Using Unsupervised Learning for Graph Construction in Semi-Supervised Learning with Graphs

Diego Alonso Chávez Escalante*, Gabriel Taubin†, Luis Gustavo Nonato‡ and Siome Klein Goldenstein*
*Institute of Computing, University of Campinas, IC-UNICAMP
Email: ra134046@students.ic.unicamp.br, siome@ic.unicamp.br
†School of Engineering, Brown University
Email: taubin@brown.edu
‡Institute of Mathematic Sciences and Computing, University of São Paulo, ICMC-USP
Email: gnonato@icmc.usp.br

***Abstract*—Semi-supervised Learning with Graphs can achieve good results in classification tasks even in difficult conditions. Unfortunately, it can be slow and use a lot of memory. The first important step of the graph-based semi-supervised learning approaches is the construction of the graph from the data, where each data-point usually becomes a vertex in the graph – a potential problem with large amounts of data. In this paper, we present a graph construction method that uses an unsupervised neural network called growing neural gas (GNG). The GNG instance presents a intelligent stopping criteria that determines when the final network configuration maps correctly the input-data points. With that in mind, we use the final trained network as a reduced input graph for the semi-supervised classification algorithm, associating original data-points to the neurons they have activated in the unsupervised training process.**

I. Introduction

Traditional supervised learning algorithms fail in the classification task when there are many data-points to classify but only a few labeled data-points for training – in the extreme case there are fewer annotated instances than the dimension of each data-point. These cases are hard because there is just too little data to work with.

If we know the unlabeled data-points at training time, semi-supervised learning (SSL) algorithms are good candidates to successfully accomplish this task, because then can use the unlabeled data that supervised learning cannot use [1]–[3] to provide more information.

SSL is an hybrid between supervised and unsupervised learning, it uses and combines concepts from both [1]–[3] approaches. SSL can learn the topological distribution of the data as unsupervised learning, and augment classification from labeled data, as supervised learning does. This combination makes SSL classification algorithms more robust than regular classifiers in extreme conditions.

There are five types of SSL methods [2], [3]:

1) semi-supervised transductive SVMs,
2) mixture models with semi-supervised EM algorithms,
3) self-training,
4) co-training,
5) and semi-supervised graph based methods.

The main advantage of semi-supervised graph based methods is that they are semi-supervised by essence while the others are adaptations of supervised algorithms. Graph based methods use the label propagation among vertices as the main intuition to deal with classification, this propagation begins from the labeled vertices to the unlabeled ones. An unlabeled vertex receives a label from the vertices that are highly related to it, and then it propagates the label to other vertices. Generally graph-based SSL has two main steps: the graph construction and the graph regularization. The size of the graph affects directly the complexity and space of graph-based SSL algorithms [2], [4]. The most common graph construction techniques represent each data-point as one vertex in the graph, so are extremely sensitive to the size of the input data.

In this paper, we use an unsupervised learning neural network, known as Growing Neural Gas (GNG) [5]–[7], to introduce a new graph construction method appropriate for use in graph-based SSL. We describe our modifications to the GNG to obtain a stopping criteria that tell us when the graph produced by the neural network correctly maps our input data, and it is appropriate for use in a SSL classification scenario. Our graph construction method can be combined to most regularization techniques in semi-supervised learning with graphs, and in our experiments we make a comparative study of different graph construction methods versus our approach combined with different regularization techniques. The experiments section shows how our graph-construction approach faithfully represents some set of patterns and saves space with little loss of accuracy during the classification.

II. Semi-Supervised Learning with Graphs

In general SSL approaches there are three types of data: labeled data X_l, known unlabeled data X_u, and unknown unlabeled data X_n. An SSL method is *transductive* when it does not know how to handle X_n and is only interested in label X_u. When the algorithms labels both X_u and X_n, it is called *inductive* [2]. Graph SSL are transductive by nature and have with two main steps, the graph construction or graph conceptualization, and the graph regularization or label propagation. At the end of the latter stage, all data-points in X_u should be labeled.

There are many ways to build the graph. Most methods treat as vertices each data-point $x \in X_l \cup X_u$, and for the edge weight they use a similarity measure based on the distances of adjacent vertices [1]–[3]. The graphs are typically either fully connected, k-graphs, or ϵ-graphs. Fully connected graphs connect each vertex to all the other vertices. In k-graphs, each vertex connects to its k nearest vertices. Finally, in ϵ-graphs each vertex connects to all the vertices that are at a distance less than ϵ.

