UTILIZING GRAPH STRUCTURE FOR MACHINE LEARNING

A Dissertation Presented

by

STEFAN E. DERNBACH

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2020

College of Information and Computer Sciences

UTILIZING GRAPH STRUCTURE FOR MACHINE LEARNING

A Dissertation Presented

by

STEFAN E. DERNBACH

Approved as to style and content by:

Don Towsley, Chair

Ben Marlin, Member

Andrew McCallum, Member

Weibo Gong, Member

James Allan, Chair
College of Information and Computer Sciences

ACKNOWLEDGMENTS

I would like to thank my advisor, Don Towsley, who took me on when I was an orphaned graduate student. Don gave me the freedom to set my own research direction while being there with advice and support throughout.

Thank you to my mentors who guided me before I joined Don's lab: Sridhar Mahadevan and Jim Kurose.

Thanks to my family. I would not be here without them. They shared in every high and every low of my journey.

Thank you to my wonderful labmates in both the Networks Lab and the Autonomous Learning Lab who provided both a partner to bounce ideas off of and to solve problems with as well as friendship: CJ Carey, Thomas Bucher, Phil Thomas, Will Dabney, Bruno Castro da Silva, Stephen Giguere, Francisco Garcia, Clemens Rosenbaum, Bo Liu, Arman Mohseni Kabir, Kun Tu, and Gayane Vardoyan. Extra thanks to Archan Ray and Blossom Metevier who were great friends in and out of the lab and whom I bonded deeply with through our shared suffering of watching Game of Thrones.

Thank you to the friends who always had my back: John and Sara Talbot and Alvin Le, who will probably continue to point out every missing comma in my future.

Thank you to the coauthors and collaborators I worked with along the way: Nina Taft, Udi Weinsberg, Siddharth Pal, and Bruno Ribeiro.

Thank you to my committee: Weibo Gong, Ben Marlin, and Andrew McCallum for the time they devoted to this thesis.

And thank you to the wonderful people at UMASS CICS who got me from A to B and made sure I didn't accidentally drop out of the college along the way: Leeanne Leclerc, Susan Overstreet, and Laurie Connors.

ABSTRACT

UTILIZING GRAPH STRUCTURE FOR MACHINE LEARNING

SEPTEMBER 2020

STEFAN E. DERNBACH

B.Sc., WHITWORTH UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Don Towsley

The information age has led to an explosion in the size and availability of data. This data often exhibits graph-structure that is either explicitly defined, as in the web of a social network, or is implicitly defined and can be determined by measuring similarity between objects. Utilizing this graph-structure allows for the design of machine learning algorithms that reflect not only the attributes of individual objects but their relationships to every other object in the domain as well.

This thesis investigates three machine learning problems and proposes novel methods that leverage the graph-structure inherent in the tasks. Quantum walk neural networks are classical neural nets that use quantum random walks for classifying and regressing on graphs. Asymmetric directed node embeddings are another neural network architecture designed to embed the nodes of a directed graph into a vector space. Filtered manifold alignment is a novel two-step approach to domain adaptation.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Networks and graphs are ubiquitous in the modern information age. Graphs form the basis for modeling everything from physical systems, such as transportation networks, to relational systems, such as social networks. The representational power of graphs makes them an essential tool in many data processing tasks. While unstructured machine learning methods are unable to distinguish two samples with identical features, graph-based algorithms can classify such objects independently based on their relationships to other data points. This is a core distinguishing advantage to machine learning methods that utilize graph-structure in the data. This thesis studies a variety of problems in machine learning in which graphs are either defined within the scope of the problem or are constructed as part of the solution.

Graph signal processing [66] is a field that studies how to apply standard signal processing techniques to signals that lie on graphs as well as develops unique methods that do not have parallels in standard signal processing. Tasks in graph signal processing deal with classifying entire graphs or individual nodes within the graphs. In these problems, the structure of the graph and the signal are intertwined and both contribute to the classifications. Alternatively, regression can replace classification as the goal such that a graph and signal map to a continuous quantity rather than a discrete label.

Node embedding is the process of mapping the individual nodes of a graph to a vector space while preserving some aspects of the relationships encoded in the original graph. These vectors are mathematical representations of the objects in the original

graph that are easily manipulated by downstream tasks. Stochastic deep learning approaches based upon random walks on the graph [41, 68] have become popular methods of node embedding due to their scalability to large web-scale graphs. These approaches have strong parallels to word embedding methods in natural language processing [62].

Domain adaptation is a subset of transfer learning concerned with applying knowledge from one source domain to another related target domain. Transferring knowledge from the source domain to the target domain often requires learning a transformation due to differing feature distributions or entirely different feature sets between the domains. Manifold based alignment procedures [43, 96] project the two domains onto a shared manifold using graphs as discrete approximations for the continuous manifolds.

This work presents several novel techniques that span both a variety of machine learning tasks as well as multiple learning paradigms. These tasks range from graph and node classification and regression to node embedding to domain adaptation. Two techniques presented are deep learning methods which sample data from a training set and iteratively improve performance on the designated task. A third technique offers a more classical machine learning method in which a closed form solution for processing the entire dataset in one go is presented.

1.1 Contributions

This dissertation makes several contributions to the study of machine learning by utilizing graph-structure and dynamics. Graphs provide powerful tools for modeling the interactions between objects in a form that can aid in processing both the individual objects as well as collections of objects. We propose three machine learning algorithms and multiple variations that cover a range of tasks and disciplines spanning supervised, semi-supervised, and unsupervised learning.

The first contribution of the dissertation is ***quantum walk neural networks***, QWNN (Chapter 3), a neural network model based on using quantum random walks to propagate information throughout a graph. QWNNs take a graph and a signal across the nodes in the graph as input and outputs a new diffused signal that is the result of a quantum random walk on the graph. This diffused signal is usable across a span of different types of tasks such as regression or classification on the graph. Quantum walk neural networks are formulated as a fully differentiable sequence of layers allowing backpropagation and the use of training data to tune the parameters of the quantum walk to improve the performance of the diffused features for the downstream learning task. Additionally, the output of QWNN is equivariant to node ordering.

The second contribution of the dissertation is ***asymmetric node similarity embeddings***, ANSE (Chapter 4), a neural network that embeds the nodes of a directed graph into a vector space using an asymmetric similarity function. Unlike embedding methods designed for undirected graphs, ANSE preserves both unidirectional relationships as well as bidirectional relationships in the embedding space. ANSE learns both the vector representations of the nodes and the parameters of the similarity function simultaneously in a stochastic setting by taking random walks on the directed graph. This allows the technique to scale to web-sized graphs.

The third contribution of the dissertation is ***semi-supervised filtered manifold alignment***, FMA (Chapter 5), a machine learning method that can efficiently embed multiple datasets into a vector space in which samples from different datasets can be directly compared. FMA takes a two-step approach to domain adaptation, first projecting each domain to a low dimensional space and then aligning these spaces. This approach leads to a reduced computational complexity of FMA compared to similar domain adaptation algorithms and provides improved alignment benefits from filtering noise in each domain. We propose both nonlinear (instance-

based) and linear (feature-based) variations of the algorithm. Additionally, FMA can embed samples not in the initial alignment set and can align any number of domains simultaneously.

The rest of the dissertation is organized as follows. Chapter 2 provides the background and related work for the research. Chapters 3, 4, and 5 present the major contributions of this work in the order listed above. Each of these chapters outlines their respective method and provides supporting experiments. Finally, Chapter 6 provides concluding remarks.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter provides an outline of the terminology necessary for developing and understanding graph based machine learningtechniques and provides a review of related work in graph signal processing and the machine learning problems addressed in this work.

2.1 Matrix and Tensor Notation

Throughout this work, lower case letters (x) are used to represent scalar variables and capital variables (X) are used to represent scalar constants. Bolded lowercase letters ($\boldsymbol{\phi}$) represent vectors and bolded uppercase letters ($\boldsymbol{\Phi}$) represent matrices and higher dimensional tensors. The i, j element of a tensor is given by subscripts $\mathbf{X}_{ij}$, the i row of a tensor is given by a single subscript $\mathbf{X}_i$, and the i section of a higher dimension of a tensor uses dots to indicate the number of preceding dimensions: e.g the i slice along the third dimension is $\mathbf{X}_{..i}$.

The inner or dot product of two vectors is given by: $\mathbf{a}^T\mathbf{b}$ or $\langle \mathbf{a}, \mathbf{b} \rangle$ and is equal to $\sum_i \mathbf{a}_i \mathbf{b}_i$. The outer product is $\mathbf{a} \otimes \mathbf{b}$ where $(\mathbf{a} \otimes \mathbf{b})_{ij} = \mathbf{a}_i \mathbf{b}_j$. Several forms of matrix and tensor multiplication are used. The inner product of $\mathbf{A}$ and $\mathbf{B}$ is given by $\mathbf{AB} = \sum_i \mathbf{A}_{..i} \otimes \mathbf{B}_{i...}$ We use a tensor double inner product, $\mathbf{A} \cdot \cdot \mathbf{B}$ to perform a contraction over the last two dimensions of $\mathbf{A}$ and the first two dimensions of $\mathbf{B}$:

$$\mathbf{A} \cdot \cdot \mathbf{B} = \sum_i \sum_j \mathbf{A}_{...ij} \otimes \mathbf{B}_{ji...}. \tag{2.1}$$

The generalization of an outer product to tensors is expressed as $\mathbf{A} \otimes \mathbf{B}$ where: $(\mathbf{A} \otimes \mathbf{B})_{m_1 m_2 m_3 \ldots n_1 n_2 n_3 \ldots} = \mathbf{A}_{m_1 m_2 m_3 \ldots} \mathbf{B}_{n_1 n_2 n_3 \ldots}$. Finally, the elementwise product of two vectors, matrices, or higher dimensional tensors is given by $\mathbf{a} * \mathbf{b}$ or $\mathbf{A} * \mathbf{B}$ where $(\mathbf{A} * \mathbf{B})_{ijk\ldots} = \mathbf{A}_{ijk\ldots} \mathbf{B}_{ijk\ldots}$.

2.2 Graph Terminology

A *simple graph*, $G = (V, E)$ is a set of vertices (nodes) $V = \{v_0, v_1, ..., v_{N-1}\}$ and a set of edges $E = \{(v_i, v_j) \ : \ v_i, v_j \in V\} \subseteq \{V \times V\}$. In general, *nodes* in the graph represent objects and *edges* represent correspondences or relationships between objects. The *degree* of a node $d(v_i)$ is equal to the number of edges incident to that node, thus $\sum_i d(v_i) = 2|E|$.

Graphs can be weighted, directed, both, or neither. A *weighted graph* defines a function w that maps each edge to a real value $w : E \to \mathbb{R}$. A *directed graph* (digraph) induces an ordering on each edge (v_i, v_j), so that it points from the source vertex v_i to the target vertex v_j. In this work, graphs that are weighted or directed will be explicitly stated as so and can otherwise be assumed to be unweighted and undirected.

Several categories of graphs have particular importance for the work in this thesis. A graph is *connected* if there is a path (a continuous sequence of edges $(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n) : (v_i, v_{i+1}) \in E$) between all pairs of vertices $(v_0, v_n) \in V \times V$. A graph is called *k-regular* if each vertex has the same degree, i.e. $\forall\ v \in V : d(v) = k$. Finally, a graph is *bipartite* if the nodes of the graph can be partitioned into two distinct non-overlapping sets such that all edges in the graph cross from one set to the other, i.e. given the partition $V{=}V_1 \cup V_2$ with $V_1 \cap V_2 = \emptyset$, for each edge $(v_i, v_j) \in E$, either $v_i \in V_1$ and $v_j \in V_2$ or $v_i \in V_2$ and $v_j \in V_1$.

Graphs are commonly represented by several different matrices. Their definitions below are given for the case of a simple graph, but most have intuitive extensions to

the weighted and directed graph cases as well. The adjacency matrix:

$$\mathbf{A}_{ij} \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

encodes the relationships between vertices and edges in the graph. Each vertex in the graph corresponds to a row and a column in the adjacency matrix and edges in the graph are represented by the value one (or alternatively the weight of the edge) in the corresponding row-column indices. The adjacency matrix is symmetric for an undirected graph: $\mathbf{A} = \mathbf{A}^T$. The degree matrix $\mathbf{D}$ is a diagonal matrix with diagonal elements $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij} = d(v_i)$. The graph Laplacian, $\mathbf{L}$, combines the degree and adjacency matrices: $\mathbf{L} = \mathbf{D} - \mathbf{A}$. The Laplacian is a diagonally dominant matrix with rows and columns both summing to 0. The eigenvalues of the Laplacian, $\lambda_i : 0 \leq i \leq min(2d_{max}, |V|)$, where d_{max} is the maximum degree of the graph, are real and nonnegative. The smallest eigenvalue of the graph Laplacian is 0 and has multiplicity equal to the number of disconnected components of the graph. The normalized Laplacian $\tilde{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ is often used in place of the regular Laplacian. The eigenvalues of the normalized Laplacian matrix are bounded: $0 = \lambda_0 \leq \lambda_i \leq 2$.

A discrete time *random walk* on a graph is a sequence of nodes produced by a walker that begins on node v of the graph and with uniform probability selects a random edge incident to v to traverse to node v' at each step in the walk. The k-neighborhood of a node in the graph is the set of all nodes reachable within a walk of length k. The probability distribution of the walker's location after t steps is given by $\mathbf{p}_t = (\mathbf{A}\mathbf{D}^{-1})^t\mathbf{p}_0$, where $\mathbf{p}_0$ is the initial distribution. All graphs have a stationary distribution $\mathbf{p}^* = \mathbf{A}\mathbf{D}^{-1}\mathbf{p}^*$, where $\mathbf{p}^*(v) = \frac{d(v)}{2|E|}$. For any connected, non-bipartite graph, any initial random walk distribution $\mathbf{p}_0$ will converge to $\mathbf{p}^*$ as $t \to \infty$. In the bipartite graph case, the walker distribution alternates between the two sets of nodes

in the graph. The graph can be amended to include a self loop (an edge from a node back to itself) on each vertex to remove the oscillatory effect that impedes the walker from reaching the stationary distribution.

2.3 Graph Signal Processing

Graph signal processing [83, 66] is a topic that focuses on computations using signals that lie on a graph. A signal over a graph is defined as a function that maps each vertex to a real value $f : V \to \mathbb{R}$. The signal is often expressed as a vector $\mathbf{f} \in \mathbb{R}^N$ where $\mathbf{f}_i = f(v_i)$.

Two common approaches to graph signal processing are spectral methods and spatial methods. Spectral methods take their name from performing a graph spectral transform, or graph Fourier transform, of the signal before processing. The graph Fourier transform is defined as the expansion of $\mathbf{f}$ onto the eigenvectors of the graph Laplacian. Given a matrix $\mathbf{U}$ whose columns are composed of the eigenvectors of $\mathbf{L}$ such that the eigenvalues of the associated eigenvectors are in nondecreasing order, the graph Fourier transform of $\mathbf{f}$ can be expressed as $\hat{\mathbf{f}} = \mathbf{U}^T\mathbf{f}$. This transform converts the graph signal from the vertex domain into the graph frequency domain. Afterwards, standard signal processing techniques for frequency signals can be applied.

The spatial approach to graph signal processing forgoes the spectral transform. Spatial methods use the adjacency and Laplacian matrices directly to perform computations based on node neighborhoods and diffusion processes (e.g. random walks) across the graph.

2.4 Graph Convolutional Neural Networks

Graph convolutional neural networks are a type of neural network architecture inspired by image convolutional neural networks. The translation of a convolutional

neural net from the image domain to the general graph domain is imprecise. Image convolutions are designed to exploit the rigid grid structure of pixels in an image, a structure that is not guaranteed for the nodes of an arbitrary graph. Several approaches to graph convolutions have thus been proposed which focus on translating specific aspects of image convolutional networks to the graph domain.

A convolution can be viewed as a function with local support over the graph. Alternatively, via the convolutional theorem, it can be viewed as a product of two functions in the frequency domain of the graph. Bruna et al. [23] propose a pair of graph convolutional neural networks that each tackled the problem of adapting a convolutional neural network from one of these two perspectives. The first approach connects inputs according to the structure of the graph. The N nodes of the graph are divided into $M << N$ (potentially overlapping) subgraphs. Input to the neural network is partitioned according to these subgraphs and the partitions are each processed separately by fully connected layers. A subsequent layer in the neural network then concatenates the outputs from these layers. This approach is computationally inexpensive and builds on previous work on locally connected neural networks. However, this approach lacks several features of classical image CNNs, including reusing a compact set of weights across the image or graph to simplify and increase learning speed.

The second proposed approach from [23] performs a convolution in the spectral domain. The eigenvectors of the graph Laplacian are used to perform a graph Fourier transform. Given a graph signal $\mathbf{X} \in \mathbb{R}^{N \times F}$ and the eigenvectors of the Laplacian $\mathbf{U} \in \mathbb{R}^{N \times N}$, the pointwise product between the transformed signal and a filter $\mathbf{\Theta}$ is computed and then transformed back to the spatial domain and passed through a nonlinear function h (e.g. a sigmoid function):

$$\mathbf{Y} = h\left(\mathbf{U}(\mathbf{\Theta} \odot \mathbf{U}^T\mathbf{X})\right). \tag{2.3}$$

Because the eigenvectors of the Laplacian are orthonormal, the inverse Fourier transform can be performed using the transpose of the eigenvector matrix. A subset of the eigenvectors corresponding to the smallest eigenvalues can be used to perform a band limited transform to filter out the high-frequency components of the original signal. The spectral convolution layer is complementary to the spatial convolution layer. The spectral convolution is a global transform with respect to the graph, can be implemented using a weight matrix whose size is independent of the graph, and is applicable to graphs of different shapes and sizes. The spatial convolution is local to neighborhoods around each node in the graph; it uses weight matrices proportional to the size of these neighborhoods; and it can only be implemented for a single graph at a time.

A downside of spectral convolution is the computational cost of calculating the eigenvectors of the graph Laplacian. Several papers have provided means of alleviating some of this cost and have formed links between the spectral and spatial approaches. Deferrard et al. [30] uses Chebychev polynomials to approximate the eigenvectors used for the spectral transform. These polynomials are less computationally expensive to compute than exact eigenvectors and have the additional benefit of enforcing local support similar to the locally connected convolutional layer. For a kth order polynomial approximation, support for the Chebyshev convolution is restricted to a k-neighborhood around each vertex. Kipf and Welling [49] further provide a first order approximation to the localized spectral filters. Their network, coined a graph convolution network (GCN), has layers of the form:

$$\mathbf{Y} = h\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta}\right). \tag{2.4}$$

Their convolution uses an augmented adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ equivalent to adding a self-loop to every node in the graph. The degree matrix is similarly augmented: $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$.

An alternative spatial construction of a graph convolution network, termed a diffusion convolution neural network (DCNN), was proposed in [9]. The architecture uses a diffusion convolution layer that has compact spatial support, but also maintains shareable weights for different graphs. The diffusion convolution layer computes the product of the input signal and successive powers of the graph random walk matrix. The output of the layer is the tensor of stacked matrices $[\mathbf{Y}_0\ \mathbf{Y}_1\ ...\ \mathbf{Y}_K]$, where each matrix $\mathbf{Y}_i$ is calculated as:

$$\mathbf{Y}_i = h(\mathbf{\Theta}_i \odot (\mathbf{A}\mathbf{D}^{-1})^i \mathbf{X}). \tag{2.5}$$

The weight matrix $\mathbf{\Theta} \in \mathbb{R}^{K \times F}$ is applied individually to each node in the graph.

CHAPTER 3

QUANTUM WALK NEURAL NETWORKS

3.1 Introduction

While classical neural network approaches for structured data have been well investigated, there is growing interest in extending neural network architectures beyond grid structured data in the form of images or ordered sequences [51] to the domain of graph-structured data [9, 23, 39, 49, 79, 92]. Following the success of quantum kernels on graph-structured data [11, 12, 13], a primary motivation of this work is to explore the application of quantum techniques and the potential advantages they might offer over classical algorithms. In this work, we propose a novel quantum walk based neural network architecture that can be applied to graph data.

3.2 Related Work

Gupta and Zia [42] and Altaivsky [5] among others proposed quantum versions of artificial neural networks. Our proposed quantum walk neural network is graph neural network architecture based on discrete quantum walks (see Biamonte et al. [16] and Dunjko et al. [33] for an overview of the larger emerging field of quantum machine learning). Various researchers have worked on quantum walks on graphs. Ambainis et al. [7] studied quantum variants of random walks on one-dimensional lattices. Farhi and Gutmann [36] reformulated interesting computational problems in terms of decision trees and devised quantum walk algorithms that could solve problem instances in polynomial time compared to classical random walk algorithms that require exponential time. Aharonov et al. [2] generalized quantum walks to arbitrary

graphs. Subsequently, Rohde et al. [71] studied the generalization of discrete time quantum walks to the case of an arbitrary number of walkers acting on arbitrary graph structures and their physical implementation in the context of linear optics. Quantum walks have recently become the focus of many graph-analytics studies because of their non-classical interference properties. Bai et al. [11, 12, 13] introduced novel graph kernels based on the evolution of quantum walks on graphs. They defined an overall similarity of two graphs in terms of the similarities between the evolution of quantum walks on the two graphs. Quantum kernel based techniques were shown to outperform classical kernel techniques in effectiveness and accuracy. In [72, 73], Rossi et al. studied the evolution of quantum walks on the union of two graphs to define the kernel between two graphs. These closely related works on quantum walks and the success of quantum kernel techniques motivated our approach in developing a quantum neural network architecture.

In recent years, new neural network techniques that operate on graph-structured data have become prominent. Gori et al. [39], followed by Scarselli et al. [79], propose recursive neural network architectures to deal with graph-structured data. Bruna et al. [23] studied the generalization of convolutional neural networks (CNNs) to graph signals through two approaches, one based on hierarchical clustering of the graph, and the second based on using the spectrum of the graph Laplacian as a Fourier transform. Defferrard et al. [30] proposed polynomial filters to reduce the learning complexity of spectral graph convolutional networks as well as to restrict their support from the entire graph to local neighborhoods.

Along with the spectral approaches described above, a number of spatial approaches have been proposed that utilize random walks to extract and learn information from the graph. For comparison, we detail several modern approaches. Atwood and Towsley [9] propose a spatial convolutional method, Diffusion Convolutional Neural Networks (DCNN), that performs random walks on the graph and combines in-

formation from spatially close neighbors. Given a graph $G = \{V, E\}$ and a feature matrix $\mathbf{X}$, DCNN uses powers of the random walk probability matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ to diffuse information across the graph. $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal degree matrix such that $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. The random walk matrix raised to the k^{th} power, $\mathbf{P}^k$, diffuses information from each node to every node exactly k hops away from it. The output $\mathbf{Y}$ of the DCNN is a weighted combination of the diffused features from across the graph, given by

$$\mathbf{Y} = h\left(\mathbf{W} \odot \mathbf{P}^{*}\mathbf{X}\right), \tag{3.1}$$

where $\mathbf{P}^*$ is the stacked tensor of powers of transition matrices from $\mathbf{P}^0$ to $\mathbf{P}^k$, the operator $\odot$ represents element-wise multiplication, $\mathbf{W}$ are the learned weights of the diffusion-convolutional layer, and h is an activation function (e.g. rectified linear unit).

