RESEARCH

Quantum Walk Neural Networks with Feature Dependent Coins

Stefan Dernbach[1*], Arman Mohseni-Kabir[1], Siddharth Pal[2], Miles Gepner[1] and Don Towsley[1]

*Correspondence:
dernbach@cs.umass.edu
[1]University of Massachusetts, College of Information and Computer Sciences, Amherst, MA, 01003 USA
Full list of author information is available at the end of the article

Abstract

Recent neural networks designed to operate on graph-structured data have proven effective in many domains. These graph neural networks often diffuse information using the spatial structure of the graph. We propose a quantum walk neural network that learns a diffusion operation that is not only dependent on the geometry of the graph but also on the features of the nodes and the learning task. A quantum walk neural network is based on learning the coin operators that determine the behavior of quantum random walks, the quantum parallel to classical random walks. We demonstrate the effectiveness of our method on multiple classification and regression tasks at both node and graph levels.

Keywords: graph neural networks; random walks; quantum random walks

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 [1] to the domain of graph-structured data [2, 3, 4, 5, 6, 7]. Following the success of quantum kernels on graph-structured data [8, 9, 10], 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 structure that can be applied to graph data. Quantum random walks differ from classical random walks through additional operators (called coins) that can be tuned to affect the outcome of the walk.

In [11] we introduced a quantum walk neural network (QWNN) for the purpose of learning a task-specific random walk on a graph. When dealing with learning problems involving multiple graphs, the original QWNN formulation suffered from a requirement that all nodes across all graphs share the same coin matrix. This paper improves upon our original network architecture by replacing the single coin matrix with a bank that learns a function to produce different coin matrices at each node in every graph. This function allows the behavior of the quantum walk to vary spatially across the graph even when dealing with multi-graph problems. Additionally, this function produces the coins based on neighboring node features so that even for structurally identical graphs, a different walk is produced if the node features change. We also improve the neural network architecture in this work. In the new architecture, each step of the quantum walk produces its own set of diffused features. The aggregated set of features, spanning the length of the walk, are passed to successive layers in the neural network. Finally, the previous work

produced results that were dependent upon the ordering of the nodes. This work provides a QWNN architecture that is invariant to node ordering.

The rest of this paper is organized as follows. Section 2 describes the background literature on graph neural network techniques in further detail. The setting of quantum walks on graphs is described in Section 3, followed by a formal description of the proposed quantum walk based neural network implementation in Section 4. Experimental results on node and graph regression, and graph classification tasks are presented in Section 5, followed by a discussion of the techniques' limitations in Section 6 and concluding remarks in Section 7.

2 Related Work

Gupta and Zia [12] and Altaivsky [13] among other researchers proposed quantum versions of artificial neural networks; See Biamonte et al. and Dunjko et al. [14, 15] for an overview of the emerging field of quantum machine learning. While not much work exists on quantum machine learning techniques for graph-structured data, in recent years, new neural network techniques that operate on graph-structured data have become prominent. Gori et al. [4] followed by Scarselli et al. [6] proposed recursive neural network architectures to deal with graph-structured data, instead of the then prevalent approach of transforming the graph data into a domain that could be handled by conventional machine learning algorithms. Bruna et al. [3] studied the generalization of convolutional neural networks (CNNs) to graph signals through two approaches, one based upon hierarchical clustering of the domain, and another based on the spectrum of the graph Laplacian. Subsequently, Defferrard et al. [16] proposed to approximate the convolutional filters on graphs through their fast localized versions.

Along with the spectral approaches described above, a number of spatial approaches have been proposed that relied on random walks to extract and learn information from the graph. For comparison, we detail several modern approaches. Atwood and Towsley [2] propose a spatial convolutional method that performs random walks on the graph and combines information from spatially close neighbors. Given a graph $G = \{V, E\}$ and a feature matrix $\mathbf{X}$, their approach, Diffusion Convolutional Neural Networks (DCNN) use powers of the transition matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ to diffuse information across the graph, where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal degree matrix such that $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. The k^{th} power of the transition matrix, $\mathbf{P}^k$, 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

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

where $\mathbf{P}^*$ is the stacked tensor of powers of transition matrices, 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).

