optimum of w can be revealed by introducing the term $\lambda^* = \frac{1}{w^{*\top}\mathbf{1}}$ and rewrite (10) as

$$w^* = \lambda^* - \theta z + l. \tag{14}$$

Then, we split the elements of w in two sets, $\mathcal{A}$ and $\mathcal{I}$ according to the activity of the inequality constraint (11), so that $w_{\mathcal{I}} > \mathbf{0}$ (inactive) and $w_{\mathcal{A}} = \mathbf{0}$ (active). Note that at the minimum, the elements of w will also be sorted in a descending order so that $w^* = [w_{\mathcal{I}}^*; \mathbf{0}]$, according to Theorem 1. We first need a condition for an element of w^* to be positive, as expressed in the following lemma:

Lemma 1. *An element* w_i^* *of the solution* w^* *of problem* (8) *is in the active set* $\mathcal{A}$ *if and only if it corresponds to an element of* z_i *for which* $\theta z_i \geq \lambda^*$.

Proof. ($\Rightarrow$)*: If* w_i *is in the active set we have* $w_i = 0$ *and* $l_i \geq 0$*, therefore from eq.* (14) *we have* $\theta z_i - \lambda^* \geq 0$. ($\Leftarrow$)*: Suppose that there exists* $i \in \mathcal{I}$ *for which* $\theta z_i \geq \lambda^*$. *The constraint being inactive means that* $w_i^* > 0$. *From* (13) *we have that* $l_i = 0$ *and* (14) *gives* $w_i^* = \lambda^* - \theta z_i \leq 0$*, a contradiction.*

We are now ready to proceed to the proof of Theorem 1.

Proof (Theorem 1)**.** *As elements of* θz *are sorted in an ascending order, the elements of* $\lambda^* - \theta z$ *will be in a*

descending order. Furthermore, we know from Lemma 1 that all positive w_i^* will correspond to $\theta z_i < \lambda^*$. Then, supposing that $|\mathcal{I}| = k$ we have the following ordering:

$$-\theta z_1 \geq \cdots \geq -\theta z_k > -\lambda^* \geq -\theta z_{k+1} \geq \cdots \geq -\theta z_n \Rightarrow$$
$$\lambda^* - \theta z_1 \geq \cdots \geq \lambda^* - \theta z_k > 0 \geq \lambda^* - \theta z_{k+1} \geq \cdots \geq \lambda^* - \theta z_n.$$

In words, the vector $\lambda^* - \theta z$ will have sorted elements so that the first k are positive and the rest are non-positive. Furthermore, we know that the elements of l in the optimal have to be 0 for all inactive variables $w_{\mathcal{I}}^*$, therefore $w_{\mathcal{I}}^* = \lambda^* - \theta z_{\mathcal{I}}$. The remaining elements of w will be 0 by definition of the active set:

$$w^* = [\underbrace{\lambda^* - \theta z_1, \cdots, \lambda^* - \theta z_k}_{w_{\mathcal{I}}^*}, \underbrace{0, \cdots, 0}_{w_{\mathcal{A}}^*}].$$

What remains is to find an expression to compute λ^* for any given z. Keeping z ordered in ascending order, let the cumulative sum of z_i be $b_k = \sum_{i=1}^{k} z_i$. Then, from the definition of $\lambda^* = \frac{1}{w^{*\top}\mathbf{1}}$ and using the structure of w^* we have

$$\begin{aligned} & w^{*\top}\mathbf{1}\lambda^* = 1 \Rightarrow \\ & \left(k\lambda^* - \theta z_{\mathcal{I}}^\top \mathbf{1}\right)\lambda^* = 1 \Rightarrow \\ & k(\lambda^*)^2 - \theta b_k \lambda^* - 1 = 0, \end{aligned} \tag{15}$$

which has only one positive solution,

$$\lambda^* = \frac{\theta b_k + \sqrt{\theta^2 b_k^2 + 4k}}{2k}. \tag{16}$$

4.2.2 Parameter selection for the non-symmetric case

While Theorem 1 gives the form of the solution for a known k, the latter cannot be known a priori, as it is also a function of z. For this, we propose Algorithm 1 that solves this problem, simultaneously finding k and λ^* in $\mathcal{O}(k)$ iterations. This algorithm will be needed for automatically setting the parameters for the symmetric case of graph learning.

As k is the number of non-zero edges per node, it can be assumed to be a small number, like the ones used for k nearest neighbor graphs. Therefore, it is cheap to incrementally try all values of k starting from $k = 1$ until we find the correct one, as Algorithm 1 does. Once we try a value of k that satisfies $\lambda \in (\theta z_k, \theta z_{k+1}]$, all KKT conditions hold and we have found the solution to our problem. A similar algorithm has been proposed in [8] for projecting a vector on the probability simplex, that could be used for a similar analysis for the ℓ_2-degree constraints model (4).