The graph regularization step aims to propagate the labels from X_l to X_u within the graph, trying to respect two main conditions. The first is to respect the labels of the labeled data-points, after the regularization takes place all labeled data should have the same labels as in the beginning. The second condition to accomplish is that label transitions should be smooth in the graph, that is if an edge has its adjacent vertices with different labels, it should have the greater possible weight, so edges with small weights should have both adjacent vertices with the same label [2], [8]. The graph regularization problem can be seen as an optimization problem, see Equation 1.

$$\arg\min_{f} \left(\overbrace{Q(f)}^{loss\ function} + \underbrace{R(f)}_{regularizer} \right) \tag{1}$$

In Equation 1 the loss function takes over the first condition while the regularizer takes over the second condition. Most of the semi-supervised learning methods with graphs work with variations of this general equation [2]. The literature presents many ways to deal with the graph regularization step. One of them is the use of label propagation algorithms that converge to a solution within a number of iterations, this could take a while, or even never converge, it depends on the graph structure. Also there are direct analytical solutions as the use of Harmonic functions or the use of smooth operators of the graph from the graph Laplacian, this methods used to deal with matrix inversion and spectral decomposition of the graph Laplacian Δ,

$$\Delta = D - W, \tag{2}$$

where W is the adjacent matrix of the graph and the diagonal matrix D is

$$D_{ii} = \sum_{j=1}^{l+u} w_{ij}, \tag{3}$$

where w_{ij} the weight of the edge between vertices i and j.

The graph Laplacian is a matrix of size $(l+u) \times (l+u)$, so when dealing with very large datasets, unless the matrices are sparse, inverting or decomposing this matrix would be computationally expensive.

Also there are method for fast computation of the classification function f, this methods work with spectral approximations [9], sub-space Krylov iterations [10], EigenFunctions of the smooth operator of the graph Laplacian [4] and the construction of backbone graphs with less vertices than the original graph [9], [11].

III. Growing Neural Gas with Stopping Criteria

A GNG neural network is an unsupervised incremental algorithm that learns topological relations of a given input data-set. This neural network can change its configuration while it is training itself, adding and removing neurons and connections whenever necessary [5]–[7], [12]. Unlike other unsupervised neural networks, this behavior optimizes the speed and memory use. And using a correct stopping criterion for our problem, it can represents correctly at the right moment, in the training process, a group of data-points.

This neural network can be seen as a graph $G = (V, E)$, where each vertex $v \in V$ is a neuron and the edges $e \in E$ are the connections between the neurons. Each vertex has the following structure: $v = (p, \xi)$, where p are the coordinates of the neuron v, $p \in \mathbb{R}^n$, like the input data-points $x \in \mathbb{R}^n$; and $\xi \in \mathbb{R}$ is the error rate of neuron v . The edges has also a configuration structure, each edge $e = (v_i, v_j, w_{ij}, t)$, v_i and v_j are the vertices adjacent to e, w_{ij} is the weight of the edge, which could be the euclidean distance between them, and t is the age of the edge, in number of training iterations.

The GNG begins with just two isolated neurons, randomly disposed in the data-points space. At each iteration one data-point x is analyzed, then the two nearest neighbors to x are picked from V, let say v_s and v_t, being v_s the closest vertex to x, so v_s was activated by x. Then the error rate ξ_s of v_s is updated as in Equation 4; where $d(x, p_s)$ is the distance between x and v_s (which is at p_s coordinates), by this way the vertices that were activated by more data-points tend to have a bigger error ξ.

$$\xi_s = \xi_s + d(x, p_s) \tag{4}$$

After updating the error, it is created an edge $e_{st} = (v_s, v_t, 0, 0)$, if it already exists then just the age t_{st} is set to 0; then the coordinates p_s of v_s and the coordinates p_m of all of its adjacent vertices including v_t are updated as in Equations 5 and 6; ϵ_s and ϵ_m are real numbers in $[0, 1]$, and $\epsilon_s \geq \epsilon_m$, those are hyper-parameters of the GNG.

$$p_s = p_s + \epsilon_s(x - p_s) \tag{5}$$

$$p_m = p_m + \epsilon_m(x - p_m) \tag{6}$$

Then the distance between v_s and all of its adjacent vertices v_m are updated, as $e_{sm} = (v_s, v_m, d(p_s, p_m), t_{sm})$. At each iteration all the edges of the GNG connected to v_s increment their age t by one. The neural network grows when a given number λ of input-patterns x are analyzed, λ is also an hyper-parameter of this algorithm, it is selected the vertex v_{max} with maximum internal error ξ_{max}, and then it is selected the adjacent vertex to v_{max} that has the maximum error too, so it is created a new vertex v_{new} between v_{max} and v_{nmax}, $p_{new} = 0.5(p_{max} + p_{nmax})$, subsequently the errors of v_{max} and v_{nmax} are decreased by $\xi_{max} = \xi_{max} - \alpha(\xi_{max})$ and $\xi_{nmax} = \xi_{nmax} - \alpha(\xi_{nmax})$, after this $\xi_{new} = 0.5(\xi_{max} + \xi_{nmax})$, α is another hyper-parameter of the algorithm.

