Neural Networks 146 (2022) 350–360

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Deep semi-supervised learning via dynamic anchor graph embedding in latent space

Enmei Tu [a,*], Zihao Wang [a], Jie Yang [a], Nikola Kasabov [b,c]

[a] Department of Automation, Shanghai Jiao Tong University, Shanghai, China
[b] School of Engineering, Computing and Mathematical Science, Auckland University of Technology, New Zealand
[c] Intelligent Systems Research Center, Ulster University, United Kingdom

ARTICLE INFO

Article history:
Received 2 July 2021
Received in revised form 5 October 2021
Accepted 24 November 2021
Available online 1 December 2021

Keywords:
Semi-supervised learning
Dynamic Anchor Graph Embedding
Grid-structured/graph-structured data
Image/text classification

ABSTRACT

Recently, deep semi-supervised graph embedding learning has drawn much attention for its appealing performance on the data with a pre-specified graph structure, which could be predefined or empirically constructed based on given data samples. However, the pre-specified graphs often contain considerable noisy/inaccurate connections and have a huge size for large datasets. Most existing embedding algorithms just take the graph off the shelf during the whole training stage and thus are easy to be misled by the inaccurate graph edges, as well as may result in large model size. In this paper, we attempt to address these issues by proposing a novel deep semi-supervised algorithm for simultaneous graph embedding and node classification, utilizing dynamic graph learning in neural network hidden layer space. Particularly, we construct an anchor graph to summarize the whole dataset using the hidden layer features of a consistency-constrained network. The anchor graph is used for sampling node neighborhood context, which is then presented together with node labels as contextual information to train an embedding network. The outputs of the consistency network and the embedding networks are finally concatenated together to pass a softmax function to perform node classification. The two networks are optimized jointly using both labeled and unlabeled data to minimize a single semi-supervised objective function, including a cross-entropy loss, a consistency loss and an embedding loss. Extensive experimental results on popular image and text datasets have shown that the proposed method is able to improve the performance of existing graph embedding and node classification methods, and outperform many state-of-the-art approaches on both types of datasets.

1. Introduction

Deep neural networks leveraging a large number of labeled data have achieved remarkable performance on computer vision (He, Zhang, Ren, & Sun, 2016; Tan & Le, 2019) and natural language processing (Brown et al., 2020; Devlin, Chang, Lee, & Toutanova, 2019) applications in recent years. However, labeling numerous data manually is extremely expensive or even impossible for many practical applications (e.g., medical image segmentation (Xie, Tu, Zheng, Gu, & Yang, 2021), destructive product testing (Baguley, Roy, & Watson, 2008)). In these situations, deep learning models are prone to overfitting and generalize badly on new data, due to the insufficiency of well-labeled training data. To mitigate this problem, deep semi-supervised learning (DSSL) that utilizes a small amount of labeled data plus many easily-available unlabeled data to train a model has received extensive attentions in recent years. For comprehensive reviews of traditional and recent semi-supervised learning (SSL) approaches, readers are referred to Qi and Luo (2020), Van Engelen and Hoos (2020), Zhu (2005). Among the existing DSSL algorithms, graph-based deep semi-supervised learning (GDSSL) is of particular interesting because of the solid mathematical background of graph theory, efficient matrix computational implementation and the great success of tradition graph-based SSL, etc. (Zhu, 2005). In addition, different from traditional grid-structured data (such as vectors, time series and images, which are defined by regular grid coordinates in Euclidean space), the emerged graph-structured data in many applications (such as social media data (Pitas, 2016), paper citation graph (Lu, Janssen, Milios, Japkowicz, & Zhang, 2007), which are defined by nodes and edges in Non-Euclidean space) are not in traditional Euclidean domain and the demand of new approaches to process them effectively also drives the study of GDSSL (Goyal & Ferrara, 2018; Ortega, Frossard, Kovačević, Moura, & Vandergheynst, 2018).

In a general form, GDSSL usually formulates a data classification task as a graph nodes partition problem. Recall that a

* Corresponding author.
E-mail address: hellotem@hotmail.com (E. Tu).

https://doi.org/10.1016/j.neunet.2021.11.026
0893-6080/

graph is a data structure which can be denoted as $G(V, E, W)$. It consists of a node set (or vertex set) V to represent a group of instances (samples), an edge set E to indicate the pairwise connections of the nodes, and a weight matrix W whose elements are the respective weights of the edges.[1] Given a graph G, in which only a small part of its nodes are tagged by some categorical labels, GDSSL algorithms aim to leverage the graph structural information and the correlation between the labeled and unlabeled nodes to train a deep model to partition all nodes into several disjoint subsets corresponding to the categories defined by the label nodes. This has broadly been accomplished with two strategies. The first one is based on graph similarity regularization (Smola & Kondor, 2003) (mainly graph Laplacian), which explicitly forces the model's predictions to be consistent with the graph structure (i.e. strong connected graph nodes must produce similar model outputs). For example, the algorithm described in Iscen, Tolias, Avrithis, and Chum (2019), Kamnitsas et al. (2018), Luo, Zhu, Li, Ren, and Zhang (2018) and Weston, Ratle, Mobahi, and Collobert (2012). However, a recent study shows that the asymptotic behavior of graph Laplacian regularization tends to be unstable while the graph grows to large (El Alaoui, Cheng, Ramdas, Wainwright, & Jordan, 2016). The second strategy is to transform the graph data into grid form, i.e. to embed the graph into a Euclidean space (or called embedding space, which is usually a much lower dimensional space than graph size.) and eventually using common tools in Euclidean space to classify nodes. Of course, to make the partition meaningful, the transformation needs to maximally preserve the graph structural information (e.g. subgraphs, node adjacency relationships, etc.). This has been shown to be a much more effective way to process graph structured data (Henaff, Bruna, & LeCun, 2015; Kipf & Welling, 2016; Yang, Cohen, & Salakhudinov, 2016). However, existing embedding approaches in this category mostly utilize a ***static graph***, which means the graph is given or constructed using raw data/hand-crafted features and the structure of the graph is fixed during training. This causes at least two possible limitations. First, the static graph usually contains noisy nodes/edges (especially for complex data such as natural images) which tend to mislead the learning process. Second, the learning model cannot benefit from the latest useful information extracted by itself. This can be inefficient and not consistent with our knowledge about human learning, in which past experiences play important roles while learning new things (Wagar & Dixon, 2005). Furthermore, the embedding learning is carried out on the original full graph, which is usually very large, and thus the algorithms may result in slow training and produce large-size models on large-scale datasets.

To address these issues, we propose a latent space Dynamic Anchor Graph Embedding (DAGE) semi-supervised learning approach in this paper. Our key observation is that for classification purpose, the sequential random walk on a full graph can be approximately represented by a random walk on an anchor graph. Besides, the anchor graph is constructed and updated in the hidden layer space of a neural network (i.e. the latent space). This enables us to include model extracted information on time and, meanwhile, to reduce the embedding learning problem size dramatically. To be more specific, we build a two-branch network architecture including a single-sample consistency branch and a dynamic graph embedding branch, as shown in Fig. 1. We construct a dynamic bipartite graph using hidden-layer features of the consistency branch and perform bipartite random walks on the graph to generate random sequences representing graph structural information. The random sequences are used in combination with node (pseudo) labels to optimize the embedding branch to generate more accurate node embedding features. Finally, the outputs of the two branches are aggregated together to accomplish the classification task. The single-sample consistency branch learns local information of a sample under a consistency regularization constraint, while the graph embedding branch learns global graph structural information using the context generated from both the anchor graph and the model prediction outcomes. So, the final classification results incorporate both local sample information and global graph information to generate more confident class labels.

To summarize, our main contributions are as follows:

- We propose a novel graph-based deep semi-supervised learning framework, Dynamic Anchor Graph Embedding (DAGE), for simultaneously graph embedding and node classification on both graph-structured data and grid-structured data. Different from existing approaches that usually rely on a pre-specified and noisy graph structure, DAGE builds a dynamic anchor graph in latent space and optimizes the graph structure continuously. As will be demonstrated by experiments in Section 5, this enables us to have more efficient model training and better model generalization.
- To make use of both labeled and unlabeled data, we present a new context sampling technique for semi-supervised graph embedding learning. It integrates graph structural information and model discriminant information extracted from both labeled and unlabeled data to better guide embedding and classification learning.
- We develop a new explicit weight decay technique to improve model training efficiency and prediction accuracy. Comparing with previous weight-ensembling techniques such as fast-SWA (Athiwaratkun, Finzi, Izmailov, & Wilson, 2019), the new technique could significantly enhance model diversity within a few training epochs and yield better model generalization.
- We perform extensive experiments to compare the proposed method with existing state-of-the-art methods on popular image and text benchmark datasets, as well as ablation experiments to study the effects of our new techniques. The results demonstrate the superiority of our approach on both two types of datasets and the validity of the new techniques.

