Graph Harmony: Denoising and Nuclear-Norm Wasserstein Adaptation for Enhanced Domain Transfer in Graph-Structured Data

Anonymous authors
Paper under double-blind review

Abstract

Graph-structured data can be found in numerous domains, yet the scarcity of labeled instances hinders its effective utilization of deep learning in many scenarios. Traditional unsupervised domain adaptation (UDA) strategies for graphs primarily hinge on adversarial learning and pseudo-labeling. These approaches fail to effectively leverage graph discriminative features, leading to class mismatching and unreliable label quality. To address these obstacles, we developed the Denoising and Nuclear-Norm Wasserstein Adaptation Network (DNAN). DNAN employs the Nuclear-norm Wasserstein discrepancy (NWD), which can simultaneously achieve domain alignment and class distinction. It also integrates a denoising mechanism via a Variational Graph Autoencoder. This denoising mechanism helps capture essential features of both source and target domains, improving the robustness of the domain adaptation process. Our comprehensive experiments demonstrate that DNAN outperforms state-of-the-art methods on standard UDA benchmarks for graph classification.

1 Introduction

While deep learning has made substantial progress in handling graph-structured data, it shares a drawback with other methods in the same category—a heavy reliance on labeled data. This requirement presents a significant obstacle in real-world applications, where the gathering and annotating of graph-structured data come with a steep price tag, both in terms of time and resources. Obtaining detailed labels for graph-structured data, such as chemical molecules, is a considerable challenge because chemical molecules are incredibly complex, comprising a large number of atoms connected in various ways through different kinds of bonds. Collecting annotated graph-structured data like social networks is also challenging due to the need to protect personal and sensitive information and the continual changes in network relationships. The scarcity of labels makes it difficult to derive meaningful insights and hinders the development of strategies and solutions based on deep learning. Therefore, it is highly desirable to relax the need for extensive graph-structured data annotation to replicate the success of deep learning in applications.

To navigate the challenge of label scarcity, Unsupervised Domain Adaptation (UDA) (Ganin & Lempitsky, 2015) has emerged as a promising frontier, aiming to leverage labeled data from a related source domain to inform an unlabeled target domain. The challenge of label scarcity necessitates Unsupervised Domain Adaptation (UDA) due to the absence of labels in the target domain. The target domain is our primary area of interest. To overcome this challenge, we utilize a source dataset rich in labels to train our model. However, the inherent differences between the source and target datasets require the application of domain adaptation strategies. These strategies enable the effective application of models trained on the well-labeled source data to achieve high performance in the target domain, despite its lack of labels. The principle of UDA is to align the data distributions between the two domains within a common embedding space, allowing a classifier trained on the source domain to perform competently in the target domain.

While UDA has been extensively applied to array-structured data (Long et al., 2016; Kang et al., 2019), its translation to graph-structured data remains underexplored. Graph samples exhibit a wide range of structural variations, including differences in connectivity patterns, node degrees, and subgraph structures. The

primary challenge in applying UDA to graph classification is the domain shift in structural patterns, or simply, the structural variations between the source and target domain graphs. These structural variations make it challenging for models to identify and leverage invariant features across domains. Pioneering methods, such as DANE (Zhang et al., 2019a), integrate generative adversarial networks (GANs) with graph convolutional networks (GCNs) to align the domains. Others, like the approach by Wu et al. (2020), introduce attention mechanisms to reconcile global and local consistencies, again employing GANs for cross-domain node embedding extraction. However, these GAN-based methods have the drawback of class mismatching, lacking clear separability between features from different classes, as they align target and source domain features irrespective of their classes. In addition, these methods are designed for node classification. The UDA strategy for graph classification has not been well explored.

In this paper, we focus on the UDA setting for graph classification. We propose the Denoising and Nuclear-Norm Wasserstein Adaptation Network (DNAN) to address the primary challenges in graph UDA tasks and problems in previous GAN-based methods. Our DNAN benefits from the denoising mechanism with a Variational Graph Autoencoder (VGAE) and the Nuclear-Norm Wasserstein Discrepancy. By leveraging the Nuclear-Norm Wasserstein Discrepancy, it tackles the class mismatch issue in existing graph-based UDA methods. Unlike previous GAN-based methods, DNAN performs a refined, class-specific alignment of source and target domain distributions within a shared embedding space, preserving the distinct separability of features across classes. The inclusion of the denoising mechanism is motivated by the structural variations between source and target domain graphs for the graph UDA setting. The denoising mechanism of VGAE reconstructs clean adjacency matrices from corrupted versions. This process forces the model to learn robust features that are more invariant to structural variations and helps the model focus on the underlying structure and features relevant to the classification task. Thus, we believe the denoising mechanism could help handle the domain shift in UDA tasks for graph classification. By using these two components, DNAN performs competitively and achieves state-of-the-art performance on major UDA benchmarks for graph classification.

Our contributions mainly lie in applying existing techniques to a new problem and introducing an effective combination of existing approaches. Our first contribution is translating the denoising criteria of variational autoencoders to variational graph autoencoders and applying denoising techniques to address the domain shift in structural patterns in the graph UDA problem. This translation and utilization of the denoising mechanism are not trivial. We believe we are the first to discuss the denoising criteria of VGAEs and apply this mechanism to address the domain shift in the structural patterns of graphs. Our second contribution is integrating Nuclear-Norm Wasserstein Discrepancy (NWD) with VGAE. Typically, VGAEs are used together with a domain classifier in previous domain adaptation methods. We use NWD to remove the domain classifier and incorporate the domain adaptation directly into our class classifier. This integration hasn't been proposed before.

2 Related Work

Unsupervised Domain Adaptation A foundational approach within UDA is to reduce the discrepancy between the source and target domain distributions using adversarial learning. A representative method in this space, the Domain Adversarial Neural Network (DANN) (Ganin & Lempitsky, 2015), employs an adversarial training framework to align domain representations by confusing a domain classifier in a shared embedding space. This strategy is adapted from generative adversarial networks (GANs) (Goodfellow et al., 2020), tailored for domain adaptation purposes. Expanding on this adversarial methodology, the FGDA technique (Gao et al., 2021) uses a discriminator to discern the gradient distribution of features, thereby achieving better performance in mitigating domain discrepancy. Furthermore, DADA (Tang & Jia, 2020) proposed an innovative strategy by integrating the domain-specific classifier with the domain discriminator to align the joint distributions of two domains more effectively.

Adversarial approaches are complemented by statistical discrepancy measures like Maximum Mean Discrepancy (MMD), utilized in the Joint Distribution Optimal Transport (JDOT) model (Courty et al., 2017b). Wasserstein Distance (WD) has been leveraged for distribution alignment in UDA methods (Courty et al., 2017a; Damodaran et al., 2018), with Redko et al. (2017) providing theoretical foundations for model generalization in the target domain when employing WD. However, the practical application of WD is computation-

ally intensive due to the absence of a closed-form solution. The Sliced Wasserstein Distance (SWD) (Rabin et al., 2011; Bonneel et al., 2015) offers a computationally feasible alternative. Reconstruction-based objectives constitute another research direction, enforcing feature invariance across domains by reconstructing source domain data from target domain features, as demonstrated in the work by Ghifary et al. (2016). Additionally, the application of Variational Autoencoders (VAEs) Kingma & Welling (2013) to UDA, such as in the Variational Fair Autoencoder (Louizos et al., 2015), showcases the capabilities of probabilistic generative models in domain-invariant feature learning. Our proposed method draws inspiration from the Variational Autoencoder's framework. Other notable approaches like ToAlign (Wei et al., 2021), SDAT (Rangwani et al., 2022), and BIWAA (Westfechtel et al., 2023), mark recent advancements in UDA, surpassing previous models in performance. These three approaches are detailed in the experiment sections as our references for current state-of-the-art methods. However, extending these existing methods to graph-structured data is often non-trivial.

Graph Representation Learning Graph representation learning (GRL) has emerged as an important approach in machine learning, tasked with distilling complex graph-structured data into a tractable, low-dimensional vector space to enable the use of architectures developed for array-structured data. Previously, spectral methods laid the foundation, leveraging graph Laplacian matrices to capture the topological structures of graphs despite limitations in scalability for larger graphs (Belkin & Niyogi, 2003; Chung, 1997). The field then evolved with algorithms such as DeepWalk (Perozzi et al., 2014) and Node2Vec (Grover & Leskovec, 2016), which utilized random walks to encode local neighborhood structures into node embeddings, balancing the preservation of local and global graph characteristics. The introduction of Graph Neural Networks (GNNs) marked a significant advancement in GRL. GNNs, specifically Graph Convolutional Networks (GCNs), offer a way to generalize neural network approaches to graph data, integrating neighborhood information into node embeddings (Kipf & Welling, 2016a). This was further refined by GraphSAGE, which scaled GNNs by learning a function to sample and aggregate local neighborhood features (Hamilton et al., 2017b;a). Moreover, Graph Attention Networks (GATs) introduced an attention mechanism, enabling the model to adaptively prioritize information from different parts of a node's neighborhood, thus enhancing the expressiveness of the embeddings (Velickovic et al., 2017). These advances, along with the development of graph autoencoders like VGAEs that focus on graph reconstruction from embeddings, have broadened the applications of GRL and continue to shape its trajectory (Kipf & Welling, 2016b). For a fair comparison, when we compare methods originally not proposed for graph domain adaptation, we replace their feature extraction backbones with GAT, as GAT is the graph encoder we use in our approach.

Denoising Diffusion Models Generative models (Ruthotto & Haber, 2021) have increasingly incorporated denoising mechanisms to enhance the clarity and fidelity of synthesized data, bridging the gap between artificial and real-world data samples. Within this framework, diffusion models stand out by iteratively applying denoising steps, transforming random noise into detailed and coherent outputs, and exemplifying the synergy between generative capabilities and denoising techniques. Denoising Diffusion Probabilistic Models (DDPMs) have revolutionized the field of generative modeling by leveraging principles from nonequilibrium thermodynamics, connecting diffusion probabilistic models with denoising score matching and Langevin dynamics to achieve high-quality image synthesis (Ho et al., 2020). Nichol & Dhariwal (2021) improved DDPMs not only to produce exceptional samples but also to refine them to attain competitive log-likelihoods while maintaining sample quality. Innovations such as the SinDDM framework demonstrate the capability of DDMs to train on single images by exploiting internal statistics through a multi-scale diffusion process (Kulikov et al., 2023). However, the computational intensity of training and evaluating these models presents a challenge. This obstacle is addressed by the introduction of a pyramidal diffusion model that streamlines the generation of high-resolution images from coarser ones (Ryu & Ye, 2022). Additionally, the Star-Shaped DDPM (SS-DDPM) introduces a star-shaped diffusion process that simplifies the computation of transition probabilities and posteriors, enhancing the model's efficiency and applicability (Okhotin et al., 2024). Despite their success in image generation, applying diffusion models to image denoising has been less straightforward due to difficulties in controlling noise placement, leading to the development of a novel diffusion process with linear interpolation (Yang et al., 2023). A multitude of diffusion model techniques for image-related tasks have been adapted for graph-related tasks (Chamberlain et al., 2021; Barbe et al., 2021; Kong et al., 2023; Li et al., 2023). Although the denoising mechanism of graph generative models is closely related to diffusion models, our work does not utilize diffusion models. In this paper, we employ a variational graph

autoencoder as the generative model and extend the denoising mechanism of the variational autoencoder, as described by (Im Im et al., 2017), to the domain of variational graph autoencoders.