This algorithm also prune vertices and edges from the graph, there is an hyper-parameter β, at each iteration all the edges which ages $t > \beta$ are pruned and if one vertex after this becomes isolated it is removed too. Finally all the errors of all

the remaining vertices are decreased by $\xi = \xi - \delta\xi$, δ is the last hyper-parameter of this algorithm.

As the literature points out, in order to get a good trained GNG it is necessary to use another stopping criteria, different than the number of iterations [12]. This way we optimize for time and space, avoiding unnecessary training iterations and the creation of unnecessary vertices.

IV. Proposed Algorithm

The main idea of this algorithm is to use the GNG with some stopping criteria to build a graph that faithfully represents $X_l \cup X_u$, and immediately after this use some traditional graph regularization algorithm in order to calculate the classification function f over the graph G built by the GNG algorithm. But as we are not trying to clusterize the data, a clustering index would not be right for our purpose, it is necessary another way that let us know when to stop the training of the neural network.

The index that will be used to stop the GNG training has to tell us if there are significant changes in the mapping of the data by the graph. First we have the vertex influence index I_{vi}, look Equation 7; where $x_i \in X_l \cup X_u$, l and u are the number of elements in X_l and X_u respectively, and p_{x_i} are the coordinates of the vertex that was the last to be activated by x_i; when each pattern x_i is analyzed by the GNG, it is annotated the vertex that was activated by it. This index is a mean of all the distances of the data-points to their respective activated neurons and tell us basically if there are enough vertices that map the data, this index has to be minimized for better results, but also depending of the topology of the input data set, this index would reach a minimum that cannot be decreased by any other graph configuration.

$$I_{vi} = \frac{1}{l+u} \sum_{i=1}^{l+u} d(x_i, p_{x_i}) \tag{7}$$

The second index that we use, is called vertex distribution index I_{vd}, this index is shown in Equation 8, $Adj(v_i)$ is a function that returns all the adjacent vertices to v_i, p_i and p_{nb} are the coordinates of vertices v_i and v_{nb}, and the distance $d(p_i, p_{nb})$ between them, are obtained from w_{inb} of the edge with v_i and v_{nb} as adjacent vertices; finally $|V|$ represents the number of vertices in V. The vertex distribution index is the mean of the larger edge of each vertex in the graph, it express if the graph is correctly distributed, this index for better results in this problem should have the less possible value; I_{vd} could reach a very low value if the graph has too many vertices, by this it has to work with the vertex influence index for a good performance and description of the data graph mapping.

$$I_{vd} = \frac{1}{|V|} \sum_{i=1}^{|V|} \underset{v_{nb} \in Adj(v_i)}{\arg\max}\ d(p_i, p_{nb}) \tag{8}$$

When a graph faithfully maps the input data, $I_{vi} \leq I_{vd}$, because we are assuming that the vertices are uniformly distributed and the original data-points are almost at the same distance to their activated neuron, the mean of all the distances of the data-points to their respective activated neuron cannot be larger than the distance between vertices, from this we get to the final graph mapping index I_{gm} in Equation 9, which should be maximized, but never greater than one. This index tell us the current state of the graph with respect to the data, every time the input patterns were analyzed by the GNG.

$$I_{gm} = \frac{I_{vi}}{I_{vd}} \tag{9}$$

The GNG training with stopping criteria would be done in the same fashion as was did by D. Chavez et al. [12] using an index error and an error factor, but replacing the SV index by the graph mapping index, which tell us the state of the data graph mapping at every configuration of the graph. Each time that a quantity of input patterns are analyzed (we set this quantity as the size of the input-patterns set $l + u$), it is calculated de I_{gm} index for all the analyzed patterns ($I_{gm} = 0$ at the beginning), then it is calculated the error of the I_{gm} index which is the difference between the actual I_{gm} and the last I_{gm}, the error factor is the number of times that the error can be less than the index error before stopping the algorithm. Another modification was made to the traditional GNG, to insert a new vertex to the graph, not only, it has to be satisfy the condition of the λ patterns analyzed, but also there has to be checked if there are no more vertices than a proportion of the input patterns, this proportion is also another hyper parameter of the algorithm. The last modification done to the traditional GNG was, that if, after the training process was done, it is obtained a not connected graph, this graph is converted to a connected graph by connecting the vertices of different subgraphs that are the closest between them.