Most interestingly, using the form of the solution given by Theorem 1 we can solve the reverse problem: *If we know the distances vector z and we want a solution w^* with exactly k non-zero elements, what should the parameter θ be?* The following theorem answers this question, giving intervals for θ as a function of k, z and its cumulative sum b.

Theorem 2. *Let $\theta \in \left(\frac{1}{\sqrt{kz_{k+1}^2 - b_k z_{k+1}}}, \frac{1}{\sqrt{kz_k^2 - b_k z_k}}\right]$, the result of problem (8) has exactly k non-zero elements.*

Proof. *See Appendix A.*

The idea of Theorem 2 is illustrated in the left part of Figure 2. For this figure we have used the distances between one image of MNIST and 999 other images. For any given sparsity level k we can know what are the intervals of the valid values of θ just by looking at the pairwise distances.

Algorithm 1 Solver of the one-node problem, eq. (8).

Input: $z \in \mathbb{R}_{*+}^n$ in ascending order, $\theta \in \mathbb{R}_{*+}$
$b_0 \leftarrow 0$ {Initialize cumulative sum}
for $i = 1, \ldots, n$ **do**
  $b_i \leftarrow b_{i-1} + z_i$ {Cumulative sum of z}
  $\lambda_i \leftarrow \frac{\sqrt{\theta^2 b_i^2 + 4i + \theta b_i}}{2i}$
  if $\lambda_i > \theta z_i$ **then**
    $k \leftarrow i - 1$
    $\lambda^* \leftarrow \lambda_k$
    $w^* \leftarrow \max\{0, \lambda^* - \theta z\}$ {k-sparse output}
    break
  end if
end for

4.2.3 Parameter selection for the symmetric case

In order to approximate the parameter θ that gives the desired sparsity of W, we use the above analysis for each row or column separately, omitting the symmetricity constraint. Then, using the arithmetic mean of the bounds of θ we obtain a good approximation of the behaviour of the full symmetric problem. In other words, to obtain a graph with approximately k edges per node, we propose to use the following intervals:

$$\begin{aligned} \theta_k \in & \left(\frac{1}{n}\sum_{j=1}^{n} \theta_{k,j}^{lower}, \quad \frac{1}{n}\sum_{j=1}^{n} \theta_{k,j}^{upper}\right] \\ = & \left(\sum_{j=1}^{n} \frac{1}{n\sqrt{k\hat{Z}_{k+1,j}^2 - B_{k,j}\hat{Z}_{k+1,i}}}, \right. \\ & \left. \sum_{j=1}^{n} \frac{1}{n\sqrt{k\hat{Z}_{k,j}^2 - B_{k,j}\hat{Z}_{k,j}}}\right], \end{aligned} \tag{17}$$

where $\hat{Z}$ is obtained by sorting each column of Z in increasing order, and $B_{k,j} = \sum_{i=1}^{k} \hat{Z}_{i,j}$. The above expression is the arithmetic mean over all minimum and maximum values of $\theta_{k,j}$ that would give a k-sparse result $W_{:,j}$ if we were to solve problem (8) for each

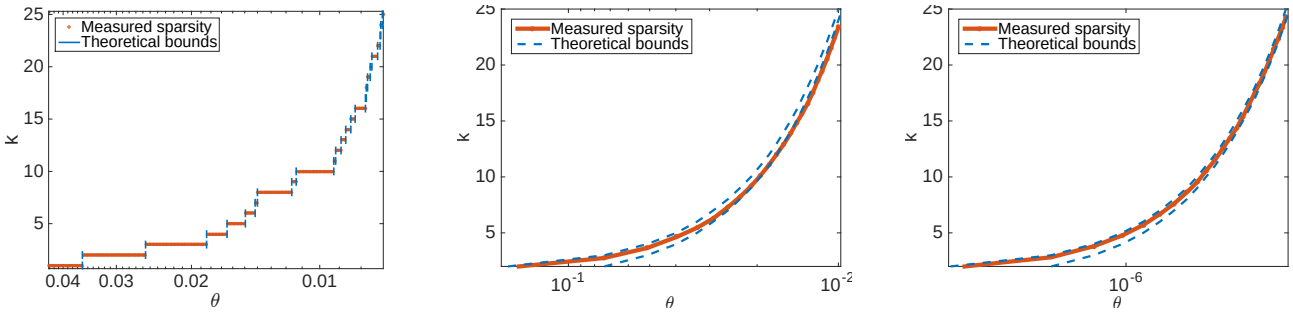

Figure 2: Theoretical bounds of θ for a given sparsity level on 1000 images from MNIST. **Left**: Solving (8) for only one column of Z. Theorem 2 applies and for each k gives the bounds of θ (blue). **Middle**: Solving (3) for the whole pairwise distance matrix Z of the same dataset. The bounds of eq. (17) (blue dashed line) are used to approximate the sparsity of the solution. The red line is the measured sparsity of the learned graphs from model (3). **Right**: Same for USPS dataset.