3 Problem Description

We operate under the assumption that there is a source domain with labeled data and a target domain with unlabeled data. In both domains, each input instance is a graph-structured data sample. Our primary objective is to develop a predictive model for the target domain by transferring knowledge from the source domain.

Graph Classification We focus on a graph classification task, where a graph sample can be represented as $G = (X, A)$. $X \in \mathbb{R}^{n \times \mathrm{F}}$, where n represents the number of nodes in G and F represents the dimension of the features for each node. Note that the number of nodes may vary across different graphs. $x_i \in X$ corresponds to the feature associated with a node v_i. $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix. The adjacency matrix A encapsulates the topological structure of G. Each graph is associated with a class, and we use y to denote the ground truth label of the graph sample G. The goal is to train a model capable of classifying graphs effectively and accurately.

Source Domain Dataset We consider a fully labeled source domain dataset as $(D_s, Y_s) = (\{G_s^k\}, \{y_s^k\})$, where G_s^k is the kth batch of graph samples in D_s and y_s^k are the ground truth labels of G_s^k.

Target Domain Dataset We consider that only an unlabeled target domain dataset $D_t = \{G_t^k\}$ is accessible, where G_t^k is the kth batch of graph samples in the target dataset D_t.

UDA for Graph Classification The comprehensive pipeline of a UDA model for graph classification begins by leveraging a neural network to extract relevant features from the input graph samples. Then, it aligns the domains by utilizing either a distance metric to minimize the discrepancy between the source and target feature distributions or employing adversarial techniques to achieve domain-invariant feature representations. Subsequently, the classification task is performed using the aligned features. Usually, the classifier for the classification task is trained with source labels, but different UDA models may have different strategies for training the classifier. The performance of the UDA model is evaluated based on how well it performs the classification task on the test portion of the target domain dataset.

4 Proposed Method

Figure 1 visualizes a high-level description of our proposed pipeline. Our algorithm benefits from a denoising mechanism via a Variational Graph Autoencoder (VGAE) and the Nuclear-norm Wasserstein discrepancy for distribution alignment. In a nutshell, our method embeds the graph-structured data from both domains into a shared feature space via a VGAE with a denoising mechanism. Then, we align the distributions of both domains in this shared feature space by minimizing the Nuclear-norm Wasserstein discrepancy.

4.1 Latent Variables Construction with Denoising Mechanism

We start this section by introducing variational inference. In a system composed of the random variables X and Z, X denotes the observed variable, while Z is the latent variable. The conditional probability density $P(X|Z)$ is known as the likelihood. From Bayes' theorem, we can compute the posterior probability density $P(Z|X)$ as

$$P(Z|X) = \frac{P(X|Z)P(Z)}{P(X)} \tag{1}$$

The marginal probability density $P(X)$ can be computed as

$$P(X) = \int_Z P(X|Z)P(Z)\, dZ \tag{2}$$

The marginal probability density $P(X)$ is known as the evidence, and $P(Z)$ is termed the prior probability density, as it encapsulates the prior information regarding Z.

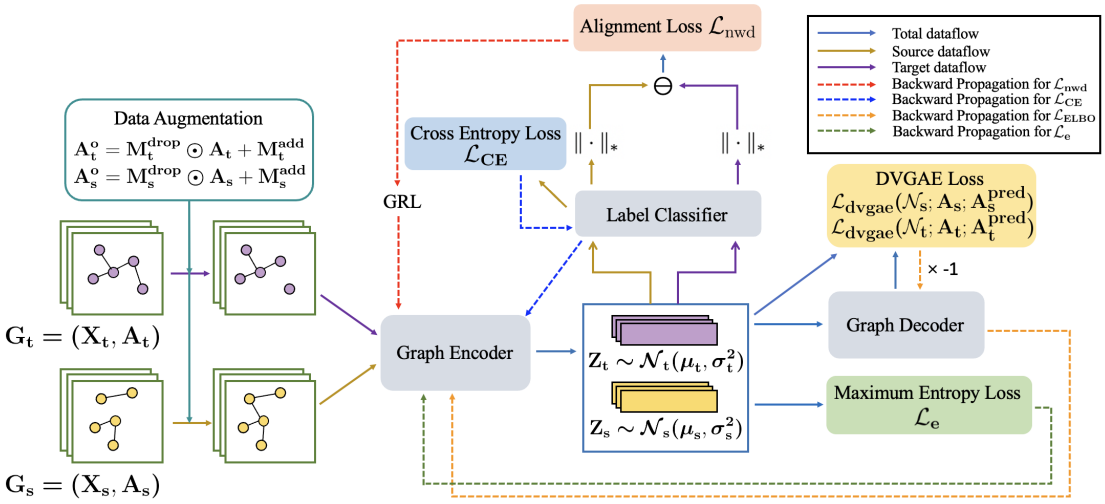

Figure 1: The block-diagram visualization of DNAN: Inputs are the source batch G_s and target batch G_t. We first add noise to the graph samples of the source and target batches by applying data augmentation to their adjacency matrices A_s and A_t, using masks M_t^{drop}, M_s^{drop}, M_s^{add}, and M_t^{add}. Then, the graph encoder of the VGAE produces the latent variables Z_s and Z_t from node features X_s, X_t and augmented adjacency matrices A_s^o, A_t^o. We train a class classifier using a cross-entropy loss $\mathcal{L}_{\text{CE}}$ between the outputs of the label classifier and the ground-truth labels. To align the latent variables of both domains, we compute a Nuclear-norm Wasserstein discrepancy (NWD) using Z_s, Z_t, and the label classifier. The graph decoder of VGAE reconstructs the original adjacency matrices A_s, A_t from Z_s, Z_t. Then, the $\mathcal{L}_{\text{dvgae}}$ loss is computed based on the outputs of the graph encoder and the original and reconstructed adjacency matrices. Lastly, the model applies maximum entropy regularization $\mathcal{L}_e$ to the latent variables Z_t, Z_s.

Variational inference (Blei et al., 2017) is an estimation technique aimed at approximating the complex, often intractable posterior distribution $P(Z|X)$ with a more computationally manageable parameterized distribution $q_\phi(Z)$. A Variational Autoencoder (VAE) is a type of generative model that uses variational inference within a probabilistic framework to encode input data into a latent space and reconstruct outputs from this space. This process allows for the generation of new data points that are similar to the original inputs. In a VAE, the model defines an approximate distribution, $q_\phi(Z|X)$, which is conditioned on an input observation X. This distribution typically takes the form of a neural network, where X serves as the input and the latent variable Z as the output. The parameters ϕ, representing the neural network's weights, are shared across all input observations X. This neural network, known as the inference network, essentially learns to encode the input data into a latent representation. Conversely, the model also defines a parameterized distribution $P_\theta(X|Z)$ that models the probability of observing X given the latent variable Z. $P_\theta(X|Z)$ is also typically chosen as a neural network with Z as the input and a distribution over possible values of X as the output. The weights of this neural network are denoted by θ, and this network is referred to as the generative network. The generative network decodes latent representations back into data points, thereby enabling the generation of new data points by sampling from the latent space. The prior distribution for the latent variable Z, denoted as $P(Z)$, is typically chosen to be an isotropic Gaussian distribution, expressed as $\mathcal{N}(0, \sigma I)$. To train a VAE, we maximize the following lower bound, known as the evidence lower bound (ELBO), with respect to the parameters θ and ϕ:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_\phi(Z|X)}\left[\frac{\log P_\theta(X,Z)}{q_\phi(Z|X)}\right] = \mathbb{E}_{q_\phi(Z|X)}\left[\log P_\theta(X|Z)\right] - \text{KL}\left(q_\phi(Z|X) \,||\, P(Z)\right) \tag{3}$$

where KL represents the Kullback-Leibler divergence.

The Variational Graph Autoencoders (Kipf & Welling, 2016b) are based on Variational Autoencoder concepts and are designed specifically for graph-structured data. Given a graph sample $G = (X, A)$ with n nodes, the graph encoder (inference network) in VGAE generates a corresponding latent variable Z. $q_\phi(Z|A, X)$ is

used to denote the graph encoder, characterized by the parameter ϕ. $q_\phi(Z|A)$ aims to approximate the real posterior distribution $P(Z|A)$. The graph decoder (generative network) of a standard VGAE is represented as $P_\theta(A|Z)$, defined by parameters θ. The prior distribution is denoted by $P(Z)$, assumed to be a normal distribution, specifically $P(Z) \sim \mathcal{N}(0, I)$. The ELBO for standard VGAE is given as:

$$\mathcal{L}_{\text{ELBO}} = E_{q_\phi(Z|A,X)}[\log P_\theta(A|Z)] - \text{KL}(q_\phi(Z|A,X)||P(Z)) \tag{4}$$

The standard VGAE is trained through maximizing the $\mathcal{L}_{\text{ELBO}}$. Instead of using plain VGAE, we propose to use VGAE with a denoising mechanism. Particularly, we adopt the denoising criterion of Denoising Variational Autoencoders (DVAE) (Im Im et al., 2017) and translate it to variational graph autoencoders. Like DVAE, the VGAE with a denoising mechanism reconstructs clean graph data from inputs perturbed with noise. Similar to the training process of VGAE, however, we have some variations. Given a graph sample $G = (X, A)$, we train the VGAE on both $G = (X, A)$ and $G^o = (X, A^o)$, where A^o is the adjacency matrix with noise. We implement data augmentation to add noise to the adjacency matrices. Specifically, we benefit from a random manipulation-based approach (Cai et al., 2021). To this end, edges are dropped and added randomly by modifying the values in the adjacency matrix A of the original graph. A^o is constructed as follows:

$$A^o = M^{\text{drop}} \odot A + M^{\text{add}}, \qquad m_{ij}^{\text{add}} \sim \text{Bernoulli}(p^{\text{add}} \cdot p^{\text{edge}}), \qquad m_{ij}^{\text{drop}} \sim \text{Bernoulli}(p^{\text{drop}}) \tag{5}$$

where p^{add}, p^{edge}, and p^{drop} denote the edge addition rate, the sparsity of the adjacency matrix A, and the edge dropping rate. $\odot$ represents the element-wise multiplication between two matrices. M^{drop} and M^{add} represent mask matrices with the same dimensions as A. For each element $m_{ij}^{\text{add}} \in M^{\text{add}}$ or $m_{ij}^{\text{drop}} \in M^{\text{drop}}$, we sample its value from a Bernoulli distribution.

We're using similar logic from the DVAE paper to explain how the denoising mechanism can be applied to the variational graph autoencoder. In the following explanation, we use "corruption" to refer to the noise addition procedure. We start with how the denoising mechanism will affect the inference process of the variational graph autoencoder. We first translate Proposition 1 from the DVAE paper to the VGAE setting as follows.