The second approach of interest due to Kipf and Welling [5], was proposed to tackle semi-supervised learning on graph-structured data through a CNN architecture that uses localized approximation of spectral graph convolutions. The proposed

technique, the Graph Convolutional Neural Network (GCN) simplified the original spectral-based frameworks of Bruna et al. [3] and Defferrard et al. [16] 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:

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

where, again, $\mathbf{W}$ 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. One such example, Graph Attention Networks (GATs) [7], is inspired by the attention mechanisms commonly applied in natural language processing for sequence-based tasks. The neural network architecture 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

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

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

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

where $\mathcal{N}(v_i)$ is the neighbor set of node v_i.

Our proposed quantum walk neural network is graph neural network architecture based on discrete quantum walks. Various researchers have worked on quantum walks on graphs – Ambainis et al. [17] studied quantum variants of random walks on one-dimensional lattices; Farhi and Gutmann [18] 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. [19] generalized quantum walks to arbitrary graphs. Subsequently, Rohde et al. [20] 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. [8, 9, 10] introduced novel graph kernels based on the evolution of quantum walks on graphs. They defined the similarity between 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 [21, 22], 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.

3 Graph Quantum Walks

Motivated by classical random walks, quantum walks were introduced by Aharonov et al. in 1993 [23]. Unlike the stochastic evolution of a classical random walk, a quantum walk evolves according to unitary process. The behavior of a quantum walk is fundamentally different from a classical walk since in a quantum walk there is interference between different trajectories of the walk. Two kinds of quantum walks have been introduced in the literature; namely, continuous time quantum walks [18, 24] and discrete time quantum walks [25]. Quantum walks have recently received much attention because they have been shown to be a universal model for quantum computation [26]. In addition, they have numerous applications in quantum information science such as database search [27], graph isomorphism [28], network analysis and navigation, and quantum simulation.

Discrete time quantum walks were initially introduced on simple regular lattices [29] and then extended to general graphs [30]. In this paper, we use the formulation of discrete time quantum walks as outlined in [31, 30]. Given an undirected graph $G = (V, E)$, we introduce a position Hilbert space $\mathcal{H}_P$ that captures the superposition over various positions, i.e., 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 can now be written as a linear combination of position state basis vectors,

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

where $\{\alpha_v,\ v \in V\}$ are coefficients satisfying the unit L_2-norm condition $\sum_v \|\alpha_v\|^2 = 1$, with the understanding that $\|\alpha_v\|^2$ is the probability of finding the walker at vertex v.

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 can now be written as a linear combination of coin state basis vectors,

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

where $\{\beta_{v,i},\ i \in 1, \ldots, d\}$ are coefficients satisfying the unit L_2-norm condition $\sum_i |\beta_{v,i}|^2 = 1$. 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 can be written as $\mathcal{H}_W = \mathcal{H}_P \otimes \mathcal{H}_C$, which is the tensor product of the two aforementioned Hilbert spaces.

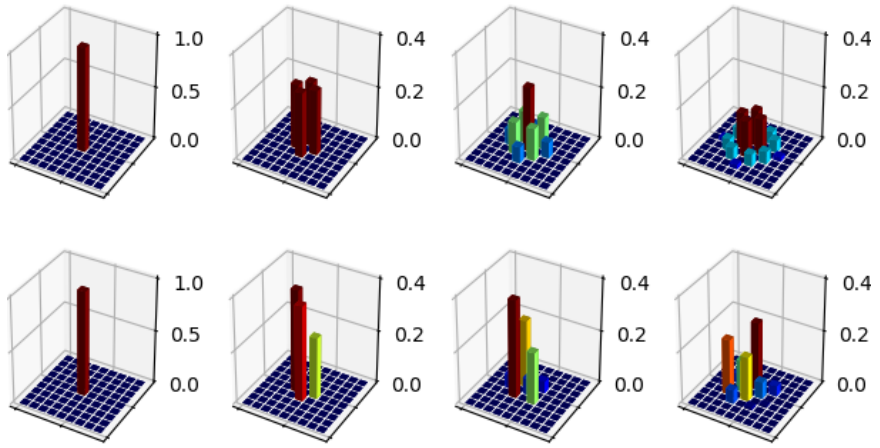


Figure 1 Classical and Quantum Walk Distributions The probability distribution of a classical random walk (Top) and a quantum random walk (Bottom) across the nodes of a lattice graph over four steps from left to right.