of columns separately, according to Theorem 2. Even though the above approach does not take into account the symmetricity constraints, it gives surprisingly good results in the vast majority of the cases.

5 Experiments

In our experiments we wish to answer questions regarding (1) the approximation quality of our large scale model, (2) the quality of our automatic parameter selection, (3) the benefit from learning versus A-NN for large scale applications and (4) the scalability of the model. We perform experiments using 4 real datasets, namely MNIST[1], USPS, US Census 1990[2], and Google's word2vec word representation[3]. Further datasets and experiments can be found in the Appendix. All timing results reported are computed using a standard desktop computer.

5.1 Effectiveness of automatic parameter selection

The plot in the middle of Figure 2 shows the approximate bounds obtained by eq. (17) for 1000 images of the MNIST dataset, in comparison to the actual sparsity obtained for each choice of θ (red line). We repeat the same experiment for images from the USPS dataset (right plot of Figure 2), and for more datasets in the Appendix B.2. Please note, that in the rare cases that the actual sparsity is outside the predicted bounds, we already have a good starting point for finding a good θ. Note also that small fluctuations in the density are tolerated, for example in k-NN graphs we always obtain results with slightly more than nk edges due to the fact that W is symmetric.

5.2 Graph between images of MNIST

To assess the quality of our model, we first learn the graph between the all 60000 images of the training set of MNIST. Using the image labels, we first show that, for the same average degree, the amount of wrong edges (connecting wrong classes) is much lower in learned graphs than in A-NN graphs. Second, we show that graph learning improves the performance of semi-supervised learning.

5.2.1 Edge quality

For graph learning, MNIST is an interesting dataset because it has a relatively uniform sampling between numbers of different labels, **except for the digits "1" that are more densely sampled**. In other words, the intra-class distances between the digits "1" is in average much smaller than for other digits (see also Appendix B.1). This affects the results of graph learning as we see in the sequel.

We analyze the quality of the connections retrieved by the large-scale log and ℓ_2 models and we compare with an A-NN graph as a baseline. Each graph is normalized so as to have equal total "connectivity" $\|W\|_{1,1} = 1$. In Figure 3, we plot the histograms of the connectivity ratio that is spent for edges between images of each label. The last bar corresponds to the total ratio of connectivity wasted on wrong edges (edges between any pair of images with different labels).

Given the almost uniform distribution of the labels

[1] We use the 60000 images of the MNIST training dataset.

[2] The dataset is available at the UCI machine learning repository and consists of approximately 2.5 million samples of 68 features. `https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990)`

[3] The dataset consists of the 10′000 most used words in English (`https://research.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html`). It uses the Google word2vec features (`https://code.google.com/archive/p/word2vec/`).

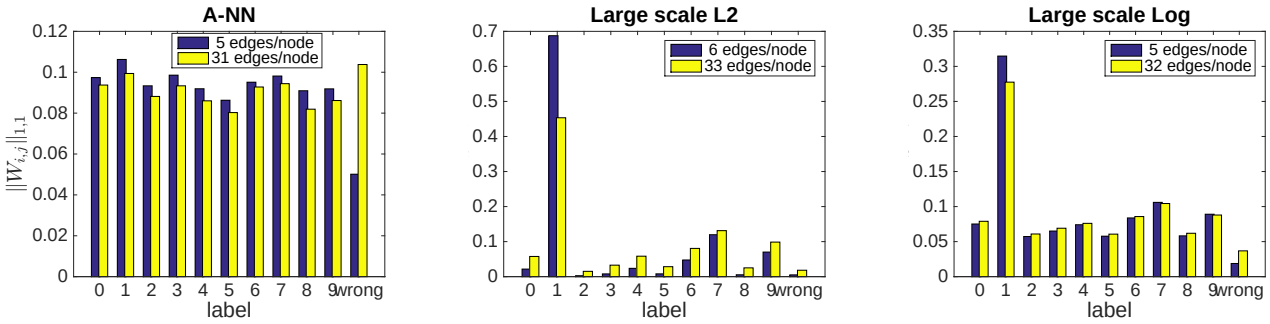

Figure 3: Connectivity across different classes of MNIST (60000 nodes). The graph is normalized so that $\|W\|_{1,1} = 1$. We measure the percentage of the total weight for connected pairs of each label. The last columns correspond to the total of the wrong edges, between images of different labels. **Left**: The A-NN graph does not take into account enough the distances and connects uniformly all digits. For 30 edges per node, it introduces many wrong edges. **Middle**: The ℓ_2 model (4) fails to connect digits with larger distance ("2"s and "8"s) even for 30 edges per node graphs. **Right**: While being sensitive to the distance between different pairs, the log model (5) does not neglect to connect any cluster, even for very sparse graphs of 5 edges per node.