A second neural network architecture by Kipf and Welling [49], Graph Convolutional Neural Networks (GCN), was proposed to tackle semi-supervised learning on graph-structured data through a CNN architecture that uses localized approximation of spectral graph convolutions. GCN simplified the original spectral-based frameworks of Bruna et al. [23] and Defferrard et al. [30] for improved scalability. The method uses the augmented adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and degree matrix $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ to diffuse the input with respect to the local neighborhood according to:

$$\mathbf{Y} = h\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}\right). \tag{3.2}$$

As in DCNN, $\mathbf{W}$ are learning weights and h is an activation function.

Many graph convolution layers are inspired by classical CNNs used in image recognition problems. However, other deep learning models have also inspired graph-based variants. Graph Attention Networks (GAT) [92] are inspired by the attention

mechanisms commonly applied in natural language processing for sequence-based tasks [10, 91]. GAT uses a graph attention layer that combines information from neighboring nodes through an attention mechanism. Unlike the prior approaches, this allows a nonuniform weighting of the features of each node's neighbors. The method uses attention coefficients

$$e_{ij} = att\left(\mathbf{W}\mathbf{X}_i, \mathbf{W}\mathbf{X}_j\right), \tag{3.3}$$

where $\mathbf{W}$ is a learned weight matrix that linearly transforms feature vectors, $\mathbf{X}_i$ and $\mathbf{X}_j$, of nodes v_i and v_j, and att is an attention function (e.g. inner product). The attention coefficients e_{ij} are normalized through the softmax function to obtain normalized coefficients

$$\alpha_{ij} = \frac{e_{ij}}{\sum_{v_k \in \mathcal{N}(v_i)} e_{ik}}, \tag{3.4}$$

where $\mathcal{N}(v_i)$ is the neighbor set of node v_i. The output of node i is given as

$$\mathbf{Y}_i = h\left(\sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij}\mathbf{W}\mathbf{X}_j\right), \tag{3.5}$$

where h is an activation function, such as $tanh$ or $reLU$.

3.3 Preliminaries

This section provieds a review of the notation and operations used to describe a quantum walk. A Hilbert space $\mathcal{H}$ is a generalization of euclidean space to any number of dimensions, finite or infinite. Hilbert spaces are vector spaces with defined inner products: $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Except where otherwise noted, $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{H}} = \mathbf{x}^H\mathbf{x}$ where $\mathbf{X}^H$ is the conjugate transpose of $\mathbf{X}$. An N-dimensional Hilbert space is spanned by N basis

vectors: $\{\hat{\mathbf{e}}_n,\ i \in 1, ..., N\}$ where $\mathbf{e}_i$ is a vector of zeros with a one in the i^{th} position. Vectors in $\mathcal{H}$ are formed from complex combinations of basis vectors:

$$\boldsymbol{\psi} = \sum_{i \in N} \alpha_i \hat{\mathbf{e}}_i \tag{3.6}$$

such that $\alpha \in \mathbb{C}$. A superposition is a vector $\boldsymbol{\psi} \in \mathcal{H}$ with unit length, i.e. $\sum_i \|\alpha_i\|^2 = 1$, that represents a combination of states of the system existing simultaneously. When we take a measurement of $\boldsymbol{\psi}$, the superposition collapses to one of the basis states.

A unitary operator $\mathbf{U}$ is an operator that acts on a Hilbert space preserving the inner product: $\mathbf{U} : \mathcal{H} \to \mathcal{H}$ such that $\langle \mathbf{U}\boldsymbol{\psi}, \mathbf{U}\boldsymbol{\phi} \rangle_{\mathcal{H}} = \langle \boldsymbol{\psi}, \boldsymbol{\phi} \rangle_{\mathcal{H}}$.

In the context of a quantum walk, the space of the system $\mathcal{H}_W$ is described by the tensor product of two Hilbert spaces, the position (vertex) space $\mathcal{H}_P$ and the coin (spin) space $\mathcal{H}_C$ such that $\mathcal{H}_W = \mathcal{H}_P \otimes \mathcal{H}_C$. The state of the system then is the product of the superposition of each of these spaces: $\boldsymbol{\Phi} = \boldsymbol{\psi}_p \otimes \boldsymbol{\psi}_c$, where $\boldsymbol{\psi}_p \in \mathcal{H}_P$ and $\boldsymbol{\psi}_c \in \mathcal{H}_C$.

3.4 Graph Quantum Walks

Motivated by classical random walks, quantum walks were introduced by Aharonov et al. in 1993 [3]. Unlike the stochastic evolution of a classical random walk, a quantum walk evolves according to a unitary process. The behavior of a quantum walk is fundamentally different from a classical walk. In a quantum walk, there is interference between trajectories of the walk. Two kinds of quantum walks have been introduced in the literature; namely, continuous time quantum walks [36, 74] and discrete time quantum walks [58]. As the name implies, a continous time quantum walk is a continuous process whereas a discrete time quantum walk evolves in discrete steps. Quantum walks have recently received much attention because they have been shown to be a universal model for quantum computation [27]. In addition, they have

numerous applications in quantum information science such as database search [81], graph isomorphism [69], network analysis and navigation, and quantum simulation.

Discrete time quantum walks were initially introduced on simple regular lattices [65] and then extended to general graphs [47]. Quantum walk neural networks (QWNN) use the formulation of discrete time quantum walks outlined in [6, 47]. Given an undirected graph $G = (V, E)$, we introduce a position Hilbert space $\mathcal{H}_P$ that captures the superposition over various positions (nodes), in the graph. We define $\mathcal{H}_P$ to be the span of the position basis vectors $\left\{\hat{\mathbf{e}}_v^{(p)},\ v \in V\right\}$. The position vector of a quantum walker is a linear combination of position state basis vectors,

$$\boldsymbol{\psi}_p = \sum_{v \in V} \alpha_v \hat{\mathbf{e}}_v^{(p)} \tag{3.7}$$

where $\{\alpha_v \in \mathbb{C},\ v \in V\}$ are coefficients whose sum has unit L_2-norm: $\sum_v \|\alpha_v\|^2 = 1$. The probability of the walker residing at vertex v is $\|\alpha_v\|^2$.

Similarly, we introduce a coin Hilbert space $\mathcal{H}_C$ that captures the superposition over various spin directions of the walker on each node of the graph. We define $\mathcal{H}_C$ to be the span of the coin basis vectors $\left\{\hat{\mathbf{e}}_i^{(c)},\ i \in 1, \ldots, d_{max}\right\}$, where i enumerates the edges incident on a vertex v and d_{max} is the maximum degree of the graph. We will use d instead of d_{max} for conciseness. The coin (spin) state of a quantum walker is a linear combination of coin state basis vectors,

$$\boldsymbol{\psi}_c = \sum_{i \in 1, \ldots, d} \beta_{v,i} \hat{\mathbf{e}}_i^{(c)} \tag{3.8}$$

where $\{\beta_{v,i},\ i \in 1, \ldots, d\}$ are coefficients whose sum has unit L_2-norm: $\sum_i |\beta_{v,i}|^2 = 1$. If a measurement is done on the coin state of the walker at vertex v, $|\beta_{v,i}|^2$ denotes the probability of finding the walker in coin state i. The Hilbert space of the quantum walk is the tensor product of the two aforementioned Hilbert spaces: $\mathcal{H}_W = \mathcal{H}_P \otimes \mathcal{H}_C$.

Two unitary operators, the coin and shift operators, govern the time-evolution of discrete time quantum walk over graph G. Let $\boldsymbol{\Phi}^{(t)} = \boldsymbol{\psi}_p^{(t)} \otimes \boldsymbol{\psi}_c^{(t)}$ in $\mathcal{H}_W$ denote the state of the walker at time t. At each time-step, the coin operator $\boldsymbol{C}$ transforms the coin state of the walker at each vertex,

$$\boldsymbol{\psi}_p^{(t)} \otimes \boldsymbol{\psi}_c^{(t+1)} = (\boldsymbol{I} \otimes \boldsymbol{C})(\boldsymbol{\psi}_p^{(t)} \otimes \boldsymbol{\psi}_c^{(t)}). \tag{3.9}$$

$\boldsymbol{I}$ denotes the identity operator. After transforming the coin (spin) states, the shift operator $\boldsymbol{S}$ swaps the states of two vertices connected by an edge (i.e. for an edge (u, v), if u is the i^{th} neighbor of v and v is the j^{th} neighbor of u, then the coefficient corresponding to the basis state $\hat{\mathbf{e}}_v^{(p)} \otimes \hat{\mathbf{e}}_i^{(c)}$ is swapped with the coefficient of the basis state $\hat{\mathbf{e}}_u^{(p)} \otimes \hat{\mathbf{e}}_j^{(c)}$). The shift operator operates on both coin and position Hilbert spaces,

$$\boldsymbol{\Phi}^{(t+1)} = \boldsymbol{\psi}_p^{(t+1)} \otimes \boldsymbol{\psi}_c^{(t+1)} = \boldsymbol{S}(\boldsymbol{\psi}_p^{(t)} \otimes \boldsymbol{\psi}_c^{(t+1)}). \tag{3.10}$$

In short, the unitary evolution of the quantum walk is governed by the operator $\boldsymbol{U} = \boldsymbol{S}\,(\boldsymbol{I} \otimes \boldsymbol{C})$. The state of the quantum walk evolves through successive applications of $\boldsymbol{U}$ over time.

The choice of coin operator and the initial superposition of the walker control how the diffusion process evolves over the graph and over time. Altering these operators provides QWNN additional degrees of freedom for controlling the flow of information over the graph that do not exist in classical random walks. Figure 3.1 demonstrates how the diffusion behavior of a classical random walk differs from a discrete time quantum walk. Ahmad et al. [4] showed that a discrete quantum walk on a line, using position-dependent coin operators can lead to quantitatively different diffusion behaviors with different choices of coins. Our work uses multiple non-interacting quantum walks acting on arbitrary graphs, as introduced in Rhode et al. [71], to learn patterns in graph data. Calculating separate quantum walks originating from

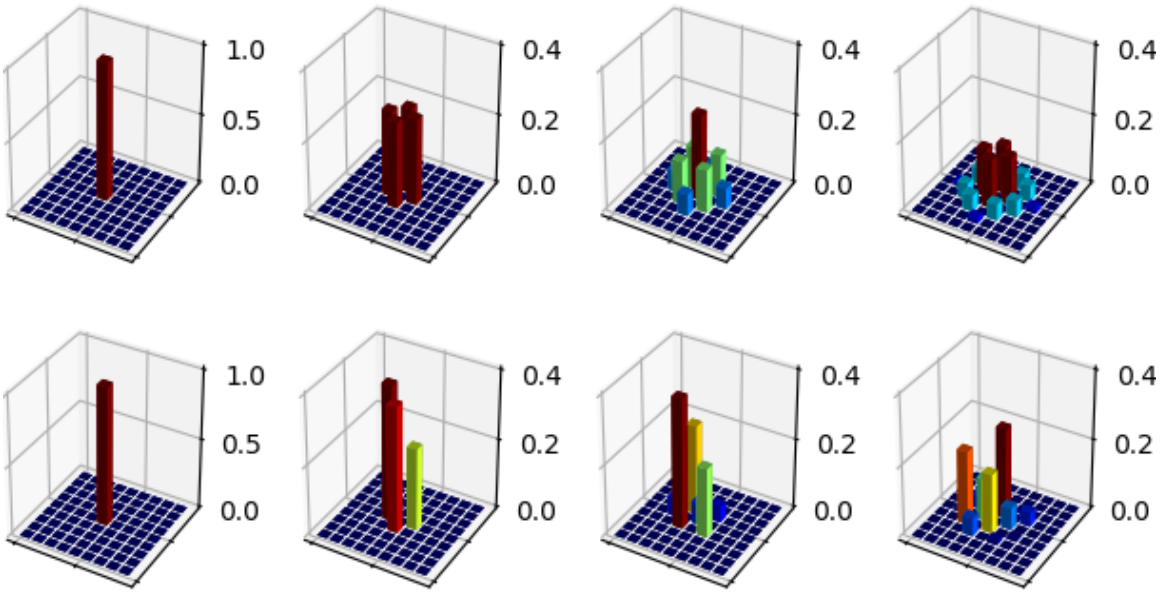


Figure 3.1: **Classical and quantum walk distributions.** The probability distribution of a classical random walk (Top) and a quantum random walk (Bottom) across the nodes of a lattice graph over four steps from left to right.

each node in the graph allows us to construct a diffusion matrix where each entry gives the relationship between the starting and ending nodes of a walk. This diffusion matrix can be used to determine the spread of node features across the graph.

3.4.1 Physical Implementation of Discrete Quantum Walks

Over the past few years, there have been several proposals for the physical implementation of a quantum walk. Quantum walks are unitary processes that are naturally implementable in a quantum system by manipulating their internal structure. The internal structure of the quantum system should be engineered to be able to encode the position and coin Hilbert spaces of the quantum walk. Quantum simulation based methods have been proposed using classical and quantum optics [105], nuclear magnetic resonance [76], ion traps [89], cavity QED [1], optical lattices [45], Bose Einstein condensate [60], and quantum dots [59] to implement the quantum walk.

Circuit implementations of quantum walks have also been proposed. While most of these implementations focus on graphs that have a high degree of symmetry [55] or are sparse [46, 26], there is some recent work on circuit implementations on non-degree regular dense graphs [56].

3.5 Quantum Walk Neural Networks

Many graph neural networks pass information between two nodes based on the distance between the nodes in the graph. This is true for both graph convolution networks and diffusion convolution networks. However, quantum walk neural networks are similar to graph attention networks in that the amount of information passed between two nodes also depends on the features of the nodes. In graph attention networks, this is achieved by calculating an attention coefficient for each of a node's neighbors. In quantum walk neural networks, the coin operator alters the spin states of the quantum walk to prioritize specific neighbors.

A QWNN, as shown in Figure 3.2, learns a quantum walk on a graph by means of backpropagating gradient updates to the coin operators used in the walk. QWNN use the walker distribution induced by the quantum random walk to form a diffusion operator which acts on the input feature matrix.

QWNN evolves a walk using a unitary coin matrix, $\mathbf{C}$ produced by a bank, to modify the spin state of the walker $\mathbf{\Phi}$ according to $\mathbf{\Phi}^{(t+1)} = \mathbf{\Phi}^{(t)}\mathbf{C}^{(t)}$ and then swaps states along the edges of the graph. Features $\mathbf{X}$ are then diffused across the graph by converting the states of the walker into a probability matrix, $\mathbf{P}$, and using it to diffuse the feature matrix: $\mathbf{Y} = \mathbf{PX}$. Learning is done by backpropagating the gradient of a loss function to the parameters that define the coin matrix.

QWNN produces the coin matrix by a node and time dependent function we call a bank. The bank forms the first of the three primary parts of a QWNN. It is followed by the walk and the diffusion. The bank produces the coin matrices used to direct

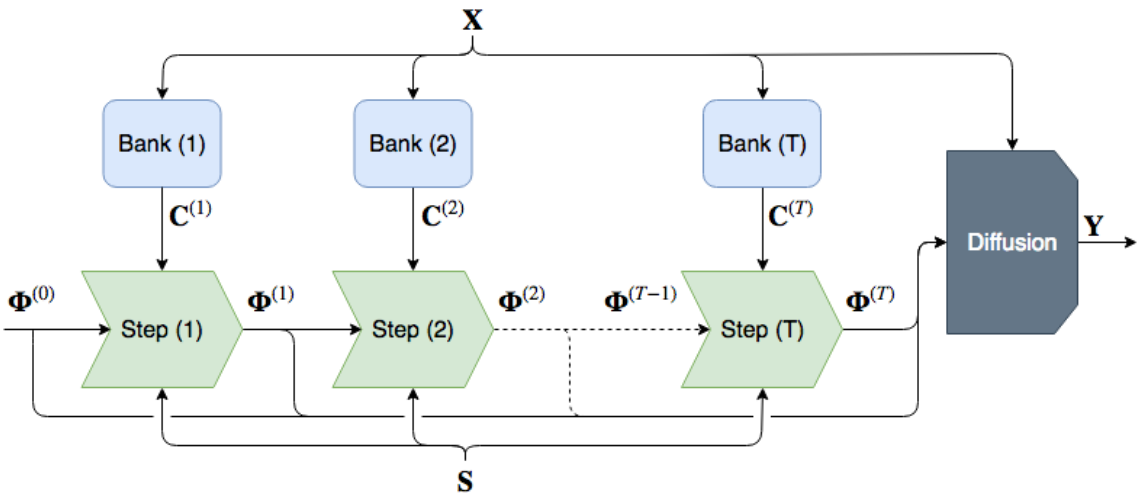


Figure 3.2: **Quantum walk neural network diagram.** The feature matrix X is used by the banks to produce the coin matrices $\mathbf{C}$ used in each step layer as well in the final diffusion process. The superposition $\mathbf{\Phi}$ evolves after each step of the walk. The diffusion layer diffuses X using each superposition $\{\mathbf{\Phi}^{(0)}, \mathbf{\Phi}^{(1)}, ...\mathbf{\Phi}^{(T)}\}$ and concatenates the results to produce the output $\mathbf{Y}$.

the quantum walk, the walk layers determine the evolution of the quantum walk at each step, and the diffusion layer uses these states to spread information throughout the graph.

3.5.1 Bank

The coin operator modifies the spin state of the walk and is thus the primary lever by which a quantum walk is controlled. The coin operator can vary spatially across nodes in the graph, temporally along steps of the walk, or remain constant in either or both dimensions. In QWNN, the bank produces these coins for the quantum walk layers.

When the learning environment is restricted to a single static graph, the bank stores the coin operators as individual coin matrices distributed across each node in the graph. However, for dynamic or multi-graph situations, the bank operates by learning a function that produces coin matrices from node features $f : X \to \mathbb{C}^{d \times d}$ where d is the maximum degree of the graph. In general, f can be any arbitrary

function that produces a matrix followed by a unitary projection to produce a coin $\mathbf{C}$. This projection step is expensive as it requires a singular value decomposition of a $d \times d$ matrix.

In recurrent neural networks (RNN), unitary matrices are employed to deal with exploding or vanishing gradients because backpropagating through a unitary matrix does not change the norm of the gradient. To avoid expensive unitary projections, several recursive neural network architectures use functions whose ranges are subsets of the set of unitary matrices. A common practice is to use combinations of low dimensional rotation matrices [8, 44].

In this work, we focus on elementary unitary matrices of the form $\mathbf{U} = \mathbf{I} - 2\mathbf{w}\mathbf{w}^T/(\mathbf{w}^T\mathbf{w})$ where $\mathbf{I}$ denotes the identity matrix and $\mathbf{w}$ is any real valued vector. These matrices can be computed efficiently in the forward pass of the neural network and their gradients can similarly be calculated efficiently during backpropagation. While this work focuses on using a single elementary matrix for each coin operator, any unitary matrix can be composed as the product of multiple elementary unitary matrices. The QWNN bank produces the coin matrix for node v_i according the following:

$$\mathbf{C}_i = \mathbf{I} - 2f(v_i)f(v_i)^T/(f(v_i)^T f(v_i)). \tag{3.11}$$

We propose two different functions $f(v_i)$.

The first function:

$$f_1(v_i) = \mathbf{W}^T vec\left(\mathbf{X}_{\mathcal{N}(v_i)}\right) + \mathbf{b}, \tag{3.12}$$

where $vec\left(\mathbf{X}_{\mathcal{N}(v_i)}\right)$ denotes the column vector of concatenated features of the neighbors of v_i, is a standard linear function parameterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{(Fd)\times d}$, with F the number of features, and a bias vector $\mathbf{b} \in \mathbb{R}^d$. This method has individual weights for each node but is not equivariant to the ordering of the nodes in the graph. This means that permuting the neighbors of v_i changes the output of the

function. We mitigate this effect by using a heuristic node ordering based on node centrality that we outline in Section 3.5.4.

The second function:

$$f_2(v_i) = \mathbf{X}_{\mathcal{N}(i)}\mathbf{W}\mathbf{X}_i^T, \tag{3.13}$$

with $\mathbf{W} \in \mathbb{R}^{F\times F}$, computes a similarity measure between the node v_i and each of its neighbors. This method is equivariant with respect to the node ordering of the graph (i.e. permuting the neighborhood of v_i equally permutes the values of $f_2(v_i)$). This in turn allows the entire neural network to be invariant to node ordering.

3.5.2 Walk

For a graph with N vertices, the QWNN processes N separate, non-interacting walks in parallel. One walk originates from each node in the graph. The walks share the same bank functions. A T-step walk produces a sequence of superpositions $\{\mathbf{\Phi}^{(0)}, \mathbf{\Phi}^{(1)}, ..., \mathbf{\Phi}^{(T)}\}$. For a graph with maximum degree d, the initial superposition tensor $\mathbf{\Phi}^{(0)} \in \mathbb{C}^{N\times N\times d}$ is initialized with equal spin along all incident edges to the node it begins at such that $(\mathbf{\Phi}_{ii\cdot}^{(0)})^H\mathbf{\Phi}_{ii\cdot}^{(0)} = 1$ and $\forall i{\neq}j : \mathbf{\Phi}_{ijk}^{(0)} = 0$. The value of $\mathbf{\Phi}_{ijk}^{(t)}$ denotes the amplitude of the i-th walker at node v_j with spin k after t steps of the walk.

A complete walk can be broken down into individual step layers. Each quantum step layer takes as input the current superposition tensor $\mathbf{\Phi}^{(t)}$, the set of coins operators $\mathbf{C}^{(t)}$ produced by the bank, and a shift tensor $\mathbf{S} \in \mathbb{Z}_2^{N\times d\times N\times d}$ that encodes the graph structure: $S_{mjni} = 1$ iff v_m is the the i^{th} neighbor of v_n and v_n is the j^{th} neighbor of v_m. The superposition evolves according to:

$$\mathbf{\Phi}^{(t+1)} = \mathbf{\Phi}^{(t)}\mathbf{C}^{(t)}{\cdot\cdot}\mathbf{S} \tag{3.14}$$

where$\mathbf{A}{\cdot\cdot}\mathbf{B}$ denotes the tensor double inner product of $\mathbf{A}$ and $\mathbf{B}$. Equivalently, for an edge (v_m, v_n), with v_m being the i^{th} neighbor of v_n and v_n being the j^{th} neighbor of

v_m:

$$\mathbf{\Phi}^{(t+1)}_{wmj} = \left(\mathbf{\Phi}^{(t)}_{n}\mathbf{C}^{(t)}_{n}\right)_{wi} \tag{3.15}$$

$$\mathbf{\Phi}^{(t+1)}_{wni} = \left(\mathbf{\Phi}^{(t)}_{m}\mathbf{C}^{(t)}_{m}\right)_{wj} \tag{3.16}$$

The output $\mathbf{\Phi}^{(t+1)}$ is fed into the next quantum step layer (if there is one) and finally is appended to the outputs of the other quantum step layers to form the input to the final diffusion layer.

