Quantum Computing for Graph Representation Learning

Candidate no. 1047849

University of Oxford

A thesis submitted for the degree of

Master of Science

Trinity Term 2021

Abstract

In this thesis, we explore applications of quantum computational and quantum-inspired classical approaches to the domain of graph representation learning. We focus on two distinct areas: firstly, we discuss a quantum-inspired perspective on knowledge graph embeddings; secondly, we analyse the capabilities of possible quantum graph neural network methods.

In Part I, we use the quantum-inspired natural language processing approach *DisCoCat* to give a novel perspective on *knowledge graph embedding* (KGE) methods. We show how DisCoCat can be seen as a generalisation of later KGE techniques, formalise deep connection between these two fields, and discuss potential directions to build on these correspondences in both fields, bringing ideas from each area to the other.

In Part II, we investigate quantum analogues for *graph neural network* (GNN) architectures, formalising what it means for quantum circuits to respect the structure of the graph domain in our proposed framework of *Equivariant Quantum Graph Circuits*. We discuss how earlier work on quantum machine learning over graphs can be seen as a special case of this framework and discuss other possible subclasses. We analyse the expressive capacity of such models, confirming experimentally that they are more powerful than many popular GNNs, and prove that they are universal approximators for functions over the graph domain.

Contents

Chapter 1

Introduction

This project aims to bring together useful ideas from two distinct areas of computer science: *quantum computing* and *graph representation learning*. In this chapter, we briefly review the past and present of these fields, then discuss our contributions and give an overview of the rest of the thesis.

1.1 Historical overview

Quantum computing is the study of computation using specialised hardware that can make use of quantum mechanical behaviour in a controlled way to go beyond what classical computers can feasibly calculate. This has been a subject of attention for around 40 years [24], and it is now getting close to being applicable in practice. While quantum methods (such as Shor's factoring algorithm [61]) have been known to provide even exponential speedups compared to the best known classical approaches for certain problems since the nineties, until recently there was no quantum hardware that could perform even the simplest operations in a controlled, reliable way. We have since entered the era of *noisy intermediate-scale quantum* (NISQ) devices: although this hardware is still very far from sufficient to run famous algorithms such as Shor's, it has already outperformed classical supercomputers on carefully selected tasks [4] and it is expected that the performance of quantum devices will keep increasing in the coming years. This progress has motivated an increasing amount of recent work into applications of near-term devices. One candidate area is *quantum machine learning*, a subfield that seeks to use quantum computers to solve machine learning tasks [51].

Graph representation learning or **relational learning** is a comparatively young subfield of computer science that arose through the wide successes of machine learning methods in the last decade. Inspired by specialised architectures for other domains (such as convolutional neural

networks for grids and recurrent neural networks for sequences), this area is focused on finding methods for effectively learning architectures for *graph-structured* data. It encompasses a range of significantly different methods itself: in most problem settings, a class of deep neural network architectures called *graph neural networks* (GNNs) are commonly used, but there are certain tasks where simpler *shallow embedding approaches* are used as we will discuss later [29].

1.2 Our contributions

While both of these two fields have seen growing interest in recent years, the amount of interdisciplinary work is still fairly small – we hope this project can advance the understanding of what can be achieved using quantum methods on graphs, and also motivate future work by pointing out interesting connections and providing a unifying perspective.

The project focuses on two distinct areas where we see potential for quantum computing and quantum-inspired classical methods to contribute to the field of graph representation learning. In Part I, we explore correspondences between *DisCoCat*, a quantum-inspired natural language processing approach [21, 19], and more recent work on *shallow node embeddings* for *knowledge graphs*, an important specialty within the field of relational learning. We show that certain knowledge graph embedding methods can be seen as a special case of DisCoCat, and establish connections that can inform further work in both fields. In Part II, we focus on quantum analogues of graph neural networks: we propose a unifying framework that subsumes existing proposals and prove powerful results about the theoretical expressive power of such models, supported by experiments.

In both parts, our focus is primarily on the theoretical side. In Part I, this is because our derivations lead to new perspectives on existing methods, but do not suggest new experiments; in Part II, we chose this focus because only very small-scale experiments are feasible today by classical simulation or using NISQ hardware, while our theoretical insights will stay relevant even as the capabilities of available quantum hardware increase.

The rest of this section gives an overview of each part of the thesis.

1.2.1 Quantum-inspired knowledge graph embeddings

In the first half of this thesis, we explore the correspondences between *DisCoCat* and certain more recent *knowledge graph embedding* (KGE) methods. We discuss how DisCoCat can be seen as a generalisation of these KGE methods and point out connections that might inspire further work in both fields.

DisCoCat is an approach to natural language processing (NLP) inspired by category theoretic approaches to formal grammars and quantum computing [21, 19]. By focusing on the grammatical

structure of sentences and mapping this to computations on classical or quantum hardware in a way that respects this structure, it seeks to combine the advantages of rigid grammar-based NLP approaches with the power of distributional methods that operate on vectorised representations of words. This is done by mapping words to vectors or higher-rank tensors with a shape depending on their grammatical role in the sentence, and contracting these tensors in a pattern corresponding to the sentence structure.