2. Related works

2.1. Consistency-based models

Consistency means that given different perturbed versions of a sample (e.g. adding noises or transformations, etc.), model output should be consistent (i.e. generate same outputs), because the inputs are essentially the same thing (Laine & Aila, 2017; Rasmus, Valpola, Honkala, Berglund, & Raiko, 2015; Sajjadi, Javanmardi, & Tasdizen, 2016; Tarvainen & Valpola, 2017). In these approaches, a student model is enforced to output similar predictions to that of a teacher model when fed different perturbed versions of the same sample. Teacher model can be same as student model (Berthelot et al., 2019; Rasmus et al., 2015) or different from student model (Laine & Aila, 2017; Tarvainen & Valpola, 2017). Main perturbations include adding Gaussian noise, dropout and data augmentation for images. Later, there are two directions to improve consistency-based models. One is to improve the quality of the teacher model. For instance, Temporal Ensembling (TempEns) (Laine & Aila, 2017) calculates the temporal average values of model predictions as consistency target. MT (Tarvainen & Valpola, 2017) averages model weights

[1] A graph can also be denoted simply as $G(V, W)$, in which edge set E is implicitly represented by the non-zero elements in W.

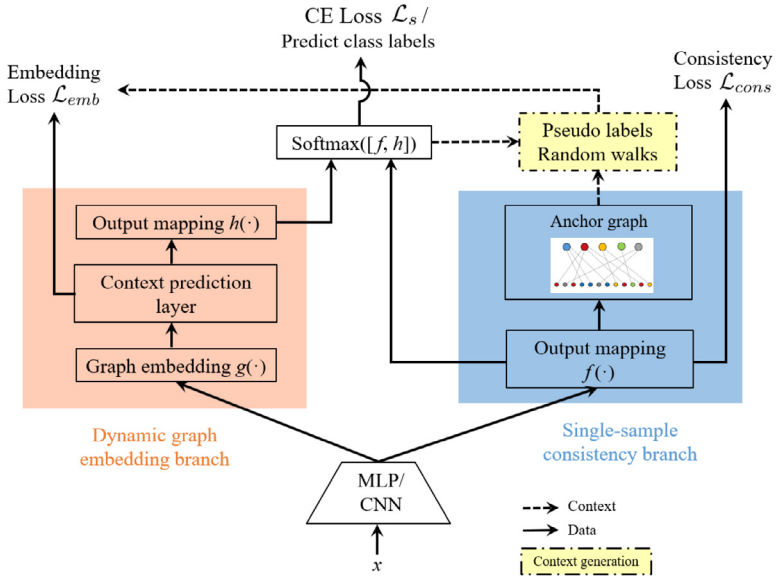


Fig. 1. The proposed neural network framework for dynamic anchor graph embedding (DAGE) learning.

instead of predictions. Fast-SWA (Athiwaratkun et al., 2019) obtains the ensemble "teacher" by averaging weights at different epochs directly. The other is to improve the quality of perturbations. For example, Adversarial perturbation is regarded as a new, effective perturbation in VAT (Miyato, Maeda, Koyama, & Ishii, 2018) and VAdD (Park, Park, Shin, & Moon, 2018); WCP (Zhang & Qi, 2020) uses additive and DropConnect perturbations and AutoAugment (Cubuk, Zoph, Mane, Vasudevan, & Le, 2019) automatically generates perturbations by learning operations in a designed search space.

2.2. Graph embedding learning

Graph embedding is to assign a unique coordinate to each graph node so that the graph can be transformed to grid data and *embedded* into a Euclidean space (called embedding space), because graph structural data are not Euclidean (i.e. not come with natural coordinates arranged in regular grids, like vectors or images). Ideally, to facilitate subsequent learning tasks such as node classification or link prediction, the embedding space should be low dimensional and the embedding transformation should maximally preserve the properties of the graph (such as topology, subgraphs, adjacency, etc.) in the embedding space.

Early graph embedding learning methods are purely unsupervised and do not leverage label information during training (e.g. Deepwalk (Perozzi, Al-Rfou, & Skiena, 2014), LINE (Tang et al., 2015)). To the best of our knowledge, Planetoid (Yang et al., 2016) firstly combines supervised classification with unsupervised graph embedding learning task and shows preferable performance on semi-supervised text recognition. Graph Convolutional Networks (GCN) recently have been shown to be effective to learn graph embeddings (Gori, Monfardini, & Scarselli, 2005; Kipf & Welling, 2016; Zhang, Tong, Xu, & Maciejewski, 2019). However, as mentioned above, most of these algorithms are designed to use "static graph", which is explicitly given or pre-constructed from coarse hand-craft features. Besides, GCN also suffers from over-smoothing effect and high computational burden on large datasets due to the nature of Laplacian graph filtering (Li, Han, & Wu, 2018).

3. Preliminaries

In this section we first define the semi-supervised learning problem. Then we briefly review the basics of consistency regularization and Planetoid (Yang et al., 2016) which are related to the proposed approach.

3.1. Semi-supervised learning

We mainly focus on semi-supervised classification problem. Suppose f_θ is a classifier parameterized by θ defined by the weights of a neural network. $\mathcal{X} = \{x_i\}_{i=1}^n$ with $x_i \in \mathcal{R}^d$ is a sample dataset for grid-structured data, or the node feature set V of the size $|V| = n$ for graph-structured data $G(V, W)$.[2] Without loss of generality, we assume the first l samples in $\mathcal{X}$ are labeled and denoted as $\mathcal{X}_L$. The rest are unlabeled and denoted as $\mathcal{X}_U$ ($l \ll n$ in most cases). $\mathcal{Y}_L = \{y_i\}_{i=1}^l$ (with $y_i \in C = \{j\}_{j=1}^c$) are categorical labels corresponding to $\mathcal{X}_L$. The binary label matrix of $\mathcal{Y}_L$ is denoted as Y, whose elements are $Y_{ij} = 1$ if and only if sample x_i is from class j, $Y_{ij} = 0$ otherwise. Semi-supervised learning is usually achieved by optimizing (1).

$$\min_\theta \sum_{i=1}^{l} \mathcal{L}_s(f_\theta(x_i), y_i) + \mathcal{L}_u(f_\theta(\mathcal{X}_L, \mathcal{X}_U)) \tag{1}$$

where $\mathcal{L}_s$ is the supervised classification loss (e.g. cross-entropy loss). $\mathcal{L}_u$ is usually the unsupervised regularization term (e.g. graph Laplacian regularization, context prediction term, etc.) which fully leverages unlabeled samples' information to prevent model from overfitting.

3.2. Consistency regularization

In consistency-based models, consistency regularization term is usually formulated as (2).

$$\mathcal{L}_{cons}(x, \theta) = \mathbb{E}_x \|f(x, \theta) - f(\tilde{x}, \theta')\|^2 \tag{2}$$

where $\tilde{x}$ is a different perturbed version of sample x, θ, θ' are model parameters of student model and teacher model, respectively. f is hidden layer feature or classification output. Except for Mean Squared Error, one can also use KL divergence to measure the perturbation difference (Kim, Oh, Kim, Cho, & Yun, 2021).

[2] To facilitate description, we will use *sample* and *node* interchangeably to mean the same thing x.

The teacher model's parameter set θ' is exponential moving average value of θ in MT (Tarvainen & Valpola, 2017), as shown in (3).

$$\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t \tag{3}$$

where θ_t is the student parameter set at time step t. α is smoothing coefficient, which is usually 0.99 or 0.999.