Time-evolution of discrete time quantum walk over graph G is governed by two unitary operators, namely, coin and shift operator. 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 we first apply a unitary coin operator $\boldsymbol{C}$ which transforms the coin state of the walker at each vertex,

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

$\boldsymbol{I}$ denotes the identity operator. After transforming the coin (spin) states, we apply a unitary shift operator $\boldsymbol{S}$ which 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 we swap the coefficient corresponding to the basis state $\hat{\mathbf{e}}_v^{(p)} \otimes \hat{\mathbf{e}}_i^{(c)}$ with that of the basis state $\hat{\mathbf{e}}_u^{(p)} \otimes \hat{\mathbf{e}}_j^{(c)}$. $\boldsymbol{S}$ operates on both coin and position Hilbert spaces,

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

In shorthand notation, the unitary evolution of the walk is governed by the operator $\boldsymbol{U} = \boldsymbol{S}\,(\boldsymbol{I} \otimes \boldsymbol{C})$. Applying $\boldsymbol{U}$ successively evolves the state of the quantum walk through time.

The choice of coin operators as well as the initial superposition of the walker control how this non-classical diffusion process evolves over the graph and therefore provides the deep learning technique additional degrees of freedom for controlling the flow of information over the graph. Figure 1 shows how the diffusion behavior of a classical random walk differs from a discrete time quantum walk with a single coin. Ahmad et al. [32] recently showed that for a discrete quantum walk on a line, having a position-dependent coin can lead to quantitatively different diffusion behaviors with different choices of coin operators. Our work uses the setting of multiple non-interacting quantum walks acting on arbitrary graphs, as introduced in Rhode et al. [20], to learn patterns in graph data. Calculating a separate quantum walk originating from each node in the graph allows us to construct a diffusion

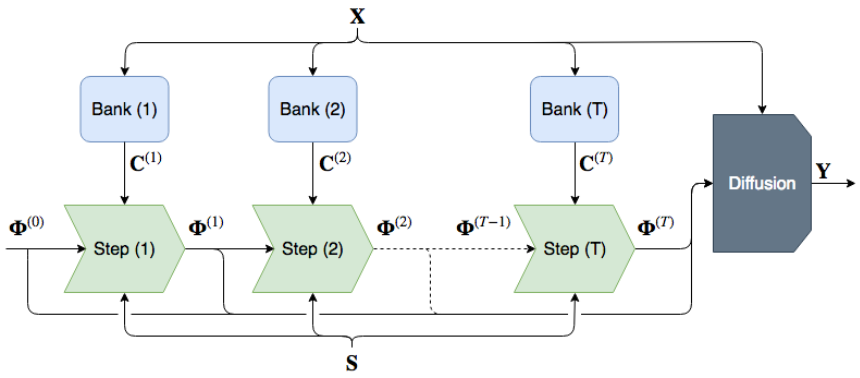


Figure 2 Quantum walk neural network diagram. The feature matrix X is used by the banks to produce the coin matrices $\mathbf{C}$ used in each step layer as well in the final diffusion process. The superposition $\mathbf{\Phi}$ evolves after each step of the walk. The diffusion layer diffuses X using each superposition $\{\mathbf{\Phi}^{(0)}, \mathbf{\Phi}^{(1)}, ...\mathbf{\Phi}^{(T)}\}$ and concatenates the results to produce the output $\mathbf{Y}$.

matrix where each entry gives the relationship between the starting and ending nodes of a walk. This matrix works like its classical counterpart, a random walk matrix, used in DCNN [2].

3.1 Physical implementation of discrete quantum walks

Over the past few years, there have been several proposals for the physical implementation of quantum walks. Quantum walks are unitary process 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 manifest the position and coin Hilbert spaces of the quantum walk. These quantum simulation based methods have been proposed using classical and quantum optics [33], nuclear magnetic resonance [34], ion traps [35], cavity QED [36], optical lattices [37], and Bose Einstein condensate [38] as well as quantum dots [39] to implement the quantum walk.