After the successful training of the GNG with the graph mapping index as the stopping criteria, we have a graph where its edge weights represent the distance between vertices, so now we have to convert this distances into similarities, after this the traditional graph regularization would be done over the graph; but just before this we have to deal also with the labels mapping. Basically a vertex would have the label that is most common for all the data-points that activated it, if those data-points do not have a label then the vertex either does not have a label, or if there are more than one common label it would be chosen randomly. After the regularization step, the label mapping is inverted, all the data-points would have the label of the neuron that was activated by them, but all the data-points in X_l would keep its original label.

V. Experiments and Results

In this section, we describe the experimental procedure and results of this paper. We compare our method with other graph constructions algorithms: full graphs, ϵ-graphs, and k-graphs; using different graph regularization algorithms: the harmonic function regularizer [13], the Krylov approximated regularizer [10] and the eigenvectors of the smooth operator of the graph Laplacian regularizer [4]. We compare graph construction-regularization combinations in terms of accuracy given a progressive proportion of labeled data-points in different input data-set size instances.

We choose image content classification problem for our evaluation: we want to classify a set of images by a concept (for example all the images with dogs in the foreground) from a little subset of labeled images with this concept, for

TABLE I. ACCURACY RESULTS FROM THE 1000 DATA-POINTS GROUP.

Graph Construction Method	Graph Regularization Method	Labeled Data Proportion				
		10%	20%	30%	40%	50%
Full Graph	Harmonic	0.578	0.597	0.656	0.714	0.758
	Krylov	0.574	0.596	0.654	0.710	0.755
	Smooth Operator	0.562	0.596	0.586	0.602	0.624
k-Graph ($k = 2$)	Harmonic	0.553	0.608	0.648	0.711	0.744
	Krylov	0.553	0.606	0.643	0.710	0.743
	Smooth Operator	0.548	0.585	0.604	0.604	0.623
k-Graph ($k = 3$)	Harmonic	0.556	0.586	0.663	0.719	0.756
	Krylov	0.557	0.587	0.663	0.718	0.757
	Smooth Operator	0.558	0.625	0.616	0.633	0.638
k-Graph ($k = 5$)	Harmonic	0.590	0.604	0.667	0.723	0.754
	Krylov	0.589	0.602	0.667	0.724	0.753
	Smooth Operator	0.583	0.586	0.610	0.623	0.627
k-Graph ($k = 10$)	Harmonic	0.570	0.600	0.658	0.722	0.747
	Krylov	0.568	0.601	0.658	0.720	0.746
	Smooth Operator	0.577	0.594	0.615	0.633	0.638
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.550	0.600	0.650	0.700	0.750
	Krylov	0.550	0.600	0.650	0.700	0.750
	Smooth Operator	0.500	0.500	0.500	0.500	0.500
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.568	0.605	0.667	0.710	0.752
	Krylov	0.568	0.605	0.667	0.710	0.752
	Smooth Operator	0.511	0.537	0.537	0.516	0.512
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.550	0.619	0.649	0.710	0.754
	Krylov	0.550	0.619	0.649	0.710	0.754
	Smooth Operator	0.550	0.585	0.601	0.596	0.610
GNG Graph ($\lambda = 5$)	Harmonic	0.568	0.610	0.669	0.696	0.739
	Krylov	0.568	0.610	0.669	0.696	0.739
	Smooth Operator	0.552	0.609	0.669	0.674	0.691
GNG Graph ($\lambda = 10$)	Harmonic	0.586	0.628	0.657	0.702	0.712
	Krylov	0.585	0.628	0.657	0.702	0.711
	Smooth Operator	0.573	0.599	0.628	0.681	0.649

this purpose we used in our experiments the CIFAR-10 label set [14] with the Tiny Images data set [15]. The CIFAR-10 label set maps 60000 images from the Tiny Images data set to ten labels (Figure 1), so each label has approximately 6000 images samples.