3.3. Graph embedding loss

Planetoid (Yang et al., 2016) trains graph embeddings to jointly predict class labels and context in the graph. Given a sample x_i, its corresponding context ct can be sampled from random walk on the graph or ground truth of labeled samples. Then, a graph embedding loss is defined in (4).

$$\begin{aligned}\mathcal{L}_{emb} = \sum_i & -\mathbb{I}(\gamma = 1)\ log\ \sigma(w_{ct}^T g(x_i)) \\ & - \mathbb{I}(\gamma = -1)\ log\ \sigma(-w_{ct}^T g(x_i))\end{aligned} \tag{4}$$

where $\mathbb{I}(\cdot)$ is indicator function that outputs 1 when the condition in parentheses is true, otherwise 0. $\gamma = 1(-1)$ means ct is positive (negative) context, w_{ct} are model weights of context prediction, $g(\cdot)$ is graph embedding. $\sigma(x) = 1/(1+\exp(-x))$ is sigmoid function, which represents the probability that x_i matches context ct. $\mathcal{L}_{emb}$ measures the cross entropy between sample pair (i, c) and binary label γ. Note that to evaluate the embedding loss for all training data, Plantoid has to define an output for each training sample to predict node context. This could be burdensome for large scale datasets. Meanwhile, the context generation only uses labeled data.

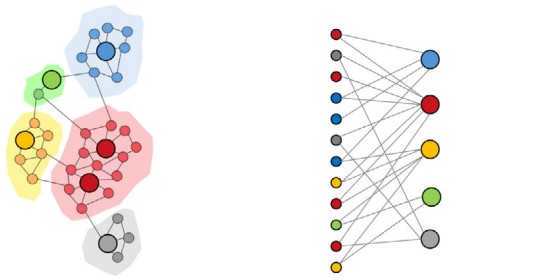
4. Dynamic anchor graph embedding

In this section, we present our Dynamic Anchor Graph Embedding (DAGE) learning approach for semi-supervised classification for both graph-structured data and grid-structured data. The key idea is to construct a dynamically updated bipartite graph in latent space to capture the model extracted information and, meanwhile, simplify graph embedding model prediction. First, we elaborate the idea and construction of dynamic anchor graph in latent space. Thereafter, we describe in detail the components of the modeling framework and training techniques, namely, the single-sample consistency branch, dynamic graph embedding branch and model optimization techniques.

4.1. Dynamic anchor graph in latent space

Traditional unsupervised graph embedding learning such as DeepWalk (Perozzi et al., 2014) first generates a node sequence (called context) for each node by fixed-length random walks and then trains a model to predict the context correctly while given a node. So the learnt embedding features could represent graph structure. While this strategy has been very successfully applied to data with a *well-defined* graph structure (e.g. social networks, paper citation graph), it generally does not perform equally well on general data tasks such as image classification, because embedding feature quality depends severely on the underlying graph quality but constructing a high quality graph (in terms of semantic relationship between connected nodes) to represent the original data is a very non-trivial task (Qiao, Zhang, Chen, & Shen, 2018).

Previous studies have shown that classification/clustering in the hidden-layer feature space is more advantageous since data distribution in that space are more explicit and thus easier to learn than that in the input data space (Kamnitsas et al., 2018; Yang, Fu, Sidiropoulos, & Hong, 2017). So, instead of predicting the context in original data space, we consider the problem of constructing an anchor graph and predicting the context of a node in latent space (i.e. the hidden-layer feature space of a neural network). We adopt a bipartite graph structure for the anchor graph to further simplify embedding feature learning. The general idea of the anchor graph construction is illustrated in Fig. 2. We first select some points (e.g. using k-means/k-medoids) from the training data and treat them as anchor points. So the whole node set could be partitioned into different cliques according to node distance to the anchor points, and two samples who are closer to each other on full graph will also be more likely to share the same anchor point. Then we construct a bipartite graph between the anchor points and the rest non-anchor points in latent space (described in next subsection). Then we perform fixed-length random walks on the bipartite graph to obtain the context of each non-anchor node. Different from previous methods such as DeepWalk (Perozzi et al., 2014) or Plantoid (Yang et al., 2016) which perform random walks and context prediction on the full graph, the bipartite random walks always pass through some anchor nodes while jumping between non-anchor nodes. Therefore, the anchor nodes are *transition hubs* for the random walkers. The sequence of visited anchor nodes can be seen as a sampled subsequences and a good summarization of the full random walk sequences. It can be served as an approximation of the original node context on full graph. So, rather than training an embedding model to predict the whole random walk sequences, we can train a model to predict the anchor node sub-sequences. This enables us to reduce the context prediction problem from full graph size to anchor node set size, which can be adjusted flexibly and is much smaller.



(a) Full graph in latent space (b) Bipartite graph representation

Fig. 2. Anchor graph in latent space. Different colors indicate different cliques and big dots are anchors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

To implement DAGE, we propose a neural network framework in Fig. 1. The feature extraction network (e.g. a CNN or MLP) maps an input sample x into hidden feature space, followed by a consistency branch and an embedding branch in parallel. The consistency branch enforces its output to be invariant to input perturbations (i.e. noise and transformations on x) and thus learns sample-wise local information. The outputs of $f(\cdot)$ are used to construct an anchor graph as shown in Fig. 2. The embedding branch learns graph node embeddings using an augmented context containing both random walks on the anchor graph and model generated labels. So the learnt embedding features contain both graph structural information and node categorical information. The outputs of the two branches are concatenated to pass a softmax function to obtain prediction labels. By encouraging output consistency of the same sample, single-sample local features are optimized but global features, which imply relationships between different samples on the graph, are not considered

in previous consistency-based models, and vice versa for existing graph embedding learning methods. In our framework, we combine local sample consistency and global graph embedding together to make use of their complementary characteristics.

It is worth mentioning that recent studies in neurology find that the visual system of mammals (or more broadly, vertebrates) also has a two-branch architecture (namely, the retinogeniculate visual pathway and retinotectal visual pathway), which is fundamentally important to visual perception (Knudsen, 2020). For different species, the relative importance of the two branches (in terms of the amount of visual information processed) is different, but generally, there is a functional and architectural similarity across all vertebrates. For example, in primates, the retinogeniculate pathway processes the majority of the visual information that is mainly for object discrimination (such as color, contrast, shape), and the retinotectal pathway analyzes spatial relationships of visual objects and generates attention information to select the target for eye's focus. The visual information from the two pathways converges in the forebrain to support visual perception (Knudsen, 2020). Interestingly, our framework in Fig. 1 has a very *similar* two-branch architecture, and the two branches also play strikingly *similar* division-cooperation roles in terms of information processing functionality. More specifically, the consistency branch learns the invariant discriminant information of an input sample (the visual target), and the embedding branch extracts the contextual information of the sample on the graph (spatial attention information). The information from the two branches is subsequently merged together for sample classification (target recognition). While the mechanism behind the similarity is very interesting, it is out of the scope of this paper and we leave it for our future work. In the following subsections, we will explain each part in detail.

4.2. Single-sample consistency branch

Consistency has been an important technique in many state-of-the-art DSSL algorithms (Berthelot et al., 2019; Luo et al., 2018; Sohn et al., 2020). We apply the same consistency constraint and model ensembling technique for $f(\cdot)$ as in (2). Notice that ν is a subset of parameter θ (i.e. the consistency branch).

$$\begin{aligned}\mathcal{L}_{cons} &= \mathbb{E}_x \|f(x,\nu) - f(\tilde{x},\nu')\|^2 \\ \nu'_t &= \alpha\nu'_{t-1} + (1-\alpha)\nu_t \end{aligned} \tag{5}$$

To construct the anchor graph, we denote $\mathcal{U} = \{u_j\}_{j=1}^m$ as anchoring point set of size m in the input space. We then calculate the anchor graph similarity matrix Z at each training step using normalized features $f_{norm}(\cdot)$ by (6). Notice that the elements of Z are dynamically adjusted during training, which enables the model to utilize newer and more accurate connections in latent space than that of the predefined static graph.

$$\begin{aligned} Z_{ij} &= \begin{cases} \frac{K_\beta(f_{norm}(x_i), f_{norm}(u_j))}{\sum_{j'\in\langle i\rangle} K_\beta(f_{norm}(x_i), f_{norm}(u'_j))}, & \forall j \in \langle i\rangle \\ 0, & \text{otherwise} \end{cases} \\ f_{norm}(x_i) &= \frac{f(x_i)}{(\sum_k (f(x_i)_k)^2)^{\frac{1}{2}}} \end{aligned} \tag{6}$$

where $Z \in \mathcal{R}^{n\times m}$ denotes the relationship between samples $\mathcal{X}$ and anchors $\mathcal{U}$. $f_{norm}(\cdot)$ is normalized with the output feature of single-sample consistency branch, which is denoted as $f(x)$ for symbolic simplicity. $K_\beta(x,y) = \exp(-\|x-y\|^2/2\beta^2)$ is Gaussian kernel function with a coefficient hyperparameter β. $\langle i\rangle \subset [1:m]$ is the s nearest anchors' index set of sample x_i. Since we only consider s closest anchors to x_i and $s \ll m$, the similarity matrix Z is highly sparse. An anchor graph representation of our method is shown in Fig. 3.