Circuit implementation of quantum walks has also been proposed. While most of these implementations focus on graphs that have a very high degree of symmetry [40] or very sparse graphs [41, 42], there is some recent work on circuit implementations on non-degree regular graphs [43].

A central question in implementing quantum walks on graphs is how to scale the physical system to achieve the complexity required for simulating large graphs. Rohde et al. [44] showed that exponentially larger graphs can be constructed using quantum entanglement as a resource for creating very large Hilbert spaces. They use multiple entangled walkers to simulate a quantum walk on a virtual graph of chosen dimensions. However, this approach has its own limitations and arbitrary graphs can not be built with this method.

4 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 neighors. 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 2, learns a quantum walk on a graph by means of backpropagating gradient updates to the coin operators used in the walk. The learned walk is then used to diffuse a signal over the graph.

In [11], the quantum walk neural network evolves a walk using a single coin matrix, $\mathbf{C}$, 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 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: $\mathbf{Y} = \mathbf{PX}$. The coin matrix is learned through backpropagating the gradient of a loss function. In this paper we replace 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 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.

4.1 Bank

The Coin operators modify the spin state of the walk and are thus the primary levers 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 the 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 operators from node features $f : X \to \mathbb{C}^{d\times d}$ where d is the maximum degree of the graph. In general, f is 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 f whose ranges are subsets of unitary matrices. A common practice is to use combinations of low dimensional rotation matrices [45, 46]. This was the model used for the coin operators in previous QWNNs [11].

In our work, we focus on elementary unitary matrices. These matrices are 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 vector. These matrices can be computed efficiently in the forward pass of the neural network and their gradients can similarly be computed 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 elementary unitary matrices. The QWNN bank produces the coin matrix for node v_i according the following:

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

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

The first function:

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

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 result of the function. We mitigate this effect by using a heuristic node ordering based on node centrality that we outline in Section 4.4.

The second function:

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

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 $f_k(v_i)$). This in turn allows the entire neural network to be invariant to node ordering.

4.2 Walk

For a graph with N vertices, the QWNN processes N separate, non-interacting walks in parallel – one walk originating 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 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 $(\mathbf{\Phi}^{(0)}_{ii\cdot})^H \mathbf{\Phi}^{(0)}_{ii\cdot} = 1$ and $\forall i{\neq}j : \mathbf{\Phi}^{(0)}_{ijk} = 0$. The value of $\mathbf{\Phi}^{(t)}_{ijk}$ 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, as well as a shift tensor $\mathbf{S} \in \mathbb{Z}_2^{N\times d\times N\times d}$ that encodes the graph structure: $S_{ujvi} = 1$ iff u is the the i^{th} neighbor of v and v is the j^{th} neighbor of u. The superposition evolves according to:

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

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

$$\mathbf{\Phi}^{(t+1)}_{wuj} = \left(\mathbf{\Phi}^{(t)}_{v}\mathbf{C}^{(t)}_{v}\right)_{wi}$$
$$\mathbf{\Phi}^{(t+1)}_{wvi} = \left(\mathbf{\Phi}^{(t)}_{u}\mathbf{C}^{(t)}_{u}\right)_{wj}$$

The output $\mathbf{\Phi}^{(t+1)}$ is fed into the next quantum step layer (if there is one) and the diffusion layer.

4.3 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 can then be 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 1.