3.5.3 Diffusion

The superpositions at each step of the walk are used to diffuse the signal $\mathbf{X}$ across the graph. Given a superposition $\mathbf{\Phi}$, the diffusion matrix is constructed by summing the squares of the spin states: $\boldsymbol{P} = \sum_k \mathbf{\Phi}_{\cdot\cdot k} \odot \mathbf{\Phi}_{\cdot\cdot k}$. The value $\boldsymbol{P}_{ij}$ gives the probability of the walker beginning at v_i and ending at v_j similar to a classical random walk matrix. Diffused features are computed as a function of $\mathbf{P}$ and $\mathbf{X}$ by $\mathbf{Y} = h(\mathbf{PX} + \mathbf{b})$ where h is an optional nonlinearity (e.g. reLU). The complete calculation for a forward pass for the QWNN is given in Algorithm 3.1.

Algorithm 3.1: QWNN Forward Pass

given : Initial Superpositions $\mathbf{\Phi}^{(0)}$, Shift $\mathbf{S}$
input : Features $\mathbf{X}$
output: Diffused Features $\mathbf{Y}$
for $t = 1$ *to* T **do**
  for *All nodes* v_i **do**
    $\mathbf{v}_i^{(t)} \leftarrow \mathbf{W}^T vec\left(\mathbf{X}_{\mathcal{N}(v_i)}\right) + \mathbf{b}$ or $\mathbf{v}_i^{(t)} \leftarrow \mathbf{X}_{\mathcal{N}(i)}\mathbf{W}\mathbf{X}_i^T$
    $\mathbf{C}_i^{(t)} \leftarrow \mathbf{I} - 2\mathbf{v}_i^{(t)}(\mathbf{v}_i^{(t)})^T/((\mathbf{v}_i^{(t)})^T\mathbf{v}_i^{(t)})$
    $\mathbf{\Phi}^{(t)}_{\cdot i\cdot} \leftarrow \mathbf{\Phi}^{(t-1)}_{\cdot i\cdot} \cdot \mathbf{C}^{(t)}_{i\cdot\cdot}$
  $\mathbf{\Phi}^{(t)} \leftarrow \mathbf{\Phi}^{(t)}\cdot\cdot\mathbf{S}$ $\left(\text{i.e., } \mathbf{\Phi}^{(t)}_{wuj} = \sum_v \sum_i \mathbf{\Phi}^{(t)}_{wvi}\mathbf{S}_{viuj}\right)$
  $\mathbf{P}^{(t)} \leftarrow \sum_{\mathrm{k}} \mathbf{\Phi}^{(t)}_{\cdot\cdot k} \odot \mathbf{\Phi}^{(t)}_{\cdot\cdot k}$
  $\mathbf{Y}^{(t)} \leftarrow h(\mathbf{P}^{(t)}\mathbf{X} + \mathbf{b}^{(t)})$
return : $\{\mathbf{Y}^{(0)}, \mathbf{Y}^{(1)}, ..., \mathbf{Y}^{(T)}\}$

3.5.4 Node and Neighborhood Ordering

Node ordering and by extension neighborhood ordering of each node can have an effect on a quantum walk if the coin is not equivariant to the ordering. Given a non-equivariant set of coins, if the order of nodes in the graph is permuted, the result of the walk may change.

This is the case for the first of the two bank functions (3.12). We address this issue using a centrality score. The betweenness centrality [22] of node v_i is calculated as:

$$g(v_i) = \sum_{j \neq i \neq k} \frac{\sigma_{jk}(v_i)}{\sigma_{jk}} \tag{3.17}$$

where σ_{jk} is the number of shortest paths from v_j to v_k and $\sigma_{jk}(v_i)$ is the number of shortest paths from v_j to v_k that pass through v_i. A larger betweenness centrality score implies a node is more central within the graph. Conversely, a leaf node connected to the rest of the graph by a single edge has a score of 0. Nodes in the graph are then ranked by their betweenness centrality and each neighborhood follows this ranking so that when ordering a node's neighbors, the most central nodes in the graph come first. In this setting, a walker moving along a higher ranked edge is moving towards a more central part of the graph compared to a walker moving along a lower ranked edge. While the walk is still variant to the node ordering of nodes with equal centrality score, this effect has been greatly diminished and the combination of this node ordering with (3.12) allows geometry of the graph to have a greater influence on the quantum walk than (3.13).

3.5.5 Relation to Attention Based Graph Neural Networks

Many spatial graph neural networks, such as GCN [49] and DCNN [9], pass information between nodes using static diffusion operators (e.g. the graph Laplacian). Attention based graph neural networks, like QWNN, focus attention on specific neighbors in the graph. This has made attention base graph neural networks a popular

model for graph classification problems [92, 101, 102] and graph pooling [52]. Attention based Transformer Networks [91] have also found implementations for graph-to-sequence tasks [24].

Using graph attention networks (GAT) [92] as our primary representative of attention based methods, we compare how information is moved throughout the graph in these models to QWNN. We observe the process at two scales: individual layers and deep networks. Graph attention networks use a self attention mechanism (3.3–3.5) to focus on information from specific neighbors. In GAT, the attention coefficients are computed by concatenating a linear function of each node's features $\mathbf{h}_i$ with a weight matrix $\mathbf{W}$ and then taking the product with an attention vector $\mathbf{a}$

$$e_{ij} = \mathbf{a}^T \begin{bmatrix} (\mathbf{W}\mathbf{h}_i)^T & (\mathbf{W}_e\mathbf{h}_j)^T \end{bmatrix}^T . \tag{3.18}$$

Using a softmax, e_{ij} is converted into a probability to determine the weight on the incoming features v_j.

The equivariant function used to form the coin matrices in QWNN (3.13) serves a parallel role to the attention coefficient in GAT. An alternative attention coefficient function to concatenation is dot product attention: $e_{ij} = \langle \mathbf{W}h_i, \mathbf{W}\mathbf{h}_j \rangle$. The function f_2 can be equivalently written in this form. Instead of directly relating to the weight of incoming information like e_{ij}, the value of $f_2(v_i)[j]$ influences the transformation of the spin states to and from spin j. Additionally, for QWNN, the direction of information flow is opposite that of GAT, the quantum walk calculates the weight on the information sent from a node, rather than received, to each neighbor of the node. The difference is that in GAT, the information flow from u to v is dependent on the neighbors of v, while in QWNN it is dependent on the neighbors of u.

The methods by which GAT and QWNN diffuse information over multiple layers are even more pronounced than the differences in a single layer. In GAT, layer l_k affects layer l_{k+1} through the changed features at each node. GAT uses output features

from the last attention layer in determining the attention scores of the current layer. In contrast, QWNN does not alter node features until the final diffusion layer. A QWNN could be constructed to mimic GAT in this respect by including a diffusion layer in between each step layer.

In QWNN layer l_k influences layer l_{k+1} through the current superposition of the walker. This gives each QWNN layer a memory of its last step. The incoming spin state of the walker at each quantum step layer provides the information of what node the walker came from because each spin state corresponds to a neighbor of a node. The coin operation interacts with each of these spin states in a way that the diffusion of information at step k in the walk is directly impacted by step $k-1$. The difference between GAT and QWNN in regards to what is passed between intermediary layers highlights possible applications where one network might be better suited than the other. Tasks which can benefit from heavier, nonlinear processing of a graph signal can take advantage of modifying the feature vectors of each node at each layer in GAT. On the other hand, tasks which require modeling more complex dynamics of passing messages along the graph may be better suited to QWNN.

3.6 Experiments

We demonstrate the effectiveness of QWNNs across three different types of tasks: node level regression, graph classification and graph regression. Our experiments focus on comparisons with three other graph neural network architectures: diffusion convolution neural networks (DCNN) [9], graph convolution networks (GCN) [49], and graph attention networks (GAT) [92].

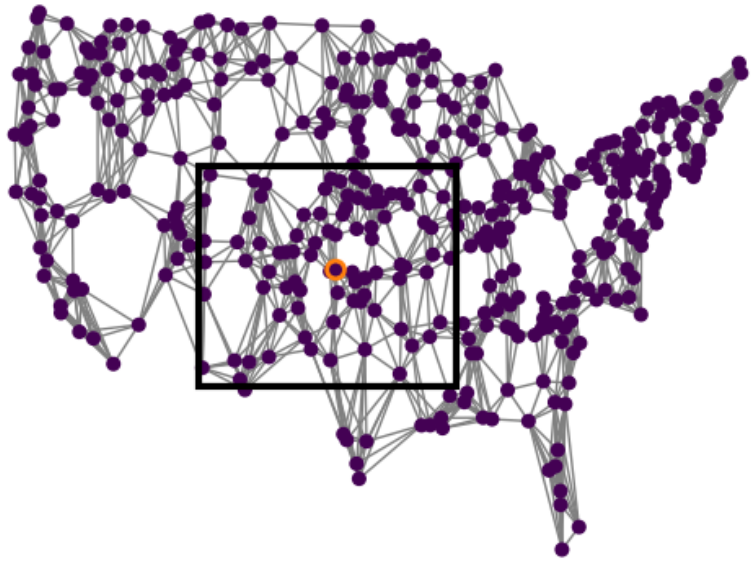
The implementation for QWNN is done in pyTorch. For compatibility and computational efficiency, the state of the quantum walk and the operators are limited to real values. For graph level experiments, we employ a set2vec layer [94] as an intermediary between the graph layers and standard neural network feed forward layers.

Set2vec has proved effective in other graph neural networks [37] as it is a permutation invariant function that converts a set of node features into a fixed length vector.

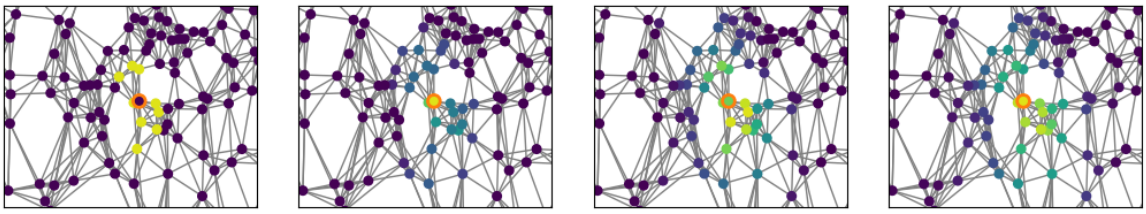
3.6.1 Node Regression

In the node regression task, daily temperatures are recorded across 409 locations in the United States during the year 2009 [103]. The goal of the task is to use a day's temperature reading to predict the next day's temperatures. A nearest neighbors graph (Figure 3.3a) is constructed using longitudes and latitudes of the recording locations by connecting each station to its closest neighbors. Adding edges from each station to its eight closest neighbors produces a connected graph. The QWNN is formed from a series of quantum step layers (indicated by walk length) followed by a diffusion layer. Since the neural network in this experiment only uses quantum walk layers, we relax the unitary constraint on the coin operators. While this can no longer be considered a quantum walk in the strictest sense, the relaxation is necessary to allow the temperature vector to grow or shrink to match increases or decreases in temperatures from day to day. For this experiment, we also compare the results with multiple DCNN walk lengths. For GCN and GAT, an effective walk length is constructed by stacking layers. Data is divided into thirds for training, validation, and testing. Learning is limited to 32 epochs.

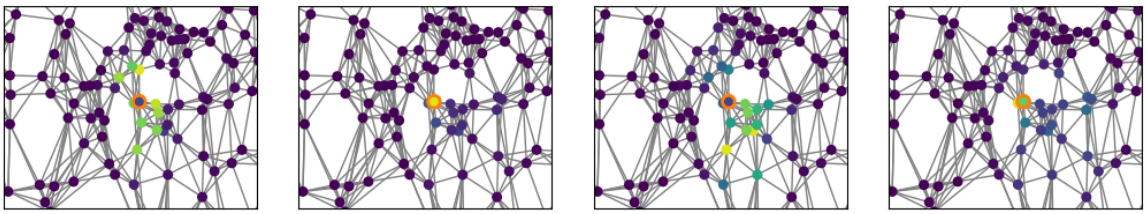
Table 3.1 gives the test results for the trained networks. The root-mean-square error (RMSE) and standard deviation (STD) are reported from five trials. We observe that quantum walk techniques yield lower errors compared to other graph neural network techniques. The two networks that control the amount of information flow between nodes, QWNN and GAT, appear to be able to take advantage of more distant relationships in the graph for learning, while DCNN and GCN perform best with more restrictive neighborhood sizes.



(a) Graph of Temperature Recording Locations



(b) Diffusion of a 4-step Classical Random Walk



(c) Diffusion of a 4-step Quantum Walk After Training

Figure 3.3: **Comparison of a classical walk and a learned quantum walk.** The classical and quantum random walks evolve from left to right over 4 steps. Both walks originate at the highlighted node. At each step, the brighter colored nodes correspond to a higher probability of the random walker at that node. A classical walk, as used in GCN and DCNN, diffuses uniformly to neighboring nodes. The learned quantum walk can direct the diffusion process to control the direction information travels. The third and fourth steps of the quantum walk show the information primarily directed southeast.

Table 3.1: Temperature Prediction Results (RMSE ± STD)

Walk Length	1	2	3	4	5
GCN	8.56 ± 0.02	8.14 ± 0.41	7.82 ± 0.13	8.55 ± 0.52	8.88 ± 0.73
DCNN	8.07 ± 0.21	7.40 ± 0.13	7.46 ± 0.06	7.44 ± 0.10	10.19 ± 0.18
GAT	7.84 ± 0.16	8.43 ± 0.42	8.47 ± 1.02	8.23 ± 0.69	7.93 ± 0.15
QWNN	6.11 ± 0.14	5.54 ± 0.16	5.38 ± 0.07	$\mathbf{5.28 \pm 0.08}$	5.65 ± 0.02

We use this experiment to provide a visualization for the learned quantum walk. Figure 3.3b and 3.3c shows the evolution of a classical random walk and the learned quantum random walk originating from the highlighted node, respectively. At each step, warmer color nodes correspond to nodes with larger superposition amplitudes. Initially, the quantum walk appears to diffuse outward in a symmetrical manner similar to a classical random walk, but in the third and fourth steps of the walk, the learned quantum walk focuses information flow towards the southeast direction. The ability to direct the walk in this way proves beneficial in the prediction task.

3.6.2 Graph Classification

The second type of graph problem we focus on is graph classification. We apply the graph neural networks to several common graph classification datasets: Enzymes [20], Mutag [29], and NCI1 [95]. Enzymes is a set of 600 molecules extracted from the Brenda database [80]. In the dataset, each graph represents a protein and each node represents a secondary structure element (SSE) within the protein structure, e.g. helices, sheets, and turns. Nodes contain a type label, and physical and chemical information as features. The task is to classify each enzyme into one of six classes. Mutag is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds that are classified into one of two categories based on whether they exhibit a mutagenic effect. NCI1 consists of 4110 graphs representing two balanced subsets of chemical compounds screened for activity against non-small cell lung cancer. For both the

Mutag and NCI1 datasets, each graph represents a molecule, with nodes representing atoms and edges representing bonds between atoms. Each node has an associated label that corresponds to its atomic number. Summary statistics for each dataset are given in Table 3.2. The experiments are run using 10-fold cross validation.

For the Enzyme and NCI1 experiment, the quantum walk neural networks are composed of a length 6 walk, followed by a set2vec layer, a hidden layer of size 64, and a final softmax layer. Because Mutag is a much smaller dataset, the walk length is reduced to 4 and the hidden layer size to 16. The reduced size helps alleviate some of the overfitting from such a small training set. We report the best results using the centrality based node ordering version of the network that uses the linear bank function: QWNN (cen) as well as the invariant QWNN using the equivariant bank function: QWNN (inv). We also report results from the three other graph networks. GCN, DCNN, and GAT are all used as an initial layer to a similar neural network followed by a set2vec layer, a hidden layer of size 64 (16 for Mutag) and a softmax output layer. DCNN uses a walk length of 2, while GCN and GAT use feature sizes of 32. Additionally, we compare with two graph kernel methods, Weisfeiler-Lehman (WL) kernels [82] and shortest path (SP) kernels [19], using the results given in [82].

Classification accuracies are reported in Table 3.3. The best neural network accuracies and the best overall accuracies are bolded. Quantum Walks are competitive with the other neural network approaches. QWNN demonstrates the best average accuracy on Mutag and Enzymes, but the other neural network approaches are within the margin of error. On the NCI1 experiment, QWNN shows a measurable improvement over the other neural networks. The WL kernels outperform all the neural network approaches on both Enzymes and NCI1.

Table 3.2: Graph Classification Datasets Summary

	Enzymes	Mutag	NCI1
Graphs	600	188	4110
Average Nodes	33	18	30
Max Nodes	126	28	111
Max Degree	9	4	4
Node Classes	3	7	37
Graph Classes	6	2	2

Table 3.3: Graph Classification Accuracy (Mean ± STD)

	Enzymes	Mutag	NCI1
GCN	0.31 ± 0.06	0.87 ± 0.10	0.69 ± 0.02
DCNN	0.27 ± 0.08	0.89 ± 0.10	0.69 ± 0.01
GAT	0.32 ± 0.04	0.89 ± 0.06	0.66 ± 0.03
WL	$\mathbf{0.59 \pm 0.01}$	0.84 ± 0.01	$\mathbf{0.85 \pm 0.00}$
SP	0.41 ± 0.02	0.87 ± 0.01	0.73 ± 0.00
QWNN (cen)	0.26 ± 0.03	$\mathbf{0.90 \pm 0.09}$	$\mathbf{0.76 \pm 0.01}$
QWNN (inv)	$\mathbf{0.33 \pm 0.04}$	0.88 ± 0.04	0.73 ± 0.02

3.6.3 Graph Regression

The graph regression task uses the QM7 dataset [17, 75], a collection of 7165 molecules each containing up to 23 atoms. The geometries of these molecules are stored in Coulomb matrix format defined as

$$\mathbf{C}_{ij} = \begin{cases} 0.5Z_i^{2.4} & i = j \\ \frac{Z_i Z_j}{|R_i - R_j|} & i \neq j \end{cases} \tag{3.19}$$

where Z_i, R_i are the charge of and position of the i-th atom in the molecule, respectively. The goal of the task is to predict the atomization energy of each molecule. Atomization energies of the molecules range from -440 to -2200 kcal/mol.

For this task, we form an approximation of the molecular graph from the Coulomb matrix by normalizing out the atomic charges and separating all atom-atom pairs into two sets based on their physical distances. One set contains the atom pairs with larger distances between them and the other the smaller distances. We create an adjacency matrix from all pairs of atoms in the smaller distance set. There is generally a significant gap between the distances of bonded and unbonded atoms in a molecule, but this approach leaves 19 disconnected graphs. For these molecules, edges are added between the least distant pairs of atoms until the graph becomes connected. We use the element of each atom, encoded as a one-hot vector, as the input features for each node.

The two variants of QWNN (node ordering invariant and centrality ordered) are constructed using a 4-step walk, followed by the set2vec layer, a hidden layer of size 10, and a final output layer. For the other graph neural networks, a single graph layer is used, followed by the same setup of a set2vec layer, a hidden layer of size 10, and the output layer. A DCNN of length 2 walk and GCN and GAT using 32 features were found to give the best results. Root-mean-square error (RMSE) and

Table 3.4: Atomization Energy Prediction Results

	RMSE	MAE
GCN	16.51 ± 0.38	12.39 ± 0.29
DCNN	11.90 ± 0.59	8.53 ± 0.42
GAT	18.75 ± 0.51	14.52 ± 1.12
QWNN (cen)	$\mathbf{9.70 \pm 0.77}$	$\mathbf{6.74 \pm 0.24}$
QWNN (inv)	10.91 ± 0.56	8.28 ± 0.47

mean absolute prediction error (MAE) are reported for each network in Table 3.4. QWNNs demonstrate a marked improvement over other methods in this task.

3.7 Alternative Formulations of QWNN

We experimented with alternative methods of parameterizing the unitary coin matrix $\mathbf{C}$ and the node ordering of the graph for the non-invairant case.

A method of parameterizing a unitary matrix proposed in [44] is to build a large unitary matrix from a sequence of rotation matrices. A $N \times N$ unitary matrix $\mathbf{U}_N$ is represented as a product of rotation matrices $\mathbf{R}_{k\ell}$ and a unitary diagonal matrix $\mathbf{D}$, such that $\mathbf{U}_N = \mathbf{D} \prod_{k=2}^{k=N} \prod_{\ell=1}^{\ell=i-1} \mathbf{R}_{k\ell}$, where $\mathbf{R}_{k\ell}$ is defined as the N-dimensional identity matrix with the elements R_{kk}, $R_{k\ell}$, $R_{\ell k}$ and $R_{\ell\ell}$ replaced as follows:

$$\begin{pmatrix} R_{kk} & R_{k\ell} \\ R_{\ell k} & R_{\ell\ell} \end{pmatrix} = \begin{pmatrix} e^{i\varphi_{k\ell}} \cos(\theta_{k\ell}) & -e^{i\varphi_{k\ell}} \sin(\theta_{k\ell}) \\ \sin(\theta_{k\ell}) & \cos(\theta_{k\ell}) \end{pmatrix} \tag{3.20}$$

with $\theta_{k\ell}$ and $\varphi_{k\ell}$ being parameters specific to $\mathbf{R}_{k\ell}$ which can be updated through backpropagation. Each of these rotation matrices acts as a unitary transformation on a 2-dimensional subspace of the N-dimensional Hilbert space and leaves the remaining $N - 2$ dimensions unchanged.

It is possible to impose local orderings on the nodes rather than a single global ordering. A local ordering orders the neighbors of each node, but does not imply that

if v_i precedes v_j in one node's set of neighbors that this is true for another node that also has v_i and v_j as neighbors. To differentiate this from a global node ordering, we refer to it as an edge ordering of the graph. We compute a similarity score between every adjacent pair of nodes in the graph, each node's neighbors are then ordered in descending order of their similarity score with itsself. We investigated using the random walk node similarity measure:

$$sim(v_i, v_j) = \mathbf{W}_i^k \left(\mathbf{W}_j^k\right)^T \tag{3.21}$$

where $\mathbf{W}^k$ is a classical random walk matrix raised to the k^{th} power.