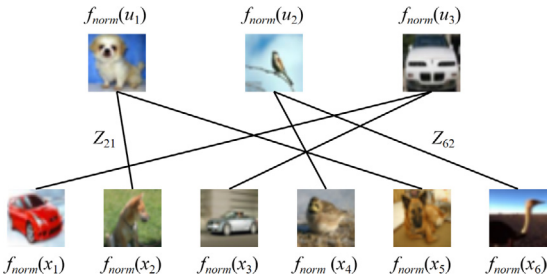


Fig. 3. An anchor graph representation of samples $x_1, \ldots, x_6$ and anchors u_1, u_2, u_3 with normalized features $f_{norm}(\cdot)$, Z_{ij} captures the relationships between samples and anchors.

The construction process of matrix Z formulated in (6) is based on features learned by the model. For some datasets (e.g. citation graph data), if a pre-existed graph is given, we could make use of the given graph structure by firstly constructing the matrix Z as in (7).

$$Z_{ij} = \begin{cases} \frac{W_{ij}}{\sum_{l\in\mathcal{U}} W_{il}}, & \text{if } j \in \mathcal{U} \wedge \exists\, l \in \mathcal{U}, W_{il} \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

where W_{ij} denotes the edge weight of the given graph. Then, we use matrix Z defined above to pre-train our model and thereafter replace it with (6) to establish new connections between samples and anchors in latent space.

4.3. Dynamic graph embedding branch

In this branch, the anchor graph constructed in previous subsection is used to learn graph embedding dynamically. To do so, two types of context (positive context ct^p and negative context ct^n, respectively) can be sampled based on the anchor graph and pseudo labels jointly. Finally, we could calculate the graph embedding loss to train graph embeddings.

More specifically, we treat m anchors as context space $\mathcal{C}$ (target space of models) that graph embeddings need to predict. Consequently, the dimension of context prediction layer connected with graph embedding layer $g(\cdot)$ is reduced from $b\times n$ to $b\times m$, where b is batch size. Since the number of anchor points m is a fixed hyperparameter, the width of context prediction layer is fixed, no longer increasing with the data size. Therefore our approach is more suitable to large-scale datasets.

A key step of graph embedding learning is to generate the context of a node. Different from existing context generation strategy as in Planetoid (Yang et al., 2016), we propose an improved context sampling strategy based on anchor graph structure and pseudo labels that makes use of both labeled and unlabeled data. The anchor graph similarity matrix Z in (6) and (7) represents the probability p of a random walker moving from a sample x to anchor u. We select simultaneously the positive contexts from nearest anchors and the negative contexts from farthest anchors according to Z. In this case, the embedding not only pull similar nodes closer, but also push dissimilar nodes farther. We also further utilize pseudo labels of all samples instead of just the labeled samples to augment context capacity. This helps exploit the full training to provide much richer discriminant information to guide the embedding learning process.

All of the above lead to our context generation strategy given in Algorithm 1. Given a sample x_i, there are two types of context in the algorithm that need to be sampled, i.e. ct^p and ct^n representing the positive and negative context, respectively. Each type of context can be generated in two ways (represented by $\gamma = +1$ and $\gamma = -1$, respectively):

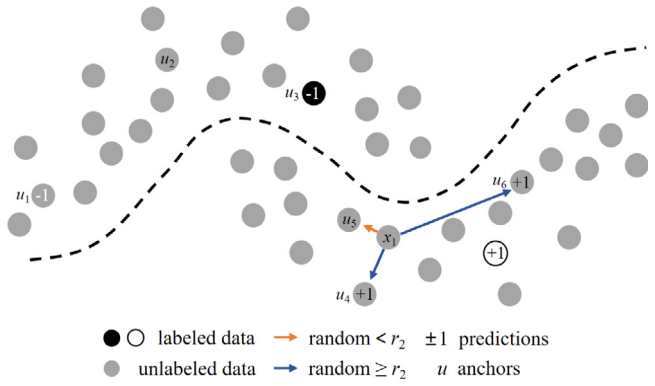


Fig. 4. The strategy of sampling ct^p on "two moons" dataset. The figure illustrates two different sampling ways based on Z and pseudo labels.

- Based on the anchor graph similarity matrix Z, which encodes the underlying graph structural information of data.
- Based on the (pseudo) labels of all training data, which injects the model discriminant information into the graph embeddings.

Algorithm 1: Sampling Context Triplet (x_i, ct, γ).

Require: anchor graph similarity matrix Z, pseudo labels $\{\tilde{y}_i\}_{i=1}^n$, number of neighbors s and non-neighbors o, hyperparameters r_1, r_2
  if *random* $< r_1$ **then**
    $\gamma = +1$
  else
    $\gamma = -1$
  end if
  if *random* $< r_2$ **then**
    Choose s closest anchors to sample x_i, uniformly sample positive context ct^p according to $Z_{ij}, j \in \langle i \rangle$
    if $\gamma = -1$ **then**
      Choose o farthest anchors to sample x_i, uniformly sample negative context ct^n
    end if
  else
    if $\gamma = +1$ **then**
      Uniformly sample positive context ct^p with $\tilde{y}_i = \tilde{y}_{ct}$
    else
      Uniformly sample negative context ct^n with $\tilde{y}_i \neq \tilde{y}_{ct}$
    end if
  end if
  return triplet (x_i, ct^p, γ) or (x_i, ct^n, γ)

The model leverages both of the two types of contextual information when training embeddings. The ratio of positive and negative contexts is controlled by parameter $r_1 \in (0, 1)$, and the ratio of two sampling ways is controlled by parameter $r_2 \in (0, 1)$. The algorithm's sampling strategy on the well-known "two moons" synthetic dataset is illustrated in Fig. 4 (in the case of sampling ct^p).

Given a triplet (x_i, ct, γ), we could use graph embedding loss in (4) to train the dynamic graph embedding branch. In this way, the global features which represent relations between samples and anchors are learned because neighboring positive sample-context pairs are encouraged to be closer and negative sample-context pairs are pushed farther. We would mention that since the context space $\mathcal{C}$ is composed of anchor set $\mathcal{U}$ here, (4) could also be regarded as a multi-label classification loss with m labels. As will be shown in experiments, the improved context generation strategy could significantly enhance model performance.

4.4. Model optimization process

As elaborated above, the two branches are trained to learn sample-perturbed local features and the graph-structured global features, respectively. Then, we concatenate the outputs of $f(\cdot)$ and $h(\cdot)$ to predict the class labels of the input sample x. The overall semi-supervised objective function is defined in (8).

$$\mathcal{L} = \mathcal{L}_s(f_\theta(x), y) + \lambda_1 \mathcal{L}_{emb} + \lambda_2 \mathcal{L}_{cons} \tag{8}$$

where $\mathcal{L}_s$ is supervised cross-entropy (CE) loss, and λ_1, λ_2 are balancing coefficients to trading off supervised and unsupervised loss terms. Given a batch of samples contains labeled and unlabeled samples, we could compute all loss terms for a labeled sample and the last two unsupervised loss terms for an unlabeled sample. Then we update model weights using gradient descent algorithms.