Proposition 1. Let $q_\phi(Z|A^o, X) = \mathcal{N}(Z|\mu_\phi(A^o, X), \sigma_\phi(A^o, X))$ be a Gaussian distribution, where $\mu_\phi(A^o, X)$ and $\sigma_\phi(A^o, X)$ are non-linear functions of (A^o, X). Let $P(A^o|A)$ be a corruption distribution around A and A^o be the corrupt adjaceny matrix. Then,

$$\mathbb{E}_{P(A^o|A)}[q_\phi(Z|A^o, X)] = \int_{A^o} q_\phi(Z|A^o, X) P(A^o|A)\, dA^o \tag{6}$$

is a mixture of Gaussian.

If the distribution is over a discrete variable, the integral in Equation 6 can be replaced by a summation. It's instructive to examine the distribution in the discrete domain to understand that Equation 6 takes on the form of a Gaussian mixture. Essentially, for each instance A^o drawn from $P(A^o|A)$, substituting it into $q_\phi(Z|A^o, X)$ results in a Gaussian distribution. In our scenario, given that each element within the adjacency matrix can only be 0 or 1, $P(A^o|A)$ is a discrete distribution. Thus, we can formulate our case as follows.

Example 1. Let $A \in \{0, 1\}^{n \times n}$ be the adjacency matrix of graph G that has n nodes, and consider a corruption distribution $P_\pi(A^o|A) = \text{Bernoulli}(\pi)$ around A and A^o is corrupted adjacency matrix. Then,

$$\mathbb{E}_{P_\pi(A^o|A)}[q_\phi(Z|A^o, X)] = \sum_{i=1}^{2^{n \times n}} q_\phi(Z|A_i^o, X) P_\pi(A_i^o|A) \tag{7}$$

has the form of a finite mixture of Gaussian and the number of mixture components is $2^{n \times n}$.

From the DVAE paper, the corruption process at the input can be viewed as adding a stochastic layer at the bottom of the inference network (q_ϕ). Specifically, $P_\pi(A^o|A)$ can be seen as a neural network, with π as its weights. $P_\pi(A^o|A)$ takes A as input and outputs the corruption distribution. In the following, we first

present Lemma 0, a result we will use in the later proof. Following Lemma 0, we present the variational lower bound when an extra stochastic layer is included, in Lemma 1.

Lemma 0. For all nonnegative measurable functions $f, g : \mathbb{R} \to [0, \infty)$ that satisfies $\int_{-\infty}^{\infty} f(X)\, dX = 1$,

$$\int_{-\infty}^{\infty} f(X) \log g(X)\, dX \leq \int_{-\infty}^{\infty} f(X) \log f(X)\, dX$$

Proof. Let X be a random variable with $f(X)$ be its probability density function. Consider the random variable $\log \left[\frac{f(X)}{g(X)}\right]$ with $\mathbb{E}_{f(X)} \left[\log \frac{g(X)}{f(X)}\right] = -\mathbb{E}_{f(X)} \left[\log \frac{f(X)}{g(X)}\right]$. By Jensen's inequality,

$$\mathbb{E}_{f(X)} \left[\log \frac{g(X)}{f(X)}\right] \leq \log \mathbb{E}_{f(X)} \left[\frac{g(X)}{f(X)}\right] = \log \left(\int_{-\infty}^{\infty} g(X)\, dX\right) = 0$$

Therefore, $\mathbb{E}_{f(X)} [\log g(X)] \leq \mathbb{E}_{f(X)} [\log f(X)]$.

Lemma 1. Consider an approximate posterior distribution of the following form:

$$q_\Phi(Z|A, X) = \int_{A^o} q_\phi(Z|A^o, X) q_\psi(A^o|A)\, dA^o \tag{8}$$

Here, $q_\psi(A^o|A)$ represents the stochastic layer for the corruption process. We use $\Phi = \{\phi, \psi\}$. Then, given $P_\theta(A, Z) = P_\theta(A|Z)P(Z)$, we obtain the following inequality:

$$\log P_\theta(A) \geq \mathbb{E}_{q_\Phi(Z|A,X)} \left[\log \frac{P_\theta(A, Z)}{q_\phi(Z|A^o, X)}\right] \geq \mathbb{E}_{q_\Phi(Z|A,X)} \left[\log \frac{P_\theta(A, Z)}{q_\Phi(Z|A, X)}\right] \tag{9}$$

Proof. By Jensen's inequality, we have

$$\begin{aligned}
\mathbb{E}_{q_\Phi(Z|A,X)} \left[\log \frac{P_\theta(A, Z)}{q_\phi(Z|A^o, X)}\right] &= \mathbb{E}_{q_\psi(A^o|A)} \left[\mathbb{E}_{q_\phi(Z|A^o,X)} \left[\log \frac{P_\theta(A, Z)}{q_\phi(Z|A^o, X)}\right]\right] \\
\mathbb{E}_{q_\psi(A^o|A)} \left[\mathbb{E}_{q_\phi(Z|A^o,X)} \left[\log \frac{P_\theta(A, Z)}{q_\phi(Z|A^o, X)}\right]\right] &\leq \log \left(\mathbb{E}_{q_\psi(A^o|A)} \left[\mathbb{E}_{q_\phi(Z|A^o,X)} \left[\frac{P_\theta(A, Z)}{q_\phi(Z|A^o, X)}\right]\right]\right) \\
&= \log \left(\mathbb{E}_{q_\psi(A^o|A)} \left[\int_Z \frac{P_\theta(A, Z)}{q_\phi(Z|A^o, X)} q_\phi(Z|A^o, X)\, dZ\right]\right) \\
&= \log \left(\mathbb{E}_{q_\psi(A^o|A)} \left[\int_Z P_\theta(A, Z)\, dZ\right]\right) \\
&= \log \left(\int_{A^o} P_\theta(A) q_\psi(A^o|A)\, dA^o\right) \\
&= \log \left(P_\theta(A) \cdot \int_{A^o} q_\psi(A^o|A)\, dA^o\right) \\
&= \log P_\theta(A)
\end{aligned}$$

Therefore, the left inequality of Equation 9 holds, and now, for the right inequality,

$$\mathbb{E}_{q_\Phi(Z|A,X)} \left[\log \frac{P_\theta(A, Z)}{q_\phi(Z|A^o, X)}\right] = \mathbb{E}_{q_\Phi(Z|A,X)}[\log P_\theta(A, Z)] - \mathbb{E}_{q_\Phi(Z|A,X)}[\log q_\phi(Z|A^o, X)]$$

Applying Lemma 0 to the second term, we have

$$\mathbb{E}_{q_\Phi(Z|A,X)}[\log q_\phi(Z|A^o, X)] = \int_Z \log q_\phi(Z|A^o, X) q_\Phi(Z|A, X)\, dZ \leq \int_Z \log q_\Phi(Z|A, X) q_\Phi(Z|A, X)\, dZ$$

Hence,

$$\mathbb{E}_{q_\Phi(Z|A,X)}[\log q_\phi(Z|A^o, X)] \leq \mathbb{E}_{q_\Phi(Z|A,X)}[\log q_\Phi(Z|A, X)]$$

Then, we have

$$\begin{aligned}\mathbb{E}_{q_\Phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right] &= \mathbb{E}_{q_\Phi(Z|A,X)}[\log P_\theta(A,Z)] - \mathbb{E}_{q_\Phi(Z|A,X)}[\log q_\phi(Z|A^o,X)] \\ &\geq \mathbb{E}_{q_\Phi(Z|A,X)}[\log P_\theta(A,Z)] - \mathbb{E}_{q_\Phi(Z|A,X)}[\log q_\Phi(Z|A,X)] \\ &= \mathbb{E}_{q_\Phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\Phi(Z|A,X)}\right]\end{aligned}$$

Therefore, we obtain

$$\log P_\theta(A) \geq \mathbb{E}_{q_\Phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right] \geq \mathbb{E}_{q_\Phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\Phi(Z|A,X)}\right]$$

Recalling Example 1, our approximate distribution can be defined as follows.

$$\tilde{q}_\phi(Z|A,X) = \sum_{i=1}^{2^{n\times n}} q_\phi(Z|A_i^o,X)P(A_i^o|A)$$

By treating adding noise via including one stochastic layer parameterized by ψ, we have:

$$\tilde{q}_\phi(Z|A,X) = \sum_{i=1}^{2^{n\times n}} q_\phi(Z|A_i^o,X)P_\psi(A_i^o|A)$$

We now can apply Lemma 1 to obtain our denoising variational lower bound for VGAE. The standard variational lower bound $\mathcal{L}_{\text{svgae}}$ for the approximation distribution $\tilde{q}_\phi(Z|A,X)$ is:

$$\log P_\theta(A) \geq \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right] \geq \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{\tilde{q}_\phi(Z|A,X)}\right] = \mathcal{L}_{\text{svgae}} \tag{10}$$

We define the denoising variational lower bound $\mathcal{L}_{\text{dvgae}}$ as:

$$\log P_\theta(A) \geq \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right] \stackrel{\text{def}}{=} \mathcal{L}_{\text{dvgae}} \geq \mathcal{L}_{\text{svgae}} \tag{11}$$

From Equation 11, $\mathcal{L}_{\text{dvgae}}$ is a tighter lower boud of $\log P_\theta(A)$ than standard variational lower bound $\mathcal{L}_{\text{svgae}}$. Thus, we need to check whether $\mathcal{L}_{\text{dvgae}}$ is a valid lower bound. To check whether $\mathcal{L}_{\text{dvgae}}$ is valid, we need to examine what is achieved by maximizing $\mathcal{L}_{\text{dvgae}}$. In fact, maximizing $\mathcal{L}_{\text{dvgae}}$ can minimize the expectation of the KL divergence between the true posterior distribution ($P(Z|A)$) and the approximate posterior distribution for each noised input ($q_\phi(Z|A^o,X)$). This is an effective objective as the inference network tries to map the noise-perturbed training data points to the true posterior distribution. In Theorem 1, we prove that maximizing $\mathcal{L}_{\text{dvgae}}$ achieves this goal.