Algorithm 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**
    $\mathbf{v}_i^{(t)} = \mathbf{W}^T vec\left(\mathbf{X}_{\mathcal{N}(v_i)}\right) + \mathbf{b}$ or $\mathbf{v}_i^{(t)} = \mathbf{X}_{\mathcal{N}(i)}\mathbf{W}\mathbf{X}_i^T$
    $\mathbf{C}_i^{(t)} = \mathbf{I} - 2\mathbf{v}_i^{(t)}(\mathbf{v}_i^{(t)})^T/((\mathbf{v}_i^{(t)})^T\mathbf{v}_i^{(t)})$
    $\mathbf{\Phi}^{(t)}_{\cdot i\cdot} \leftarrow \mathbf{\Phi}^{(t-1)}_{\cdot i\cdot} \cdot \mathbf{C}^{(t)}_{i\cdot\cdot}$
  $\mathbf{\Phi}^{(t)} \leftarrow \mathbf{\Phi}^{(t)}\cdot\cdot\mathbf{S}$ (i.e., $\mathbf{\Phi}^{(t)}_{wuj} = \sum_v \sum_i \mathbf{\Phi}^{(t)}_{wvi}\mathbf{S}_{viuj}$)
  $\mathbf{P}^{(t)} \leftarrow \sum_k \mathbf{\Phi}^{(t)}_{\cdot\cdot k} \odot \mathbf{\Phi}^{(t)}_{\cdot\cdot k}$
  $\mathbf{Y}^{(t)} \leftarrow h(\mathbf{P}^{(t)}\mathbf{X} + \mathbf{b}^{(t)})$
return : $\{\mathbf{Y}^{(0)}, \mathbf{Y}^{(1)}, ..., \mathbf{Y}^{(T)}\}$

4.4 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. We address this issue using a centrality score. The betweenness centrality [47] of node v_i is calculated as:

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

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.

5 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) [2], graph convolution networks (GCN) [5], and graph attention networks (GAT) [7].

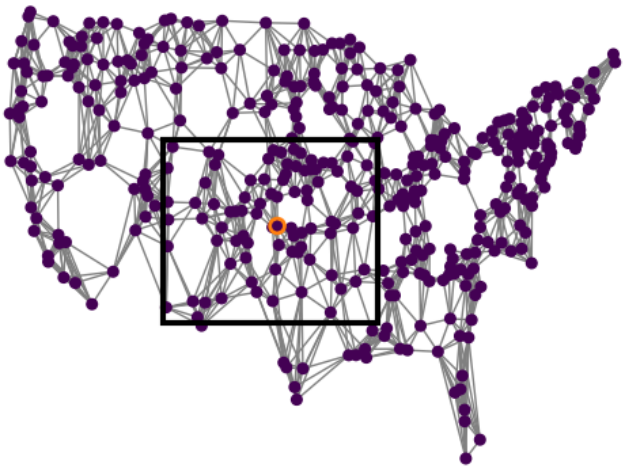
For graph level experiments, we employ a set2vec layer [48] as an intermediary between the graph layers and standard neural network feed forward layers. Set2vec has proved effective in other graph neural networks [49] as it is a permutation invariant function that converts a set of node features into a fixed length vector.

5.1 Node Regression

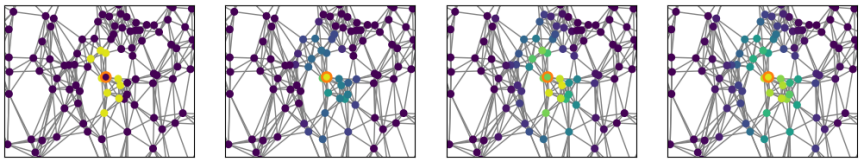
In the node regression task, daily temperatures are recorded across 409 locations in the United States during the year 2009[50]. The goal of the task is to use a day's temperature reading to predict the next day's temperatures. A nearest neighbors graph (Figure 3a) is constructed using longitudes and latitudes of the recording locations by connecting each station to its closest neighbors. Adding edges to each station's 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 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 which 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.

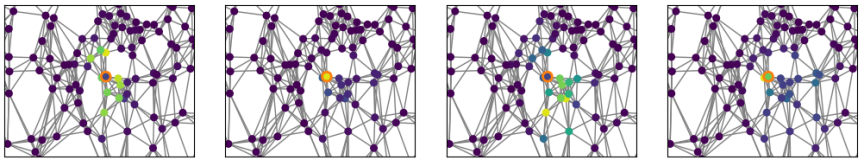
We use this experiment to provide a visualization for the learned quantum walk. Figure 3b and 3c shows the evolution of a classical random walk and the learned quantum random walk originating from the highlighted node respectively. At each step, warmer color nodes correspond to nodes with higher superposition amplitudes. Initially, the quantum walk appears to diffuse outward in a symmetrical manner



(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 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.

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.

