with the smallest d eigenvectors of the normalized Laplacian matrix of the graph.

- DeepWalk [24]: DeepWalk is a skip-gram [21] based model which learns the graph embedding with truncated random walks.
- node2vec [13]: This approach combines the advantage of breadth-first traversal and depth-first traversal algorithms. The random walks generated by node2vec can better represent the structural equivalence.
- Structural Deep Network Embedding (SDNE) [39]: SDNE is a deep learning based network embedding model which uses autoencoder and locality-preserving constraint to learn vertex representations that capture the highly non-linear network structure.
- Adversarial Network Embedding (ANE) [7]:ANE proposes to train a discriminator to push the embedding distribution to match the fixed prior.

For fair comparison [19], we run each algorithm to generate 300 dimensional vertex representations on different datasets, unless noted otherwise. The number of walks per vertex in DeepWalk and node2vec is set to 10 with walk length 30, which is the same as the random walk generation step of NetRA. The window size of DeepWalk and node2vec is optimized to 10. node2vec is optimized with grid search over its return and in-out parameters $(p, q) \in \{0.25, 0.50, 1, 2, 4\}$. For SDNE, we utilize the default parameter setting as described in [39]. For NetRA, the gradient clipping is performed in every training iteration to avoid the gradient explosion, and we use stochastic gradient descent as the optimizer of autoencoder networks. The multilayer perceptron (MLP) is used in the generator and discriminator. The evaluation of different algorithms is based on applying the embeddings they learned to the downstream tasks, such as link prediction, network reconstruction, and multi-label classification as will be illustrated in the subsequent sections.

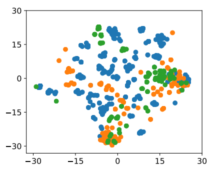
4.3 Visualization

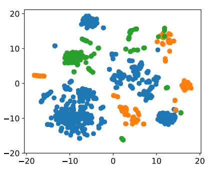
In order to demonstrate how well key properties of network structure are captured by the network embedding models, we visualize the embeddings of each compared method. We run different embedding algorithms described in Section 4.2 to obtain low dimensional representations of each vertex and map vertex vectors onto a two dimensional space using t-SNE [37]. With vertex colored by its label, we perform the visualization task on JDK dependency network, as shown in Figure 3.

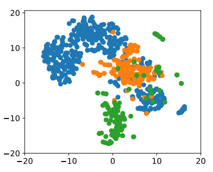
As observed in Figure 3, three classes are presented: red points for *org.omg*, green points for *org.w3c* and blue points for *java.beans*. It can be seen that the eigenvector-based method Spectral Clustering cannot effectively identify different classes. Other baselines can detect the classes to varying extents. NetRA performs best as it can separate these three classes with large boundaries, except for a small overlap between green and red vertices.

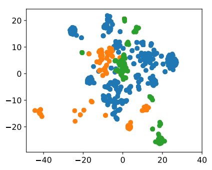
4.4 Link Prediction

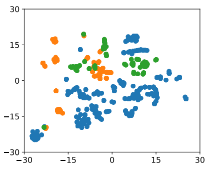
The objective of link prediction task is to infer missing edges given a network with a certain fraction of edges removed. We randomly remove 50% of edges from the network, which serve as positive samples, and select an equal number of vertex pairs without linkage between them as negative samples. With vertex representation learned by network embedding algorithms, we obtain the edge feature from the ℓ_2 norm of two vertex vectors, and use it directly to predict missing edges. Because our focus is network embedding model, this simple experimental setup can evaluate the performance based on the assumption that the representations of two connected vertices should be closer in the Euclidean space. We use the area under curve (AUC) score for evaluation on link prediction task. The results are shown in Table 2.


(a) Spectral Clustering


(b) DeepWalk


(c) node2vec


(d) SDNE


(e) ANE

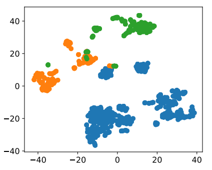

(f) NetRA

Figure 3: Visualization results of the compared methods on JDK dependency network: the red points belong to class *org.omg*; the green points belong to class *org.w3c*; the blue points belong to class *java.beans*.

Table 2: AUC score of link prediction

Method	UCI	JDK	BLOG	DBLP
SC	0.6128	0.6686	0.6014	0.5740
DeepWalk	0.6880	0.8506	0.7936	0.8605
node2vec	0.6040	0.8667	0.8105	0.8265
SDNE	0.7806	0.7226	0.6621	0.7712
ANE	0.6402	0.7409	0.7025	0.7935
NetRA	**0.8879**	**0.8913**	**0.8627**	**0.8902**

Obviously, we observe that NetRA outperforms the baseline algorithms across all datasets by a large margin. It can be seen that NetRA achieves 3% to 32% improvement based on the AUC score on the four datasets. By comparing NetRA, node2vec and DeepWalk, which all use random walks as inputs, we can see the effectiveness of generative adversarial regularization for improving the generalization performance in NetRA model. With same random walk sequences, NetRA can overcome the sparsity issue from the sampled sequences of vertices.

We also plot the ROC curve of these four datasets, as shown in Figure 4(a)-(d). The ROC curve of NetRA dominates other approaches and is very close to the (0, 1) point. We train the NetRA model with different epochs for different datasets and embed the vertices to get representations after each training epoch. The results are shown in Figure 4(e)-(h). Generally, we can observe that