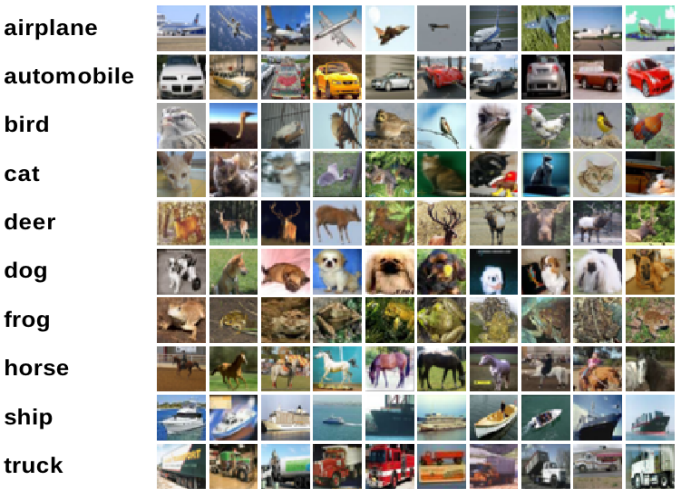

Fig. 1. CIFAR-10 label set.

.

The elements of the Tiny Images data set are 32x32 pixel images, in our experiments this images were described using the GIST descriptor [16], which is a holistic descriptor for complete scenes of 384 dimensions. Each datapoint was projected to a 64 dimensional space using principal component analysis, this projection maintain the 81.3 % of the original data's variance.

We have three groups of experiments, in all of them the number of known positive labeled data-points (labels from the CIFAR-10 label set) where the same as the known negative labeled data-points. The first group of experiments were made with just 1000 data-points, the second group was made with 4000 data-points and the third with 8000, each group with 5% to 50% of labeled data.

As we compare the graph regularization performance with different graph construction approaches, it is necessary to know how each regularization method behaves with each graph construction method by itself, in order to accomplish this, there are the charts in Figures 2, 3 and 4. In Figure 2 we show the performance in terms of accuracy (with 0.5 as decision threshold, being 0 the negative labels and 1 the positive ones) of the Harmonic regularizer with the different graph construction methods, at a progressive amount of labeled data (from 5% to 50%); the four graph methods were very tied with the Harmonic regularizer, but the best performance was obtained by the ϵ-graph at a 50% of labeled data, as can

TABLE II. ACCURACY RESULTS FROM THE 4000 DATA-POINTS GROUP.

Graph Construction Method	Graph Regularization Method	Labeled Data Proportion				
		10%	20%	30%	40%	50%
Full Graph	Harmonic	0.54725	0.60525	0.65950	0.70725	0.75000
	Krylov	0.54800	0.60525	0.65850	0.70900	0.75000
	Smooth Operator	0.55450	0.60725	0.61275	0.62225	0.63300
k-Graph ($k = 2$)	Harmonic	0.56350	0.60975	0.65550	0.71025	0.75900
	Krylov	0.56300	0.60875	0.65575	0.70925	0.75850
	Smooth Operator	0.56425	0.58825	0.60050	0.61600	0.62800
k-Graph ($k = 3$)	Harmonic	0.56075	0.60100	0.65475	0.70950	0.76450
	Krylov	0.56025	0.60075	0.65500	0.70950	0.76400
	Smooth Operator	0.55575	0.60375	0.60125	0.61500	0.62350
k-Graph ($k = 5$)	Harmonic	0.55875	0.60625	0.65375	0.70300	0.75175
	Krylov	0.55625	0.60375	0.65350	0.70250	0.75175
	Smooth Operator	0.56000	0.58850	0.60350	0.61050	0.62200
k-Graph ($k = 10$)	Harmonic	0.56675	0.61325	0.65350	0.70475	0.75850
	Krylov	0.56625	0.61350	0.65400	0.70400	0.75775
	Smooth Operator	0.55975	0.60125	0.62075	0.61625	0.62100
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.55000	0.60000	0.65000	0.70000	0.75000
	Krylov	0.55000	0.60000	0.65000	0.70000	0.75000
	Smooth Operator	0.50000	0.50000	0.50000	0.50000	0.50000
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.55450	0.60100	0.65225	0.69600	0.75250
	Krylov	0.55450	0.60325	0.65275	0.69700	0.75250
	Smooth Operator	0.52000	0.53050	0.54725	0.54750	0.55300
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.54700	0.59975	0.65175	0.69700	0.74825
	Krylov	0.54700	0.59975	0.65175	0.69700	0.74825
	Smooth Operator	0.55575	0.58475	0.60850	0.62175	0.63200
GNG Graph ($\lambda = 5$)	Harmonic	0.55900	0.59350	0.64650	0.67600	0.72675
	Krylov	0.55900	0.59350	0.64625	0.67625	0.72675
	Smooth Operator	0.55900	0.58550	0.64325	0.65925	0.67425
GNG Graph ($\lambda = 10$)	Harmonic	0.55175	0.58950	0.62500	0.64700	0.67825
	Krylov	0.55175	0.58950	0.62500	0.64750	0. 67875
	Smooth Operator	0.55500	0.58650	0.62075	0.60475	0.62575