When using the global centrality measure, the coin operator has some measure of control in determining whether the walk will move to a more central node in the graph or a less central node. With the random walk score edge ordering, the coin operator sends the walker either towards similar nodes (with respect to their position and roll in the graph) or to less similar nodes. Note that this similarity measure does not account for similarity or differences in the features of neighboring nodes. This ordering has a similar downside to the centrality ordering in that two neighbors of a node can have equal similarity scores. Different orderings of these neighbors can then lead to different outcomes of the quantum walk.

Table 3.5 compares the alternative coin matrix parameterization using both centrality node ordering and similarity edge ordering to the invariant QWNN formulation. Classification accuracy is reported for the Enzymes, Mutag, and NCI1 datasets and the RMSE is given for QM7. The results show that the elementary unitary matrix formulation of the coin operator generally outperforms the product of rotation matrices. Only on Enzymes do the alternative QWNN formulations have a measurable edge. The results continue to demonstrate a heuristic ordering of the graph can have substantive effects on the accuracy of QWNN, often improving results over the

Table 3.5: Comparison of QWNN Formulations

	Enzymes	Mutag	NCI1	QM7
QWNN (cen)	0.26 ± 0.03	$\mathbf{0.90 \pm 0.09}$	$\mathbf{0.76 \pm 0.01}$	$\mathbf{9.70 \pm 0.77}$
QWNN (inv)	$\mathbf{0.33 \pm 0.04}$	0.88 ± 0.04	0.73 ± 0.02	10.91 ± 0.56
altQWNN (cent)	$\mathbf{0.32 \pm 0.0}3$	0.85 ± 0.06	0.68 ± 0.02	12.52 ± 0.91
altQWNN (sim)	$\mathbf{0.32 \pm 0.0}3$	$\mathbf{0.92 \pm 0.03}$	0.71 ± 0.01	16.01 ± 0.96

invariant formulation. This can likely be attributed to increasing the influence of the graph structure on the diffusion process of the quantum walk.

3.8 Limitations

Storing the superposition of a single walker requires $O(Nd)$ space, where N is the number of nodes in the graph, and d is the maximum degree of the graph. To calculate a complete diffusion matrix requires that a separate walker begin at every node, increasing the space requirement to $O(N^2d)$ which starts to become intractable for very large graphs, especially when doing learning on a graphics processing unit (GPU). Some of this cost can be alleviated using sparse tensors. At time t=0 the superpositions are localized to single nodes so only $O(Nd)$ space is used by nonzero amplitudes. At time t=1 the first step increases this to $O(Nd^2)$ as each neighboring node becomes nonzero. Given a function $s(G, t)$ which determines the number of nodes in a graph reachable after a t-length random walk, the space complexity for a t-length walk is $O(Nds(G, t))$.

The majority of graph neural networks are invariant to the ordering of the nodes in the graph. This is true for GCN, DCNN, and GAT. We provide one formulation for a QWNN that is also invariant, however the second formulation is not. Although we have greatly reduced the effect, node ordering can still affect the walk produced in this variant of QWNN and thus the overall output of the network. This can occur when two otherwise distinguishable nodes have the same betweenness centrality.

3.9 Conclusion

Quantum walk neural networks outperformed other graph neural network approaches across a range of different experiment types. QWNN is specifically suited for applications involving small graph regression problems. The ability of quantum walks to adapt spatially over the graph provide QWNN with the ability to fine-tune the output in regression tasks further than the other comparison networks. While outperformed by WL kernels in two of the graph classification, QWNN offers increased flexibility in the type of problems it can be applied to compared to the kernel method which is restricted to classification tasks.

CHAPTER 4

ASYMMETRIC NODE SIMILARITY EMBEDDING FOR DIRECTED GRAPHS

4.1 Introduction

The representational power of graphs–the ability to model relationships between objects–make them an essential tool in data visualization and processing. The common matrix forms of graphs, namely adjacency ($\mathbf{A}$) and Laplacian ($\mathbf{L}$) matrices, represent nodes in the graph by their connections to other nodes. Each row of the matrix forms a sparse vector representation in $\mathbb{R}^{|V|}$ of a node. While this is an accurate depiction of the local geometry around a node, it is impossible to determine, for example, the distance of two nodes, which do not share an edge or neighbor, from only their vector representations. In data processing tasks where the rows of the adjacency or Laplacian matrices are insufficient, an embedding of the nodes into a dense vector format can provide a richer representation of the graph. Node embedding techniques are also used for nonlinear dimensionality reduction of data matrices by forming nearest neighbor graphs of the data points [15].

Many node embedding schemes utilize the spectra of graph matrix (e.g. Laplacian or shortest path matrix) to form vector representations of nodes [15, 28, 88]. These approaches, while effective, require an eigen decomposition of the matrices and do not scale well as the number of nodes in the graph increases. Stochastic sampling methods have been proposed to improve scaling issues. Recent embedding methods sample random walks from the graph and use a stochastic gradient descent process to learn vector representations for the nodes [68, 41]. These methods have been shown to be efficient on graphs scaling up to millions of nodes.

Implementing graph embedding methods designed for undirected graphs (such as those above) on a directed graph often require making sacrifices to the graph structure because these methods rely on symmetric relationships between nodes. This leads to the unrecoverable loss of the asymmetric relationships between nodes, such as the direction of an edge between two nodes. Node embeddings have been proposed for digraphs that rely on dual, independent embeddings for each node [106] to preserve graph asymmetries. The two embeddings, the *source* embedding and the *target* embedding, correspond to the a node's role as either the source or the target node of a directed edge, respectively. A node's role as a source and target are not independent in a graph. A graph with high reciprocity (a large portion of edges between nodes in the graph are bidirectional) obviously should have similar source and target embeddings. The set of edges coming into a node can also often provide insight into the set of edges leaving a node. Because of this, our goal is an embedding method that recognizes the codependency between source and target roles of a node.

In this chapter, we propose a directed random walk based approach to learning node embeddings that requires only a single embedding for each node: asymmetric node similarity embedding (ANSE). ANSE simultaneously learns the node embedding vectors and the parameters of an asymmetric similarity function $K_{\mathbf{A}}(v_i, v_j)$ on the embeddings. Because $K_{\mathbf{A}}(v_i, v_j)$ and $K_{\mathbf{A}}(v_j, v_i)$ can have different values, ANSE preserves the direction of edges so that they can be recovered from the embeddings. Crucially, ANSE embeddings can also represent an edge between two nodes that exist in both directions or neither. Like dual embedding digraph approaches, ANSE preserves asymmetric relationships in the graph. However, ANSE's single embedding scheme ties source and target information about each node together, improving the embedding of nodes when information about one or the other is lacking in the observable graph (e.g. a node without incoming edges). Like other random walk approaches, ANSE scales linearly with the size of the graph. Additionally, we pro-

vide a modified form of ANSE, ANSE-H, that embeds the nodes of the graph onto a unit hypersphere. Spherical embeddings have been shown to be effective in similar domains to preserve directional similarities [61, 54]. We provide an illustration of a 2-dimensional embedding of a small lattice graph and demonstrate the effectiveness of our approach in several link prediction tasks on real world directed networks.

4.2 Problem Definition

An undirected graph $G = (V, E)$ is a set of vertices (nodes), $V = \{v_0, v_1, ..., v_{N-1}\}$, and a set of edges, $E = \{(v_i, v_j) \ : \ v_i, v_j \in V\}$, representing objects and relationships between pairs of objects, respectively. A directed graph (digraph) imposes an ordering on the vertices of each edge such that (v_i, v_j) denotes an edge pointing from node v_i, the source, to node v_j, the target. A random walk on a digraph is a sequence of nodes $\left(v^{(0)}, v^{(1)}, ..., v^{(K)}\right)$ ordered such that $v^{(k+1)}$ is selected at random from the neighbors incident to the outgoing edges of $v^{(k)}$.

A node embedding is a function on a graph that maps each node in the graph to a d-dimensional vector $f_G : V \to \mathbb{R}^d$. The embedding of node v_i is expressed as $\boldsymbol{\phi}_i$. The embedding matrix $\boldsymbol{\Phi} \in \mathbb{R}^{|V| \times d}$ is built up from rows of these embeddings, i.e. $\boldsymbol{\Phi}_i = \boldsymbol{\phi}_i^T = f_G(v_i)$. Our goal is to preserve the asymmetric similarity between nodes in the embedded space. We use the probability of a random walk beginning at node v_i visiting node v_j within k steps as our similarity measure $sim(v_i, v_j)$ between two nodes. This probability is an effective similarity measure for embedding because it is greatest when both the path between two nodes is short and when there exist many such paths. The maximum similarity between nodes, $sim(v_i, v_j) = 1$, occurs when all k-length paths originating from v_i travel through v_j and the minimum value, $sim(v_i, v_j) = 0$, when v_i and v_j are more than k edges apart in the graph. The function is asymmetric, $sim(v_i, v_j) \neq sim(v_j, v_i)$ for most complex graphs, directed or otherwise. This can be demonstrated by setting $k = 1$ and looking at the similarity

of two adjacent nodes with different degrees. Additionally, for a directed graph: $sim(v_i, v_j) > 0 \implies sim(v_j, v_i) > 0$. ANSE learns an embedding matrix $\mathbf{\Phi}$ and a similarity measure $K_{\mathbf{A}} : \mathbf{\Phi} \times \mathbf{\Phi} \to \mathbb{R}$ such that $K_{\mathbf{A}}(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) \approx sim(v_i, v_j)$.

4.3 Background and Related Work

There are a limited number of machine learning methods that operate directly on graphs; graph convolutional neural networks use the structure of the graph to operate on signals that lie on a graph rather than operate on the graph itself. *Graph embeddings* are methods that translate a graph or portions of a graph into a low dimensional space often with the intent to form embeddings that are better suited as inputs to machine learning algorithms.

A *node embedding* is a function on a graph that maps each node in the graph to a d-dimensional vector $f_G : V \to \mathbb{R}^d$ with the goal of preserving the relationships between vertices. These relationships are usually defined by neighborhoods around each node. An ideal node embedding produces vector representations for each node in this neighborhood that are similar to the embedding of the central node. Skip-gram methods are popular because of their effectiveness at doing just that.

The skip-gram model proposed for word embedding in natural language processing [62, 63] extracts sentences from a document corpus and embeds each word in a low-dimensional space to maximize the probability of predicting surrounding words. DeepWalk [68] and subsequent algorithms such as Node2Vec [41] adapt the skip-gram model to node embedding. These methods generate random walks on an undirected graph in lieu of extracting sentences from a corpus. Randomly sampling sentences or random walks produces similar frequency distributions of the appearances of words and nodes.

Algorithm 4.1 outlines a generic skip-gram style node embedding method. Lines 6 and 9 are the two key steps in the algorithm and consequently are where many node

Algorithm 4.1: Skip-Gram Model of Node Embedding

1 **input** : Graph: $G = (V, E)$
Window size: w
Embedding dimesnion: d
Walks per vertex: γ
Walk length: k
2 **output** : Node Embedding $\mathbf{\Phi}$
3 **initialize:** $\mathbf{\Phi} \in \mathbb{R}^{|V| \times d}$
4 **for** $i = 1$ *to* γ **do**
5 **for** *All nodes* $v_i \in V$ **do**
6 $W_{v_i} =$ `RandomWalk` (G, v_i, k)
7 **for** $v_j \in W_{v_i}$ **do**
8 **for** $v_k \in W_{v_i}[j - w : j + w]$ **do**
9 $J(\mathbf{\Phi}) = -\log P(v_k|\boldsymbol{\phi}_j)$
10 $\mathbf{\Phi} = \mathbf{\Phi} - \alpha * \frac{\partial J}{\partial \mathbf{\Phi}}$
11 **return** : $\mathbf{\Phi}$

embedding methods differ from one another. Function $RandomWalk(G, v_i, k)$ on line 6 collects a sequence of k nodes on the graph from a random walk originating at vertex v_i. DeepWalk samples classical random walks, while Node2Vec uses weighted random walks that can be tuned to either encourage the walker to remain close to the initial node or to explore further away in the graph. In both methods, after a walk is performed, each pair on nodes in the walk within w steps of each other form positive learning samples (i.e. $P(v_k|\boldsymbol{\phi}_j)$ is large).

The probability in line 9 of Alg. 4.1 is calculated using a softmax:

$$P(v_k|\boldsymbol{\phi}_j) = \frac{\exp\langle \boldsymbol{\phi}_j, \boldsymbol{\phi}_k \rangle}{\sum_l \exp\langle \boldsymbol{\phi}_j, \boldsymbol{\phi}_l \rangle}. \tag{4.1}$$

In practice, this calculation is prohibitively expensive because it requires an inner product between the embedding of the source node and the embeddings of each other node in the graph. Many algorithms employ alternative, less computationally expensive, methods of approximating the conditional probabilities. DeepWalk utilizes a hierarchical softmax [64] for computing probabilities, while Node2Vec uses a negative

sampling approach [63]. Negative sampling obtains N nodes from a noise distribution $P(v)$ that are outside the walk range of the current node and thus should have very dissimilar embeddings to it. The distribution $P(v)$ in node2vec is proportional to the frequency distribution of nodes in the random walks raised to the 3/4 power. Negative sampling methods use the following approximation of the log softmax:

$$\log P(v_k|\boldsymbol{\phi}_j) \approx \log\left(\sigma\left\langle \boldsymbol{\phi}_j, \boldsymbol{\phi}_k \right\rangle\right) + \sum_{n=1}^{N} \mathbb{E}_{v_n \sim P(v)} \log(\sigma\left\langle -\boldsymbol{\phi}_j, \boldsymbol{\phi}_n \right\rangle) \tag{4.2}$$

where σ is the sigmoid function: $(1 + exp(x))^{-1}$. The first term in (4.2) seeks to maximize the probability of positive node pair samples $(\boldsymbol{\phi}_j, \boldsymbol{\phi}_k)$, while the second term seeks to drive down the probabilities of the negative samples $(\boldsymbol{\phi}_j, \boldsymbol{\phi}_n)$ in order to maximize the overall log probability $\log P(v_k|\boldsymbol{\phi}_j)$. The hierarchical softmax in DeepWalk reduces the complexity of each softmax calculation from $O(|V|)$ to $O(\log |V|)$ by using a binary tree to calculate the conditional probabilities. The negative sampling approach of Node2Vec reduces the complexity further to $O(N)$ where N is the number of negative samples.

To translate node embedding methods from undirected graphs to digraphs, several digraph node embedding methods embed the nodes twice, once into a source space and a second time into a target space [48, 67, 106]. APP [106] is a skip-gram model that learns dual embeddings for each node in the graph. Node pair samples are collected from the endpoints of (directed) random walks. These pairs are ordered so that the similarity score between nodes is calculated using the source embedding of the first node and the target embedding of the second.

4.4 Method

Many symmetric properties of undirected graphs are asymmetric on digraphs. For example the graph geodesic, the length of the shortest path between nodes, is not

symmetric for a directed graph. The existence of a path from v_i to v_j does not even imply that a path exists for v_j to v_i. The similarity measure used in an embedding scheme for digraphs must reflect these differences if they are to be preserved in the embeddings. This issue is compounded by the fact that some nodes in a directed graph may have edges between them in both directions. This fact rules out skew-symmetric functions as possible similarity measures.

To learn an embedding that can retain the asymmetric relationships between nodes, we replace the standard (symmetric) inner product used in the softmax (4.1) and the negative sampling approximation (4.2) with an asymmetric bilinear product parameterized by a square matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$:

$$K_{\mathbf{A}}(v_i, v_j) = \langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle_{\mathbf{A}} = \boldsymbol{\phi}_i^T \mathbf{A} \boldsymbol{\phi}_j. \tag{4.3}$$

In general, when $\mathbf{A}$ is asymmetric, $K_{\mathbf{A}}(v_i, v_j) \neq K_{\mathbf{A}}(v_j, v_i)$. Matrix $\mathbf{A}$ can be learned in tandem with the embedding matrix $\boldsymbol{\Phi}$ through standard backpropagation methods such as stochastic gradient descent. From a geometric perspective, $\mathbf{A}$ can be viewed as defining the direction of the most similar embedding vectors at any point (See Figure 4.1b for an example).

4.4.1 Sampling Random Walks

Digraphs exhibit two properties not exhibited by undirected graphs that must be taken into account when sampling random walks. First, the relative order of node pairs sampled from a walk must be maintained. Second, a digraph, unlike an undirected graph, has two levels of connectivity. A weakly connected directed graph has an undirected path from any to node to every other node. A strongly connected digraph has a directed path from any node to every other node. Strongly connected digraphs are much rarer than weakly connected digraphs or connected undirected graphs and so we often can't assume strong connectivity. Walks on digraphs that

are weakly connected, but not strongly connected, may dead end. In a connected undirected graph or a strongly connected digraph, a random walk can continue indefinitely because each node in the graph must have an outgoing edge. This is not the case for a weakly connected digraph in which nodes may not have any outgoing edges. In such events, the walk must either jump to a non-neighbor node or be forced to terminate.

A consequence of these two properties is that nodes without outgoing edges will never be sampled first in a pair. In these cases, there is an increased importance in reaching the node from random walks beginning at other nodes so that positive samples containing the node are still collected. In scenarios in which a node has an arbitrarily small likelihood of being reached in a random walk, e.g. the only incoming edge to a node comes from another node with a high out-degree, the node is unlikely to appear as the second node in any positive sample pair. To address these issues, we introduce a reverse random walk sampling method in addition to regular random walk sampling. Reverse walks are sampled from a dual graph, $G^* = (V, (v_j, v_i) : (v_i, v_j) \in E)$, where all edges are reversed (bidirectional edges remain so). The sequence of nodes in the walk is then reversed again to provide a random walk on G whose transition probabilities are proportional to the in-degrees of nodes rather than their out-degrees. This guarantees that there are sample pairs containing each node with at least one incoming edge as the second node.

Negative sampling is used to approximate the softmax function. Nodes are randomly sampled from the graph and used as negative samples as in (4.2). Combining the positive and negative sampling methods with the asymmetric similarity function produces our method given in Algorithm 4.2. The overall loss is split into 4 parts: J_1, J_2, J_3, and J_4. The positive samples produce losses J_1 and J_3 and the negative sampling produces losses J_2 and J_4. The focus node is the preceding node in losses J_1 and J_2 and the succeding node in losses J_3 and J_4.

Algorithm 4.2: Asymmetric Node Similarity Embedding

input : graph: G(V,E)
embedding dimesnion: d
walks per vertex: γ
walk length: k

output: embeddings matrix: $\mathbf{\Phi}$
similarity matrix: $\mathbf{A}$

Initialize $\mathbf{\Phi} \in \mathbb{R}^{|V| \times d}$, $\mathbf{A} \in \mathbb{R}^{d \times d}$
for *All nodes* $v_i \in V$ **do**
  for $w = 1$ *to* γ **do**
    $W_{v_i} = RandomWalk(G, v_i, k)$
    $W_{v_i}^{rev} = ReverseRandomWalk(G^*, v_i, k)$
    $J_1(\mathbf{\Phi}) = -\sum_{v_j \in W_{v_i}} \log(\sigma \left\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \right\rangle_{\mathbf{A}})$
    $J_2(\mathbf{\Phi}) = -\sum_{n=1}^{N} \mathbb{E}_{v_n \sim P(v)} \log(\sigma \left\langle -\boldsymbol{\phi}_i, \boldsymbol{\phi}_n \right\rangle_{\mathbf{A}})$
    $J_3(\mathbf{\Phi}) = -\sum_{v_j \in W_{v_i}^{rev}} \log(\sigma \left\langle \boldsymbol{\phi}_j, \boldsymbol{\phi}_i \right\rangle_{\mathbf{A}})$
    $J_4(\mathbf{\Phi}) = -\sum_{n=1}^{N} \mathbb{E}_{v_n \sim P(v)} \log(\sigma \left\langle -\boldsymbol{\phi}_n, \boldsymbol{\phi}_i \right\rangle_{\mathbf{A}})$
    $J(\mathbf{\Phi}) = \sum_{n=1}^{4} J_n(\mathbf{\Phi})$
    $\mathbf{\Phi} = \mathbf{\Phi} - \alpha * \frac{\partial J}{\partial \mathbf{\Phi}}$

4.4.2 Hypersphere Embedding

ANSE can be adapted to embed nodes of a graph onto the unit hypersphere. This has been shown to be effective for word [61] and face [54] embeddings by removing the length of the embedding vector as a factor when evaluating the embeddings. To embed nodes in the unit hypersphere, the embedding vectors are constrained to have unit length: $||\boldsymbol{\phi}_i||_2^2 = 1$. This is accomplished by renormalizing the length of the vector following each backpropagation update.

The bilinear matrix $\mathbf{A}$ should also be constrained such that $\forall v_i, v_j \in V : -1 \leq k_{\mathbf{A}}(v_i, v_j) \leq 1$. This constraint allows the similarity function to match the range of a standard inner product between two points on the unit hypersphere. If $\mathbf{A}$ is a unitary matrix, the product $\boldsymbol{\phi}_i^T \mathbf{A}$ will also have unit length and thus $-1 \leq \boldsymbol{\phi}_i^T \mathbf{A} \boldsymbol{\phi}_j \leq 1$, guaranteeing the constraint will hold.

We can project $\mathbf{A}$ onto the set of unitary matrices whenever it diverges during learning, similar to renormalizing the embedding vectors. Unfortunately, this projec-

tion is computationally costly, requiring a singular value decomposition of the matrix. Alternatively, we compose $\mathbf{A}$ as the product of a set of elementary reflector matrices of the form $\mathbf{A}^{(m)} = \mathbf{I} - 2 * \frac{\mathbf{v}_m \mathbf{v}_m^T}{\mathbf{v}_m^T \mathbf{v}_m}$, where $\mathbf{v}_m$ is any vector and $\mathbf{I}$ is the identity matrix. Any unitary matrix can be decomposed into a product of elementary reflectors. We use this decomposition to efficiently construct a unitary matrix $\mathbf{A} = \prod_{m=1}^{M} \mathbf{A}^{(m)}$, where $M \in \{1, ..., d\}$. Smaller values of M restrict the space of possible unitary matrices but also reduces both the computational cost to calculate $\mathbf{A}$ and the number of parameters for the model to learn. The vectors $\mathbf{v}_m$ are learned by backpropagating the loss through $\mathbf{A}$.

4.4.3 Comparison to Other Embedding Spaces

ANSE can be viewed as learning a dual embedding, similar to [106], where the source embedding for node v_i is $\boldsymbol{\phi}_i$ and the target embedding is $\mathbf{A}\boldsymbol{\phi}_i$. Compared to learning two embeddings, ANSE reduces the number of embedding parameters from $2|V|d$ to $|V|d + d^2$, as typically $d \ll |V|$. Additionally, tying the source and target embeddings together in our approach overcomes a potential issue in [106], in which the source or target embeddings of a node in a digraph may have no positive samples if the node has no outgoing or incoming edges, respectively.