To improve learning efficiency, we introduce a new technique to increase the diversity of model weights directly. Fast-SWA (Athiwaratkun et al., 2019) shows that weight diversity of the student model is crucial for model generalization. However, it applies the averaging strategy to epochs with a large training interval in order to maintain the diversity of student model parameters. Therefore it needs longer training stage to integrate more diverse models. As a result, the training process of fast-SWA is very slow. Here, we propose an explicit weight decay model ensembling method calculated by (9).

$$\theta_t = (1 - \epsilon)\theta_{t-1} \tag{9}$$

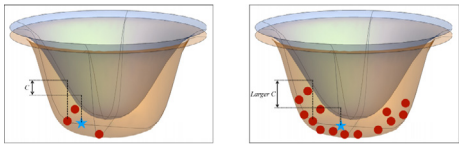
where ϵ is a coefficient hyperparameter. Note that it is decoupled with learning rate when using SGD optimizer (Bottou, 2010) and different from L2 regularization when applying Adam optimizer (Kingma & Ba, 2015). Comparing with (Loshchilov & Hutter, 2019), we remove the scaling factor η_t, aiming to add a larger penalty that greatly improves the parameters' diversity. Eq. (9) could be applied to the online teacher model ensembling. At each training step, we first take a gradient descent step on θ_{t-1}. Then we use (9) to update the parameters of student model directly. Finally, we apply (3) to integrate the teacher model with more diverse "students" generated by (9). Notice that we apply (3) and (9) to the whole parameter set θ, which could increases the quality of consistency target $f(\tilde{x}, \nu')$ in (5).

To understand why the new method could improve parameters' diversity of the student model at each iteration and enhance training efficiency, the performance gains of fast-SWA and our proposed method with the same training length are depicted in Fig. 5. In experiment we will show that the ensemble teacher model's performance and training efficiency is dramatically improved by combining (9) and (3).

Given these, the training process are as follows. Firstly, taking a randomly selected batch of samples and m anchors as network's input,[3] we could obtain their output features and pseudo labels $\tilde{y}$. Then we sample corresponding triplets (x_i, ct, γ) based on (6) and Algorithm 1. Finally we take the batch of samples as input again and calculate loss $\mathcal{L}$ in (8) to update the parameters of the student model and the teacher model.

Optimizing a deep neural network is always a challenging task due to its highly nonlinear, non-convex properties in high dimensional space, as well as other factors such as modeling

[3] Here, we simply select m samples randomly from training data to make the anchoring point set and the rest samples are treated as non-anchoring points, because k-means or k-medoids cost much computations on large dataset but the final results are generally similar.



(a) Fast-SWA [24] (b) Ours

Fig. 5. Train error surface (orange) and test error surface (blue) (approximately convex refer to Goodfellow and Vinyals (2015)). Red dots depict different model weights, blue star represents the ensemble model. C denotes the performance gain. Our method could integrate more diverse models than fast-SWA with the same training length to achieve a larger C. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Details of image datasets.

Dataset	Train	Test	Classes	Features
SVHN	73257	26032	10	3072
CIFAR10	50000	10000	10	3072

errors and various uncertainties (Tao, Li, Chen, Stojanovic, & Yang, 2020). Therefore, a large number of local minima may exist and an improper optimization may lead the network to converge to a bad local minima (suboptimal) which hurts the model stability and generalization ability substantially (Safran & Shamir, 2018; Swirszcz, Czarnecki, & Pascanu, 2016). Fortunately, recent studies show that the number of bad local minima does not increase with the network depth and under some mild conditions (i.e. no bottleneck hidden layers and convex loss function on network output logits), the quality of all local minima can be equally as good as the global optimal one (Laurent & Brecht, 2018; Lu & Kawaguchi, 2017). Most modern deep neural networks models (including ours) meet these conditions (e.g. Kawaguchi and Bengio (2019)). We will also show this in the following experiments.

5. Experiments

In this section, we conduct experiments on two image datasets (SVHN (Netzer et al., 2011) and CIFAR-10 (Krizhevsky, 2009)) and three text datasets (Citeseer, Cora and Pubmed (Sen et al., 2008)) to demonstrate the effectiveness of our approach. We compare our results with popular methods, including recently proposed state-of-the-art methods.

5.1. Datasets and preprocessing

Image datasets. The image datasets are summarized in Table 1. In SVHN, there are 73257 training samples and 26032 test samples of size $32\times32\times3$. Each sample is any digit from 0 to 9. We scale each image to zero mean and unit variance. CIFAR-10 has a training set of 50000 samples and a test set of 10000 samples, with the same size as SVHN. There are also 10 classes including dog, bird, car, etc. We normalize the samples in CIFAR-10 by per-channel standardization. In addition, we perform standard data augmentation for SVHN and CIFAR-10 (2-pixel random translation on both datasets and random horizontal flip on CIFAR-10).

Text datasets. In text datasets, sample features are bag-of-words representations of documents (i.e., binary vectors that represent corresponding words whether appear or not in Citeseer and Cora, TF-IDF vectors in Pubmed). Each dataset comes with a well-defined graph, in which edges are citation links between the documents. We set $W_{ij} = W_{ji} = 1$ in (7) if document x_i cites x_j. The information of text datasets is listed in Table 2.

Table 2
Details of text datasets.

Dataset	Train	Test	Classes	Features	Edges
Citeseer	2312	1000	6	3703	4732
Cora	1708	1000	7	1433	5429
Pubmed	18717	1000	3	500	44338

5.2. Implementation details

For image datasets, we evaluate the test error rate with different number of labels. The labeled samples are randomly selected with a random seed. Following the common practice (Berthelot et al., 2019; Laine & Aila, 2017; Tarvainen & Valpola, 2017), we run the model 5 times with different seeds to report the mean and standard deviation of the test errors. We adopt the commonly used "CNN-13" architecture (Athiwaratkun et al., 2019; Laine & Aila, 2017; Tarvainen & Valpola, 2017; Zhang & Qi, 2020) as the feature extractor module in Fig. 1 for fair comparison. $f(\cdot)$ is obtained by global average pooling of the feature extractor output. The parameter settings of image datasets are tuned by cross validation and listed in Table 3. Specifically, the output dimension of both $f(\cdot)$ and $h(\cdot)$ is $b\times6\times6\times128$, which is determined by the architecture of CNN-13 model. To be consistent with existing works (e.g. Berthelot et al. (2019), Laine and Aila (2017), Tarvainen and Valpola (2017)), we do not change it. We choose a large output of $g(\cdot) = 1024$ to make sure the embedding feature vectors have enough expressiveness for the dynamic graph embedding. The number of anchors $m = 500$ is chosen to make a trade-off between performance and computational cost, as described in Section 5.4. s and o are rather flexible parameters and we just set empirical values based on several trials. β is tuned by cross validation. Context ratio $r_1 = 0.5$ is to attach equal importance to positive and negative contexts. $r_2 = 0.2$ means 80% of context information comes from model discriminant information (i.e. pseudo labels, the other 20% is graph structural information Z), for it is a classification task and class information should be dominant. Regularization coefficients λ_1 and λ_2 are tuned by cross validation ($\lambda_2 = 1.5$ for SVHN and $\lambda_2 = 8$ for CIFAR-10). Training batch parameters b and b_l are same as that in Athiwaratkun et al. (2019). To our experiences, they usually do not need tuning and empirical values between [50, 200] work fine. We also do not tune the weight decay ϵ and just use the one from Athiwaratkun et al. (2019), since it works fine. We choose Adam optimizer because it is one of the most popular optimizers in Deep Learning community and is able to achieve good results in most cases (Choi et al., 2019). We adjust the default learning rate of Adam (0.001) a bit higher to speed up training at the earlier training stage. Training length can be longer, but we found 400 epochs already work fine. Since we randomly select a batch of samples from the training set, the number of labeled samples in a batch varies. So, we define a epoch when the batches traverse all training samples. Because the model is inaccurate at early training stage, we ramp up λ_1, λ_2 and learning rate from 0 to their maximum values at the first 80 epochs. And the learning rate is ramp down to 0 at the last 50 epochs. We tune the parameters following the technique in Oliver, Odena, Raffel, Cubuk, and Goodfellow (2018).

For text datasets, we use the same data splits as in Yang et al. (2016). Specifically, we randomly select 20 samples from each class as labeled data, 1000 samples as test data, and the rest remain as unlabeled data. We compute the average accuracy here. Notice that our approach is inductive (i.e., model could predict unobserved samples directly when testing) similar to Planetoid-I (Yang et al., 2016). We use direct connections as input feature extractor module in Fig. 1 because we find complex CNN/MLP lead to overfitting easily. The parameter settings of text datasets

Table 3
Parameter settings of image datasets.