and distances in the MNIST dataset, we expect to see a similar behavior in the connectivity between the images, with a small variation due to different average distances (with label "1" more connected). Ideally, the wrong edges should be minimal.

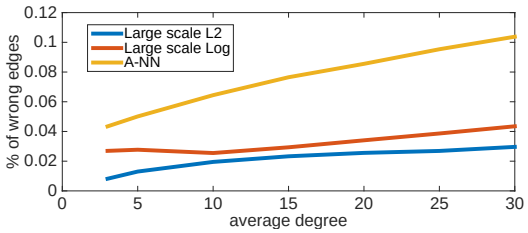

Figure 4: Edge accuracy of large scale algorithms for the full MNIST training dataset.

The ℓ_2 model (4) generally fails to give consistent connectivities across different labels, being too sensitive to the average distances and assigns the vast majority of its connectivity only to the label "1" that has the smallest intra-label image distance. This effect becomes smaller when more dense graphs (30 edges per node, yellow bars) are sought. On the other hand, the log model does not suffer from this problem and gives consistent connectivities without depending too much on the sampling density. Similarly for a k-NN graph, the connectivity is constant across all labels. Note, however, that we use it here only as a baseline, and that a weighted k-NN graph would also follow the intra-label distances depending on its weighting scheme.

Figure 4 summarizes the number of wrong edges with respect to k. Clearly, learned graphs have much less wrong edges at any given graph density. The ℓ_2 model has the least number of wrong edges. However, as shown in Figure 3, the majority of its edges connects the digit one and digits such as $2, 3, 8$ are almost not connected. On the contrary, the log model is much less affected from this drawback, while keeping a relatively low amount of wrong edges.

5.2.2 Semi-supervised learning

Using the graph learned in the previous subsection, we perform a semi-supervised learning experiment on the MNIST dataset. We use only 1% of the labels as observations, and apply label propagation [34] to predict the rest of the labels. The results are plotted in Figure 5. Again, the log model performs best. To have a fair comparison we used the best weighting scheme for k-NN, while the quality of the label propagation did not vary significantly by changing weighting schemes.

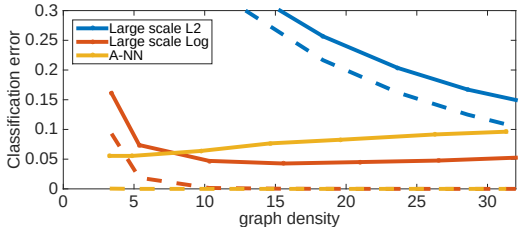

Figure 5: Digit classification error with 1% known labels (MNIST train dataset, 60000 nodes). Dashed lines represent the rates of disconnected nodes

Given the performance of the A-NN graph, one might wonder why pay the additional cost of learning a graph only for a small improvement in classification. Note, however, that the additional cost is not significant. Asymptotically, the cost of learning an A-NN graph