A related asymmetric product is used in [87] for text retrieval. The asymmetric Hermitian inner product of two complex vector embeddings $\langle \mathbf{a}_i, \mathbf{b}_j \rangle = \mathbf{a}_i^H \mathbf{b}_j$ is used to score co-attention between complex valued word vectors: $\mathbf{a}_i, \mathbf{b}_j \in \mathbb{C}^d$, where $\mathbf{a}^H$ is the conjugate transpose of $\mathbf{a}$. The product $s_{ij} = Re(a_i^H \mathbf{M} b_j)$, where $\mathbf{M} \in \mathbb{C}^{d \times d}$ and $Re(x)$ indicates the real portion of the complex number x, is also studied. Unlike our work, however, the matrix $\mathbf{M}$ is tuned as a hyperparameter and fixed rather than treated as a learnable parameter.

Complex vector embeddings have also been studied for link prediction. In [90], a given observation matrix $\mathbf{X} \in \mathbf{R}^{n \times n}$ is decomposed as $\mathbf{X}_{so} = Re\left(\mathbf{E}_s^H \mathbf{W} \mathbf{E}_o\right)$, where

$\mathbf{E} \in \mathbb{C}^{n \times d}$ is a matrix of eigenvectors and $\mathbf{W} \in \mathbb{C}^{d \times d}$ is a diagonal matrix of eigenvalues. Like dual embeddings, a d-dimensional complex embedding requires $2d$ parameters per node (the real and imaginary part). But, like our method, the two roles of each object (e.g. source and target) are tied together because the scoring functions use both the real and imaginary parts of each of the complex embeddings.

Gaussian embeddings were proposed for word embedding as a means of better capturing asymmetries [93] and have also been applied to node embedding in attributed graphs [18]. Nodes are mapped to a mean $\mu \in \mathbb{R}^d$ and covariance $\Sigma \in \mathbb{R}^{d \times d}$. Gaussian distribution embeddings lead to a natural asymmetric dissimilarity measure via Kullback–Leibler divergence:

$$D_{KL}(\mathcal{N}_j||\mathcal{N}_i) = \frac{1}{2}\left[tr(\Sigma_i^{-1}\Sigma_j) + (\mu_i - \mu_j)^T\Sigma_i^{-1}(\mu_i - \mu_j) - d - log\frac{det(\Sigma_j)}{det(\Sigma_i)}\right]. \quad (4.4)$$

During implementation, the covariance matrix for each node is restricted to a diagonal matrix to make updates efficient. This means $\mathcal{N}$ is a set of d independent univariate Gaussians. Letting $\sigma_{ij} = \Sigma_i[j, j]$, the KL-divergence reduces to the following:

$$D_{KL}(\mathcal{N}_j||\mathcal{N}_i) = \frac{1}{2}\sum_k \sigma_{jk}/\sigma_{ik} + (\mu_{ik} - \mu_{jk})^2/\sigma_{ik} - \log\sigma_{jk} + \log\sigma_{ik} - d/k, \quad (4.5)$$

where there are no interactions between dimensions. Because ANSE uses a dense matrix for its bilinear similarity function, it has the advantage of incorporating interactions between dimensions of the node embeddings allowing for more complex relationships between nodes to be modeled. An inherent advantage of Gaussian embeddings is the benefit of providing confidence scores for predictions.

Hyperbolic embeddings, specifically embeddings onto the Poincaré ball, have been shown to be effective for link prediction in knowledge graphs [14]. The radius $c^{-1/2}$

Poincaré ball is a d-dimensional manifold: $\mathbb{B}_c^d = \{x \in \mathbb{R}^d : c||x||^2 < 1\}$. The distance between points u and v in the Poincaré ball is given by:

$$d_{\mathbb{B}}(u,v) = cosh^{-1}\left(1 + 2\frac{||u-v||^2}{(1-||u||^2)(1-||v||^2)}\right). \tag{4.6}$$

This distance measure can be used to develop a score function for embeddings of a hypergraph: $S(e_s, r, e_o) = -d_{\mathbb{B}}(\mathbf{h}_s^{(r)}, \mathbf{h}_o^{(r)})^2 + b_s + b_o$. Here r represents a specific type of relationship and introduces asymmetry into the otherwise symmetrical function, e_s and e_o are entities i and o, respectively, and b_s and b_o are biases. The vectors $\mathbf{h}_s^{(r)}$ and $\mathbf{h}_o^{(r)}$ are functions of hyperbolic embedding vectors $\mathbf{h}_s$ and $\mathbf{h}_o$ of e_s and e_o, respectively, and dependent on their role in the relationship r.

Hyperbolic embeddings have been shown to outperform euclidean embeddings for hierarchical relationships and are also a logical choice for node embedding of graphs. An arbitrary sized tree can be embedded onto the Poincaré disk ($\mathbb{B}_c^2$) with arbitrarily small distortion in distances between vertices [78] and more sophisticated complex networks have also been demonstrated to have hyperbolic structure [50]. One advantage euclidean embeddings (such as ANSE) retain over hyperbolic embeddings is their general applicability to downstream methods, which are often designed to operate in euclidean space. However, this advantage is diminishing as methods designed to operate in hyperbolic space are developed. For example, hyperbolic graph convolution networks [25] map euclidean graph signals to hyperbolic spaces and operate on them in that space.

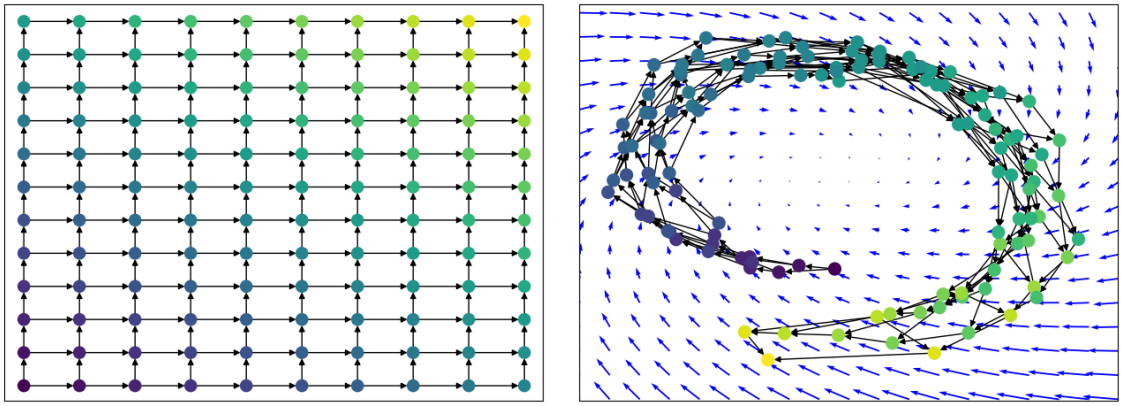
4.5 Experiments

In this section, we conduct multiple experiments to demonstrate both quantitatively and qualitatively the effectiveness of our approach to embedding nodes of a graph. We first provide a visual of how ANSE embeds the nodes of a digraph and

learns a similarity function that allows the direction of an edge to be recovered from the embeddings. We then measure the effectiveness of ANSE compared to other node embedding methods by how well the embeddings can recover missing edges from the graph. To measure this, a subset of the edges of a graph is withheld during the training phase and used in conjunction with a random set of node pairs that do not contain an edge between them to determine how well the embeddings can differentiate the two groups.

4.5.1 Lattice Example

A 2D lattice graph provides a simple but useful visual representation of the embedding produced by ANSE. The lattice is composed of 10 rows and 12 columns of vertices. All lateral edges in the graph are oriented to point right and all vertical edges to point up. Eight walks of length three are sampled from every node in the graph. The lattice is shown in Figure 4.1a and the learned 2-dimensional embedding of the vertices is shown in 4.1b. Additionally, the effect of the matrix $\mathbf{A}$ in the similarity function is drawn as a vector field in the background such that a source node is most similar to target nodes that lie in the direction of the local arrows. The embedded nodes form a spiral pattern with the bottom-left-most node of the original lattice innermost in the spiral and the top-right-most one outermost. Diagonal sets of nodes in the original lattice representation are structurally similar in the graph and are roughly clustered together along the spiral. Edges are also oriented along the direction of the field induced by $\mathbf{A}$. Because of the effect of $\mathbf{A}$, nodes have a higher affinity with nodes immediately ahead of them when moving outwards from the center of the spiral than nodes behind or even side by side to them.



(a) Original Lattice Graph (b) Embedded Lattice Graph

Figure 4.1: **Example embedding of a directed lattice.** The original lattice (a) and vector embeddings (b). The direction and magnitude of the vector field illustrates the bias of the similarity measure across the space.

4.5.2 Link Prediction

Node embeddings can be used to predict missing or future edges in a graph by measuring pairwise similarities. Node pairs with a high similarity score are more likely to form an edge. We evaluate the area under the receiver-operator curve (AUC) for several real world networks to evaluate our method and compare to several other skip-gram algorithms.

Arxiv [53] is a co-authorship network consisting of 5,242 nodes representing authors and 28,980 edges linking authors who have co-authored a paper together. Arxiv is the only undirected network in this set of experiments. **Cora** [84] is a citation network where the 23,166 nodes in the graph are papers and the 91,500 edges points from one paper to another if the first paper cites the second. **Epinions** [70] is a social network dataset with 75,879 users (nodes) and 508,837 edges indicating trust placed by one user in another.

The reciprocity of a graph is the ratio of the number of bidirectional edges to the total number of edges. The three datasets we evaluate on provide a diverse sample of graphs with various reciprocities. Arxiv, an undirected graph, has a reciprocity

of 1.00. Cora with nearly 0 bidirectional edges and a reciprocity of 0.05. Epinions sits in the middle at 0.40. Together, the three graphs provide a range of node-pair relations covering bidirectional, one-directional, and unrelated.

We use the same hyper-parameters for asymmetric node similarity embeddings (ANSE) and the hypersphere variant (ANSE-H) across all three experiments. Nodes are embedded into a 32-dimensional space and 16 total walks (8 forward, 8 backward) are collected at each node. Each walk continues for three steps. In ANSE-H, we use a single elementary reflector matrix for $\mathbf{A}$. We evaluate our method against several modern skip-gram style embedding methods. Deepwalk [68] and Node2Vec [41] are methods developed for undirected graphs. Line [86] and asymmetric proximity preserving embeddings (APP) [106] are methods for embedding nodes of either undirected or directed graphs. We compare ANSE against the scores of the other methods reported in [106]. We also compare against HOPE [67], a recent spectral method for directed graphs that learns both a source and target embedding for each node.

The embedding methods are trained using 70% of the edges of the graph, while the remaining 30% appear as positive examples in the test set along with an equal number of random node pairs without edges to form the negative examples. The pairwise node score is used to whether or not an edge exists between each pair of nodes in the test set. The AUC for each method is given in Table 4.1. On two of the three graph domains, ANSE and ANSE-H demonstrates significant improvements over all the other methods tested. On Cora, APP joins ANSE and ANSE-H in outperforming the other methods tested. Despite Arxiv being an undirected graph, our method learns asymmetric random walk similarities between pairs of nodes resulting in the high AUC.

The effect of the length of the walk on the AUC is shown for Cora in Figure 4.2. Other parameters are held constant to the values given above. While there is some amount of robustness to the length of the walk, the AUC begins to rapidly drop off

Table 4.1: Link Prediction Area Under Curve (AUC)

	Arxiv	Cora	Epinions
DeepWalk	0.887	0.936	0.823
Node2Vec	0.810	0.734	0.865
Line	0.750	0.694	0.867
APP	0.887	**0.944**	0.926
HOPE	0.596	0.874	0.629
ANSE	0.902	0.925	**0.948**
ANSE-H	**0.920**	**0.942**	0.924

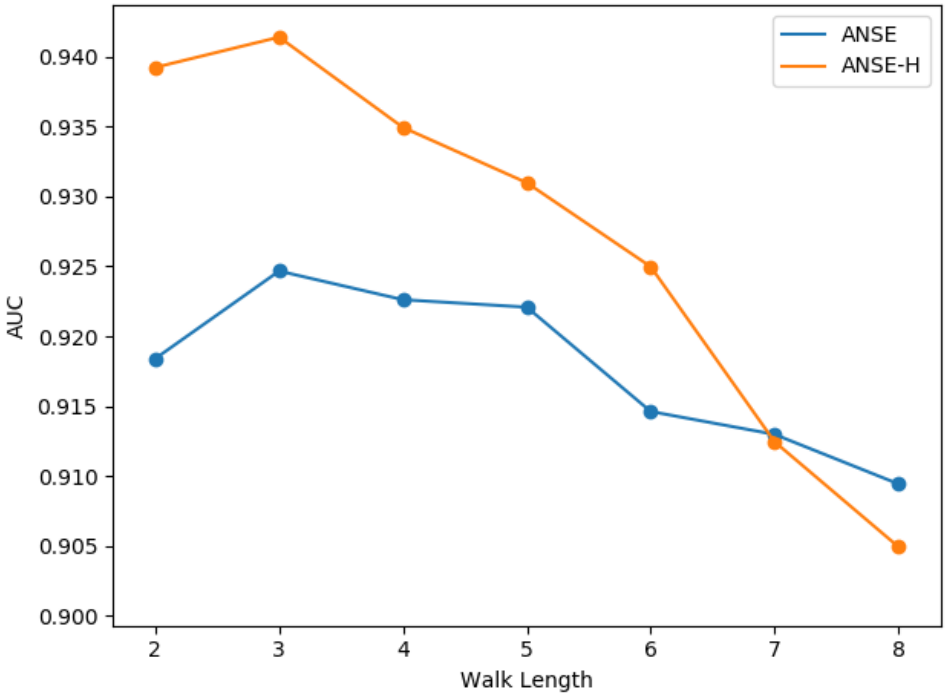


Figure 4.2: **AUC as a function of walk length.** The AUC for varying the walk length of ANSE and ANSE-H when embedding the nodes of the Cora dataset. Both ANSE and ANSE have an optimal walk length of 3.

for longer walks. This isn't unexpected because, although Cora has a diameter of 20, it a much smaller effective diameter. Long walks quickly spread across a large portion of the graph causing embeddings to converge to similar values which hampers the ability to differentiate between likely adjacent nodes and separated nodes.

4.6 Conclusion

We proposed ANSE, a scalable method to embed a digraph into a vector space, and ANSE-H, a variant of ANSE that embeds the graph onto a hypersphere. ANSE simultaneously learns a vector representations of nodes and an asymmetric similarity function for embedding directed graphs. Learning both the embedding and the similarity function offers the ability to recover the direction of edges from the embedded nodes. Additionally, we proposed a random walk sampling method to improve learning for nodes without either incoming or outgoing edges. On multiple real world datasets, ANSE and ANSE-H outperforms other skip-gram embedding schemes for link prediction.

CHAPTER 5

FILTERED MANIFOLD ALIGNMENT

5.1 Introduction

In many domains, raw data has become abundant and inexpensive, while labeled data often remains sparse and expensive to create. Methods which can leverage available labeled data to bolster learning on related unlabeled or sparsely labeled datasets are essential to cost-effective machine learning. Transfer learning is the process of adapting knowledge from one problem to another and domain adaptation is a subset of transfer learning focused on applying knowledge of a specific problem from one domain, the source, to a second domain, the target. The data distribution of the source domain often varies from that of the target domain. For example, the pose, lighting, and backgrounds of a set of images from an amateur photographer may differ significantly from precompiled, labeled photos taken by a professional. These differences can cause a significant degradation of the accuracy of a machine learning model trained on one dataset and applied to another when they are not adjusted for.

Manifold alignment (MA) is a domain adaptation approach based on the theory that data in both the source and target domains are drawn from the same underlying, low-dimensional manifold. MA projects and aligns two or more datasets onto this manifold so that learning can be done in the joint space. In this chapter, we introduce a new semi-supervised filtered manifold alignment (FMA) technique in which we align two datasets by first learning an individual embedding for each domain based on a discrete graph approximation of the underlying manifold and then subsequently aligning these two embeddings by connecting the two graphs via cross-domain cor-

respondences. The initial embeddings filter noise in the original domains prior to aligning the two domains. This ensures that the filter of one domain does not affect the second domain or the cross-domain correspondences. This approach offers low computational complexity, can be applied to new samples even after the alignment is complete, and is applicable to heterogeneous domains (i.e. domains with different feature sets). Additionally, a feature-based version of our method further simplifies the task of embedding new data points that weren't part of the original alignment. The complexity of the feature-based method scales with the number of features rather than the number of samples in the dataset allowing one to choose the method most suited for their task.

5.2 Manifold Alignment

Manifold alignment [96] facilitates knowledge transfer between two domains by utilizing correspondences across the domains to align their underlying manifolds. Semi-supervised manifold alignment (SMA) [43] combines the task of projecting data down to its underlying low-dimensional manifold and aligning instances across datasets. Given datasets $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$ with possibly different numbers of samples and features, SMA seeks low n-dimensional embeddings $\mathbf{Z}^{(1)}$ and $\mathbf{Z}^{(2)}$ that preserve intra-dataset relationships and inter-dataset correspondences. The former can be calculated using a similarity measure, $sim(a, b)$, between two samples such as the cosine similarity of their feature vectors. The latter, the inter-dataset correspondences, $cor(a, b)$, are determined by matching a small set of labeled instances from the target domain with labeled instances in the source domain. Weight matrices $\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{*}\}$, where $\mathbf{W}^{(a)}_{ij} = sim\left(\mathbf{X}^{(a)}_{i}, \mathbf{X}^{(a)}_{j}\right)$ encodes the similarity (e.g. cos similarity) between samples i and j in dataset $a \in \{1, 2\}$ and $\mathbf{W}^{*}_{ij} = cor\left(\mathbf{X}^{(1)}_{i}, \mathbf{X}^{(2)}_{j}\right)$ encodes the correspondence between sample i in the source domain and sample j in the target domain (e.g. 1 when they share a label and 0 if the labels differ or are unknown). In an optimization

setting, the two goals of preserving intra- and inter-dataset relationships are expressed as a pair of loss functions:

$$L_1(Z) = \sum_{a=1}^{2} \sum_{i,j} \mathbf{W}_{ij}^{(a)} ||Z_i^{(a)} - Z_j^{(a)}||_{l^2}^2 \tag{5.1}$$

$$L_2(Z) = \sum_{i,j} \mathbf{W}_{ij}^{*} ||Z_i^{(1)} - Z_j^{(2)}||_{l^2}^2. \tag{5.2}$$

The loss in (5.1) penalizes the embedding distance between related examples within a dataset and (5.2) penalizes the embedding distance between corresponding examples across the two datasets. Combining the matrices such that $\mathbf{Z} = \begin{bmatrix} \mathbf{Z}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}^{(2)} \end{bmatrix}$ and $\mathbf{W} = \begin{bmatrix} \mathbf{W}^{(1)} & \mathbf{W}^{*} \\ \mathbf{W}^{*} & \mathbf{W}^{2} \end{bmatrix}$ and summing the two loss functions produces the combined loss:

$$L(\mathbf{Z}) = \sum_{i,j} \mathbf{W}(i,j) ||\mathbf{Z}[i] - \mathbf{Z}[j]||_{l^2}^2 \tag{5.3}$$

$$= tr(\mathbf{Z}^T \mathbf{L} \mathbf{Z}), \tag{5.4}$$

where $\mathbf{L} = \begin{bmatrix} \mathbf{D}^{(1)} - \mathbf{W}^{(1)} & -\mathbf{W}^{*} \\ -\mathbf{W}^{*} & \mathbf{D}^{(2)} - \mathbf{W}^{(2)} \end{bmatrix}$ is referred to as the joint Laplacian. Given the joint degree matrix $\mathbf{D} = \begin{bmatrix} \mathbf{D}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^{(2)} \end{bmatrix}$, the constraint $\mathbf{Z}^T \mathbf{D} \mathbf{Z} = \mathbf{I}$ removes trivial solutions to the optimization problem. Minimizing (5.4), subject to this constraint, is equivalent to solving the generalized eigenvalue problem for the first n nontrivial eigenvectors $\mathbf{\Phi}$ corresponding to the smallest eigenvalues $\mathbf{\Lambda}$:

$$\mathbf{L}\mathbf{\Phi} = \mathbf{D}\mathbf{\Phi}\mathbf{\Lambda}. \tag{5.5}$$

The rows of the eigenvector matrix provide low dimensional embeddings of each sample, $\mathbf{X}_i \rightarrow \mathbf{\Phi}_i$, that allows samples from different domains to be compared.

5.3 Filtered Manifold Alignment

The approach in filtered manifold alignment (FMA) is to separate the process of embedding each dataset into a low dimensional space from the process of aligning the embeddings. FMA calculates the spectra of the graph Laplacians associated with the source and target domains individually and combines them into an approximation of the spectra of the joint graph Laplacian.

In order to separate the embedding process from the alignment process, FMA first converts the generalized eigenvector equation in (5.5) into a standard eigenvector problem. Because $\mathbf{D}$ is a diagonal matrix, this conversion is straightforward and produces the standard eigenproblem:

$$\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}\mathbf{\Phi} = \mathbf{\Phi}\mathbf{\Lambda}. \tag{5.6}$$

The n-dimensional embeddings of each sample for semisupervised manifold alignment are now given by the rows of the matrix $\mathbf{Z} = \mathbf{D}^{-1/2}\mathbf{\Phi}_{*,1:n}\mathbf{\Lambda}^{-1/2}_{1:n,1:n}$.

Once converted into a standard eigenproblem, the next step is to divide the embedding and aligning tasks. This is done by separating the joint Laplacian into two matrices. The joint graph Laplacian is the sum of the disconnected Laplacian $\mathbf{L}^* = \begin{bmatrix} \mathbf{L}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}^{(2)} \end{bmatrix}$ and the product of the cross-domain incidence matrix with its transpose: $\mathbf{L} = \mathbf{L}^* + \mathbf{A}\mathbf{A}^T$. The cross-domain incidence matrix $\mathbf{A}$ maps each cross-domain correspondence pair (i, j) to a unique column k in $\mathbf{A}$, such that $\mathbf{A}_{ik} = 1$ and $\mathbf{A}_{jk} = -1$. All other elements of $\mathbf{A}$ are 0. The L.H.S. of (5.6) becomes:

$$\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}\mathbf{\Phi} = \left(\mathbf{D}^{-1/2}\mathbf{L}^*\mathbf{D}^{-1/2} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{A}^T\mathbf{D}^{-1/2}\right)\mathbf{\Phi}. \tag{5.7}$$

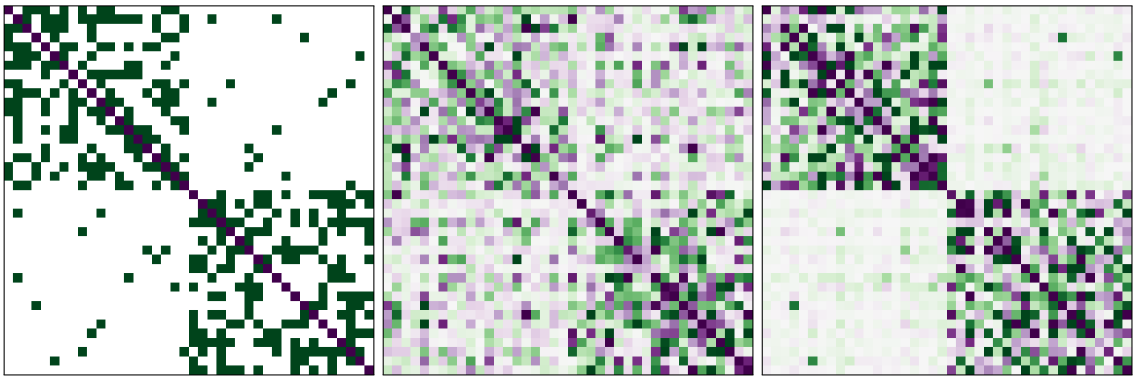
This eigenproblem is solved by determining the eigenvectors and eigenvalues of $\mathbf{D}^{-1/2}\mathbf{L}^*\mathbf{D}^{-1/2}$ and updating each using $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{A}^T\mathbf{D}^{-1/2}$ to match $\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$.