5.2 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 [51], Mutag [52], and NCI1 [53]. Enzymes is a set of 600 molecules extracted from the Brenda database [54]. 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 are connected if certain conditions are satisfied, with each node bearing a type label, and its physical and chemical information. The task is to classify each enzyme into one of six classes. Mutag is a dataset

Table 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

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 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. In Mutag, the walk length is reduced to 4 and the hidden layer 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 [55] and shortest path (SP) kernels [56], using the results given in [55].

Classification accuracies are reported in Table 2. 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 Enzyme 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.

5.3 Graph Regression

Our graph regression task uses the QM7 dataset [57, 58], a collection of 7165 molecules each containing up to 23 atoms. The geometries of these molecules are stored in Coulomb matrix format defined as

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

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.

Table 2 Graph Classification Datasets Summary and Results

	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
	Classification Accuracy ± STD		
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
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
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

Table 3 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

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 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 mean absolute prediction error (MAE) are reported for each network in Table 3. QWNNs demonstrate a marked improvement over other methods in this task.

6 Limitations

Storing the superposition of a single walker requires $O(Nd)$ space, with N the number of nodes in the graph, and d the max degree of the graph. To calculate a complete diffusion matrix requires that a separate walker begin at every node, increasing the space requirement to $O(N^2 d)$ 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 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 QWNN and thus the overall output of the network. This can occur when two otherwise distinguishable nodes have the same betweenness centrality.

7 Concluding Remarks

Quantum walk neural networks provide a unique neural network approach to graph classification and regression problems. Unlike prior graph neural networks, QWNNs fully integrate the graph structure and the graph signal into the learning process. This allows QWNN to learn task dependent walks on complex graphs. The benefit of using the distributions produced by these walks as diffusion operators is especially clear in regression problems where QWNN demonstrate considerable improvement over other graph neural network approaches. This improvement is demonstrated at both the node and the graph level.

An added benefit of QWNN is that the learned walks provide a human understandable glimpse of the neural network determination of where information originating from each node is most beneficial in the graph. In the current work, each walker on the graph operates independently. A future research direction is to investigate learning multi-walker quantum walks on graphs. Reducing the number of independent walkers and allowing interactions can reduce the space complexity of the quantum walk layers.

Abbreviations

QWNN: Quantum Walk Neural Networks; CNN: Convolutional Neural Networks; DCNN: Diffusion Convolutional Neural Network; GCN: Graph Convolutional Neural Network; GAT: Graph Attention Network; RNN: Recursive Neural Network; RMSE: Root Mean Squared Prediction Error; STD: Standard Deviation; WL: Weisfeiler-Lehman; SP: Shortest Path; MAE: Mean Absolute Error; GPU: Graphics Processing Unit;

Availability of data and Material

The US Temperature dataset [50] was compiled from recordings prepared by the Carbon Dioxide Information Analysis Center and is available at http://cdiac.ornl.gov/epubs/ndp/ushcn/usa.html. The Mutag [52], Enzymes [51], and NCI1 [53] datasets are part of the benchmark datasets for graph kernels available at https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets. The QM7 dataset [57, 58] is available at http://quantum-machine.org/datasets/.

Competing interests

The authors declare that they have no competing interests.

Funding

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053 (the ARL Network Science CTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This document does not contain technology or technical data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations.

Author's contributions

SD worked on conceptualization, methodology, software writing, experiments, writing, and review and editing of the paper. AMK worked on conceptualization, writing, and review and editing of the paper. SP worked on conceptualization, writing, review and editing of the paper, and acquisition of funding for the research. MG helped with the methodology and worked on software. DT worked on conceptualization, review and editing, supervision of the research, acquisition of funding, and methodology.

Acknowledgements

Not applicable.