Parameters	Value
Dimension of $f(\cdot)$	$b\times 128$
Dimension of $g(\cdot)$	$b\times 1024$
Dimension of $h(\cdot)$	$b\times 128$
Number of anchors m	500
Neighbors and non-neighbors (s, o)	(5, 100)
Gaussian kernel parameter β	1.0
Ratios for context generation (r_1, r_2)	(0.5, 0.2)
Regularization coefficients λ_1, λ_2	0.2, (1.5, 8.0)
Batch size and labels in a batch (b, b_l)	(100, 'vary')
Explicit weight decay ϵ	0.0001
Optimizer	Adam
Max learning rate	0.003
Training length (epochs)	400

Table 4
Parameter settings of text datasets.

Parameters	Value
Dimension of $f(\cdot)$ for 3 datasets	6, 7, 3
Dimension of $g(\cdot)$	$b\times 20$
Dimension of $h(\cdot)$ for 3 datasets	6, 7, 3
Number of anchors m	200
Neighbors and non-neighbors (s, o)	(5, 50)
Gaussian kernel parameter β	1.0
Ratios for context generation (r_1, r_2)	(0.5, 0.2)
Regularization coefficients λ_1, λ_2	3.25, 0
Batch size and labels in a batch (b, b_l)	(100, 20)
Explicit weight decay ϵ	0.00005
Optimizer	SGD, Momentum=0.5
Max learning rate	0.1
Training length (iterations)	30000

are tuned by cross validation and listed in Table 4.[4] The motivations of parameters tuning in this table are similar to that in the previous Table 3. Particularly, we reduce the networks' dimension and the number of anchors using cross-validation tuning, because text datasets are smaller ones than image datasets, so large-capacity networks may easily overfit the data. We also set $\lambda_2 \equiv 0$ since there is no direct way to perturb a graph, so the dynamic anchor graph embedding loss $\mathcal{L}_{emb}$ acts as the only unsupervised regularization of the model. Similarly, we ramp up λ_1 from 0 to its maximum value at the first 4000 iterations. Besides, we do not use data augmentation on text datasets for fair comparison. Thus $\mathcal{L}_{cons}$ hardly works ($|f(x, v) - f(x, v')| \to 0$).

5.3. Image classification results

We perform experiments on SVHN with 250, 500, 1000 labels and CIFAR-10 with 1000, 2000, 4000 labels respectively. The number of labeled samples are balanced referring to other DSSL algorithms. We compare our method with recently state-of-the-art consistency-based DSSL models (i.e., Π model (Laine & Aila, 2017), TempEns (Laine & Aila, 2017), MT (Tarvainen & Valpola, 2017), VAT (Miyato et al., 2018), VAdD (Park et al., 2018), fast-SWA (Athiwaratkun et al., 2019), WCP (Zhang & Qi, 2020)) and GDSSL models (i.e., LPDSSL (Iscen et al., 2019), SNTG (Luo et al., 2018)). Notice that we use the same CNN model and data augmentation as these algorithms for fair comparison. And the results of LPDSSL on SVHN are reproduced by us. The test error rates are listed in Tables 5 and 6 respectively. Numbers in parentheses indicate the training length (epochs). Notably, our approach achieves state-of-the-art results on SVHN and CIFAR-10 with standard data augmentation. The error rates are 0.59%, 0.07% and 0.07% lower than WCP on SVHN and 1.58%, 0.32% and 0.04% lower than fast-SWA (1200) on CIFAR-10. The superiority of our approach becomes more obvious as the number of labels decreases. The strong performance suggests that the local output features of single-sample consistency branch and global outputs of graph embedding branch could effectively cooperate to predict class labels in our model. Besides, explicit weight decay in (9) largely improves the parameters' diversity of the student model, which in turn improves the teacher's performance. Specifically, our model is much more efficient than fast-SWA. We obtain better results with a training length of 400 than fast-SWA with 1200 epochs.

From the results, we can also see that the standard deviation of the error rates of our approach is comparable to existing approaches. Since the algorithms are run independently with randomly initialization each time, they have equal chance to converge to "good" local minima and "bad" local minima. So the standard deviations reflect the stability of the algorithms, or the effectiveness of the optimization process that may be influenced by the quality of the local minima. The results in the tables show that for most approaches (including ours), the standard deviation is one/two order smaller than error rate. This indicates that the performances of the algorithms are very stable, much less likely to converge to a bad local minima. It also experimentally confirms our discussion in Section 4.4.

The t-SNE (Maaten & Hinton, 2008) visualizations of our method and Planetoid (Yang et al., 2016) on CIFAR-10 with 4000 labels are depicted in Fig. 6. From the figure one can see that in our method, different classes are better separated than Planetoid.

We also provide ablation studies on CIFAR-10 with 4000 labels to verify the effectiveness of explicit weight decay and the combination of two sampling branches, which is shown in Tables 7 and 8 respectively.

Table 7 indicates that as ϵ increases, the prediction ability of student model is reduced to some extent. However, the performance gains of teacher model greatly improves since the parameters' diversity of the "student" increases. And Table 8 shows that combination of two sampling branches makes the model generalize better.

5.4. Text classification results

We compare our approach with three inductive graph-based semi-supervised learning approaches including manifold regularization (MR) (Belkin, Niyogi, & Sindhwani, 2006), semi-supervised embedding (SSEmb) (Weston et al., 2012) and Planetoid-I (Yang et al., 2016). The results are displayed in Table 9. "Feat" is a baseline method, which is a linear softmax model taking the feature x as input directly.

From Table 9, we can see that our dynamic anchor graph embedding learning approach outperforms other approaches by a large margin on Citeseer and Cora dataset. Specifically, an accuracy improvement of 2.4% and 3.9% compared to Planetoid-I respectively. And on Pubmed, the performance of our model is very close to Planetoid-I (only 0.7% lower). Notice that we do not use augmentation or consistency regularization. All of these demonstrate that dynamic graph could discover new, accurate connections between samples and anchors.

We also conduct parameter sensitivity experiment with different number of anchors on Citeseer dataset, as presented in Fig. 7. When the number of anchors is small, they cannot cover the whole dataset, which has a bad influence on model performance. However, as the anchor size m increases, the performance gradually improves and finally tends to be saturated, which conforms to the statement in the above section.

[4] We use a different "o" in vision tasks because there are more samples on image datasets.

Table 5
Test error rates (%) on SVHN.

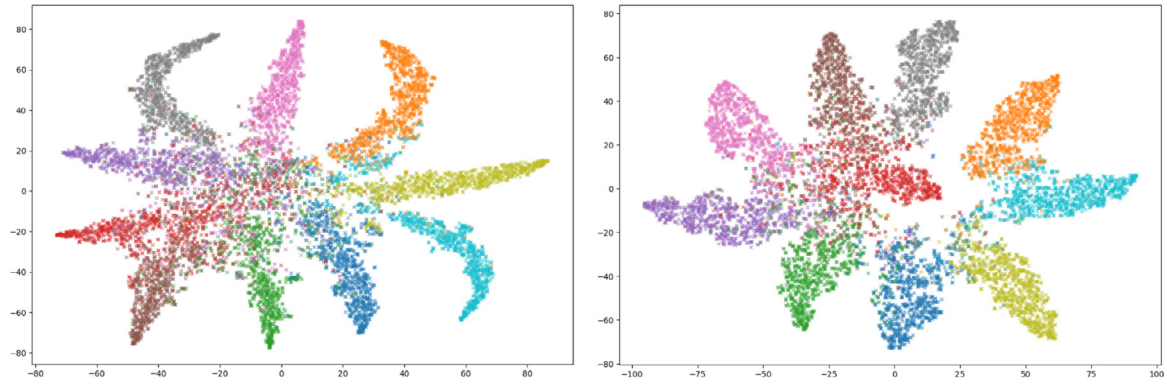
Method	250 labels	500 labels	1000 labels
Π model (Laine & Aila, 2017)	9.93 ± 1.15	6.65 ± 0.53	4.82 ± 0.17
TempEns (Laine & Aila, 2017)	12.62 ± 2.91	5.12 ± 0.13	4.42 ± 0.16
MT (Tarvainen & Valpola, 2017)	4.35 ± 0.50	4.18 ± 0.27	3.95 ± 0.19
VAT (Miyato et al., 2018)	–	–	5.42 ± 0.22
VAdD (Park et al., 2018)	–	–	4.26 ± 0.14
WCP (Zhang & Qi, 2020)	4.29 ± 0.10	3.75 ± 0.11	3.58 ± 0.19
LPDSSL (Iscen et al., 2019)	18.45	9.49	7.38
Π+SNTG (Luo et al., 2018)	5.07 ± 0.25	4.52 ± 0.30	3.82 ± 0.25
TempEns+SNTG (Luo et al., 2018)	5.36 ± 0.57	4.46 ± 0.26	3.98 ± 0.21
Ours[a] (400)	**3.70 ± 0.21**	**3.68 ± 0.24**	**3.51 ± 0.12**

[a]Teacher model.