Because the disconnected Laplacian is block diagonal and symmetric (each block being the Laplacian of one domain), the eigendecomposition of $\mathbf{D}^{-1/2}\mathbf{L}^*\mathbf{D}^{-1/2}$ can be efficiently computed by computing the eigendecomposition of each block and then combining them:

$$\mathbf{D}^{-1/2}\mathbf{L}^*\mathbf{D}^{-1/2} = \begin{bmatrix} \mathbf{\Phi}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Phi}^{(2)} \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}^{(2)} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Phi}^{(2)} \end{bmatrix}^T . \tag{5.8}$$

FMA retains the smallest $n/2$ eigenvectors of each block in (5.8). The eigendecomposition process of each block is equivalent to using Laplacian eigenmaps [15] to embed each of the original datasets. The rows of the matrix $\mathbf{\Phi}^{(i)}$ produces an n-dimensional embedding of the original samples in $\mathbf{X}^{(i)}$ that best preserves the local geometry, according to the loss in (5.1). This subspace embedding has the effect of denoising the manifold represented by the graphs of each domain by filtering out eigenvector-eigenvalue pairs associated with the highest frequencies on the graph [31]. By minimizing (5.1) separately from (5.2), this filtering focusses on noise in the intra-dataset relationships and not the cross domain correspondences. This filtering effect can be seen in Figure 5.1 where the Joint Laplacian of two graphs is decomposed, filtered, and reassembled.

The second step in FMA is to align the projections for each dataset by updating the joint eigenvectors and eigenvalues using the cross-domain correspondences. We adapt the low-rank spectral update in [21] (Algorithm 5.1). Given $\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{X}$, the spectral update computes new singular values and vectors $\hat{\mathbf{U}}\hat{\mathbf{S}}\hat{\mathbf{V}}^T = \mathbf{X}+\mathbf{A}\mathbf{B}^T$ without ever directly referencing $\mathbf{X}$. We apply the update algorithm to the eigenvectors and eigenvalues computed above. Alg. 5.2 reduces the complexity of the spectral update



(a) Original Laplacian (b) Standard Filter (c) Subgraph Filter

Figure 5.1: **Comparison of graph filtering methods**. The original Laplacian (a) is composed of two 20 node Erdős–Rényi graphs with several random connecting edges. The standard filter (b) shows the matrix recomposed from half of its eigenvectors. The subgraph filter (c) performs the same eigen-filter on each subgraph before combining them via the linking edges resulting in smoother off-diagonal blocks.

Algorithm 5.1: SVD Update

Solves $\mathbf{U}'\mathbf{S}'\mathbf{V}'^T = \mathbf{X} + \mathbf{A}\mathbf{B}^T$ given $\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{X}$

1: **Input:** $\mathbf{U}$, $\mathbf{S}$, $\mathbf{V}$, $\mathbf{A}$, $\mathbf{B}$

2: $\begin{bmatrix}\mathbf{U} & \mathbf{P}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{U}^T\mathbf{A} \\ \mathbf{0} & \mathbf{R}_A\end{bmatrix} \leftarrow QR\left(\begin{bmatrix}\mathbf{U} & \mathbf{A}\end{bmatrix}\right)$

3: $\begin{bmatrix}\mathbf{V} & \mathbf{Q}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{V}^T\mathbf{B} \\ \mathbf{0} & \mathbf{R}_B\end{bmatrix} \leftarrow QR\left(\begin{bmatrix}\mathbf{V} & \mathbf{B}\end{bmatrix}\right)$

4: $\mathbf{Z} = \begin{bmatrix}\mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} - \begin{bmatrix}\mathbf{U}^T\mathbf{A} \\ \mathbf{R}_A\end{bmatrix}\begin{bmatrix}\mathbf{V}^T\mathbf{B} \\ \mathbf{R}_B\end{bmatrix}^T$

5: $\mathbf{U}^*, \mathbf{S}^*, \mathbf{V}^* \leftarrow svd(\mathbf{Z})$

6: $\hat{\mathbf{U}} \leftarrow \begin{bmatrix}\mathbf{U} & \mathbf{P}\end{bmatrix}\mathbf{U}^*$

7: $\hat{\mathbf{S}} = \mathbf{S}^*$

8: $\hat{\mathbf{V}} \leftarrow \begin{bmatrix}\mathbf{V} & \mathbf{Q}\end{bmatrix}\mathbf{V}^*$

9: **Return:** $\hat{\mathbf{U}}, \hat{\mathbf{S}}, \hat{\mathbf{V}}$

Algorithm 5.2: Block SVD Update

Updates the eigenvectors and eigenvalues for $\mathbf{X} + \mathbf{A}\mathbf{A}^T$ given $\mathbf{X} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^T$

1: **Input:** $\mathbf{\Phi}$, $\mathbf{\Lambda}$, $\mathbf{A}$

2: $\mathbf{\Phi}', \mathbf{\Lambda}', \mathbf{\Phi}'^T \leftarrow svd\left(\mathbf{\Lambda} + \mathbf{\Phi}^T\mathbf{A}\mathbf{A}^T\mathbf{\Phi}\right)$

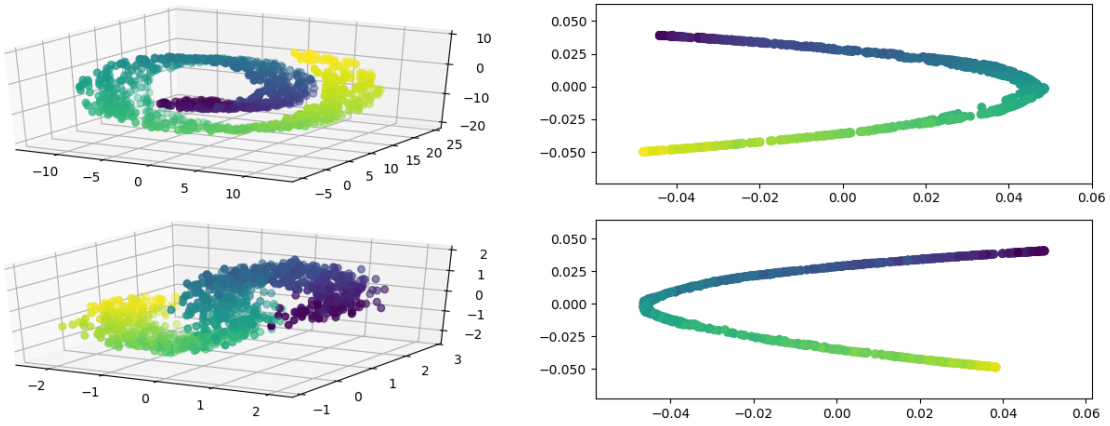
3: $\mathbf{\Phi}'' \leftarrow \mathbf{\Phi}\mathbf{\Phi}'$

4: **Return:** $\mathbf{\Phi}'', \mathbf{\Lambda}'$

under the following assumptions: $\mathbf{X}$ is symmetric, $\mathbf{A} = \mathbf{B}$, and only the projection of $\mathbf{A}$ onto the span of $\mathbf{X}$ is relevant.

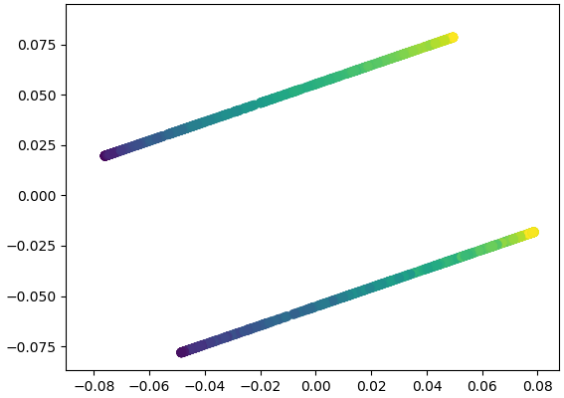
These assumptions hold when updating the eigenvectors of $\mathbf{L}^* + \mathbf{A}\mathbf{A}^T$. The last condition, specifically, is met by FMA because only the smallest eigenvalues and eigenvectors are used in the embedding. Any information in $\mathbf{A}$ that lies outside of the smallest eigenvectors of $\mathbf{L}^*$ will only increase the eigenvalues of the corresponding vectors and thus is likely to continue to be filtered out. Alg. 5.2 is efficient when $rank(\mathbf{X})$ is small because it relies on a spectral decomposition of a square matrix whose size is equal to its rank. By performing the filtering in the first step of FMA, the rank of the disconnected Laplacian has been reduced, making the update step efficient. The complete FMA algorithm to embed two datasets is given in Alg. 5.3.

An illustrative example of the process FMA performs is given in Figure 5.2. The two datasets being aligned are composed of 400 points sampled from two different three-dimensional manifolds: the swiss roll manifold and the S-curve manifold (Figure 5.2a). Noise is added to the points and they are colored according to their location along the true intrinsic 1 dimensional manifold. A nearest neighbor graph, using the 5 nearest neighbors of each point is formed for each of the two datasets. Each point is embedded onto the first two nontrivial eigenvectors of the Laplacian matrix of the nearest neighbor graphs (Figure 5.2b). The joint graph is formed using 40 corresponding points across the two manifolds and the block SVD update is used to combine and align the two embeddings and reduce the embeddings to 1D (Figure 5.2c).



(a) Original Data Points

(b) Independent Spectral Embeddings



(c) Joint Embedding

Figure 5.2: **Example FMA process.** (a) A random sample of 400 points are collected from a noisy 3D swiss roll manifold and a noisy 3D S-curve manifold. (b) The datapoints from the two manifold are embedded independently onto a two dimensional manifold via spectral embedding. (c) The two embeddings are aligned onto the same 1D manifold.

5.3.1 Feature-Level Alignment

Semi-supervised manifold alignment can be modified to find a linear mapping from the original feature spaces to the embedding space. This is referred to as linear or feature-level alignment in contrast to instance-level alignment because it maps features of the original domains into the joint embedding space rather than mapping instances directly. Replacing (5.5) with the new generalized eigenproblem [98]:

$$\mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{\Phi} = \mathbf{X}^T\mathbf{D}\mathbf{X}\mathbf{\Phi}\mathbf{\Lambda}, \tag{5.9}$$

allows us to solve for the eigenvectors $\mathbf{\Phi}$, which can be used to map features from the original domains into the joint embedding space $\mathbf{X} \rightarrow \mathbf{X}\mathbf{\Phi}$. The R.H.S of (5.9) now contains the dense matrix $\mathbf{X}$ in addition to the diagonal matrix $\mathbf{D}$ in (5.5). The conversion to a regular eigenvector problem requires an extra step to calculate $(\mathbf{X}^T\mathbf{D}\mathbf{X})^{-1/2}$. The adjusted algorithm is given in Alg. 5.4.

Algorithm 5.3: Filtered Manifold Alignment

Input: Data Matrices: $\mathbf{X}^{(1)} \in \mathbb{R}^{m_1 \times n_1}$, $\mathbf{X}^{(2)} \in \mathbb{R}^{m_2 \times n_2}$
Cross Correspondence Incidence Matrix: $\mathbf{A}$
Neighbors: k
Embedding Dimension: n
for all $\mathbf{X}^{(i)}$ **do**
  $\mathbf{L}^{(i)} \leftarrow knn(\mathbf{X}^{(i)}, k)$
  $\mathbf{\Phi}^{(i)}, \mathbf{\Lambda}^{(i)} \leftarrow eig_{n/2}\left((\mathbf{D}^{(i)})^{-1/2}\mathbf{L}^{(i)}(\mathbf{D}^{(i)})^{-1/2}\right)$
end for
$\mathbf{\Phi}, \mathbf{\Lambda} \leftarrow \begin{bmatrix} \mathbf{\Phi}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Phi}^{(2)} \end{bmatrix}, \begin{bmatrix} \mathbf{\Lambda}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}^{(2)} \end{bmatrix}$
$\mathbf{A}' \leftarrow \mathbf{D}^{-1/2}\mathbf{A}$
$\mathbf{\Phi}', \mathbf{\Lambda}' \leftarrow svdu(\mathbf{\Phi}, \mathbf{\Lambda}, \mathbf{A}')$; `// Alg. 5.2`
$\mathbf{Z} \leftarrow \mathbf{D}^{-1/2}\mathbf{\Phi}'\mathbf{\Lambda}'^{-1/2}$
Return: $\mathbf{Z}_{1:m_1,1:n}$, $\mathbf{Z}_{m_1+1:m_1+m_2,1:n}$

Algorithm 5.4: Linear Filtered Manifold Alignment

Input: Data Matrices: $\mathbf{X}^{(1)} \in \mathbb{R}^{m_1 \times n_1}$, $\mathbf{X}^{(2)} \in \mathbb{R}^{m_2 \times n_2}$
Cross Correspondence Incidence Matrix: $\mathbf{A}$
Neighbors: k
Embedding Dimension: n
for all $\mathbf{X}^{(i)}$ **do**
  $\mathbf{L}^{(i)}, \mathbf{D}^{(i)} \leftarrow knn(\mathbf{X}^{(i)}, k)$
  $\mathbf{T} = \left((\mathbf{X}^{(i)})^T \mathbf{D}^{(i)} \mathbf{X}^{(i)}\right)^{-1/2}$
  $\mathbf{\Phi}^{(i)}, \mathbf{\Lambda}^{(i)} \leftarrow eig_{n/2}\left((\mathbf{X}^{(i)}\mathbf{T}^{(i)})^T \mathbf{L}^{(i)} \mathbf{X}^{(i)} \mathbf{T}^{(i)}\right)$
end for
$\mathbf{\Phi}, \mathbf{\Lambda} \leftarrow \begin{bmatrix} \mathbf{\Phi}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Phi}^{(2)} \end{bmatrix}, \begin{bmatrix} \mathbf{\Lambda}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}^{(2)} \end{bmatrix}$
$\mathbf{A}' \leftarrow \mathbf{AT}$
$\mathbf{\Phi}', \mathbf{\Lambda}' \leftarrow bsvdu(\mathbf{\Phi}, \mathbf{\Lambda}, \mathbf{A}')$; // Alg. 5.2
$\mathbf{Z} \leftarrow \mathbf{XT\Phi}'(\mathbf{\Lambda}')^{-1/2}$
Return: $\mathbf{Z}_{1:n_1,1:n}$, $\mathbf{Z}_{n_1+1:n_1+n_2,1:n}$

5.3.2 Complexity Analysis and Extensions

Filtered manifold alignment computation is dominated by the matrix spectral decompositions. These include decompositions of each of the sparse graph Laplacian matrices, each of which has a naïve complexity of $O(N^3)$, and a decomposition of an $n \times n$ matrix where $n \ll N$ is the dimension of the final embedding space. The summed complexity of three smaller singular value decompositions is a large improvement over single step semi-supervised manifold alignment which diagonalizes the joint Laplacian matrix $O((2N)^3)$. Taking advantage of sparse solvers can further speed up both methods. Feature-level filtered manifold alignment scales with the size of the feature space rather than the number of samples. The spectral decompositions of the graph Laplacian matrices are replaced with $M \times M$ matrices where M is the number of features leading to a complexity of $O(M^3)$. As the number of samples in a dataset grows in comparison to the number of features per sample, linear FMA becomes more efficient compared to nonlinear FMA.

Feature-level filtered manifold alignment can be trivially applied to samples not in the initial alignment because it learns a linear mapping from the feature space of a

sample to the joint embedding space. Nonlinear alignment, which learns a direct map for each sample, can not be directly applied to new samples. However, the spectral update (Alg. 5.2) can be applied to embed a new sample using the incidence matrix of the nearest neighbors as the update matrix. Additionally, both instance-level and feature-level schemes can align more than 2 datasets at a time by extending the block diagonal formatting of the joint Laplacian and related matrices.

5.4 Experiments

In this section, we perform multiple experiments to illustrate the effectiveness of nonlinear instance-level filtered manifold alignment (FMA-I) and linear, feature-level filtered manifold alignment (FMA-F). We compare the performance of both filtered manifold alignment approaches to that of several state-of-the-art domain adaptation methods: Geodesic Flow Kernel (GFK) [38], Manifold Embedded Distribution Alignment (MEDA) [100], Correlation Alignment (CORAL) [85], Semi-Supervised Subspace Alignment (SSA) [104], and Semi-Supervised Manifold Alignment (SMA) [43]. GFK, MEDA, and CORAL are unsupervised methods. SSA and SMA are semi-supervised.

5.4.1 Datasets

We use 9 different datasets organized into 3 different experimental groups based upon shared classes between the datasets.[1] Each dataset is composed of a collection of images for which SURF (speeded up robust features) [77] and/or DeCaf6 [32] features are provided. The DeCaf6 features are the hidden representations of the images extracted from the second to last layer of the deep DeCaf convolutional neural network. In general, Decaf6 features are more descriptive than SURF features and

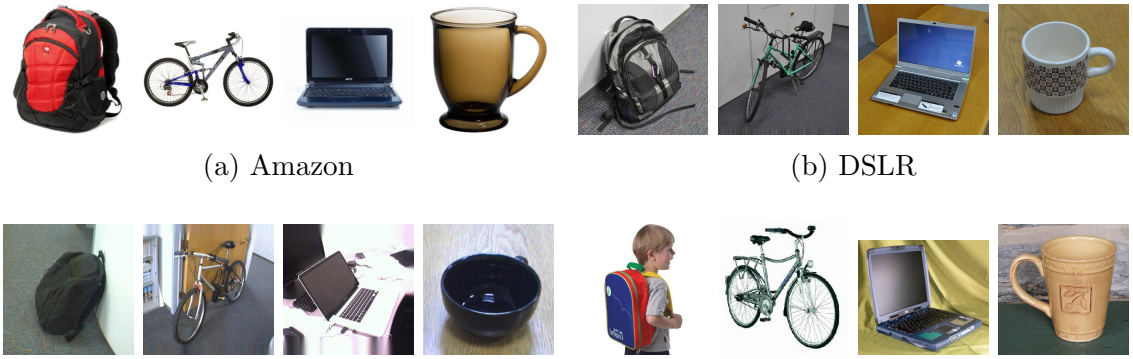
[1]Datasets available at `https://github.com/jindongwang/transferlearning`

provide better classification accuracy. The datasets were chosen because they are popular benchmarks composed of real world images.

The Office+Caltech [77] benchmark is formed from a set of 4 individual image datasets that share 10 categories and have SURF and DeCaf6 features available. The four domains that make up the Office+Caltech benchmark Amazon (A), Caltech (C), DSLR (D), and Webcam (W), are comprised of 958, 1123, 157, and 295 images, respectively. Example images from each domain and from four of the classes are shown in Figure 5.3. Differences in pose, lighting, resolution, and background activity between the datasets are apparent in the images motivating the need for domain alignment. Each ordered pair of domains forms a separate experiment with a source and target domain for a total of 12 combinations. We use $X \rightarrow Y$ to denote aligning the source domain X with target domain Y. Combined with the two separate feature sets for each domain, this provides 24 individual alignment experiments.

The MNIST-USPS [57] benchmark uses SURF features from a reduced set of 2000 images from the MNIST image dataset and 1800 from the USPS dataset. Each digit, 0-9, is represented in the images providing 10 class labels. A random sample of the digits in both datasets is given in Figure 5.4. Using both MNIST and USPS as the source and target domains provides 2 separate experiments.

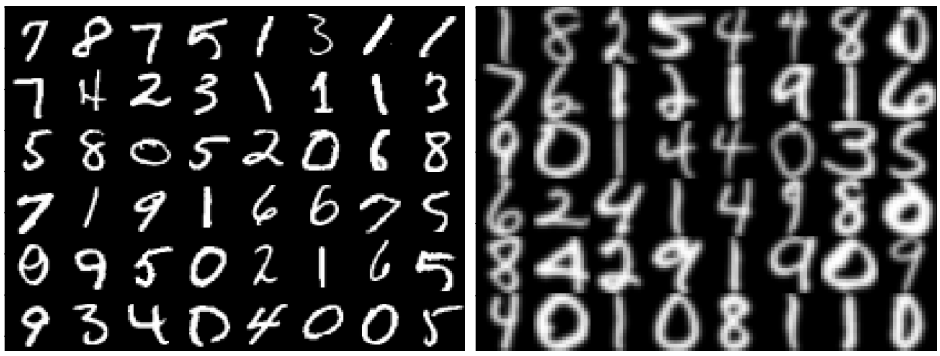
The final experiment uses 5 shared classes across the Caltech101 [40], ImageNet [35], and VOC2007 [34] datasets. The domains contain the DeCaf6 features for 1415, 7341, and 3376 images, respectively. The source/target combinations of the 3 domains provide 6 total experiments.



(a) Amazon (b) DSLR

(c) Webcam (d) Caltech

Figure 5.3: **Example Images from the Office+Caltech Datasets.** Each set of images contains a random image with each of the labels: backpack, bike, laptop, and mug.



(a) MNIST (b) USPS

Figure 5.4: **Example from the MNIST and USPS datasets.** Each set of digits contains a random sample of 48 images. MNIST (a) images are 48x48 pixels. USPS (b) images are 16x16 pixels.

5.4.2 Experimental Setup

We follow the same experimental setup as previous work wherever possible and report the best performance for each method. In Office-Caltech, 800 SURF and 4096 DeCaf6 features are extracted from each image. In the MNIST-USPS experiment, 256 SURF features are calculated for each image. Caltech-ImageNet-VOC use 4096 DeCaf6 features. All features are normalized to have zero mean and unit variance. For each experiment, 20 randomized splits are created and results are given as the average across each split. The splits for the Office-Caltech datasets are the same splits as provided in [38]. In each experiment, the training set for the semi-supervised methods is composed of 20 labeled instances per class for the source domain (except for DSLR in which only 8 are used) and 3 labeled instances per class for the target domain. The target domain labels are used only for correspondences and not for training the final classifier. Additionally, we introduce a new domain and feature adaptation task using the Caltech-Office dataset to demonstrate FMA's applicability to heterogeneous domains. The source domain and target domains in this task differ in using SURF or DeCaf6 features.