Theorem 1. Maximizing $\mathcal{L}_{\text{dvgae}}$ is equivalent to minimizing the following objective

$$\mathbb{E}_{P_\psi(A^o|A)}\left[\text{KL}\left(q_\phi(Z|A^o,X)||P(Z|A)\right)\right] \tag{12}$$

In other words,

$$\log P_\theta(A) = \mathcal{L}_{\text{dvgae}} + \mathbb{E}_{P_\psi(A^o|A)}\left[\text{KL}\left(q_\phi(Z|A^o,X)||P(Z|A)\right)\right]$$

Proof. Let us consider θ being fixed just for the sake of simpler analysis.

$$\begin{aligned}\log P_\theta(A) - \mathcal{L}_{\text{dvgae}} &= \log P_\theta(A) - \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right] \\ &= \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}[\log P_\theta(A)] - \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}\left[\log \frac{P(Z|A)P_\theta(A)}{q_\phi(Z|A^o,X)}\right] \\ &= \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}\left[\log \frac{q_\phi(Z|A^o,X)}{P(Z|A)}\right] \\ &= \mathbb{E}_{P_\psi(A^o|A)}\left[\mathbb{E}_{q_\phi(Z|A^o,X)}\left[\log \frac{q_\phi(Z|A^o,X)}{P(Z|A)}\right]\right] \\ &= \mathbb{E}_{P_\psi(A^o|A)}\left[\text{KL}(q_\phi(Z|A^o,X)||P(Z|A))\right]\end{aligned}$$

Now we prove that $\mathcal{L}_{\text{dvgae}}$ is a valid variational lower bound. To train the VGAE with $\mathcal{L}_{\text{dvgae}}$, in the DVAE paper, the authors adopt Monte Carlo sampling. In Monte Carlo sampling, we randomly select points from the domain of a function and compute the function's values at these points. We then calculate the average of these values to estimate the expected value of the function. Essentially, it involves taking the mean of the function's values at the sampled points. The authors in the DVAE paper apply Monte Carlo sampling twice: once to the inner expectation $\mathbb{E}_{P_\psi(A^o|A)}$ and once to the outer expectation $\mathbb{E}_{q_\phi(Z|A,X)}$. Their approximation is shown below.

$$\mathcal{L}_{\text{dvgae}} = \mathbb{E}_{\tilde{q}_\phi(Z|A,X)}\left[\log \frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right] = \mathbb{E}_{q_\phi(Z|A^o,X)}\left[\mathbb{E}_{P_\psi(A^o|A)}\left[\frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right]\right]$$

$$\mathcal{L}_{\text{dvgae}} = \mathbb{E}_{q_\phi(Z|A^o,X)}\left[\mathbb{E}_{P_\psi(A^o|A)}\left[\frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right]\right] \approx \frac{1}{JM}\sum_{j=1}^{J}\sum_{m=1}^{M}\log\frac{P_\theta(A,Z^{(k|m)})}{q_\phi(Z^{(k|m)}|A^o_m,X)} \tag{13}$$

where $A^o_m \sim P_\psi(A^o|A)$, $Z^{(j|m)} \sim q_\phi(Z|A^o_m,X)$, and J, M are Monte Carlo sample sizes. We also use Monte Carlo sampling in our scenario, but only for the inner expectation $\mathbb{E}_{P_\psi(A^o|A)}$, since we adopt the procedure used to train the regular VGAE: (i) sample a corrupted input (A^o, X), (ii) sample a latent variable from $q_\phi(Z|A^o,X)$, (iii) reconstruct the original adjacency matrix A. As the authors of DVAE state in their section 3.2 (Training Procedure), our procedure can be viewed as a special case of Equation 13. Our estimation is shown as follows.

$$\begin{aligned}
\mathcal{L}_{\text{dvgae}} &= \mathbb{E}_{q_\phi(Z|A^o,X)}\left[\mathbb{E}_{P_\psi(A^o|A)}\left[\frac{P_\theta(A,Z)}{q_\phi(Z|A^o,X)}\right]\right] \\
&\approx \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q_\phi(Z|A^o_m,X)}\left[\log\frac{P_\theta(A,Z)}{q_\phi(Z|A^o_m,X)}\right] \\
&= \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q_\phi(Z|A^o_m,X)}\left[\log\frac{P_\theta(A|Z)P(Z)}{q_\phi(Z|A^o_m,X)}\right] \\
&= \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q_\phi(Z|A^o_m,X)}\left[\log P_\theta(A|Z)\right] + \mathbb{E}_{q_\phi(Z|A^o_m,X)}\left[\log\frac{P(Z)}{q_\phi(Z|A^o_m,X)}\right] \\
&= \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q_\phi(Z|A^o_m,X)}\left[\log P_\theta(A|Z)\right] - \text{KL}(q_\phi(Z|A^o_m,X)||P(Z))
\end{aligned}$$

$$\mathcal{L}_{\text{dvgae}} \approx \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q_\phi(Z|A^o_m,X)}\left[\log P_\theta(A|Z)\right] - \text{KL}(q_\phi(Z|A^o_m,X)||P(Z)) \tag{14}$$

With this approximation and setting the Monte Carlo sample size to 1, the training procedure becomes similar to how the regular VGAE is trained, except that the input is corrupted by a noise distribution. For clean inputs, we train VGAE with Equation 4, and for corrupted inputs, we train VGAE with 14.

For the structure of VGAE, we utilize Graph Attention Networks (GAT) as the graph encoder q_ϕ of the VGAE. The encoding equations when the input is A^o are given as follows:

$$\begin{aligned}
\mu &= \text{GAT}_\mu(A^o, X) \\
\log\sigma &= \text{GAT}_\sigma(A^o, X) \\
z_i &= \mu_i + \varepsilon_i \cdot \sigma_i,\ \ \varepsilon_i \sim \mathcal{N}(0,1) \\
q_\phi(z_i|A^o,X) &= \mathcal{N}(z_i|\mu_i, \text{diag}(\sigma_i^2)) \\
q_\phi(Z|A^o,X) &= \prod_{i=1}^{n} q_\phi(z_i|A^o,X)
\end{aligned} \tag{15}$$

The element z_i corresponds to the ith row of Z. This same row-wise correspondence applies to μ_i and $\log\sigma_i$ as well. Using the reparameterization trick, we transform the generated μ_i and σ_i into the latent variable z_i.

To achieve a cleaner construction of the latent variable, we apply an element-wise maximum entropy loss, $\mathcal{L}_e$, as a regularization term. This maximum entropy loss helps remove irrelevant information from the latent variable, enhancing its clarity and effectiveness. The specifics of the maximum entropy loss are described below:

$$\begin{aligned} \mathcal{L}_{\mathrm{e}} &= \frac{1}{N^{\mathrm{train}}} \sum_{G^k \in (D_s, D_t)} \mathrm{ME}(Z^k) \\ \mathrm{ME}(Z^k) &= \frac{1}{n_k \times \mathrm{D}_h} \sum_{i=1}^{n_k} \sum_{j=1}^{\mathrm{D}_h} \sigma(z_{ij}) \log \sigma(z_{ij}) \end{aligned} \tag{16}$$

where n_k is the number of nodes in the graph sample G^k and D_h denotes the dimension of G^k's latent variable Z^k. N^{train} is the number of training samples in the training dataset. After obtaining the latent variable Z, an inner product decoder $P_\theta(A|Z)$ is applied to Z to reconstruct the adjacency matrix before data augmentation. This decoder translates each pair of node representations into a binary value, indicating whether an edge exists in the reconstructed adjacency matrix A^o. Specifically, we first use an MLP (multilayer perceptron) described by parameters $\{W_0, W_1\}$ to improve the expressive capacity of the latent variable Z. Then, we compute the dot product for each node representation pair as:

$$\begin{aligned} H &= \mathrm{ReLU}((Z \cdot W_0) \cdot W_1) \\ p(A^o_{ij} = 1 | h_i, h_j) &= \sigma(h_i^T h_j) \\ p(A^o|Z) &= \prod_{i=1}^{n} \prod_{j=1}^{n} p(A^o_{ij} | h_i, h_j) \end{aligned} \tag{17}$$

where h_i represents the ith row of H and A^o_{ij} is an element of reconstructed adjacency matrix A^o. The parameter θ describes the graph decoder includes $\{W_0, W_1\}$.

4.2 Distribution Alignment

By using $\mathcal{L}_{\mathrm{dvgae}}$, the graph encoder of VGAE is better equipped to grasp the essential features. However, we still face a crucial challenge: addressing the performance degradation that occurs when a model trained on data from a source domain is applied to a target domain with a different data distribution. As mentioned in the previous sections, traditional approaches in unsupervised domain adaptation often use a domain discriminator that engages in a min-max game with a feature extractor to produce domain-invariant features. However, these methods primarily focus on confusing features at the domain level, which might negatively impact class-level information and lead to the mode collapse problem (Kurmi & Namboodiri, 2019; Tang & Jia, 2020). To address these challenges, our approach integrates the Nuclear-norm Wasserstein discrepancy (NWD) (Chen et al., 2022) to effectively align the source and target domains' feature representations while maintaining class-level discrimination by considering it as a loss function. We begin explaining NWD by discussing intra-class and inter-class correlations.

From Intra-class and Inter-class Correlations to Domain Discrepancy Consider a prediction matrix $P \in \mathbb{R}^{b \times c}$ predicted by classifier C, where b represents the number of samples and c represents the number of classes. P has the following properties:

$$\sum_{j=1}^{c} P_{ij} = 1, \;\; P_{ij} \geq 0, \;\; \forall i \in \{1, 2, ...b\} \tag{18}$$

The self-correlation matrix $R \in \mathbb{R}^{c \times c}$ can then be computed by $R = P^T P$. The intra-class correlation I_a is defined as the sum of the main diagonal elements in R, and the inter-class correlation I_e is defined as the sum of the off-diagonal elements in R:

$$I_a = \sum_{i,j=1}^{c} R_{ij}, \;\; I_e = \sum_{i \neq j}^{c} R_{ij} \tag{19}$$

The I_a and I_e are very different for source and target domains. For the source domain, the I_a is large while the I_e is relatively small, as we train with labels available so that most samples are correctly classified. For the target domain, the I_a is small while the I_e is relatively large due to the lack of supervised training. Based on linear algebra, we can represent $I_a = \|P\|_F$, the Frobenius norm of P, and

$$I_a - I_e = 2\|P\|_F - b \tag{20}$$

For the source domain, $I_a - I_e$ will be large; for the target domain, $I_a - I_e$ will be small. Therefore, $I_a - I_e$ can represent the discrepancy between two domains. Since the prediction matrix P is generated by the classifier C, we can rewrite $P_s = C(Z_s), P_t = C(Z_t)$, where Z_s is the feature representation of a batch of samples from the source domain and Z_t is the feature representation of a batch of samples from the target domain. With inspiration from WGAN (Arjovsky et al., 2017) and 1-Wasserstein distance, the domain discrepancy can be formally formulated as

$$W_F(D_s, D_t) = \sup_{\|\|C\|_F\|_L \leq K_L} \mathbb{E}_{Z_s \sim D_s}[\|C(Z_s)\|_F] - \mathbb{E}_{Z_t \sim D_t}[\|C(Z_t)\|_F] \tag{21}$$

We call $W_F(D_s, D_t)$ the Frobenius norm-based 1-Wasserstein distance, where D_s denotes the source domain, D_t denotes the target domain, $\|\cdot\|_L$ denotes the Lipschitz semi-norm (Villani et al., 2009), and K_L denotes the Lipschitz constant.