Table 6
Test Error Rates (%) on CIFAR-10.

Method	1000 labels	2000 labels	4000 labels
Π model (Laine & Aila, 2017)	31.65 ± 1.20	17.57 ± 0.44	12.36 ± 0.31
TempEns (Laine & Aila, 2017)	23.31 ± 1.01	15.64 ± 0.39	12.16 ± 0.24
MT (Tarvainen & Valpola, 2017)	21.55 ± 1.48	15.73 ± 0.31	12.31 ± 0.28
VAT (Miyato et al., 2018)	–	–	11.36 ± 0.34
VAdD (Park et al., 2018)	–	–	11.32 ± 0.11
WCP (Zhang & Qi, 2020)	17.62 ± 1.52	11.93 ± 0.39	9.72 ± 0.31
fast-SWA (480) (Athiwaratkun et al., 2019)	16.84 ± 0.62	12.24 ± 0.31	9.86 ± 0.27
fast-SWA (1200) (Athiwaratkun et al., 2019)	15.58 ± 0.12	11.02 ± 0.23	9.05 ± 0.21
LPDSSL (Iscen et al., 2019)	22.02 ± 0.88	15.66 ± 0.35	12.69 ± 0.29
Π+SNTG (Luo et al., 2018)	21.23 ± 1.27	14.65 ± 0.31	11.00 ± 0.13
TempEns+SNTG (Luo et al., 2018)	18.41 ± 0.52	13.64 ± 0.32	10.93 ± 0.14
Ours[a] (400)	**14.00 ± 0.49**	**10.70 ± 0.43**	**9.01 ± 0.24**

[a]Teacher model.



(a) Planetoid (b) Ours

Fig. 6. t-SNE Visualizations of graph embeddings of our model and Planetoid on the test set of CIFAR-10 respectively. Each color denotes a class. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 7
Ablation studies with different ϵ.

Explicit weight	Test error rates	
Decay coefficient	*Teacher Model*	*Student Model*
$\epsilon = 0$	11.51	12.53
$\epsilon = 0.00005$	9.46	10.63
$\epsilon = 0.0001$	**8.77**	19.07

Table 8
Ablation studies with different sampling branches.

Sampling branch	Test error rates
Only based on matrix Z	9.32
Only based on pseudo labels	8.96
Combination of two branches	**8.77**

Table 9
Average classification accuracy (%) on text datasets.

Method	Citeseer	Cora	Pubmed
Feat	57.2	57.4	69.8
MR (Belkin et al., 2006)	60.1	59.5	70.7
SSEmb (Weston et al., 2012)	59.6	59.0	71.1
Planetoid-I (Yang et al., 2016)	64.7	61.2	**77.2**
Ours	**67.1**	**65.1**	76.5

6. Computational complexity

To evaluate the computational cost of our DAGE method, we conduct experiments to compare the running time of DAGE with other state-of-the-art methods. Specifically, two methods are chosen as the baselines, the fast-SWA (Athiwaratkun et al., 2019) and the Planetoid (Yang et al., 2016), because they are the

Table 10
Computational time (Time Unit: s).

Data type	Method	Overall time	Per Epoch
Image Data (CIFAR-10)	fast-SWA (Athiwaratkun et al., 2019)	54939.03	305.22
	Ours	**31632.51**	**79.08**
Text Data (Citeseer)	Planetoid-I (Yang et al., 2016)	338.32	1.13
	Ours	**251.08**	**0.84**

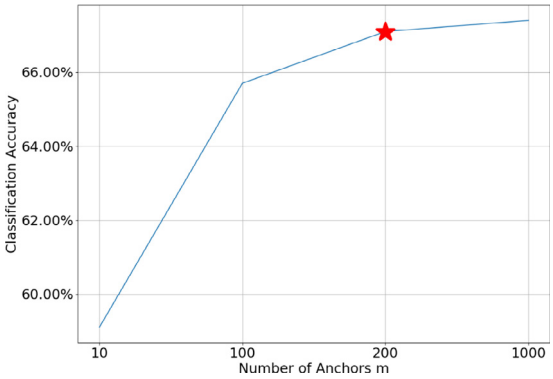


Fig. 7. Parameter sensitivity experiment with different number of anchors m on text dataset. We set $m = 200$ (star) in our text classification experiment.

most closely related to our DAGE. We compare the running time of DAGE with that of fast-SWA on CIFAR-10 dataset for image classification with 4000 labels. Similarly, we compare the running time of DAGE with that of Planetoid on Citeseer dataset for text classification with 120 labels. For both experiments, DAGE uses the same training setting as its counterpart (e.g. same batch size, training/testing split). We run all algorithms on a computer with a GeForce RTX 2080Ti GPU. The experimental results are shown in Table 10. The "Overall Time" column means the total time from training beginning to end and the "Per Epoch" time is the overall time divided by training epoch number (for text data it is the time of 100 iterations because Planetoid does not have an epoch-wise training process).

From these results, we can see that for the image dataset, fast-SWA takes almost twice long overall time as that of our DAGE, and its per epoch time is nearly four times longer than DAGE, because it needs a large training interval to maintain model diversity. For the text dataset, the anchor graph design in our DAGE could significantly reduce the size of the graph embedding network. To embed a graph $G(V, E)$ with $|V| = n$ into a d' dimensional space, the output layer size of the embedding network in Planetoid is $d' \times n$, while in our DAGE framework the size is $d' \times m (m \ll n)$, which is much smaller and does not grow with graph size. This indicates that the computational advantage can be even more prominent on large graph embedding.

7. Conclusions and discussions

Different DSSL algorithms have been developed to classify either grid-structured data or graph-structured data (Goyal & Ferrara, 2018; Qi & Luo, 2020; Wang, Tu, Zhou, & Yang, 2020). In this paper, we proposed a dynamic anchor graph embedding learning algorithm which is capable of classifying both types of data and is verified on image and text data classification. The key characteristic of our model is a dynamically updated anchor graph in latent space that comes with several benefits. First, the graph in latent space could utilize the abstract and semantically meaningful features to define its connections more accurately than use raw data or hand-crafted features. Second, dynamic updated graph could use latest model extracted information to improve graph quality and hence the node embedding feature quality. Third, the bipartite graph structure enables us to use anchors as the context to simplify context prediction in embedding learning. The model is trained by exploiting complementary information jointly extracted by different network components, i.e. the local features from a single-sample consistency branch and global features from a dynamic graph embedding branch, to minimize a uniform SSL objective function. In addition, graph embeddings are trained learn the graph structural information and the node categorical information from both labeled and unlabeled data. After training, the model could be used for either sample classification or/and graph embedding tasks. We also introduce an explicit weight decay that largely increases student model's diversity. Its combination with model ensembling (temporal weight averaging) dramatically improves training efficiency and generalization performance. Experimental results on five benchmark datasets have demonstrated the effectiveness of our approach.

Finally, we would like to point out that the success of all semi-supervised learning in practical applications depends upon the so-called manifold assumption or cluster assumption (Chapelle, Scholkopf, & Zien, 2009; Zhu & Goldberg, 2009), which can be understood roughly as that same-class data distribute on the same smooth manifold or exhibit a cluster structure in the data space. This limitation also applies to our approach. How to test whether a given dataset meets such assumption is still an open research problem in the machine learning field. There has not been an effective way to do so yet. On the other side, for most practical applications, we usually do not need to conduct such an assumption testing since we are more concerned if semi-supervised learning approaches can perform better or not (Mey & Loog, 2019). To this end, it is sufficient to compare the performances of semi-supervised learning approaches with that of their supervised counterparts.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by NSFC China (61806125, 61802247, 61876107) and Shanghai Natural Science Foundation (19ZR1476300).

References

Athiwaratkun, B., Finzi, M., Izmailov, P., & Wilson, A. G. (2019). There are many consistent explanations of unlabeled data: Why you should average. In *I*nternational conference on learning representations.

Baguley, P., Roy, R., & Watson, J. (2008). Cost of physical vehicle crash testing. In *Collaborative product and service life cycle management for a sustainable world* (pp. 113–121). Springer.