We use the same hyper-parameters for both versions of filtered manifold alignment across both the Office-Caltech and the Caltech-ImageNet-VOC experiment sets. Nearest neighbor graphs are formed from instances within each dataset using cosine similarity. The edge weights for the 12 nearest neighbors of each sample are set to $\alpha * cos(v_i, v_j)$, where $\alpha = 0.2$. Cross dataset correspondences between samples that share labels in the training set have edges weights of 1. We calculate 20 singular vectors from both the source and target domains and then combine them into a $n = 40$ dimensional aligned embedding space. The edge weights and embedding dimensions were selected for high scores across the experiments. We report on the sensitivity of these parameters in section 5.4.5. After alignment, a logistic classifier using categorical cross-entropy is trained on the labeled subset of the source domain and evaluated

on the entire target domain. In the MNIST-USPS, experiments the number of nearest neighbors is reduced to 4 and the coefficient of the edge weights is decreased to $\alpha = 1.0$. All other parameters match those above.

The same parameters used in FMA are used for MA where applicable. Results for the unsupervised methods (MEDA, GFK, CORA) are reported from [100] for both the Office+Caltech and MNIST-USPS experiments. For the remaining methods, the default parameters given for each method are used. SSA requires the embedding size be less than or equal to the minimum number of labeled source samples for any class. To this end, we used an embedding dimension of 8 across most tasks and reduced it when necessary to facilitate convergence.

5.4.3 Evaluation

The classification accuracies for the Office-Caltech datasets using SURF and DeCaf6 features are presented in Tables 5.1 and 5.2, respectively. In addition, Table 5.3 presents the accuracies for the task where the source and target domains use different feature sets. The classification results for MNIST-USPS as well as Caltech-ImageNet-VOC are given in Table 5.4. The best accuracies for each task are presented in boldface.

FMA demonstrated the second highest average accuracy across all the Office-Caltech experiments falling just below MEDA. Semi-supervised subspace alignment also has higher classification accuracy than FMA when using the SURF features, but does considerably worse with the Decaf6 features. Additionally, FMA had the highest classification accuracy on both the MNIST-USPS tasks and the Caltech-ImageNet-VOC tasks. These results demonstrate the effectiveness of filtered manifold alignment across a large range of different datasets and experiments.

Table 5.3 shows the accuracy of FMA on the Office-Caltech experiments when the source and target domain have different feature sets. The accuracy on the target

Table 5.1: Office + Caltech 10 Classification Accuracy using SURF Features

	MEDA	GFK	CORAL	SSA	SMA	FMA-I	FMA-F
A→C	43.9	40.7	**45.1**	72.4	29.8	33.4	32.8
A→D	45.9	40.1	39.5	**58.3**	56.8	56.9	57.4
A→W	53.2	37.0	44.4	54.7	**69.2**	65.4	67.8
C→A	56.5	46.0	52.1	**59.8**	47.9	49.2	48.1
C→D	50.3	40.8	45.9	**59.5**	56.1	53.7	58.0
C→W	53.9	37.0	46.4	54.0	**68.7**	65.3	67.2
D→A	41.2	28.7	37.7	**59.4**	40.8	50.8	49.8
D→C	34.9	29.3	33.8	**71.2**	27.0	34.8	33.2
D→W	**87.5**	80.3	84.7	53.9	68.1	72.2	70.8
W→A	42.7	27.6	36.0	**60.1**	45.5	50.0	50.3
W→C	34.1	24.8	33.7	**74.2**	28.5	31.4	31.5
W→D	**88.5**	85.4	86.6	54.3	56.5	56.1	59.0
Ave	52.7	43.1	48.8	**61.0**	49.6	51.6	52.2

domains roughly matches those of the homogeneous domain tasks using the same features as the target domains. None of MEDA, GFK, CORAL, or SSA are applicable to domains with different feature sets.

Figure 5.5 shows the time in seconds to align each pair of domains in the Caltech-ImageNet-VOC experiments from a cold start. Each method was implemented in Python3.7.5 and ran on an 3.60GHz AMD Ryzen 7 3700X Processor with 16GB of RAM. Across every experiment, FMA and linear FMA provide significant time advantages over the other compared techniques. Additionally, experiments involving the larger ImageNet dataset as either the source or target domain demonstrate the superior complexity of feature-level FMA when the number of samples eclipses the number of features in the dataset as compared to instance-level FMA.

Table 5.2: Office + Caltech 10 Classification Accuracy using DeCaf6 Features

	MEDA	GFK	CORAL	SSA	SMA	FMA-I	FMA-F
A→C	**87.4**	79.2	83.2	75.9	69.2	84.6	85.8
A→D	88.1	82.2	84.1	70.8	88.9	**95.4**	91.9
A→W	88.1	70.9	74.6	85.6	88.7	**95.0**	94.5
C→A	**93.4**	86.0	92.0	69.0	90.7	91.4	91.4
C→D	91.1	86.6	84.7	59.8	87.9	**95.1**	90.6
C→W	**95.6**	77.6	80.0	72.1	89.1	92.5	92.0
D→A	**93.0**	76.3	85.5	70.0	55.8	91.3	90.8
D→C	**87.5**	71.4	76.8	70.6	48.4	85.6	85.7
D→W	97.6	**99.3**	**99.3**	72.5	72.4	94.2	92.7
W→A	**99.4**	76.8	81.2	68.8	66.7	91.6	91.9
W→C	**93.2**	69.1	75.5	72.4	55.7	85.7	86.2
W→D	99.4	**100.0**	**100**	71.7	86.1	96.7	91.9
Ave	**92.9**	74.0	84.7	71.6	74.6	91.6	90.5

Table 5.3: Office 10 Classification Accuracy SURF to DeCaf6 Features

	SURF to DeCaf6		DeCaf6 to SURF	
	FMA-I	FMA-F	FMA-I	FMA-F
A→C	82.8	82.9	32.6	33.4
A→D	90.2	88.6	58.3	58.9
A→W	92.9	94.4	69.9	70.0
C→A	89.8	89.6	49.9	49.7
C→D	91.7	89.6	58.3	58.4
C→W	90.7	91.5	68.4	68.8
D→A	90.2	90.1	50.0	49.8
D→C	84.1	84.7	32.4	33.0
D→W	92.7	93.5	71.8	70.8
W→A	90.8	91.2	50.0	50.4
W→C	83.8	84.6	30.3	31.7
W→D	92.7	91.1	60.1	59.7
Ave	89.4	89.3	52.7	52.8

Table 5.4: MNIST-USPS and Caltech-ImageNet-VOC Classification Accuracy

	MEDA	GFK	CORAL	SSA	SMA	FMA-I	FMA-F
M→U	**89.5**	31.2	49.2	63.9	66.0	87.3	76.3
U→M	72.1	46.5	30.5	73.3	58.0	**81.0**	66.5
Ave	80.8	38.9	39.9	68.6	62.0	**84.2**	71.4
C→I	76.1	49.3	21.3	50.0	17.4	**81.9**	76.0
C→V	54.3	30.7	48.6	**63.0**	45.0	58.9	62.5
I→C	73.1	78.7	35.5	26.4	61.6	**99.8**	99.6
I→V	**67.3**	64.8	34.0	65.0	38.3	60.5	62.5
V→C	95.6	56.0	69.6	24.8	67.4	**99.8**	93.3
V→I	74.7	61.3	42.3	50.7	48.8	**84.4**	78.4
Ave	73.5	56.8	41.9	46.9	46.4	**80.9**	78.7

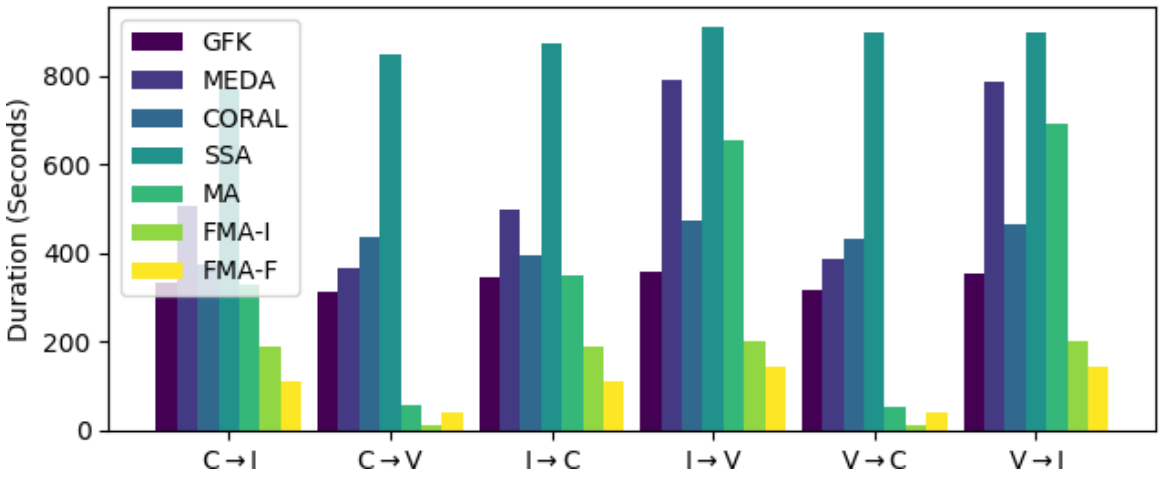
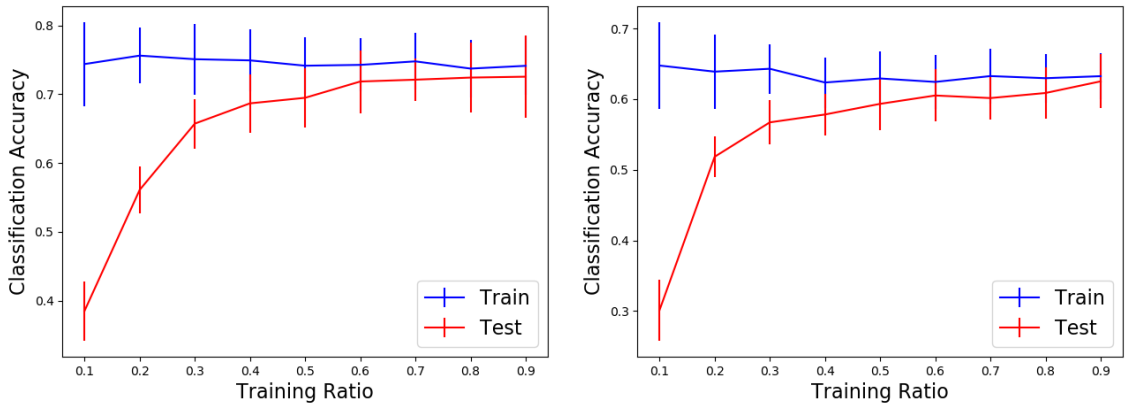


Figure 5.5: **Runtime comparison of alignment methods**. The time to align each pair of domains in the Caltech-ImageNet-VOC dataset is given for each of the 7 domain alignment methods tested.

5.4.4 Inductive Performance

Using the MNIST-USPS [57] benchmark, we evaluate the ability of feature-level filtered manifold alignment to generalize to unseen samples in the target domain. Samples that aren't initially available can't used for the graph construction portion of alignment. This can lead to forming a graph that is not an accurate representation of the domain's underlying manifold. However, unlike instance-level FMA, feature-level FMA learns a transformation matrix that can be applied to project new samples onto the joint manifold even though they weren't used in the original alignment phase.

For the inductive experiment, the hyper-parameters and experimental setup are the same as the MNIST-USPS experiment above, but only a portion of the target domain samples are used in the alignment phase. The remaining portion are withheld and then embedded using the learned linear transform on the features. We vary the portion of the target domain used for training. Figure 5.6 illustrates the results on both the MNIST to USPS transfer and the reverse. As the portion of the target domain samples used in training grows, the classification accuracy of the withheld samples approaches the accuracy of the training samples. Using 60% or roughly 1200 samples from the target domain builds an accurate graph of the domain manifold so that the learned joint embedding and transformation can generalize to new samples with only a minor loss in classification accuracy.



(a) MNIST→USPS (b) USPS→MNIST

Figure 5.6: **Inductive classification accuracy of FMA-F on MNIST-USPS** Classification accuracy on the training samples and the withheld test samples are given for USPS as the target domain (a) and for MNIST as the target (b). As the training size expands, the classification accuracy approaches the training accuracy.

5.4.5 Hyper-Parameter Evaluation

Using the office-Caltech dataset with SURF and DeCaf6 features, we study FMA's sensitivity to hyper-parameters for both instance-level and feature-level algorithms. In each experiment, we vary one parameter while keeping the others constant. The constant hyper-parameters match those used in the previous office-Caltech experiments: the edge weight coefficient is $\alpha = 0.2$ and the embedding dimension is set to 40.

Figure 5.7 presents the overall classification accuracy for the office-Caltech experiment for different final embedding dimensions. The initial embedding size for each individual dataset is always half of the final embedding size in each experiment.

Figure 5.8 demonstrates the effect of varying the scaling on the edge weights of the nearest neighbor graphs for each domain. A smaller scalar places less emphasis on the final embeddings of like instances within a domain having similar embeddings while a large value de-emphasizes the cross-domain correspondences in comparison.

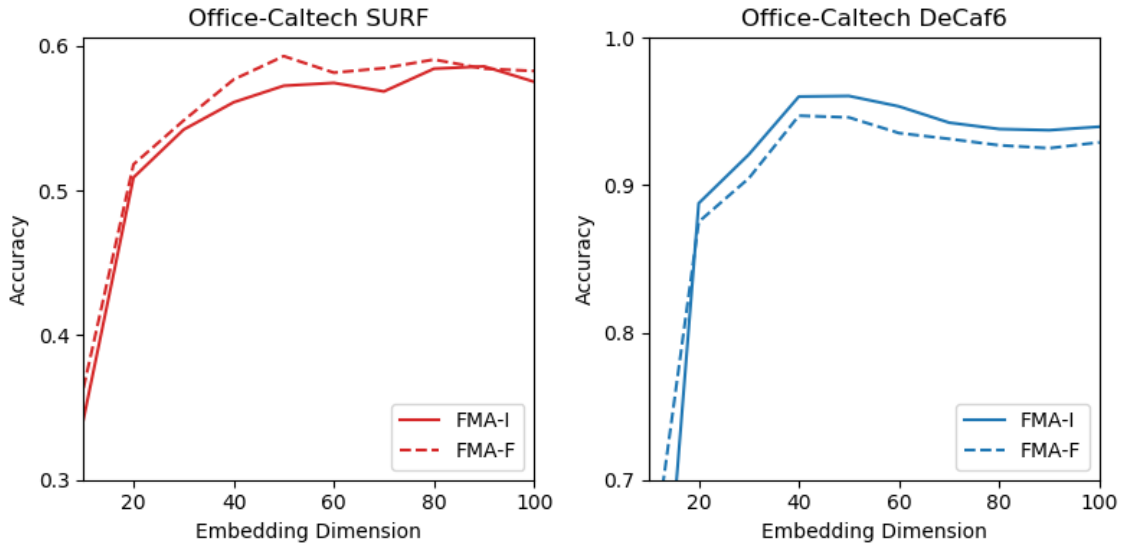


Figure 5.7: **Classification accuracy as a function of embedding dimension**. The effect on the overall classification accuracy for the office-Caltech dataset due to varying the final embedding dimension.

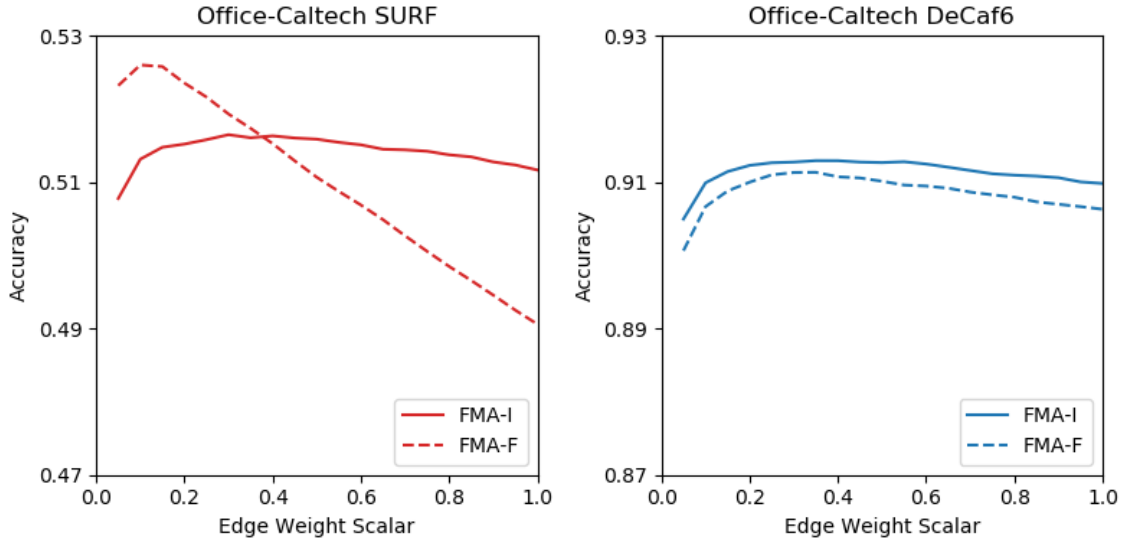


Figure 5.8: **Classification accuracy as a function of** α. The effect on the overall classification accuracy for the office-Caltech dataset due to varying the edge weights of the nearest neighbor graphs.

Note that increments on the y-axes are much smaller in comparison to Figure 5.7 due to scaling having a much smaller effect on the overall classification accuracy.

While the effect of the embedding dimension on the classification accuracy is large, it is due in part to the logistic classifier and the accuracy does somewhat level off for sufficiently large dimensions. Combined with the small effect of the edge weights on the final accuracy, this shows that FMA is not reliant on finding the best hyper-parameters and can perform well over a wide range of them.

5.5 Related Work

Domain Adaptation is a large area of study that stretches across disciplines such as computer vision, natural language processing, and machine learning. It is studied as an unsupervised, semi-supervised, or supervised problem. We focus on reviewing unsupervised and semi-supervised methods below.

Geodesic Flow Kernel (GFK) [38] is an unsupervised domain adaptation method. GFK treats the source domain and the target domain as points on a Grassmanian manifold defined by their principal components. An infinite-dimensional feature space, $\mathcal{H}^\infty$, can be constructed by interpolating between the source and target points. An inner product can be computed in $\mathcal{H}^\infty$ to construct a kernelized classifier: $\langle z_i, z_j \rangle = x_i^T \mathbf{G} x_j$, where z_i are the transformed features of sample point x_i and $\mathbf{G}$ is the kernel matrix.

Manifold Embedded Distribution Alignment (MEDA) [100], like FMA, takes a two step approach to dataset alignment. MEDA first performs manifold feature learning on each domain followed by dynamic distribution alignment. The manifold feature learning is done using GFK where learned features $z_i = \sqrt{\mathbf{G}x_i}$. The second step, dynamic distribution alignment, balances the importance of the marginal distribution and conditional distributions of samples between the domains. Soft labels, using a

classifier on z_i are used in place of actual labels in the target domain when calculating the distributions.

Correlation Alignment (CORAL) [85] transforms the source domain to match the target domain by first whitening the domain using the inverse square root of the covariance matrix of the source domain and then recoloring using the target covariance matrix: $Z_S = X_S \mathbf{C}_S^{-1/2} \mathbf{C}_T^{1/2}$, where $\mathbf{C}_i = cov(X_i) + \mathbf{I}$.

Semi-Supervised Subspace Alignment (SSA) [104] learns orthogonal linear maps for source and target domains by simultaneously preserving intra-domain and inter-domain relationships and reducing classification error. CORAL does so by minimizing a loss function that is a weighted sum of the empirical risk, the distance between corresponding labeled points across the two domains, and a manifold regularizer on the pairwise similarities of the unlabeled samples in the target domain.

Manifold alignment methods [43, 97, 99] are nonlinear embedding methods that work by forming a joining graph between samples within and across domains. The eigenvectors of the Laplacian of this graph corresponding to the smallest eigenvalues embed the original domains in a way to preserve the embedded distance between points.

5.6 Conclusion

Filtered manifold alignment provides a novel approach to performing manifold alignment and domain adaptation. Across a large number of canonical experimental test beds, FMA meets or exceeds other state-of-the-art methods, while being considerably faster in calculating the alignment. Additionally, because we present both an instance-based and a feature-based version, the method can scale with the smaller of the two options.

CHAPTER 6

CONCLUSION

Graphs are useful models of the relationships between objects in datasets and can be utilized by machine learning algorithms to great effect. This thesis proposed three innovative machine learning methods that incorporate the graph-structure of data to solve a wide range of tasks.

Quantum walk neural networks (QWNN) are designed for supervised learning problems in which there is an information rich signal on a graph. QWNN innovates over previous graph convolution networks by closely tying the graph dynamics to the learning process. A quantum walk determines how the signal is diffused across objects in the graph to be used for a classification or regression problem. In turn, the results of the problem change the parameters of the quantum walk to learn a better future diffusion process. We compare quantum walk neural networks against a range of other graph neural networks and across a wide variety of tasks. The results demonstrate the effectiveness of our method to diffuse information across the graph in a manner best suited to the current learning task. We also compare a model of QWNN invariant to the ordering of nodes in the graph to a model that uses a heuristic node ordering. While not offering the same guarantees, the heuristic model demonstrated the effectiveness of incorporating global graph information into the quantum walk process.

Asymmetric node similarity embeddings (ANSE) tackle the unsupervised node embedding problem for directed graphs. Directed graphs provide additional information compared to undirected graphs about the relationships between the objects

in the graph by orienting the edges between objects. Unfortunately, this additional information is lost when node embedding algorithms designed for undirected graphs are applied to directed graphs. ANSE preserves the asymmetric relationships between nodes in the graph. Unlike several other digraph node embedding methods, ANSE uses a single rather than a dual embedding for each node in the graph. Along with learning an asymmetric similarity function, this better encapsulates the codependency between a node's role as the source and the target of edges in the graph. We provide experimental results that show that not only does ANSE improve node embeddings for digraphs, it can provide better node embeddings for undirected graphs as well by better preserving asymmetric relationships between nodes, such as expected hitting time, that exist in undirected graphs.