From Frobenius Norm to Nuclear Norm From the domain discrepancy formulated above, we can see that the classifier C works like a discriminator in a GAN. Therefore, we can perform adversarial training to train the feature encoder via $W_F(D_s, D_t)$. However, adversarial training with $W_F(D_s, D_t)$ limits the diversity of predictions. This is because it tends to push the samples in a class with fewer samples near the decision boundary closer to a neighboring class with a significantly larger number of samples far from the decision boundary (Cui et al., 2021). To address this limitation, the author of NWD proposes using the nuclear norm instead of the Frobenius norm. The nuclear norm has been shown to be bound by the Frobenius norm (Chen et al., 2022). In addition, maximizing the nuclear norm maximizes the rank of the prediction matrix P when $\|\cdot\|_F$ is near $\sqrt{b}$ (Cui et al., 2020; 2021). Consequently, the diversity of predictions will be enhanced. Thus, the domain discrepancy can be improved to be:

$$W_N(D_s, D_t) = \sup_{\|\|C\|_*\|_L \leq K_L} \mathbb{E}_{Z_s \sim D_s}[\|C(Z_s)\|_*] - \mathbb{E}_{Z_t \sim D_t}[\|C(Z_t)\|_*] \tag{22}$$

$W_N(D_s, D_t)$ is called the Nuclear-norm 1-Wasserstein discrepancy (NWD). To integrate NWD into implementation, we can approximate the W_N by maximizing $\mathcal{L}_{\text{nwd}}$ that is defined below:

$$\mathcal{L}_{\text{nwd}} = \frac{1}{N_s^{\text{train}}} \sum_{k=1}^{N_s^{\text{train}}} \|C(Z_s^k)\|_* - \frac{1}{N_t^{\text{train}}} \sum_{k=1}^{N_t^{\text{train}}} \|C(Z_t^k)\|_*, \; W_N(D_s, D_t) \approx \max \mathcal{L}_{\text{nwd}} \tag{23}$$

where Z_s^k represents the latent variables for the k-th batch of graph samples G_s^k and Z_t^k represents the latent variables for the k-th batch of graph samples G_t^k. $\|\cdot\|_*$ denotes the Nuclear norm. To avoid complex alternating updates, we employ a Gradient Reverse Layer (GRL) (Ganin et al., 2016), which allows for updating in a single backpropagation step. The distribution alignment is achieved through a min-max game, optimized as:

$$\min_{\phi} \max_{\theta_c} \mathcal{L}_{\text{nwd}} \tag{24}$$

The NWD addresses the class mismatch issue by incorporating class information into the domain adaptation process. The class classifier not only performs class classification but also serves as a domain discriminator. The class classifier is capable of identifying correlations both within and among different classes. These correlations vary between the source domain data and the target domain data. To achieve domain alignment, NWD aligns the correlations within and between classes in the target domain with those in the source domain. This alignment ensures consistency of classes across different domains, so the class mismatch problem is mitigated.

Algorithm 1 DNAN Method

Input: $(D_s, Y_s), D_t$
Parameters: VGAE parameters $\{\phi$ (Graph Encoder)$, \theta$ (Graph Decoder)$\}$, Classifier parameter $\{\theta_c\}$
Output: Trained Parameters ϕ, θ, θ_c

1: Randomly sample a batch of $\{(G_s^k, y_s^k)\}$
2: Randomly sample a batch of $\{G_t^k\}$
3: Forward Propagation
4: Update ϕ, θ, θ_c based on Equation (26)
5: Add noise to $\{G_s^k\}, \{G_t^k\}$ based on Equation (5)
6: Forward Propagation
7: Update ϕ, θ, θ_c based on Equation (26)
8: **return** ϕ, θ, θ_c

4.3 Algorithm Summary

In addition to distribution alignment, to ensure accurate classification, we optimize the graph encoder in VGAE and the classifier C using a supervised classification loss $\mathcal{L}_{\text{cls}}$ for the source domain:

$$\mathcal{L}_{\text{cls}} = \frac{1}{N_s^{\text{train}}} \sum_{k=1}^{N_s^{\text{train}}} \mathcal{L}_{\text{CE}}(C(Z_s^k, y_s^k)) \tag{25}$$

Then, by combining all the losses described in the previous sections, our total optimization object is formulated as follows:

$$\min_{\phi,\theta,\theta_c} \left\{ \mathcal{L}_{\text{cls}} - \mathcal{L}_{\text{ELBO}}(\text{or } \mathcal{L}_{\text{dvgae}}) + \lambda_e \mathcal{L}_{\text{e}} \right\} + \min_{\phi} \max_{\theta_c} \mathcal{L}_{\text{nwd}}, \tag{26}$$

where ϕ is the parameter of the graph encoder of the VGAE, θ is the parameter of the graph decoder of the VGAE, θ_c is the parameter of the classifier, and λ_e is a hyperparameter that weighs the maximum entropy loss $\mathcal{L}_{\text{e}}$. It is worth noting that we balance the supervised classification loss and the NWD loss equally. In this case, our model effectively learns transferable and distinct features, leading to accurate and diverse predictions in the target domain. The complete procedures of our UDA approach for graph-structured data are summarized in Algorithm 1.

5 Experimental Validation

We validate our algorithm using two graph classification benchmarks. Our code is provided as a supplement.

5.1 Experimental Setup

Datasets We use the IMDB&REDDIT Dataset (Yanardag & Vishwanathan, 2015) and the Ego-network Dataset (Qiu et al., 2018) in our experiments. Following previous works, we include Coreness (Batagelj & Zaversnik, 2003), Pagerank (Page et al., 1999), Eigenvector Centrality (Bonacich, 1987), Clustering Coefficient (Watts & Strogatz, 1998), and Degree/Rarity (Adamic & Adar, 2003) as node features for both datasets.

IMDB&REDDIT Dataset IMDB&REDDIT consists of the IMDB-BINARY and REDDIT-BINARY datasets, each denoting a single domain. The Reddit-Binary dataset comprises an average of approximately 430 nodes per graph, with a total of 859,254 nodes across 2,000 samples. The IMDB-Binary dataset comprises an average of approximately 20 nodes per graph, with a total of 19,773 nodes across 1,000 samples. We use "I to R" to denote that IMDB-BINARY is the source domain and REDDIT-BINARY is the target domain. Conversely, "R to I" indicates that REDDIT-BINARY is the source domain and IMDB-BINARY is the target domain.

- **IMDB-BINARY** Each graph in this dataset represents an ego network for an actor/actress. Nodes correspond to actors/actresses. There will be an edge between two actors/actresses who appear in

the same movie. A graph is generated from either romance or action movies. The task is to classify the graph into romance or action genres.

- **REDDIT-BINARY** Each graph represents an online discussion thread. Nodes correspond to users. If one user has responded to another's comments, then an edge exists between them. The discussion threads are drawn from four communities: AskREDDIT and IAmA are question/answer-based communities; and Atheism and TrollXChromosomes are discussion-based communities. The classification task is to classify a graph into discussion-based or question/answer-based communities.

Ego-network Dataset The Ego-network dataset comprises data from four social network platforms: Digg, OAG, Twitter, and Weibo, each representing a domain. We pair two domains to form twelve tasks, and we test our model on these twelve tasks. Each network in these four domains is modeled as a graph. Each graph consists of 50 nodes, with nodes representing users. Every graph includes an ego user. An edge is drawn between two nodes if a social connection occurs between two users. The definitions of social connections for these four social network platforms are different. We extract the descriptions of social connections and social actions for each social network according to Qiu et al. (2018):

- **Digg:** Users can vote for web content such as stories and news (up or down). The social connection represents users' friendship, and the social action is voting for content.
- **OAG:** Generated from AMiner and Microsoft Academic Graph, the social connection is represented as co-authorship among users, and the social action is citation behavior.
- **Twitter:** Currently known as X, the social connection on Twitter represents users' friendships, and the social action is posting tweets related to the Higgs boson, a particle discovered in 2012.
- **Weibo:** Similar to Twitter, the Weibo dataset includes posting logs between September 28th, 2012, and October 29th, 2012, among 1,776,950 users. The social connection is defined as users' friendships and the social action is reposting messages on Weibo.

All the graphs in the four domains are labeled as active or inactive, indicating the ego user's action status. If the user takes the social action, then the user is active. The task is to classify whether the ego users are active or inactive. In addition to the previously referenced node features, the Ego-network dataset also contains DeepWalk embeddings for each node (Perozzi et al., 2014), the number/ratio of active neighbors (Backstrom et al., 2006), the density of the subnetwork induced by active neighbors (Ugander et al., 2012), and the number of connected components formed by active neighbors (Ugander et al., 2012).

Baselines for Comparison There is a limited number of UDA algorithms specifically designed for graph classification tasks. As a result, we conduct a comparative analysis between our proposed method and updated versions of several representative methods (DANN, MDD, DIVA) as well as current state-of-the-art UDA methods (SDAT, BIWAA, ToAlign) for array-structured data. To facilitate the adaptation of these algorithms to graph-structured data, and consistent with the structure of our VGAE's graph encoder, we replace the feature extraction backbones, originally designed for array-structured data, with GATs. Below, these methods are explained:

- **Sources:** Plain GATs trained without domain adaptation techniques.
- **DANN:** Domain Adversarial Neural Network (DANN) (Ganin et al., 2016) adopts an adversarial learning strategy. It contains a domain classifier. The domain classifier tries to distinguish the samples from which domain and the feature extractor aims to confuse the domain classifier.
- **MDD:** Margin Disparity Discrepancy (MDD) (Zhang et al., 2019b) is first proposed for computer vision tasks. It measures the distribution discrepancy and is tailored to the minimax optimization for training.

- **DIVA:** Domain Invariant Variational Autoencoders (DIVA) (Ilse et al., 2020) disentangles the inputs into three latent variables, domain latent variables, semantic latent variables, and residual variations latent variables. It is proposed to solve problems in fields such as medical imaging.
- **SDAT:** Smooth Domain Adversarial Training (SDAT) (Rangwani et al., 2022) focuses on achieving smooth minima with respect to classification loss, which stabilizes adversarial training and improves the performance on the target domain.
- **BIWAA:** Backprop Induced Feature Weighting for Adversarial Domain Adaptation with Iterative Label Distribution Alignment (BIWAA) (Westfechtel et al., 2023) employs a classifier-based backprop-induced weighting of the feature space, allowing the domain classifier to concentrate on features that are important for classification and coupling the classification and adversarial branch more closely.
- **ToAlign:** Task-oriented Alignment for Unsupervised Domain Adaptation (ToAlign) (Wei et al., 2021) decomposes features in the source domain into classification task-related and classification task-irrelevant parts under the guidance of classification meta-knowledge, ensuring that the domain adaptation is beneficial for the performance on the classification task.

Evaluation Metrics Following the literature (Cai et al., 2021), the F1-Score is employed as a metric for the quantitative assessment of all methods. This score is the harmonic mean of precision and recall. The formula for the F1-Score is as follows:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{27}$$

Training Scheme In our evaluation, we rigorously train five models for each baseline method by employing five distinct random seeds for parameter initialization and either dropping or adding edges during the data augmentation phase. We report both the average performance and standard deviation of the obtained F1 scores. To ensure a fair comparison across all methods, we maintain the same seed for data shuffling. The optimization process utilizes the Adam (Kingma & Ba, 2014) optimizer. Please refer to Section A in the Appendix for a comprehensive description of our training scheme.

5.2 Performance Results and Comparison

Tables 1 and 2 present our performance results. The bold font denotes the highest performance in each column.