be seen by this the GNG graph construction algorithm can obtain comparable results to the traditional graph construction algorithms but reducing the cost of the regularization and saving memory, because it induce a new graph with less vertices, in this case was induced a graph with $0.9(l + u)$ vertices (0.9 was set as hyper parameter of the algorithm) with a graph mapping index of 0.89, a GNG growing rate $\lambda = 5$, index error of 0.5 and error factor of 2.

The results from the Krylov regularizer with 1000 data-points can be seen in Figure 3, they are very similar to the Harmonic regularizer results, all the four graph construction methods were also tied, and the best result was obtained again by the ϵ-graph at 50% with $\epsilon = 0.8$, the experiments with the Krylov method were done with 100 iterations in the MINRES algorithm [10]. One thing that is hard to tell is which are the better graph construction method for each instance, how to know that an ϵ-graph with $\epsilon = 0.8$ was the right choice, with out many tests with different ϵs, or a detailed analysis of the topology of the patterns, that is the main problem, the literature shows that in most cases it is better to regularize complete connected graphs, but with the drawback of the cost of space and complexity.

In Figure 4 is shown the results from the eigenvectors of the smooth operator regularizer with $0.3(l+u)$ eigenvectors, in this case the results were different from those with the Harmonic regularizer and the Krylov regularizer, in this experiments the

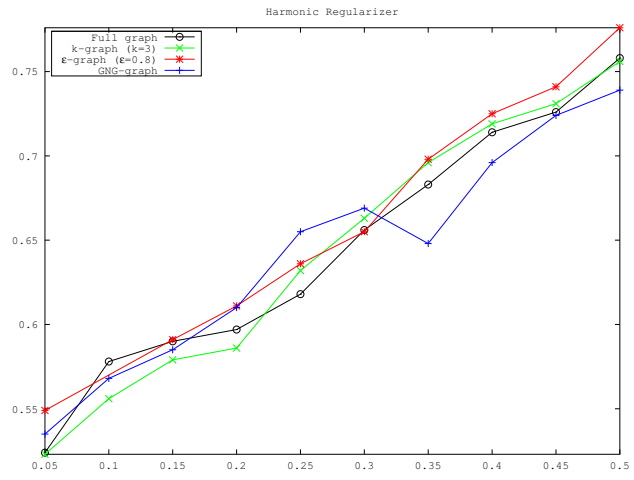

Fig. 2. Results from the Harmonic Regularizer experiments with 1000 data-points (percentage of labeled data vs accuracy).

GNG graph construction clearly outperforms the other three graph construction method, this was because as the graph induced by the GNG graph construction algorithm is smaller ($|V| < l + u$), so it is $0.3|V| < 0.3(l + u)$, and in the regularization we were using $0.3(l+u)$ eigenvectors of a graph

TABLE III. ACCURACY RESULTS FROM THE 8000 DATA-POINTS GROUP.