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

Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., & Raffel, C. (2019). Mixmatch: A holistic approach to semi-supervised learning. arXiv preprint arXiv:1905.02249.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *P*roceedings of the international conference on computational statistics (pp. 177–186).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. In *Advances in neural information processing systems*.

Chapelle, O., Scholkopf, B., & Zien, A. (2009). Semi-supervised learning (chapelle, o. eds.; 2006)[book reviews]. *vol. 20*, In *IEEE transactions on neural networks* (3), (p. 542).

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. arXiv preprint arXiv:1910.05446.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. In *P*roceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 113–123).

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *P*roceedings of the 2019 conference of the north american chapter of the association for computational linguistics: human language technologies. Vol. 1 (long and short papers) (pp. 4171–4186).

El Alaoui, A., Cheng, X., Ramdas, A., Wainwright, M. J., & Jordan, M. I. (2016). Asymptotic behavior of\ell_p-based laplacian regularization in semi-supervised learning. In *C*onference on learning theory (pp. 879–906).

Goodfellow, I. J., & Vinyals, O. (2015). Qualitatively characterizing neural network optimization problems. In *I*nternational conference on learning representations.

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks, Vol. 2005* (2), (pp. 729–734). IEEE.

Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, *151*, 78–94.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *P*roceedings of the IEEE conference on computer vision and pattern recognition (pp. 770–778).

Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163.

Iscen, A., Tolias, G., Avrithis, Y., & Chum, O. (2019). Label propagation for deep semi-supervised learning. In *P*roceedings of the IEEE conference on computer vision and pattern recognition (pp. 5070–5079).

Kamnitsas, K., Castro, D., Le Folgoc, L., Walker, I., Tanno, R., Rueckert, D., et al. (2018). Semi-supervised learning via compact latent space clustering. In *I*nternational conference on machine learning (pp. 2459–2468).

Kawaguchi, K., & Bengio, Y. (2019). Depth with nonlineaty creates no bad local minima in resnets. *Neural Networks*, *118*, 167–174.

Kim, T., Oh, J., Kim, N., Cho, S., & Yun, S.-Y. (2021). Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. arXiv preprint arXiv:2105.08919.

Kingma, D. P., & Ba, J. (2015). Adam A method for stochastic optimization. In *I*nternational conference on learning representations.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

Knudsen, E. I. (2020). Evolution of neural processing for visual perception in vertebrates. *Journal of Comparative Neurology*, *528*(17), 2888–2901.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* Master's thesis, University of Toronto.

Laine, S., & Aila, T. (2017). Temporal ensembling for semi-supervised learning. In *I*nternational conference on learning representations.

Laurent, T., & Brecht, J. (2018). Deep linear networks with arbitrary loss: All local minima are global In *I*nternational conference on machine learning (pp. 2902–2907).

Li, Q., Han, Z., & Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *P*roceedings of the AAAi conference on artificial intelligence. Vol. 32.

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In *I*nternational conference on learning representations.

Lu, W., Janssen, J., Milios, E., Japkowicz, N., & Zhang, Y. (2007). Node similarity in the citation graph. *Knowledge and Information Systems*, *11*(1), 105–129.

Lu, H., & Kawaguchi, K. (2017). Depth creates no bad local minima. arXiv preprint arXiv:1702.08580.

Luo, Y., Zhu, J., Li, M., Ren, Y., & Zhang, B. (2018). Smooth neighbors on teacher graphs for semi-supervised learning. In *P*roceedings of the IEEE conference on computer vision and pattern recognition (pp. 8896–8905).

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, *9*, 2579–2605.

Mey, A., & Loog, M. (2019). Improvability through semi-supervised learning: a survey of theoretical results. arXiv preprint arXiv:1908.09574.

Miyato, T., Maeda, S.-i., Koyama, M., & Ishii, S. (2018). Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(8), 1979–1993.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *A*dvances in neural information processing systems workshops.

Oliver, A., Odena, A., Raffel, C. A., Cubuk, E. D., & Goodfellow, I. (2018). Realistic evaluation of deep semi-supervised learning algorithms. In *Advances in neural information processing systems* (pp. 3235–3246).

Ortega, A., Frossard, P., Kovačević, J., Moura, J. M., & Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, *106*(5), 808–828.

Park, S., Park, J., Shin, S.-J., & Moon, I.-C. (2018). Adversarial dropout for supervised and semi-supervised learning. In *P*roceedings of the AAAI conference on artificial intelligence.Vol. 32.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *P*roceedings of ACM SIGKDD international conference on knowledge discovery and data mining (pp. 701–710).

Pitas, I. (2016). *Graph-based social media analysis, Vol. 39*. CRC Press.

Qi, G.-J., & Luo, J. (2020). Small data challenges in big data era: A survey of recent progress on unsupervised and semi-supervised methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Qiao, L., Zhang, L., Chen, S., & Shen, D. (2018). Data-driven graph construction and graph learning: A review. *Neurocomputing*, *312*, 336–351.

Rasmus, A., Valpola, H., Honkala, M., Berglund, M., & Raiko, T. (2015). Semi-supervised learning with ladder networks. arXiv preprint arXiv:1507.02672.

Safran, I., & Shamir, O. (2018). Spurious local minima are common in two-layer relu neural networks. In *I*nternational Conference on Machine Learning (pp. 4433–4441).

Sajjadi, M., Javanmardi, M., & Tasdizen, T. (2016). Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in neural information processing systems* (pp. 1163–1171).

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., & Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, *29*(3), 93.

Smola, A. J., & Kondor, R. (2003). Kernels and regularization on graphs. In *Learning theory and kernel machines* (pp. 144–158). Springer.

Sohn, K., Berthelot, D., Li, C.-L., Zhang, Z., Carlini, N., Cubuk, E. D., et al. (2020). Fixmatch: simplifying semi-supervised learning with consistency and confidence. arXiv preprint arXiv:2001.07685.

Swirszcz, G., Czarnecki, W. M., & Pascanu, R. (2016). Local minima in training of neural networks. arXiv preprint arXiv:1611.06310.

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks In *i*nternational conference on machine learning (pp. 6105–6114).

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). Line: Large-scale information network embedding. In *P*roceedings of the 24th international conference on world wide web (pp. 1067–1077).

Tao, H., Li, J., Chen, Y., Stojanovic, V., & Yang, H. (2020). Robust point-to-point iterative learning control with trial-varying initial conditions. *IET Control Theory & Applications*, *14*(19), 3344–3350.

Tarvainen, A., & Valpola, H. (2017). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems* (pp. 1195–1204).

Van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, *109*(2), 373–440.

Wagar, B. M., & Dixon, M. J. (2005). Past experience influences object representation in working memory. *Brain and Cognition*, *57*(3), 248–256.

Wang, Z., Tu, E., Zhou, M., & Yang, J. (2020). End-to-end graph-based deep semi-supervised learning with extended graph laplacian. In *2020 chinese automation congress* (pp. 5948–5953). IEEE.

Weston, J., Ratle, F., Mobahi, H., & Collobert, R. (2012). Deep learning via semi-supervised embedding. In *Neural networks: tricks of the trade* (pp. 639–655).

Xie, Z., Tu, E., Zheng, H., Gu, Y., & Yang, J. (2021). Semi-supervised skin lesion segmentation with learning model confidence. In *ICASSP 2021-2021 ieee international conference on acoustics, speech and signal processing* (pp. 1135–1139). IEEE.

Yang, Z., Cohen, W., & Salakhudinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *I*nternational conference on machine learning (pp. 40–48).

Yang, B., Fu, X., Sidiropoulos, N. D., & Hong, M. (2017). Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *i*nternational conference on machine learning (pp. 3861–3870).

Zhang, L., & Qi, G.-J. (2020). Wcp: Worst-case perturbations for semi-supervised deep learning. In *P*roceedings of the IEEE conference on computer vision and pattern recognition (pp. 3912–3921).

Zhang, S., Tong, H., Xu, J., & Maciejewski, R. (2019). Graph convolutional networks: a comprehensive review. *Computational Social Networks*, *6*(1), 1–23.

Zhu, X. J. (2005). *1503, Semi-supervised learning literature survey*: *Technical report*, University of Wisconsin-Madison Department of Computer Sciences.

Zhu, X., & Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *3*(1) 1–130.