Table 1: Performance results on Ego-network dataset

Method	O→T	O→W	O→D	T→O	T→W	T→D	W→O	W→T	W→D	D→O	D→T	D→W	Avg
Source	40.0$_{\pm 0.0}$	40.4$_{\pm 0.3}$	43.8$_{\pm 2.4}$	40.2$_{\pm 0.0}$	48.0$_{\pm 1.2}$	41.3$_{\pm 0.0}$	40.2$_{\pm 0.0}$	46.6$_{\pm 0.9}$	41.3$_{\pm 0.1}$	40.2$_{\pm 0.0}$	40.0$_{\pm 0.0}$	39.8$_{\pm 0.0}$	41.8
DANN	42.0$_{\pm 0.6}$	41.7$_{\pm 0.6}$	51.3$_{\pm 0.8}$	40.7$_{\pm 0.7}$	42.0$_{\pm 0.9}$	49.9$_{\pm 1.7}$	40.3$_{\pm 0.1}$	41.3$_{\pm 0.6}$	50.9$_{\pm 0.4}$	40.3$_{\pm 0.0}$	40.4$_{\pm 0.3}$	42.5$_{\pm 0.7}$	43.6
MMD	40.2$_{\pm 0.1}$	41.2$_{\pm 1.1}$	48.0$_{\pm 3.2}$	40.2$_{\pm 0.0}$	45.5$_{\pm 1.5}$	41.3$_{\pm 0.0}$	40.2$_{\pm 0.0}$	46.2$_{\pm 2.9}$	41.9$_{\pm 1.3}$	40.2$_{\pm 0.0}$	40.1$_{\pm 0.1}$	40.0$_{\pm 0.2}$	42.1
DIVA	42.1$_{\pm 0.5}$	42.4$_{\pm 1.4}$	48.9$_{\pm 0.7}$	40.3$_{\pm 0.2}$	42.0$_{\pm 0.4}$	50.3$_{\pm 0.5}$	40.4$_{\pm 0.3}$	41.2$_{\pm 0.3}$	48.7$_{\pm 0.9}$	**40.6**$_{\pm 0.5}$	41.3$_{\pm 0.4}$	42.5$_{\pm 0.5}$	43.4
SDAT	40.2$_{\pm 0.1}$	40.1$_{\pm 0.5}$	42.2$_{\pm 1.6}$	40.2$_{\pm 0.0}$	41.6$_{\pm 1.3}$	42.9$_{\pm 3.2}$	40.3$_{\pm 0.2}$	41.1$_{\pm 0.7}$	43.1$_{\pm 2.8}$	40.2$_{\pm 0.0}$	40.1$_{\pm 0.1}$	39.9$_{\pm 0.1}$	41.0
BIWAA	40.1$_{\pm 0.2}$	41.5$_{\pm 0.2}$	45.1$_{\pm 1.3}$	40.3$_{\pm 0.2}$	48.0$_{\pm 2.9}$	43.3$_{\pm 3.0}$	40.2$_{\pm 0.0}$	**50.4**$_{\pm 0.5}$	45.7$_{\pm 3.8}$	40.3$_{\pm 0.1}$	41.0$_{\pm 0.9}$	**43.1**$_{\pm 0.3}$	43.2
ToAlign	36.5$_{\pm 13.3}$	43.0$_{\pm 0.9}$	49.1$_{\pm 1.4}$	**40.8**$_{\pm 0.3}$	43.1$_{\pm 1.1}$	50.2$_{\pm 0.5}$	40.7$_{\pm 0.4}$	42.8$_{\pm 1.3}$	48.4$_{\pm 3.2}$	40.5$_{\pm 0.2}$	43.4$_{\pm 0.9}$	42.6$_{\pm 1.6}$	43.4
DNAN	42.9$_{\pm 1.6}$	43.4$_{\pm 1.2}$	**53.7**$_{\pm 0.6}$	**40.8**$_{\pm 0.2}$	45.3$_{\pm 3.0}$	**53.9**$_{\pm 1.0}$	**40.8**$_{\pm 0.4}$	48.6$_{\pm 0.8}$	**53.4**$_{\pm 2.9}$	**40.6**$_{\pm 0.2}$	**44.1**$_{\pm 1.5}$	42.8$_{\pm 0.9}$	**45.9**
DNAN*	**43.2**$_{\pm 1.4}$	**44.9**$_{\pm 1.2}$	51.1$_{\pm 0.7}$	40.7$_{\pm 0.3}$	**49.8**$_{\pm 2.8}$	50.3$_{\pm 1.2}$	40.6$_{\pm 0.3}$	**50.8**$_{\pm 1.3}$	48.5$_{\pm 1.6}$	40.4$_{\pm 0.2}$	43.1$_{\pm 0.6}$	43.0$_{\pm 0.9}$	45.5

Ego-network Results Results for this dataset are presented in Table 1. In this benchmark, twelve UDA tasks can be defined by pairing the four domains. Our experimental results indicate that DNAN performs the best on average and achieves state-of-the-art performance on nine tasks: O to T, O to W, O to D, T to O, T to D, W to O, W to D, D to O, and D to T. DNAN has good performances on T to W and W to T,

Table 2: Perofmrance results on IMDB&REDDIT dataset

Task	Source	DANN	MMD	DIVA	SDAT	BIWAA	ToAlign	DNAN	DNAN*
I→R	$63.4_{\pm 0.2}$	$63.9_{\pm 0.8}$	$63.7_{\pm 0.4}$	$63.6_{\pm 0.5}$	$63.6_{\pm 0.6}$	$64.0_{\pm 0.8}$	$63.3_{\pm 0.2}$	$64.2_{\pm 0.6}$	$\mathbf{65.1}_{\pm 0.4}$
R→I	$72.3_{\pm 1.7}$	$72.0_{\pm 1.7}$	$73.6_{\pm 1.7}$	$71.1_{\pm 0.3}$	$74.1_{\pm 2.0}$	$71.4_{\pm 1.0}$	$73.4_{\pm 0.8}$	$\mathbf{74.9}_{\pm 2.0}$	$73.7_{\pm 1.4}$
Avg	67.8	68.0	67.3	68.0	68.8	67.7	68.3	**69.6**	69.4

and achieves SOTA performance on D to T and T to D tasks, showing that DNAN can successfully handle similar domains, as Digg, Twitter, and Weibo are similar content-sharing platforms. Notably, it exceeds the second-best methods by about 4% on T to D and about 3% on W to D. Additionally, DNAN can also achieve SOTA performance when there is a large distribution gap between domains, such as on tasks between OAG and Twitter or OAG and Weibo. It is important to underscore that no single method can achieve the best performance on all tasks, likely due to the diverse range of domain gaps.

IMDB&REDDIT Results Results for this dataset are presented in Table 2. We note that the experiments demonstrate that all UDA methods perform better on the REDDIT to IMDB task (R to I) than on the IMDB to REDDIT task (I to R), indicating that the two tasks are not equally challenging. We hypothesize that the smaller size of IMDB-BINARY compared to REDDIT-BINARY may result in performance degradation when testing on the larger REDDIT-BINARY dataset, as less knowledge can be transferred to the target domain. Our experimental results show that DNAN outperforms all other methods on both the R to I and I to R tasks, leading to state-of-the-art results on average. On the I to R task, we observe that all methods perform similarly. Although DNAN does not outperform other methods by a large margin on the I to R task, the results still indicate that DNAN has competitive performance compared to other methods. It is worth noting that the performance of a UDA algorithm may vary to some extent based on hyperparameter tuning. Therefore, when comparing two UDA algorithms with similar performance, they should be considered equally competitive. Based on this consideration, we can conclude that our proposed method performs competitively on all the UDA tasks and outperforms other UDA methods on average. These findings suggest that DNAN can serve as a robust UDA algorithm.

Our theoretical framework demonstrates that any augmentation method applied to the adjacency matrix can be modeled as adding an extra stochastic layer, and Theorem 1 remains valid. In addition to the augmentation method in Equation 5, we performed experiments using "node dropping" data augmentation. We tested our method with the "node dropping" data augmentation technique, which involves removing a node and its connections in a graph with a given probability. The results of our tests (DNAN*) show that while not as effective as the edge-adding/dropping augmentation method, it achieves the second-best average result.

5.3 Analytic and Ablative and Experiments

We first perform analytic experiments to offer deeper insight into our approach. We then perform an ablative experiment to demonstrate that all components in our algorithm are important for achieving optimal performance.

The effect of DNAN on data representations in the output space of the classifier To evaluate the effectiveness of our proposed approach, we analyze how DNAN influences the target domain's distribution in the classifier's output space on the Oag to Weibo task (O to W). We chose this task to demonstrate the effect of our model because it is challenging; Weibo and Oag are very dissimilar platforms, one connected by co-authorship and the other by friendship. We utilize the UMAP (McInnes et al., 2018) visualization tool and compare the representations of the source domain's test data, the target domain's test data before using DNAN, and the target domain's test data after applying DNAN. In Figure 2, each point represents a single data point in the output space of the classifier before the softmax activation. Blue and red colors denote the two classes. In Figure 2, the middle plot shows that the classifier does not work well with the target domain data before adaptation. It is hard to distinguish the class boundary as red dots are mixed with

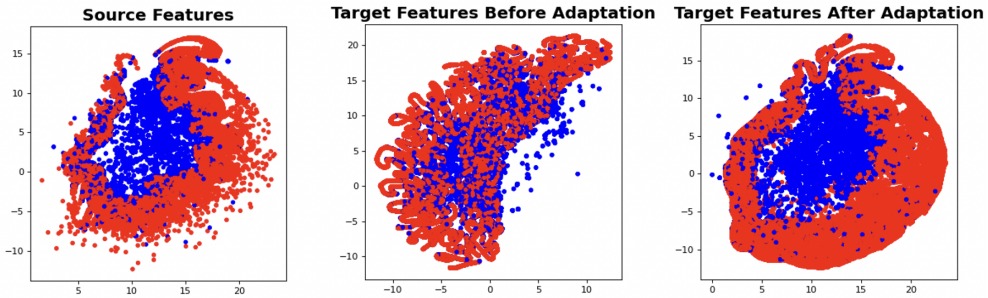

Figure 2: UMAP visualizations show the test data representations before softmax activation for the Oag to Weibo task. Blue and red points denote the different classes. The middle plot displays the target domain data representations obtained from a model trained on the source dataset before adaptation. The left and right plots show the source and target domain data representations after adaptation using DNAN.

blue ones. However, after applying DNAN, the class boundary becomes clearer, and the data representation distribution of the target domain matches well with the source domain. This is evident in the left and right plots of Figure 2, where the patterns of dots are consistent. These visualization results demonstrate that DNAN successfully mitigates the performance degradation caused by the domain shift.