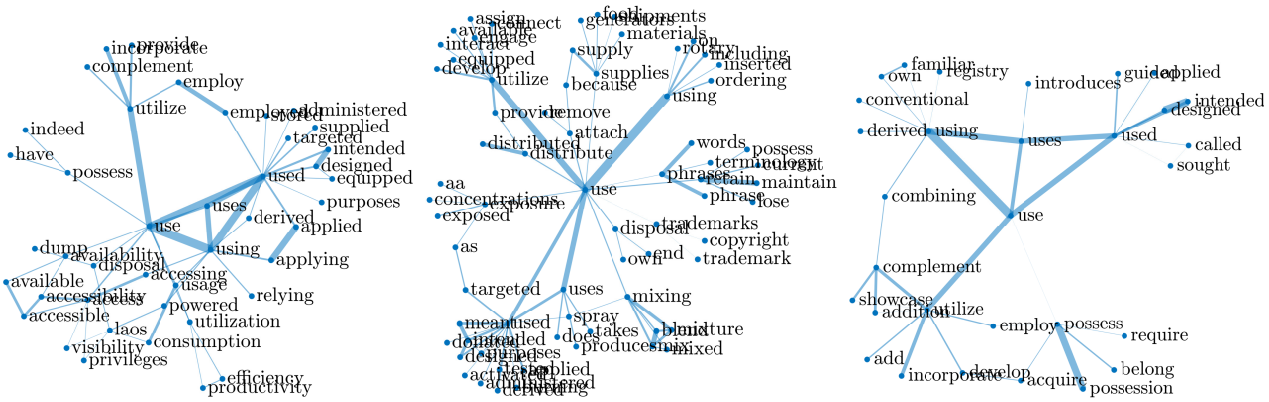

Figure 6: A realization of the 2-hop sub graph centered around the word "use". **Center:** The nodes of the approximate graph NN have the tendency to be connected to further points because of its randomness. **Left:** The k-NN graph does not suffer from this drawback and resemble the learned graph (**right**) but with a higher computational cost. The average number of neighbors per node is 5.0, 5.4 and 5.7 respectively for the k-NN, the A-NN and the learned graph.

is $\mathcal{O}(n \log(n) d)$ for a graph of n nodes and data with dimensionality d, while additionally learning the edge weights costs only $\mathcal{O}(kn)$. The asymptotical complexity is thus dominated by the cost of computing an A-NN, and not by the cost of learning the weights. For the relatively small size of the problem we solve, this corresponds to 20 seconds for the A-NN graph (using compiled C code), and additional 45 seconds for our graph learning (using a Matlab-only implementation)[4]. With a full C implementation for our algorithm, we would expect the second number to be significantly smaller. Furthermore, for really large scale problems the asymptotical complexity would show that the bottleneck is not the graph learning.

5.3 Graph between words using word2vec representations

Motivated by the application of graphs between words in graph convolutional neural networks [6], we investigate the quality of our large scale graph learning in this type of data. For this, we learn a graph connecting similar words using Google word2vec features. A first observation is made in Figure 7, where we plot the diameters of different graphs for different density levels. We see that given the same density, the learned graph has a significantly larger diameter than both k-NN and A-NN. This indicates that our learned graph is closer to a manifold-like structure, unlike the other two types that are closer to a small-world graph. Manifold-like graphs reveal better the structure of data, while in small world graphs are related to randomness [31].

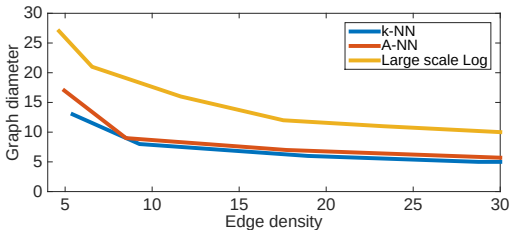

Figure 7: Diameter of the words graphs (built from word2vec features on the 10'000 most used English words). The learned graph has a significantly higher diameter for same node density.

We see this also qualitatively in Figure 6, where we plot nodes within two hops from the word "use" in the three types of graphs. We see that the NN graphs span a larger part of the entire graph just with 2 hops, the A-NN being closer to small-world. While the k-NN graph does better in terms of quality, it seems to do worse than our graph learning, that is actually cheaper to compute. In this sense, graph learning is a response to the randomness of A-NN while improving in speed upon k-NN. Additionally, graph learning seems to assign more meaningful weights as we show in Table 1 of the Appendix B.3. Note that we do not include results for the ℓ_2 model as it performs very poorly, leaving many disconnected nodes.

5.4 Approximation quality of large scale model

When computing $\mathcal{E}^{\text{allowed}}$, we use an approximate nearest neighbor (A-NN) graph using the publicly available

[4]Our implementation is available in the GSP-Box [26]. A tutorial based on this paper can be found in `https://epfl-lts2.github.io/gspbox-html/doc/demos/gsp_demo_learn_graph_large.html`.

FLANN library[5] that implements the work of Muja and Lowe [25]. To learn a graph with on average k neighbors per node (k-NN), we first compute a rk-A-NN graph and use its edges as $\mathcal{E}^{\text{allowed}}$. The graph is then learned on this subset. The choice of the number of allowed edges does not only affect the time needed to learn the graph, but also its quality. A too restrictive choice might prevent the final graph form learning useful edges. In Figure 8, we study the effect of this

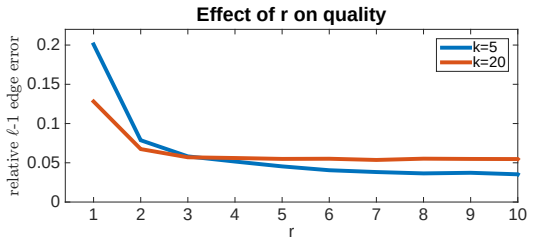

Figure 8: Effect of r on final graph quality for 1000 images of MNIST, average over 10 runs. Vertical axis is the error between our approximate log model and the exact log model by Kalofolias [15].

restriction on the final result. The vertical axis is the relative ℓ_1 error between our approximate log model and the actual log model by Kalofolias [15] when learning a graph between 1000 images of MNIST, averaged over 10 runs. Note that the result will depend on the A-NN algorithm used, while a comparison between different types of A-NN is beyond the scope of this paper.