Author details

[1]University of Massachusetts, College of Information and Computer Sciences, Amherst, MA, 01003 USA.
[2]Raytheon BBN Technologies, Cambridge, MA, 02138 USA.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
2. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1993–2001 (2016)
3. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. In: International Conference on Learning Representations (ICLR) (2014)
4. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference On, vol. 2, pp. 729–734 (2005). IEEE
5. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
6. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009)
7. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: Proceedings of the International Conference on Learning Representations (ICLR) (2017)
8. Bai, L., Hancock, E.R., Torsello, A., Rossi, L.: A quantum jensen-shannon graph kernel using the continuous-time quantum walk. In: International Workshop on Graph-Based Representations in Pattern Recognition, pp. 121–131 (2013). Springer
9. Bai, L., Rossi, L., Cui, L., Zhang, Z., Ren, P., Bai, X., Hancock, E.: Quantum kernels for unattributed graphs using discrete-time quantum walks. Pattern Recognition Letters **87**, 96–103 (2017)
10. Bai, L., Rossi, L., Torsello, A., Hancock, E.R.: A quantum jensen–shannon graph kernel for unattributed graphs. Pattern Recognition **48**(2), 344–355 (2015)
11. Dernbach, S., Mohseni-Kabir, A., Pal, S., Towsley, D.: Quantum walk neural networks. In: Seventh International Conference on Complex Networks and Their Applications (2018)
12. Gupta, S., Zia, R.: Quantum neural networks. Journal of Computer and System Sciences **63**(3), 355–383 (2001)
13. Altaisky, M.: Quantum neural network. arXiv preprint quant-ph/0107012 (2001)
14. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**(7671), 195 (2017)
15. Dunjko, V., Briegel, H.J.: Machine learning\& artificial intelligence in the quantum domain. arXiv preprint arXiv:1709.02779 (2017)
16. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
17. Ambainis, A., Bach, E., Nayak, A., Vishwanath, A., Watrous, J.: One-dimensional quantum walks. In: Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing, pp. 37–49 (2001). ACM
18. Farhi, E., Gutmann, S.: Quantum computation and decision trees. Physical Review A **58**(2), 915 (1998)
19. Aharonov, D., Ambainis, A., Kempe, J., Vazirani, U.: Quantum walks on graphs. In: Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing, pp. 50–59 (2001). ACM
20. Rohde, P.P., Schreiber, A., Štefaňák, M., Jex, I., Silberhorn, C.: Multi-walker discrete time quantum walks on arbitrary graphs, their properties and their photonic implementation. New Journal of Physics **13**(1), 013001 (2011)
21. Rossi, L., Torsello, A., Hancock, E.R.: A continuous-time quantum walk kernel for unattributed graphs. In: International Workshop on Graph-Based Representations in Pattern Recognition, pp. 101–110 (2013). Springer
22. Rossi, L., Torsello, A., Hancock, E.R.: Measuring graph similarity through continuous-time quantum walks and the quantum jensen-shannon divergence. Physical Review E **91**(2), 022815 (2015)
23. Aharonov, Y., Davidovich, L., Zagury, N.: Quantum random walks. Physical Review A **48**(2), 1687 (1993)
24. Rossi, M.A., Benedetti, C., Borrelli, M., Maniscalco, S., Paris, M.G.: Continuous-time quantum walks on spatially correlated noisy lattices. Physical Review A **96**(4), 040301 (2017)
25. Lovett, N.B., Cooper, S., Everitt, M., Trevers, M., Kendon, V.: Universal quantum computation using the discrete-time quantum walk. Physical Review A **81**(4), 042330 (2010)
26. Childs, A.M.: Universal computation by quantum walk. Physical review letters **102**(18), 180501 (2009)
27. Shenvi, N., Kempe, J., Whaley, K.B.: Quantum random-walk search algorithm. Physical Review A **67**(5), 052307 (2003)
28. Qiang, X., Yang, X., Wu, J., Zhu, X.: An enhanced classical approach to graph isomorphism using continuous-time quantum walk. Journal of Physics A: Mathematical and Theoretical **45**(4), 045305 (2012)
29. Nayak, A., Vishwanath, A.: Quantum walk on the line. arXiv preprint quant-ph/0010117 (2000)
30. Kendon, V.: Quantum walks on general graphs. International Journal of Quantum Information **4**(05), 791–805 (2006)
31. Ambainis, A.: Quantum walks and their algorithmic applications. International Journal of Quantum Information **1**(04), 507–518 (2003)
32. Ahmad, R., Sajjad, U., Sajid, M.: One-dimensional quantum walks with a position-dependent coin. arXiv preprint arXiv:1902.10988 (2019)
33. Zhang, P., Ren, X.-F., Zou, X.-B., Liu, B.-H., Huang, Y.-F., Guo, G.-C.: Demonstration of one-dimensional quantum random walks using orbital angular momentum of photons. Physical Review A **75**(5), 052310 (2007)