Table 3: Ablation Study Results on Ego-network Dataset

Method	O→T	O→W	O→D	T→O	T→W	T→D	W→O	W→T	W→D	D→O	D→T	D→W	Avg
DNAN-D	42.8$_{\pm 1.3}$	42.5$_{\pm 1.7}$	52.3$_{\pm 2.6}$	40.5$_{\pm 0.2}$	**46.9**$_{\pm 2.0}$	50.0$_{\pm 3.5}$	40.4$_{\pm 0.2}$	**50.1**$_{\pm 0.6}$	52.8$_{\pm 2.4}$	**40.6**$_{\pm 0.2}$	42.1$_{\pm 1.3}$	42.5$_{\pm 1.8}$	45.4
DNAN-N	**44.4**$_{\pm 2.2}$	43.1$_{\pm 0.7}$	52.6$_{\pm 1.0}$	41.0$_{\pm 0.3}$	45.0$_{\pm 2.4}$	52.7$_{\pm 2.7}$	40.7$_{\pm 0.3}$	46.9$_{\pm 2.2}$	53.3$_{\pm 2.5}$	40.5$_{\pm 0.2}$	43.0$_{\pm 1.1}$	43.6$_{\pm 0.9}$	45.6
DNAN-L	44.1$_{\pm 1.2}$	43.1$_{\pm 1.2}$	53.6$_{\pm 1.4}$	40.8$_{\pm 0.3}$	46.6$_{\pm 2.7}$	52.9$_{\pm 3.4}$	**40.8**$_{\pm 0.3}$	48.0$_{\pm 2.6}$	53.3$_{\pm 3.3}$	40.5$_{\pm 0.2}$	43.0$_{\pm 0.6}$	**44.3**$_{\pm 1.3}$	**45.9**
Denoise	43.6$_{\pm 1.6}$	42.8$_{\pm 1.3}$	53.2$_{\pm 2.5}$	40.7$_{\pm 0.3}$	47.2$_{\pm 1.9}$	53.8$_{\pm 1.8}$	40.6$_{\pm 0.3}$	44.3$_{\pm 7.6}$	**53.9**$_{\pm 1.8}$	**40.6**$_{\pm 0.2}$	42.2$_{\pm 0.3}$	43.4$_{\pm 1.3}$	45.5
NWD	40.9$_{\pm 0.7}$	41.9$_{\pm 0.3}$	52.8$_{\pm 1.4}$	40.4$_{\pm 0.2}$	50.4$_{\pm 1.7}$	42.4$_{\pm 1.2}$	40.3$_{\pm 0.1}$	49.8$_{\pm 1.3}$	47.9$_{\pm 1.5}$	40.3$_{\pm 0.1}$	46.2$_{\pm 0.3}$	43.8$_{\pm 2.1}$	44.8
$\mathcal{L}_e$	40.8$_{\pm 0.3}$	41.8$_{\pm 0.8}$	49.2$_{\pm 0.6}$	**42.0**$_{\pm 0.2}$	45.7$_{\pm 3.2}$	49.9$_{\pm 0.3}$	**40.8**$_{\pm 0.2}$	47.1$_{\pm 2.2}$	48.9$_{\pm 0.7}$	40.4$_{\pm 0.2}$	42.5$_{\pm 0.3}$	41.4$_{\pm 2.2}$	44.2
DNAN	42.9$_{\pm 1.6}$	**43.4**$_{\pm 1.2}$	**53.7**$_{\pm 0.6}$	40.8$_{\pm 0.2}$	45.3$_{\pm 3.0}$	**53.9**$_{\pm 1.0}$	**40.8**$_{\pm 0.4}$	48.6$_{\pm 0.8}$	53.4$_{\pm 2.9}$	**40.6**$_{\pm 0.2}$	**44.1**$_{\pm 1.5}$	42.8$_{\pm 0.9}$	**45.9**

Table 4: Ablation Study Results on IMDB&REDDIT Dataset

Task	DNAN-D	DNAN-N	DNAN-L	Denoise	NWD	$\mathcal{L}_e$	DNAN
I to R	63.8$_{\pm 0.4}$	64.0$_{\pm 0.5}$	64.2$_{\pm 0.4}$	**65.2**$_{\pm 0.6}$	63.8$_{\pm 0.4}$	64.8$_{\pm 1.5}$	64.2$_{\pm 0.6}$
R to I	72.3$_{\pm 2.4}$	74.2$_{\pm 1.8}$	73.9$_{\pm 2.0}$	74.0$_{\pm 1.8}$	72.8$_{\pm 1.4}$	74.1$_{\pm 1.6}$	**74.9**$_{\pm 2.0}$
Avg	68.0	69.0	69.1	**69.6**	68.3	69.4	**69.6**

Ablative study The ablation experiments were conducted to demonstrate the effectiveness of the two main ideas we used to develop DNAN, as well as the auxiliary maximum entropy regularization. To achieve this, we removed one of the three components at a time and reported our performance. We denote the ablated versions of DNAN as: (i) **DNAN-D**: We exclude the denoising mechanism and apply only the NWD loss and the maximum entropy loss. (ii) **DNAN-N**: We exclude the NWD loss and apply only the denoising mechanism and the maximum entropy loss. (iii) **DNAN-L**: We exclude the maximum entropy loss and apply only the NWD loss and the denoising mechanism. (iv) **Denoise**: We only apply the denoising mechanism. (v) **NWD**: We only apply the NWD loss. (vi) $\mathcal{L}_e$: We only apply the maximum entropy term.

For the Ego-network dataset, the results reveal that the integration of all components yields the highest average performance at 45.9%. The DNAN-D configuration, which lacks the denoising mechanism, shows competitive performance with an average of 45.4%. However, the DNAN-N configuration, which excludes the NWD loss, displays an even smaller decrease in performance, with an average of 45.6%. Additionally,

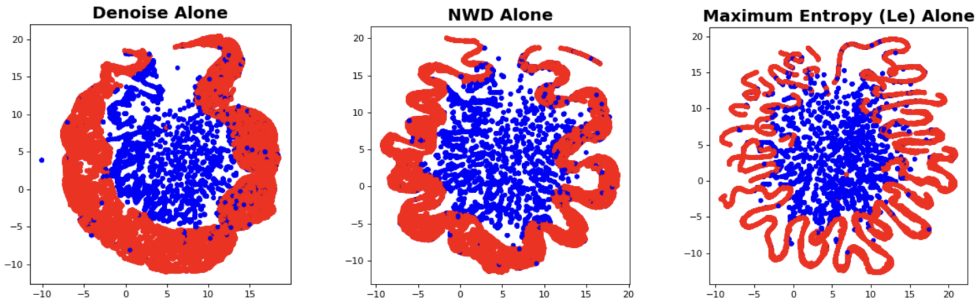

Figure 3: UMAP visualizations show the test data representations in the target domain before softmax activation for the Oag to Twitter task. Blue and red points denote different classes. The first plot displays the data representations obtained from a model trained using only the denoising mechanism. The second and third plots show the data representations after adaptation using only NWD or Maximum Entropy, respectively.

the results from DNAN-L indicate that the entropy term does not play as significant a role, with an average performance similar to DNAN but falling short in specific tasks. Applying each component individually, the denoising mechanism (Denoise) provides the highest performance on average (45.5%), highlighting the crucial role of the denoising mechanism in enhancing feature extraction and robustness. The Nuclear-Norm Wasserstein Discrepancy (NWD) and maximum entropy regularization ($\mathcal{L}_e$) also improve the performance but to a lesser extent.

For the IMDB&REDDIT dataset, the full DNAN model again demonstrates superior performance with an average score of 69.6%. Interestingly, the DNAN-N variant outperforms the DNAN-D with averages of 69% and 68.0%, respectively, indicating the greater importance of the denoising mechanism in this context. The DNAN-L variant shows a slight reduction in performance, with a 0.5% decrease in the average score compared to DNAN. When analyzing the contribution of individual components, the denoising mechanism (Denoise) again provides the highest performance on average (69.6%). The maximum entropy regularization ($\mathcal{L}_e$) improves the performance more (69.4%), while the Nuclear-Norm Wasserstein Discrepancy (NWD) has the lowest performance (68.3%) on this dataset. This suggests that the IMDB&REDDIT dataset contains more variations and noise in graph structure, which hinders distribution alignment more significantly compared to the Ego-network dataset. Overall, these results demonstrate the importance of each component in our proposed method. The denoising mechanism plays the most critical role, underscoring the effectiveness of using denoising techniques to tackle graph domain adaptation.

In addition to quantitative evaluation, we provide qualitative UMAP visualizations to show the target domain test data representations before softmax activation for the Oag to Twitter task in Figure 3. The first plot (Denoise Alone) exhibits the simplest and clearest decision boundary, with blue dots densely clustered in the center and red dots forming a distinct boundary around them with minimal overlap, resulting in a well-defined separation between the two classes. The red points are uniformly distributed around the blue cluster, enhancing the clarity of the boundary. The second plot (NWD Alone) shows a moderately clear boundary, where blue dots are still centrally clustered but with slightly less density, and the red dots are more intricately intertwined with the blue dots, leading to a less sharp but still noticeable boundary. The red points here are less uniformly distributed, causing more intermixing with the blue points. The third plot (Maximum Entropy (Le) Alone) has the most intricate and unclear decision boundary, with significant intermixing of blue and red dots. The red points are densely scattered throughout, resulting in indistinct clusters and a blurred boundary. The qualitative visualization results are consistent with the quantitative evaluations, and the effect of each component is more clearly reflected.

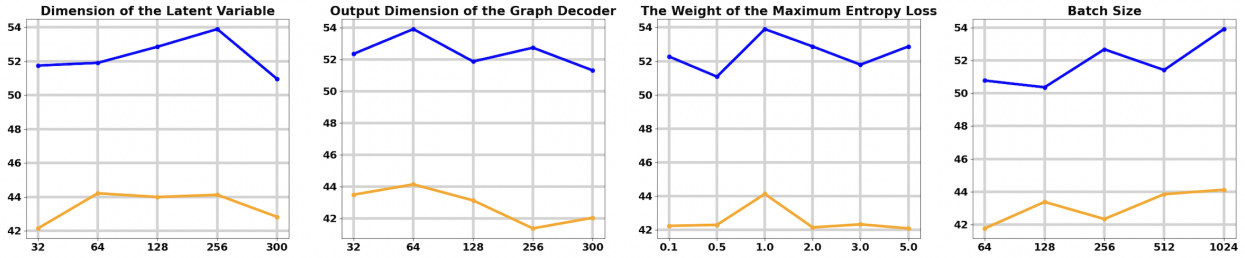

Figure 4: The performance of DNAN with different hyperparameter settings on Twitter to Digg (Blue lines) and Digg to Twitter (Yellow lines) tasks.

5.4 Hyperparameters Sensitivity Analysis

An important concern for most algorithms is tuning the hyperparameters and measuring the performance sensitivity with respect to them. We evaluate the sensitivity of DNAN with respect to various hyperparameters on two tasks: Twitter to Digg (T to D) and Digg to Twitter (D to T). We varied the dimension of the latent embedding space, the output dimension of the graph decoder (before taking the dot products), the weight of the maximum entropy loss ($\mathcal{L}_e$), and the batch size. We present the F1-scores of the DNAN model as a linear function of these hyperparameters in Figure 4, with the blue lines representing the T to D task and the yellow lines representing the D to T task. Through inspecting this figure, we deduce:

- **Dimension of the Latent Embedding Space:** We test the performance of DNAN on five different dimension sizes for the embedding space: 32, 64, 128, 256, and 300. The performance of DNAN peaks at a latent variable size of 256 for the T to D task and shows a less pronounced peak on the D to T task, indicating that a moderately large value for the dimension of the latent variable is beneficial for capturing the salient features of the data. Performance declines when the dimension is too small to capture the complexity or too large, potentially introducing noise or overfitting. However, the results indicate that the performance remains relatively decent across a wide range of embedding sizes.
- **Output Dimension of the Graph Decoder:** Similar to the experiments on the dimension of the latent variable, we test the performance of DNAN on five dimension sizes: 32, 64, 128, 256, and 300. The output dimension of the graph decoder shows a performance peak at 64 for both the T to D task and the D to T task. This observation suggests that a moderately small representation capacity in the graph decoder is more beneficial. Compared with performances on the D to T task, the T to D task is less sensitive to this hyperparameter.
- **Weight of the $\mathcal{L}_e$:** We test DNAN with six weights: 0.1, 0.5, 1.0, 2.0, 3.0, 5.0. The weight of the maximum entropy loss presents a clear peak at 1.0 for both the T to D and D to T tasks, suggesting that a balanced contribution of the entropy loss is helpful for performance.
- **Batch Size:** We test five batch sizes: 64, 128, 256, 512, 1024. For batch size, there is a trend of increasing performance as the size grows, with a notable peak at a batch size of 1024 for both the T to D and D to T tasks. This implies that the performance of DNAN benefits from larger batch sizes, possibly due to more stable gradient estimates. Compared with the T to D task, the D to T task is less affected by batch size variations.