Graph Construction Method	Graph Regularization Method	Labeled Data Proportion				
		10%	20%	30%	40%	50%
Full Graph	Harmonic	0.54863	0.59713	0.64475	0.69575	0.74738
	Krylov	0.54863	0.59713	0.64475	0.69575	0.74738
	Smooth Operator	0.54950	0.60150	0.64100	0.67825	0.68588
k-Graph ($k = 2$)	Harmonic	0.55488	0.60600	0.65263	0.70125	0.74863
	Krylov	0.55475	0.60600	0.65263	0.70125	0.74863
	Smooth Operator	0.54050	0.60200	0.64688	0.66788	0.69013
k-Graph ($k = 3$)	Harmonic	0.54875	0.59863	0.65050	0.70275	0.75088
	Krylov	0.54863	0.59863	0.65050	0.70275	0.75088
	Smooth Operator	0.55200	0.59825	0.64863	0.67425	0.69063
k-Graph ($k = 5$)	Harmonic	0.54875	0.60013	0.64988	0.70013	0.75125
	Krylov	0.54875	0.60013	0.64988	0.70013	0.75125
	Smooth Operator	0.55288	0.59075	0.64588	0.68450	0.69763
k-Graph ($k = 10$)	Harmonic	0.54563	0.59313	0.64538	0.69925	0.74650
	Krylov	0.54563	0.59313	0.64538	0.69925	0.74650
	Smooth Operator	0.54813	0.59963	0.64175	0.68113	0.69663
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.55000	0.60000	0.65000	0.70000	0.75000
	Krylov	0.55000	0.60000	0.65000	0.70000	0.75000
	Smooth Operator	0.50000	0.50013	0.50013	0.50013	0.50013
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.54888	0.59725	0.65088	0.69925	0.74763
	Krylov	0.54900	0.59850	0.65138	0.70050	0.74825
	Smooth Operator	0.51875	0.53525	0.61450	0.57125	0.58850
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.54788	0.59838	0.64388	0.69863	0.74663
	Krylov	0.54763	0.59850	0.64388	0.69863	0.74675
	Smooth Operator	0.54500	0.59625	0.65363	0.68350	0.69950
GNG Graph ($\lambda = 5$)	Harmonic	0.53750	0.58963	0.63675	0.66950	0.69938
	Krylov	0.53750	0.58963	0.63650	0.66950	0.69963
	Smooth Operator	0.54563	0.59188	0.62138	0.64938	0.64788
GNG Graph ($\lambda = 10$)	Harmonic	0.54063	0.59013	0.63088	0.65375	0.65638
	Krylov	0.54075	0.59025	0.63100	0.64388	0.65638
	Smooth Operator	0.54738	0.58750	0.61125	0.61450	0.60150

of size $|V|$, so the proportion of eigenvectors in the GNG induced graph is greater than 0.3; and by this it is shown also that the induced graph has similar spectral decomposition as the graph instances with $l + u$ vertices, pointing out the right data mapping that was accomplished by the GNG graph construction algorithm.

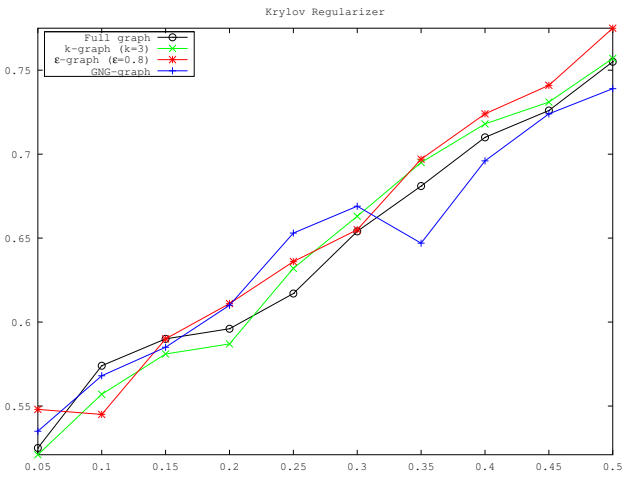

Fig. 3. Results from the Krylov Regularizer experiments with 1000 data-points (percentage of labeled data vs accuracy).

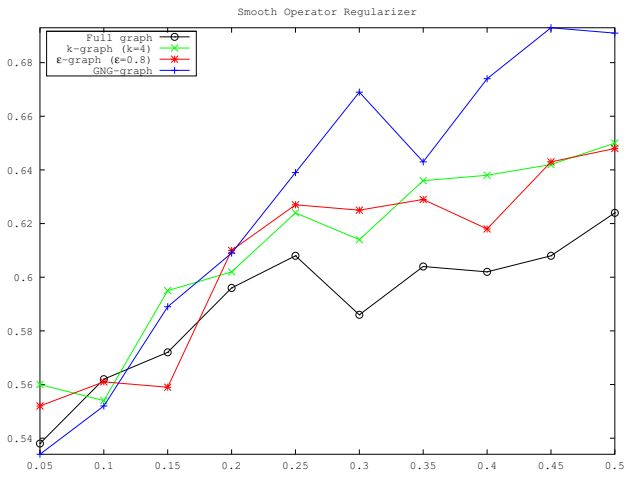

Fig. 4. Results from the Smooth Operator Regularizer experiments with 1000 data-points (percentage of labeled data vs accuracy).

The detailed results of the most meaningful experiments are in tables I, II and III for 1000, 4000 and 8000 data-points respectively. For 8000 data-points were induced graphs with $0.8(l+u)$ vertices and a error factor of 1 in order to save space and get a faster convergence of the algorithm, and by this the accuracy was less compared to the cases where were induced

graphs with $0.9(l+u)$ vertices. But the behavior is very similar in all instances, but as the input size increases the accuracy decrease in all the construction-regularization combinations.