34. Ryan, C.A., Laforest, M., Boileau, J.-C., Laflamme, R.: Experimental implementation of a discrete-time quantum random walk on an nmr quantum-information processor. Physical Review A **72**(6), 062317 (2005)
35. Travaglione, B.C., Milburn, G.J.: Implementing the quantum random walk. Physical Review A **65**(3), 032310 (2002)
36. Agarwal, G.S., Pathak, P.K.: Quantum random walk of the field in an externally driven cavity. Physical Review A **72**(3), 033815 (2005)
37. Joo, J., Knight, P.L., Pachos, J.K.: Single atom quantum walk with 1d optical superlattices. Journal of Modern Optics **54**(11), 1627–1638 (2007)
38. Manouchehri, K., Wang, J.: Quantum random walks without walking. Physical Review A **80**(6), 060304 (2009)
39. Manouchehri, K., Wang, J.: Quantum walks in an array of quantum dots. Journal of Physics A: Mathematical and Theoretical **41**(6), 065304 (2008)
40. Loke, T., Wang, J.: An efficient quantum circuit analyser on qubits and qudits. Computer Physics Communications **182**(10), 2285–2294 (2011)
41. Jordan, S.P., Wocjan, P.: Efficient quantum circuits for arbitrary sparse unitaries. Physical Review A **80**(6), 062301 (2009)
42. Chiang, C.-F., Nagaj, D., Wocjan, P.: Efficient circuits for quantum walks. arXiv preprint arXiv:0903.3465 (2009)
43. Loke, T., Wang, J.: Efficient circuit implementation of quantum walks on non-degree-regular graphs. Physical Review A **86**(4), 042338 (2012)
44. Rohde, P.P., Schreiber, A., Štefaňák, M., Jex, I., Gilchrist, A., Silberhorn, C.: Increasing the dimensionality of quantum walks using multiple walkers. Journal of Computational and Theoretical Nanoscience **10**(7), 1644–1652 (2013)
45. Arjovsky, M., Shah, A., Bengio, Y.: Unitary evolution recurrent neural networks. In: International Conference on Machine Learning, pp. 1120–1128 (2016)
46. Jing, L., Shen, Y., Dubček, T., Peurifoy, J., Skirlo, S., Tegmark, M., Soljačić, M.: Tunable efficient unitary neural networks (eunn) and their application to rnn. arXiv preprint arXiv:1612.05231 (2016)
47. Brandes, U.: A faster algorithm for betweenness centrality. Journal of mathematical sociology **25**(2), 163–177 (2001)
48. Vinyals, O., Bengio, S., Kudlur, M.: Order matters: Sequence to sequence for sets. arXiv preprint arXiv:1511.06391 (2015)
49. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 1263–1272 (2017). JMLR. org
50. Williams, C., Vose, R., Easterling, D., Menne, M.: 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)
51. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.-P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), 47–56 (2005)
52. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. Journal of medicinal chemistry **34**(2), 786–797 (1991)
53. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems **14**(3), 347–375 (2008)
54. Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., Schomburg, D.: Brenda, the enzyme database: updates and major new developments. Nucleic acids research **32**(suppl_1), 431–433 (2004)
55. Shervashidze, N., Schweitzer, P., Leeuwen, E.J.v., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(Sep), 2539–2561 (2011)
56. Borgwardt, K.M., Kriegel, H.-P.: Shortest-path kernels on graphs. In: Fifth IEEE International Conference on Data Mining (ICDM'05), p. 8 (2005). IEEE
57. Blum, L.C., Reymond, J.-L.: 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. J. Am. Chem. Soc. **131**, 8732 (2009)
58. Rupp, M., Tkatchenko, A., Müller, K.-R., von Lilienfeld, O.A.: Fast and accurate modeling of molecular atomization energies with machine learning. Physical Review Letters **108**, 058301 (2012)