5.5 Computation time

To show the scalability of our algorithm, we plot the time needed for learning different graph sizes between words. As we see in Figure 1, the cost is almost linear for our method, but quadratic for the original work of Kalofolias [15]. We also tested our Matlab implementation on $2^{19} \approx 500$K and $2^{20} \approx$ **1M samples** of the US census dataset. Setting $k = 5$, it used **29 minutes** to perform 500 iterations of graph learning. In Figure 11 of Appendix B.4, we also illustrate the linear scalability of our model w.r.t. the size of the set of allowed edges.

6 Conclusions

We propose the first scalable solution to learn a weighted undirected graph from data, based on A-NN and the current state-of-the-art graph learning model. While it costs roughly as mush as A-NN, it achieves quality very close to state-of-the-art. Its success is based primarily on reducing the variables used for learning and secondarily on selecting all parameters controlling the graph sparsity completely automatically. We assess its quality and scalability by providing an extensive set of experiments on many real datasets. Learning a graph of 1 million nodes only takes 29 minutes using our simple Matlab implementation on a desktop computer.

Code

A MATLAB implementation of our large scale graph learning is available in the GSPBOX [26]. A tutorial with most important functionality is available here: `https://epfl-lts2.github.io/gspbox-html/doc/demos/gsp_demo_learn_graph_large.html`

Thanks

The authors would like to especially thank Pierre Vandergheynst for his helpful comments during the preparation of this work at LTS2.

[5] Compiled C code run through Matlab, available from `http://www.cs.ubc.ca/research/flann/`.

References

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

[2] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research*, 7:2399–2434, 2006.

[3] K. Benzi, V. Kalofolias, X. Bresson, and P. Vandergheynst. Song recommendation with non-negative matrix factorization and graph total variation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2439–2443. IEEE, 2016.

[4] D. Cai, X. He, J. Han, and T. S. Huang. Graph regularized nonnegative matrix factorization for data representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(8): 1548–1560, 2011.

[5] S. I. Daitch, J. A. Kelner, and D. A. Spielman. Fitting a graph to vector data. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 201–208. ACM, 2009.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3837–3845, 2016.

[7] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *arXiv preprint arXiv:1406.7842v2*, 2015.

[8] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

[9] A. Elmoataz, O. Lezoray, and S. Bougleux. Nonlocal discrete regularization on weighted graphs: a framework for image and manifold processing. *IEEE transactions on Image Processing*, 17(7): 1047–1060, 2008.

[10] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[11] C. Hu, L. Cheng, J. Sepulcre, G. El Fakhri, Y. M. Lu, and Q. Li. A graph theoretical regression model for brain connectivity learning of alzheimer's disease. In *2013 IEEE 10th International Symposium on Biomedical Imaging*, pages 616–619. IEEE, 2013.

[12] T. Jebara, J. Wang, and S.-F. Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 441–448. ACM, 2009.

[13] B. Jiang, C. Ding, B. Luo, and J. Tang. Graph-laplacian pca: Closed-form solution and robustness. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3492–3498. IEEE, 2013.

[14] V. Kalofolias. *From data to structures: graph learning under smoothness assumptions and applications in data science*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2016.

[15] V. Kalofolias. How to learn a graph from smooth signals. In *The 19th International Conference on Artificial Intelligence and Statistics (AISTATS 2016)*. Journal of Machine Learning Research (JMLR), 2016.

[16] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst. Matrix completion on graphs. *arXiv preprint arXiv:1408.1717*, 2014.

[17] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[18] N. Komodakis and J.-C. Pesquet. Playing with duality: An overview of recent primal-dual approaches for solving large-scale optimization problems. *arXiv preprint arXiv:1406.5429*, 2014.

[19] B. Lake and J. Tenenbaum. Discovering structure by learning sparse graph. In *Proceedings of the 33rd Annual Cognitive Science Conference*. Citeseer, 2010.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[21] Y. A. Malkov and D. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*, 2016.

[22] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *arXiv preprint arXiv:1611.08402*, 2016.