Filtered Manifold Alignment (FMA) is a semi-supervised method for dataset alignment. The method incorporates the graph construction into the process of projecting two datasets onto a joint low-dimensional manifold. Unlike previous manifold alignment methods, FMA fully separates the embedding step from the alignment step. Not only does this improve the computational complexity of the method, it separates the natural filtering process that embedding in a lower dimensional space has on each dataset from interfering with the other dataset. Our method improves on both the efficiency and the quality of the alignments produced by previous manifold alignment methods. We offer both a nonlinear instance-level version and a linear feature-level version of our algorithm. Across multiple experiments, filtered manifold alignment demonstrates lower runtime and higher alignment accuracy compared to other modern domain adaptation methods. Additionally, the feature-level FMA algorithm can be used inductively to embed new data points that were not present in the initial alignment of the source and target datasets.

6.1 Future Work

In Chapter 3, we showed the effectiveness of quantum walk neural networks to perform regression and classification problems on graphs up to several hundred nodes with a high degree of accuracy. However, we also showed that the space complexity of our model scales quadratically with the size of the graph. In order for this neural network to be applicable to larger graphs (such as webscale graphs in the range of millions or billions of nodes), future work is needed to reduce this complexity. Another research direction of interest is the use of entangled walkers. Currently, each walk on the graph is considered independent. However, multiple walkers can interact causing entangled states in the quantum walk. Future work will study the effect entanglement has on the effectiveness of the QWNN.

Finally, the impressive results of the QWNN models that used heuristic node ordering imply that QWNN and potentially other graph convolutional neural networks would benefit from incorporating additional graph information into the process. In the case of QWNN, this currently comes at the cost of losing the invariant property of the neural network to different node ordering. Additional work to incorporate node centrality into the invariant QWNN may produce a method that includes the best aspects of both models.

In Chapter 4, we provide a model for learning a single embedding for each node in a digraph and a matrix used to parameterize the node similarity function. Our model comes from adapting the inner product (or cos similarity) of node embeddings as our similarity function. Alternative methods of computing the similarity of node embeddings exist, such as the absolute difference: $|\boldsymbol{\phi}_i - \boldsymbol{\phi}_j|$, or the euclidean distance between embeddings: $\sqrt{(\boldsymbol{\phi}_i - \boldsymbol{\phi}_j)^T(\boldsymbol{\phi}_i - \boldsymbol{\phi}_j)}$. These functions should be investigated in the pursuit of producing alternative asymmetric functions for use with digraphs.

In Chapter 5, we provide a method for aligning two datasets on a n-dimensional manifold. The dimension n is determined by searching a set of potential values. Sev-

eral methods of automatically choosing the embedding dimension for other alignment algorithms exist. Future work is necessary to determine if any of these approaches are applicable to filtered manifold alignment or if a new method of determining the optimal embedding dimension is needed.

Both the linear and nonlinear versions of the FMA algorithms can be adapted to align more than two datasets at one time but extending the block formulation of the joint Laplacian to include an additional block along the diagonal for each additional dataset as well as correspondences on the new off-diagonal blocks. Ideally, including more source domains would improve, or at worst have no effect on the classification accuracy of the target domain. Testing this on the Caltech101 [40], ImageNet [35], and VOC2007 [34] datasets using FMA-I led to a decrease in overall target accuracy as shown in Table 6.1. A possible explanation is that a drastically different edge and correspondence weighting scheme is necessary to manage the much larger set of cross-domain correspondences. Future work will investigate this discrepancy and hopefully provide the means to overcome it.

Table 6.1: Caltech-ImageNet-VOC Multi-Source Classification Accuracy

Target	Source			
	Caltech101	ImageNet	VOC2007	Multi-Source
Caltech101	-	99.8	99.8	78.1
ImageNet	81.9	-	84.4	53.0
VOC2007	58.9	60.5	-	55.8

BIBLIOGRAPHY

[1] Agarwal, Girish S, and Pathak, Pradyumna K. Quantum random walk of the field in an externally driven cavity. *Physical Review A 72*, 3 (2005), 033815.

[2] Aharonov, Dorit, Ambainis, Andris, Kempe, Julia, and Vazirani, Umesh. Quantum walks on graphs. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2001), STOC '01, ACM, pp. 50–59.

[3] Aharonov, Yakir, Davidovich, Luiz, and Zagury, Nicim. Quantum random walks. *Physical Review A 48*, 2 (1993), 1687.

[4] Ahmad, Rashid, Sajjad, Uzma, and Sajid, Muhammad. One-dimensional quantum walks with a position-dependent coin. *arXiv preprint arXiv:1902.10988* (2019).

[5] Altaisky, MV. Quantum neural network. *arXiv preprint quant-ph/0107012* (2001).

[6] Ambainis, Andris. Quantum walks and their algorithmic applications. *International Journal of Quantum Information 1*, 04 (2003), 507–518.

[7] Ambainis, Andris, Bach, Eric, Nayak, Ashwin, Vishwanath, Ashvin, and Watrous, John. One-dimensional quantum walks. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2001), STOC '01, ACM, pp. 37–49.

[8] Arjovsky, Martin, Shah, Amar, and Bengio, Yoshua. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning* (2016), pp. 1120–1128.

[9] Atwood, James, and Towsley, Don. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., Red Hook, NY, USA, 2016, pp. 1993–2001.

[10] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[11] Bai, Lu, Hancock, Edwin R, Torsello, Andrea, and Rossi, Luca. A quantum jensen-shannon graph kernel using the continuous-time quantum walk. In *International Workshop on Graph-Based Representations in Pattern Recognition* (Berlin / Heidelberg, Germany, 2013), Springer, pp. 121–131.

[12] Bai, Lu, Rossi, Luca, Cui, Lixin, Zhang, Zhihong, Ren, Peng, Bai, Xiao, and Hancock, Edwin. Quantum kernels for unattributed graphs using discrete-time quantum walks. *Pattern Recognition Letters 87* (2017), 96–103.

[13] Bai, Lu, Rossi, Luca, Torsello, Andrea, and Hancock, Edwin R. A quantum jensen–shannon graph kernel for unattributed graphs. *Pattern Recognition 48*, 2 (2015), 344–355.

[14] Balazevic, Ivana, Allen, Carl, and Hospedales, Timothy. Multi-relational poincaré graph embeddings. In *Advances in Neural Information Processing Systems* (2019), pp. 4463–4473.

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

[16] Biamonte, Jacob, Wittek, Peter, Pancotti, Nicola, Rebentrost, Patrick, Wiebe, Nathan, and Lloyd, Seth. Quantum machine learning. *Nature 549*, 7671 (2017), 195.

[17] Blum, L. C., and Reymond, J.-L. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc. 131* (2009), 8732.

[18] Bojchevski, Aleksandar, and Günnemann, Stephan. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).

[19] Borgwardt, K. M., and Kriegel, H. P. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)* (Houston, TX, USA, Nov 2005), IEEE, pp. 8 pp.–.

[20] Borgwardt, Karsten M., Ong, Cheng Soon, Schönauer, Stefan, Vishwanathan, S. V. N., Smola, Alex J., and Kriegel, Hans-Peter. Protein function prediction via graph kernels. *Bioinformatics 21*, suppl_1 (2005), i47–i56.

[21] Brand, Matthew. Fast online svd revisions for lightweight recommender systems. In *Proceedings of the 2003 SIAM international conference on data mining* (2003), SIAM, pp. 37–46.

[22] Brandes, Ulrik. A faster algorithm for betweenness centrality. *Journal of mathematical sociology 25*, 2 (2001), 163–177.

[23] Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and LeCun, Yann. Spectral networks and locally connected networks on graphs. In *International conference on learning representations (ICLR)* (Amherst, MA, USA, 2014), OpenReview.net.

[24] Cai, Deng, and Lam, Wai. Graph transformer for graph-to-sequence learning. In *AAAI* (2020), pp. 7464–7471.

[25] Chami, Ines, Ying, Zhitao, Ré, Christopher, and Leskovec, Jure. Hyperbolic graph convolutional neural networks. In *Advances in neural information processing systems* (2019), pp. 4868–4879.

[26] Chiang, Chen-Fu, Nagaj, Daniel, and Wocjan, Pawel. Efficient circuits for quantum walks. *Quantum Info. Comput. 10*, 5 (May 2010), 420–434.

[27] Childs, Andrew M. Universal computation by quantum walk. *Physical review letters 102*, 18 (2009), 180501.

[28] Coifman, Ronald R, and Lafon, Stéphane. Diffusion maps. *Applied and computational harmonic analysis 21*, 1 (2006), 5–30.

[29] Debnath, Asim Kumar, Lopez de Compadre, Rosa L, Debnath, Gargi, Shusterman, Alan J, and Hansch, Corwin. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry 34*, 2 (1991), 786–797.

[30] Defferrard, Michaël, Bresson, Xavier, and Vandergheynst, Pierre. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., Red Hook, NY, USA, 2016, pp. 3844–3852.

[31] Deutsch, Shay, Ortega, Antonio, and Medioni, Gerard. Manifold denoising based on spectral graph wavelets. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), IEEE, pp. 4673–4677.

[32] Donahue, Jeff, Jia, Yangqing, Vinyals, Oriol, Hoffman, Judy, Zhang, Ning, Tzeng, Eric, and Darrell, Trevor. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning* (2014), pp. 647–655.

[33] Dunjko, Vedran, and Briegel, Hans J. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics 81*, 7 (2018), 074001.

[34] Everingham, Mark, Van Gool, Luc, Williams, Christopher KI, Winn, John, and Zisserman, Andrew. The pascal visual object classes (voc) challenge. *International journal of computer vision 88*, 2 (2010), 303–338.

[35] Fang, Chen, Xu, Ye, and Rockmore, Daniel N. Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias. In *Proceedings of the IEEE International Conference on Computer Vision* (2013), pp. 1657–1664.

[36] Farhi, Edward, and Gutmann, Sam. Quantum computation and decision trees. *Physical Review A 58*, 2 (1998), 915.

[37] Gilmer, Justin, Schoenholz, Samuel S., Riley, Patrick F., Vinyals, Oriol, and Dahl, George E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning* (International Convention Centre, Sydney, Australia, 06–11 Aug 2017), Doina Precup and Yee Whye Teh, Eds., vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 1263–1272.

[38] Gong, Boqing, Shi, Yuan, Sha, Fei, and Grauman, Kristen. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), IEEE, pp. 2066–2073.

[39] Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* (Montreal, Que., Canada, July 2005), vol. 2, IEEE, pp. 729–734 vol. 2.

[40] Griffin, Gregory, Holub, Alex, and Perona, Pietro. Caltech-256 object category dataset.

[41] Grover, Aditya, and Leskovec, Jure. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2016), KDD '16, ACM, pp. 855–864.

[42] Gupta, Sanjay, and Zia, RKP. Quantum neural networks. *Journal of Computer and System Sciences 63*, 3 (2001), 355–383.

[43] Ham, Jihun, Lee, Daniel D, Saul, Lawrence K, et al. Semisupervised alignment of manifolds. In *AISTATS* (2005), vol. 120, Citeseer, p. 27.

[44] Jing, Li, Shen, Yichen, Dubcek, Tena, Peurifoy, John, Skirlo, Scott, LeCun, Yann, Tegmark, Max, and Soljačić, Marin. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, Australia, 2017), ICML'17, JMLR.org, pp. 1733–1741.

[45] Joo, Jaewoo, Knight, Peter L, and Pachos, Jiannis K. Single atom quantum walk with 1d optical superlattices. *Journal of Modern Optics 54*, 11 (2007), 1627–1638.

[46] Jordan, Stephen P, and Wocjan, Pawel. Efficient quantum circuits for arbitrary sparse unitaries. *Physical Review A 80*, 6 (2009), 062301.

[47] Kendon, Viv. Quantum walks on general graphs. *International Journal of Quantum Information 4*, 05 (2006), 791–805.

[48] Khosla, Megha, Leonhardt, Jurek, Nejdl, Wolfgang, and Anand, Avishek. Node representation learning for directed graphs. *ArXiv abs/1810.09176* (2018).

[49] Kipf, Thomas N., and Welling, Max. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017* (Amherst, MA, USA, 2017), OpenReview.net.

[50] Krioukov, Dmitri, Papadopoulos, Fragkiskos, Kitsak, Maksim, Vahdat, Amin, and Boguná, Marián. Hyperbolic geometry of complex networks. *Physical Review E 82*, 3 (2010), 036106.

[51] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., Red Hook, NY, USA, 2012, pp. 1097–1105.

[52] Lee, Junhyun, Lee, Inyeop, and Kang, Jaewoo. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082* (2019).

[53] Leskovec, Jure, Kleinberg, Jon, and Faloutsos, Christos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD) 1*, 1 (2007), 2.

[54] Liu, Weiyang, Wen, Yandong, Yu, Zhiding, Li, Ming, Raj, Bhiksha, and Song, Le. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 212–220.

[55] Loke, T, and Wang, JB. An efficient quantum circuit analyser on qubits and qudits. *Computer Physics Communications 182*, 10 (2011), 2285–2294.

[56] Loke, T, and Wang, JB. Efficient circuit implementation of quantum walks on non-degree-regular graphs. *Physical Review A 86*, 4 (2012), 042338.

[57] Long, Mingsheng, Wang, Jianmin, Ding, Guiguang, Sun, Jiaguang, and Yu, Philip S. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision* (2013), pp. 2200–2207.

[58] Lovett, Neil B, Cooper, Sally, Everitt, Matthew, Trevers, Matthew, and Kendon, Viv. Universal quantum computation using the discrete-time quantum walk. *Physical Review A 81*, 4 (2010), 042330.

[59] Manouchehri, K, and Wang, JB. Quantum walks in an array of quantum dots. *Journal of Physics A: Mathematical and Theoretical 41*, 6 (2008), 065304.

[60] Manouchehri, K, and Wang, JB. Quantum random walks without walking. *Physical Review A 80*, 6 (2009), 060304.

[61] Meng, Yu, Huang, Jiaxin, Wang, Guangyuan, Zhang, Chao, Zhuang, Honglei, Kaplan, Lance, and Han, Jiawei. Spherical text embedding. In *Advances in Neural Information Processing Systems* (2019), pp. 8208–8217.

[62] Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings* (2013).

[63] Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.

[64] Mnih, Andriy, and Hinton, Geoffrey E. A scalable hierarchical distributed language model. In *Advances in neural information processing systems* (2009), pp. 1081–1088.

[65] Nayak, Ashwin, and Vishwanath, Ashvin. Quantum walk on the line. *arXiv preprint quant-ph/0010117* (2000).

[66] Ortega, Antonio, Frossard, Pascal, Kovačević, Jelena, Moura, José MF, and Vandergheynst, Pierre. Graph signal processing. *arXiv preprint arXiv:1712.00468* (2017).

[67] Ou, Mingdong, Cui, Peng, Pei, Jian, Zhang, Ziwei, and Zhu, Wenwu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2016), KDD '16, ACM, pp. 1105–1114.

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

[69] Qiang, Xiaogang, Yang, Xuejun, Wu, Junjie, and Zhu, Xuan. An enhanced classical approach to graph isomorphism using continuous-time quantum walk. *Journal of Physics A: Mathematical and Theoretical 45*, 4 (2012), 045305.

[70] Richardson, Matthew, Agrawal, Rakesh, and Domingos, Pedro. Trust management for the semantic web. In *International semantic Web conference* (2003), Springer, pp. 351–368.

[71] Rohde, Peter P, Schreiber, Andreas, Štefaňák, Martin, Jex, Igor, and Silberhorn, Christine. Multi-walker discrete time quantum walks on arbitrary graphs, their properties and their photonic implementation. *New Journal of Physics 13*, 1 (2011), 013001.

[72] Rossi, Luca, Torsello, Andrea, and Hancock, Edwin R. *A Continuous-Time Quantum Walk Kernel for Unattributed Graphs*, vol. 7877 of *Lecture Notes in Computer Science*. Springer, Berlin / Heidelberg, Germany, 2013, pp. 101–110.

[73] Rossi, Luca, Torsello, Andrea, and Hancock, Edwin R. Measuring graph similarity through continuous-time quantum walks and the quantum jensen-shannon divergence. *Physical Review E 91*, 2 (2015), 022815.

[74] Rossi, Matteo AC, Benedetti, Claudia, Borrelli, Massimo, Maniscalco, Sabrina, and Paris, Matteo GA. Continuous-time quantum walks on spatially correlated noisy lattices. *Physical Review A 96*, 4 (2017), 040301.

[75] Rupp, M., Tkatchenko, A., Müller, K.-R., and von Lilienfeld, O. A. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters 108* (2012), 058301.

[76] Ryan, Colm A, Laforest, Martin, Boileau, Jean-Christian, and Laflamme, Raymond. Experimental implementation of a discrete-time quantum random walk on an nmr quantum-information processor. *Physical Review A 72*, 6 (2005), 062317.

[77] Saenko, Kate, Kulis, Brian, Fritz, Mario, and Darrell, Trevor. Adapting visual category models to new domains. In *European conference on computer vision* (2010), Springer, pp. 213–226.

[78] Sarkar, Rik. Low distortion delaunay embedding of trees in hyperbolic plane. In *International Symposium on Graph Drawing* (2011), Springer, pp. 355–366.

[79] Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, and Monfardini, Gabriele. The graph neural network model. *IEEE Transactions on Neural Networks 20*, 1 (2009), 61–80.

[80] Schomburg, Ida, Chang, Antje, Ebeling, Christian, Gremse, Marion, Heldt, Christian, Huhn, Gregor, and Schomburg, Dietmar. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research 32*, suppl_1 (2004), D431–D433.

[81] Shenvi, Neil, Kempe, Julia, and Whaley, K Birgitta. Quantum random-walk search algorithm. *Physical Review A 67*, 5 (2003), 052307.

[82] Shervashidze, Nino, Schweitzer, Pascal, Leeuwen, Erik Jan van, Mehlhorn, Kurt, and Borgwardt, Karsten M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research 12*, Sep (2011), 2539–2561.

[83] Shuman, David I, Narang, Sunil K, Frossard, Pascal, Ortega, Antonio, and Vandergheynst, Pierre. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine 30*, 3 (2013), 83–98.

[84] Šubelj, Lovro, and Bajec, Marko. Model of complex networks based on citation dynamics. In *Proceedings of the 22nd international conference on World Wide Web* (2013), ACM, pp. 527–530.

[85] Sun, Baochen, Feng, Jiashi, and Saenko, Kate. Return of frustratingly easy domain adaptation. In *Thirtieth AAAI Conference on Artificial Intelligence* (2016).

[86] Tang, Jian, Qu, Meng, Wang, Mingzhe, Zhang, Ming, Yan, Jun, and Mei, Qiaozhu. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web* (2015), International World Wide Web Conferences Steering Committee, pp. 1067–1077.

[87] Tay, Yi, Luu, Anh Tuan, and Hui, Siu Cheung. Hermitian co-attention networks for text matching in asymmetrical domains. In *IJCAI* (2018), pp. 4425–4431.

[88] Tenenbaum, Joshua B, De Silva, Vin, and Langford, John C. A global geometric framework for nonlinear dimensionality reduction. *science 290*, 5500 (2000), 2319–2323.

[89] Travaglione, Ben C, and Milburn, Gerald J. Implementing the quantum random walk. *Physical Review A 65*, 3 (2002), 032310.

[90] Trouillon, Théo, Welbl, Johannes, Riedel, Sebastian, Gaussier, Éric, and Bouchard, Guillaume. Complex embeddings for simple link prediction. International Conference on Machine Learning (ICML).

[91] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, and Polosukhin, Illia. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[92] Velickovic, Petar, Cucurull, Guillem, Casanova, Arantxa, Romero, Adriana, Lio, Pietro, and Bengio, Yoshua. Graph attention networks. In *Proceedings of the International Conference on Learning Representations (ICLR)* (Amherst, MA, USA, 2017), OpenReview.net.

[93] Vilnis, Luke, and McCallum, Andrew. Word representations via gaussian embedding. *arXiv preprint arXiv:1412.6623* (2014).

[94] Vinyals, Oriol, Bengio, Samy, and Kudlur, Manjunath. Order matters: Sequence to sequence for sets. In *4th International Conference on Learning Representations, ICLR 2016* (Amherst, MA, USA, 2016), OpenReview.net.

[95] Wale, Nikil, Watson, Ian A, and Karypis, George. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems 14*, 3 (2008), 347–375.

[96] Wang, Chang, Krafft, Peter, Mahadevan, Sridhar, Ma, Y, and Fu, Y. Manifold alignment. In *Manifold Learning: Theory and Applications*. CRC Press, 2011.

[97] Wang, Chang, and Mahadevan, Sridhar. Manifold alignment using procrustes analysis. In *Proceedings of the 25th international conference on Machine learning* (2008), pp. 1120–1127.

[98] Wang, Chang, and Mahadevan, Sridhar. A general framework for manifold alignment. In *2009 AAAI Fall Symposium Series* (2009).

[99] Wang, Chang, and Mahadevan, Sridhar. Heterogeneous domain adaptation using manifold alignment. In *Twenty-second international joint conference on artificial intelligence* (2011).

[100] Wang, Jindong, Feng, Wenjie, Chen, Yiqiang, Yu, Han, Huang, Meiyu, and Yu, Philip S. Visual domain adaptation with manifold embedded distribution alignment. In *Proceedings of the 26th ACM international conference on Multimedia* (2018), pp. 402–410.

[101] Wang, Lei, Huang, Yuchun, Hou, Yaolin, Zhang, Shenman, and Shan, Jie. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 10296–10305.

[102] Wang, Xiao, Ji, Houye, Shi, Chuan, Wang, Bai, Ye, Yanfang, Cui, Peng, and Yu, Philip S. Heterogeneous graph attention network. In *The World Wide Web Conference* (2019), pp. 2022–2032.

[103] Williams, CN, Vose, RS, Easterling, DR, and Menne, MJ. United states historical climatology network daily temperature, precipitation, and snow data. *ORNL/CDIAC-118, NDP-070. Available on-line [http://cdiac.ornl.gov/epubs/ndp/ushcn/usa.html] from the Carbon Dioxide Information Analysis Center, Oak Ridge National Laboratory, USA* (2006).

[104] Yao, Ting, Pan, Yingwei, Ngo, Chong-Wah, Li, Houqiang, and Mei, Tao. Semi-supervised domain adaptation with subspace learning for visual recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (2015), pp. 2142–2150.

[105] Zhang, Pei, Ren, Xi-Feng, Zou, Xu-Bo, Liu, Bi-Heng, Huang, Yun-Feng, and Guo, Guang-Can. Demonstration of one-dimensional quantum random walks using orbital angular momentum of photons. *Physical Review A 75*, 5 (2007), 052310.

[106] Zhou, Chang, Liu, Yuqiong, Liu, Xiaofei, Liu, Zhongyi, and Gao, Jun. Scalable graph embedding for asymmetric proximity. In *Thirty-First AAAI Conference on Artificial Intelligence* (2017).