The sensitivity analysis of hyperparameters for the DNAN model on the T to D and D to T tasks demonstrates the stability of DNAN models when using different hyperparameter values, as there is moderate fluctuation around $\pm 3\%$. However, fine-tuning hyperparameters to the specific characteristics of the task and dataset is beneficial. Although optimal performance is achieved with a latent variable dimension of 256, a decoder output dimension of 64, an entropy loss weight of 1.0, and a batch size of 1024, tuning the hyperparameters is not essential to achieve performance in the competitive range.

Table 5: Training time for IMDB&REDDIT dataset

Task	DANN	MMD	DIVA	SDAT	BIWAA	ToAlign	DNAN
I→R	10	260	14	12	899	10	10
R→I	3	2	5	6	34	3	5

Table 6: Model complexity on IMDB&REDDIT-BINARY dataset. F represents the input feature dimension, D_h represents the hidden dimension, and D represents the output dimension of the decoder.

DANN	MMD	DIVA	SDAT	BIWAA	ToAlign	DNAN
$(F+3D_h+3)D_h$	$(F+4D_h+4)D_h$	$(F+11D_h+3D+3)D_h$	$(F+3D_h+4)D_h$	$(F+3D_h+3)D_h$	$(F+2D_h+11)D_h$	$(F+4D_h+D+2)D_h$

5.5 Time Complexity and Model Complexity Analysis

In this section, we present an analysis of the training time (in minutes) and model complexity for our methods compared to others. The time complexity is detailed in Table 5, and the model complexity is outlined in Table 6.

- **Training Time:** The training times reported in Table 5 illustrate the efficiency of the DNAN model relative to its counterparts. For the I to R task, DNAN required 10 minutes, positioning it as the fastest in terms of training time, alongside DANN and ToAlign, compared to other methods. In the R to I task, DNAN again demonstrated moderate efficiency with 5 minutes, with MMD being the fastest at 2 minutes and BIWAA the slowest at 34 minutes. These results suggest that DNAN provides a balanced trade-off between model performance and training efficiency without adding significant computational overload.
- **Model Complexity:** The model complexity, as shown in Table 6, is assessed based on the number of parameters in the models, which is a function of the input feature dimension (F), hidden dimension (D_h), and the output dimension of the decoder before taking the dot products (D). Compared to other methods like DANN and SDAT, which have similar forms, DNAN introduces additional complexity due to the parameters in the graph decoder. However, it remains less complex than DIVA, which includes an extra $(7D_h+2D+1)D_h$ term.

We conclude that the DNAN model shows competitive training times that are significantly lower than the most time-consuming method (BIWAA) while maintaining better performance. Model complexity analysis reveals that DNAN, while not the simplest, avoids the higher complexity seen in more complex methods such as DIVA. DNAN balances the computational cost with the capacity to learn and transfer knowledge effectively for better UDA performance. This observation is important because, in certain applications, it is crucial to perform UDA quickly due to the constant changes in the input distribution and the limited time available to update the model.

6 Conclusions

We developed a new UDA method specifically designed for graph-structured data. Our proposed method includes denoising and using the NWD for domain alignment in a shared embedding space. The experiments demonstrate that our approach is a promising method. By innovatively combining domain alignment through NWD with a denoising mechanism via a Variational Graph Autoencoder, DNAN has outperformed state-of-the-art methods across two major benchmarks without adding significant computational overload. The ability of our method to handle both subtle and significant domain differences showcases its versatility and robustness. From ablative studies, the two ideas that DNAN benefits from are proven to be crucial for optimal performance. Future work can explore extending our approach to partial domain adaptation scenarios or situations where the source domain data is not directly accessible.

References

Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 44–54, 2006.

Amélie Barbe, Paulo Gonçalves, Marc Sebban, Pierre Borgnat, Rémi Gribonval, and Titouan Vayer. Optimization of the diffusion time in graph diffused-wasserstein distances: Application to domain adaptation. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 786–790. IEEE, 2021.

Vladimir Batagelj and Matjaz Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Phillip Bonacich. Power and centrality: A family of measures. *American journal of sociology*, 92(5):1170–1182, 1987.

Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and Radon Wasserstein barycenters of measures. *Journal of Mathematical Imaging and Vision*, 51(1):22–45, 2015.

Ruichu Cai, Fengzhu Wu, Zijian Li, Pengfei Wei, Lingling Yi, and Kun Zhang. Graph domain adaptation: A generative view. *arXiv preprint arXiv:2106.07482*, 2021.

Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pp. 1407–1418. PMLR, 2021.

Lin Chen, Huaian Chen, Zhixiang Wei, Xin Jin, Xiao Tan, Yi Jin, and Enhong Chen. Reusing the task-specific classifier as a discriminator: Discriminator-free adversarial domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7181–7190, 2022.

Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

N. Courty, R. Flamary, D. Tuia, and A. Rakotomamonjy. Optimal transport for domain adaptation. *IEEE TPAMI*, 39(9):1853–1865, 2017a.

Nicolas Courty, Rémi Flamary, Amaury Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. *Advances in neural information processing systems*, 30, 2017b.

Shuhao Cui, Shuhui Wang, Junbao Zhuo, Liang Li, Qingming Huang, and Qi Tian. Towards discriminability and diversity: Batch nuclear-norm maximization under label insufficient situations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3941–3950, 2020.

Shuhao Cui, Shuhui Wang, Junbao Zhuo, Liang Li, Qingming Huang, and Qi Tian. Fast batch nuclear-norm maximization and minimization for robust domain adaptation. *arXiv preprint arXiv:2107.06154*, 2021.

B. Damodaran, B. Kellenberger, R. Flamary, D. Tuia, and N. Courty. Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation. *arXiv preprint arXiv:1803.10081*, 2018.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pp. 1180–1189. PMLR, 2015.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

Zhiqiang Gao, Shufei Zhang, Kaizhu Huang, Qiufeng Wang, and Chaoliang Zhong. Gradient distribution alignment certificates better adversarial domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8937–8946, 2021.

Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. In *European Conference on Computer Vision*, pp. 597–613. Springer, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144, 2020.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017a.

William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Maximilian Ilse, Jakub M Tomczak, Christos Louizos, and Max Welling. Diva: Domain invariant variational autoencoders. In *Medical Imaging with Deep Learning*, pp. 322–348. PMLR, 2020.

Daniel Im Im, Sungjin Ahn, Roland Memisevic, and Yoshua Bengio. Denoising criterion for variational auto-encoding framework. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Guoliang Kang, Lu Jiang, Yi Yang, and Alexander G Hauptmann. Contrastive adaptation network for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4893–4902, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In *International conference on machine learning*, pp. 17391–17408. PMLR, 2023.

Vladimir Kulikov, Shahar Yadin, Matan Kleiner, and Tomer Michaeli. Sinddm: A single image denoising diffusion model. In *International Conference on Machine Learning*, pp. 17920–17930. PMLR, 2023.

Vinod Kumar Kurmi and Vinay P Namboodiri. Looking back at labels: A class based domain adaptation technique. In *2019 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2019.

Xujia Li, Yuan Li, Xueying Mo, Hebing Xiao, Yanyan Shen, and Lei Chen. Diga: Guided diffusion model for graph recovery in anti-money laundering. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4404–4413, 2023.

Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain adaptation with residual transfer networks. *Advances in neural information processing systems*, 29, 2016.

Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. *arXiv preprint arXiv:1511.00830*, 2015.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pp. 8162–8171. PMLR, 2021.

Andrey Okhotin, Dmitry Molchanov, Arkhipkin Vladimir, Grigory Bartosh, Viktor Ohanesian, Aibek Alanov, and Dmitry P Vetrov. Star-shaped denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 36, 2024.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2110–2119, 2018.

Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pp. 435–446. Springer, 2011.

Harsh Rangwani, Sumukh K Aithal, Mayank Mishra, Arihant Jain, and R. Venkatesh Babu. A closer look at smoothness in domain adversarial training. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Ievgen Redko, Amaury Habrard, and Marc Sebban. Theoretical analysis of domain adaptation with optimal transport. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part II 10*, pp. 737–753. Springer, 2017.

Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling. *GAMM-Mitteilungen*, 44(2): e202100008, 2021.

Dohoon Ryu and Jong Chul Ye. Pyramidal denoising diffusion probabilistic models. *arXiv preprint arXiv:2208.01864*, 2022.

Hui Tang and Kui Jia. Discriminative adversarial domain adaptation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5940–5947, 2020.

Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. Structural diversity in social contagion. *Proceedings of the national academy of sciences*, 109(16):5962–5966, 2012.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684): 440–442, 1998.

Guoqiang Wei, Cuiling Lan, Wenjun Zeng, Zhizheng Zhang, and Zhibo Chen. Toalign: task-oriented alignment for unsupervised domain adaptation. *Advances in Neural Information Processing Systems*, 34:13834–13846, 2021.

Thomas Westfechtel, Hao-Wei Yeh, Qier Meng, Yusuke Mukuta, and Tatsuya Harada. Backprop induced feature weighting for adversarial domain adaptation with iterative label distribution alignment. *Winter Conference on Applications of Computer Vision (WACV)*, 2023.

Man Wu, Shirui Pan, Chuan Zhou, Xiaojun Chang, and Xingquan Zhu. Unsupervised domain adaptive graph convolutional networks. In *Proceedings of The Web Conference 2020*, pp. 1457–1467, 2020.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Cheng Yang, Lijing Liang, and Zhixun Su. Real-world denoising via diffusion model. *arXiv preprint arXiv:2305.04457*, 2023.

Yizhou Zhang, Guojie Song, Lun Du, Shuwen Yang, and Yilun Jin. Dane: Domain adaptive network embedding. *arXiv preprint arXiv:1906.00684*, 2019a.

Yuchen Zhang, Tianle Liu, Mingsheng Long, and Michael Jordan. Bridging theory and algorithm for domain adaptation. In *International Conference on Machine Learning*, pp. 7404–7413. PMLR, 2019b.

A Implementation Details of DNAN

In this section, we present our implementations of DNAN. Our codes are in Python, mainly with PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey & Lenssen, 2019) libraries. We train five models for every baseline using five random seeds for parameter initialization. The five random seeds are 27, 28, 29, 30, and 31. We also conducted a hyperparameter search, described in the hyperparameter sensitivity section in the paper, to find suitable hyperparameters for optimal performance. The hyperparameters we use to achieve the results listed in the main paper are presented in Table 7.

Parameter	Ego-network	IMDB&REDDIT
Batch size	1024	64
Learning rate	0.01	0.001
Dropout rate	0.5	0.2
Encoder hidden size	256	128
Decoder output size	64	128
Learning decay rate	0.75	0.75
Entropy weight	1.0	1.0
Weight decay	0.0005	0.0005
p_{add}	0.1	0.1
p_{drop}	0.1	0.1

Table 7: Hyper-parameters of DNAN