[23] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[24] M. Muja and D. G. Lowe. Fast matching of binary features. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pages 404–410. IEEE, 2012.

[25] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[26] N. Perraudin, J. Paratte, D. Shuman, V. Kalofolias, P. Vandergheynst, and D. K. Hammond. GSPBOX: A toolbox for signal processing on graphs. *ArXiv e-prints*, Aug. 2014.

[27] N. Shahid, V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst. Robust principal component analysis on graphs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2812–2820, 2015.

[28] N. Shahid, N. Perraudin, V. Kalofolias, G. Puy, and P. Vandergheynst. Fast robust pca on graphs. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):740–756, 2016.

[29] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[30] F. Wang and C. Zhang. Label propagation through linear neighborhoods. *Knowledge and Data Engineering, IEEE Transactions on*, 20(1):55–67, 2008.

[31] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684): 440–442, 1998.

[32] T. Zhang, A. Popescul, and B. Dom. Linear prediction models with graph regularization for web-page categorization. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 821–826. ACM, 2006.

[33] M. Zheng, J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, and D. Cai. Graph regularized sparse coding for image representation. *Image Processing, IEEE Transactions on*, 20(5):1327–1336, 2011.

[34] X. Zhu, Z. Ghahramani, J. Lafferty, et al. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, volume 3, pages 912–919, 2003.

Supplementary Material

A Proof of Theorem 2

Proof. *From the proof of Theorem 1, we know that* $\|w^*\|_0 = k$ *if and only if* $\lambda^* \in [\theta z_k, \theta z_{k+1})$. *We can rewrite this condition as*

$$\theta z_k \leq \frac{\theta b_k + \sqrt{\theta^2 b_k^2 + 4k}}{2k} < \theta z_{k+1} \Leftrightarrow$$
$$2k\theta z_k - \theta b_k \leq \sqrt{\theta^2 b_k^2 + 4k} < 2k\theta z_{k+1} - \theta b_k \Leftrightarrow$$
$$4k^2\theta^2 z_k^2 - 4k\theta^2 b_k z_k \leq 4k < 4k^2\theta^2 z_{k+1}^2 - 4k\theta^2 b_k z_{k+1} \Leftrightarrow$$
$$\theta^2(kz_k^2 - b_k z_k) \leq 1 < \theta^2(kz_{k+1}^2 - b_k z_{k+1}).$$

As θ *is constrained to be positive, the only values that satisfy the above inequalities are the ones proposed in the theorem.*

B Experiments

B.1 MNIST irregular intra-class distances

Figure 9 illustrates one irregularity of the MNIST dataset. One could expect that the average distance between two digits of the same class (intra-class) is more or less independent of the class. Nevertheless, for the MNIST dataset, the distance between the 1 is significantly smaller than the one of the other digits. For this reason, the L2 model connects significantly more the digits 1 than the others.

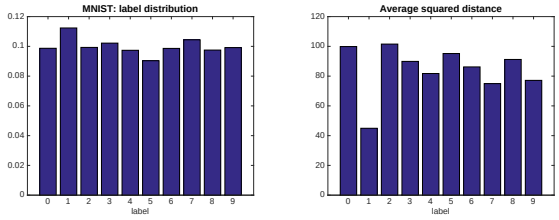

Figure 9: Label frequency (**left**) and average squared distribution (**right**) of MNIST train data (60000 nodes). The distances between digits "2" are significantly smaller than distances between other digits.

B.2 Accuracy of the tuning of the θ parameter

We already saw in Figure 2 (middle) that for the MNIST dataset eq. (17) predicts very well the sparsity of the final graph for any choice of θ. This is further illustrated on the USPS and ATT faces datasets in Figure 10. Note, that in the rare cases that the actual sparsity is outside the predicted bounds, we already have a good starting point for finding a good θ. For example, in the COIL dataset, if we want a graph with 15 edges per node we will set $\theta = 1.2$, obtaining instead a graph with 12 edges per node. This kind of fluctuations are usually tolerated, while even in k-NN we always obtain graphs with more than nk edges due to the fact that W is symmetric.

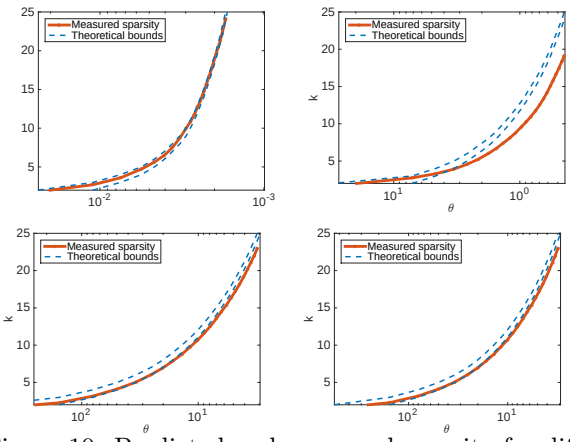

Figure 10: Predicted and measured sparsity for different choices of θ. Note that θ is plotted in logarithmic scale and decreasing. **Up left**: 400 ATT face images. **Up right**: 1440 object images from the COIL dataset. **Down left**: Graph between 1000 samples from a multivariate uniform distribution. **Down right**: Graph between 1000 samples from a multivariate Gaussian distribution.