The GNG graph construction algorithm have a complexity of $O((l+u)|V|I)$ where I is the number of executed iterations (by executed iterations we refer to each time the I_{gm} index is evaluated); most of the implementations of the other graph construction algorithms would have a complexity of $O((l+u)^2)$, so if $|V|I < (l+u)$ the GNG construction algorithm would be faster, because it would have a better best case, but its worst case (when $|V|I > (l+u)$) could be much more expensive than $O((l+u)^2)$ if the algorithm does not converge fast or there were created too many neurons too soon and few of them are removed in the process. It can be guarantee to always get a best case by setting as input parameters I and $|V|$, initializing them with values that enforce $|V|I < (l+u)$, but if they are too small they will have a drawback in graph mapping correctness and hence classification accuracy. In order to solve this issues there have to be set appropriate values for the hyper-parameters of the algorithm, and this values could variate for each data-set.

From the final results of the experiments, it can be shown that the Growing Neural Gas with stopping criteria graph construction algorithm reduce the space and complexity for the regularization step with no drawbacks in terms of classification accuracy (as it induced a smaller graph), and as long as the stopping criteria works, it also reduce the space in the graph construction step. Also in order to optimize the graph mapping index, $|V|$ has to be very close to $l+u$, but not close enough to make the GNG graph construction useless in terms of space saving. But the main problem with GNG construction algorithm is all the hyper-parameters it has to deal with, mainly λ, the index error and the error factor. As we try to make the construction as fast as possible and with the better graph mapping index; λ has to be small (in this case we chose a small value as 5 for λ), by this the GNG leaves small room for adaptation, depending on $l+u$; the index error should not be too small and the error factor should not be large, by this way the GNG would stop quickly with a suitable number of vertices, which would not be greater than the proportion (we used 0.9 and 0.8) of $l+u$ specified as hyper-parameter of the algorithm. As the proportion of maximum allowed vertices is lesser, depending on the topology of the data, the performance of the transductive classification could decrease, but more memory would be saved in the process; so it has to be choose a equilibrium between saved space and desired performance.

VI. Conclusion

We have presented a new approach to graph construction for semi-supervised learning classification, the *GNG graph* construction algorithm, based in the GNG (Growing Neural Gas) algorithm. This method faithfully maps the input patterns for the graph based SSL algorithm to a smaller graph. This allows a faster and more compact regularization step of any graph-based SSL algorithm. The new stopping criterion introduced in this paper, based in the graph mapping index, allows a fast and efficient GNG graph construction. Finally, the experiments shown that this approach does not affect the performance of the classification task compared to the traditional graph construction algorithms, and in some cases it even improves it.

References

[1] X. Zhu, "Semi-supervised learning with graphs," Ph.D. dissertation, Carnegie Mellon University, 2005.

[2] X. Zhu and A. B. Goldberg, *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers, 2009.

[3] X. Zhu, "Semi-supervised learning literature survey," Survey, 2008.

[4] R. Fergus, Y. Weiss, and A. Torralba, "Semi-supervised learning in gigantic image collections," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[5] B. Fritzke, "A growing neural gas network learns topologies," in *Advances in Neural Information Processing Systems 7*. MIT Press, 1995, pp. 625–632.

[6] J. Graham and J. Starzyk, "A hybrid self-organizing neural gas based network," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2008, pp. 3806–3813.

[7] I. Sledge and J. Keller, "Growing neural gas for temporal clustering," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, 2008, pp. 1–4.

[8] A. Blum, J. Lafferty, M. R. Rwebangira, and R. Reddy, "Semi-supervised learning using randomized mincuts," in *In Proceedings of the 21st International Conference on Machine Learning*, 2004, pp. 97–104.

[9] A. Talwalkar, S. Kumar, and H. Rowley, "Large-scale manifold learning," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008, pp. 1–8.

[10] M. Mahdaviani, N. de Freitas, B. Fraser, and F. Hamze, "Fast computational methods for visually guided robots," in *ICRA'05*, 2005, pp. 138–143.

[11] X. Zhu and J. Lafferty, "Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning," in *Proc. Int. Conf. Machine Learning*. ACM Press, 2005, pp. 1052–1059.

[12] D. Chavez, G. Laures, K. Loayza, and R. Patino, "A stopping criteria for the growing neural gas based on a validity separation index for clusters," in *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*, 2011, pp. 578–583.

[13] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, 2003, pp. 912–919.

[14] A. Krizhevsky, "Learning multiple layers of features from tiny images," Master's thesis, 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[15] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2008.128

[16] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, pp. 145–175, 2001.