B.3 Connectivity example of the graph of words

In Table 1, we look in more detail at the graph constructed from the word2vec features. We present the connectivity for the word "glucose" and "academy". Looking at different words, we observe that the learned graph is able to associate meaningful edge weights to the different words according to the confidence of their similarity.

B.4 MNIST Computational time

The cost of learning a graph with a subset of allowed edges $\mathcal{E}^{\text{allowed}}$ is linear to the size of the set as illustrated in Figure 11. For this experiment, we use the MNIST data set. To learn a graph with approximately 10 edges per node, we needed 20 seconds to compute $\mathcal{E}^{\text{allowed}}$, and 20 seconds to learn the final graph of 60000 nodes (around 250 iterations). Note that the time necessary to search for the nearest neighbors is in the same order of magnitude than the learning process.

Word	k-NN	A-NN	Learned
glucose	0.1226 insulin 0.0233 protein 0.0210 oxygen 0.0148 hormone	0.0800 insulin 0.0337 protein 0.0306 oxygen 0.0295 cholesterol 0.0263 calcium 0.0225 hormone	0.5742 insulin 0.0395 calcium 0.0151 metabolism 0.0131 cholesterol
academy	0.0996 training 0.0953 school 0.0918 institute	0.0901 young 0.0863 department 0.0841 bizrate	0.3549 training 0.2323 institute 0.1329 school 0.0135 camp 0.0008 vocational

Table 1: Weight comparison between k-NN, A-NN and learned graphs. The weights assigned by graph learning correspond much better to the relevance of the terms.

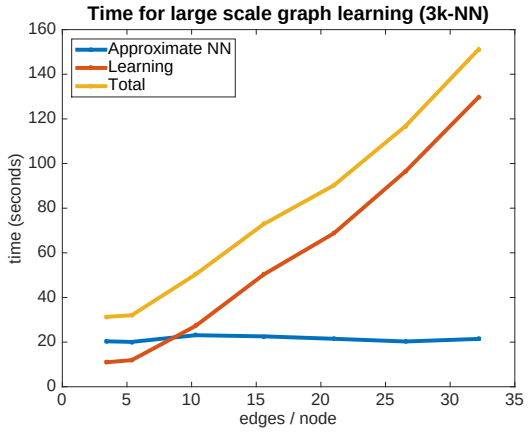

Figure 11: Time needed for learning a graph of 60000 nodes (MNIST images) using the large-scale version of (3). Our algorithm converged after 250 to 450 iterations with a tolerance of $1e-4$. The time needed is linear to the number of variables, that is linear to the average degree of the graph.

B.5 Another measure to test the parameter r

The percentage of nodes of the final graph that have the maximum number of edges allowed by $\mathcal{E}^{\text{allowed}}$ is a measure of this quality deterioration due to the edge restriction imposed by the A-NN. Using the MNIST dataset, it is plotted for our large scale log-degree model in the bottom part of Figure 12. In the horizontal axis we have k, and different lines correspond to different densities of the A-NN graph, from $2k$ to $4k$. We find that an A-NN of $3k$ (i.e. $r = 3$) is a good compromise between time and quality, and use it for most of the experiments of this paper.

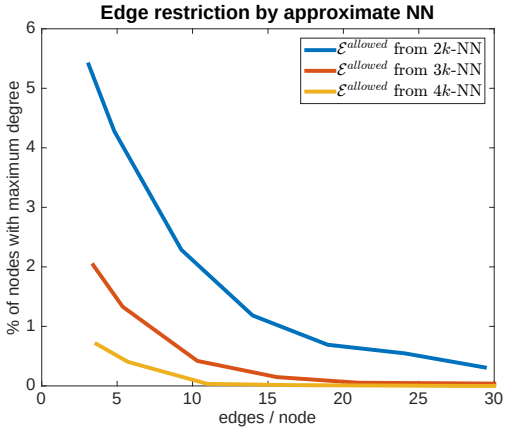

Figure 12: Percentage of nodes that after learning have the maximum allowed number of edges. For these nodes, additional neighbors are probably needed.