Knowledge graphs are datasets storing facts about various entities (people, objects, concepts etc.) in a simple but very general form. They consist of a set of entities (such as 'Alice', 'Bob'), a set of possible relations between the entities (such as 'likes'), and a set of known true facts built from these ('Alice likes Bob'). Unfortunately, these datasets tend to be highly incomplete, with only a fraction of true facts listed. This has motivated the application of machine learning to make predictions about unknown facts and reason over knowledge graphs in the presence of uncertainty. Most state-of-the-art methods do this using *shallow node/relation embeddings*: each entity and relation is represented as a learnable vector, with a scoring function combining these vectors to assign a likelihood to each fact. By optimising the error over a set of ground-truth facts, these models can pick up on patterns and generalise to new facts beyond their training data.

We discuss the background in these two domains in detail in Chapter 2, then build on this to explore connections between them in Chapter 3. We reduce KGE tasks to sentence classification tasks in natural language by mapping entities and relations to phrases and facts to sentences. Applying DisCoCat methods to such sentences is shown to respect the structure of the knowledge graph domain, allowing us to derive algorithms that are not intrinsically tied to the unstructured realm of natural language. The derived algorithms turn out to match well-known KGE methods, giving us a novel perspective on existing approaches. In Chapter 4, we discuss potential directions for future work in both fields based on these connections.

1.2.2 Quantum graph neural networks

In the second part of the project, we investigate quantum alternatives of graph neural networks. We propose a unifying framework of *Equivariant Quantum Graph Circuits* (EQGCs) to formalise the notion of quantum circuits that respect the invariances of the graph domain, and show that it subsumes existing methods as special cases. We explore design choices within this framework, characterising several practical subclasses for graphs as well as a restricted class that is suitable for learning over sets instead. We then analyse the representational capacity of these models in comparison with their classical counterparts empirically and theoretically: we experimentally confirm that they go beyond the expressive power of popular GNN models and then prove that they are in fact universal approximators for functions over the graph domain.

GNNs are deep neural networks that are used to predict properties of nodes or entire graphs in a way that respects their structural invariances: the ordering of nodes and edges in the representation should not matter [29, 8]. They achieved impressive results on a wide range of benchmarks in recent years, and there has also been significant work on the theoretical capabilities of such models. This included results showing limitations of the expressive power of a broad class of popular GNN models – particularly, some pairs of distinct graphs will always result in the same predictions with these methods (see, e.g., [7, 67, 43]). Recent work has focused on getting around this limitation at the cost of more computational power [48, 43] or by introducing randomisation [2, 56].

Meanwhile, there have been a number of proposals for quantum analogues of GNNs, typically using sets of qubits to represent information about each node, and performing multi-node entangling operations according to the structure of the graph in question [64, 72]. This lets us make use of the exponentially large Hilbert space of the joint system to model complex interactions via entanglement between the node states. Such methods have been supported by empirical data from small-scale experiments, particularly when applied to modelling quantum systems, but to date there has been no clarity on how their expressivity compares to classical approaches.

Our framework of EQGCs gives a unifying perspective on these methods and allows us to compare their expressive power to that of classical approaches in a highly general way. We find that certain EQGC subclasses can represent arbitrary Boolean and real-valued functions over bounded-size graphs, which is more than what many popular GNN methods are capable of [67, 42].

This part of the thesis is organised as follows. We first introduce the relevant background on GNNs and their properties, including recent results involving randomised models and existing quantum models for graph representation learning, in Chapter 5. We then introduce Equivariant Quantum Graph Circuits and explore the space of possible methods in this framework in Chapter 6. Chapter 7 is dedicated to analyzing the representation power of practical EQGC subclasses: this is where we get to our main result, showing that they are universal models for functions over bounded-size graphs. Finally, we summarise our results and discuss directions for future work in Chapter 8

Although our constructions do not show quantum advantage over classical models with randomisation, our results serve as important steps towards a better theoretical understanding of quantum methods. Additionally, through the EQGC framework, we provide an analysis of possible quantum architectures that can help inform future work.

Part I

Quantum-Inspired Knowledge Graph Embedding

Chapter 2

Background and Related Work

2.1 DisCoCat

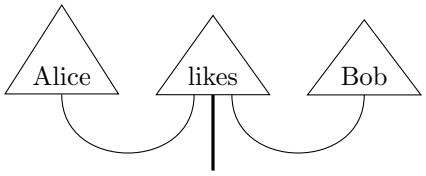
Categorical compositional distributional semantics, also known as *DisCoCat* for short, is a natural language processing (NLP) approach inspired by category theoretic approaches to quantum computing and formal grammars [21, 19]. It aims to combine the benefits of distributional models of language, where words are represented as abstract vectors that are processed through neural models, with more structured grammar-based approaches. There are also possible benefits from running DisCoCat models on quantum hardware, and several small-scale experiments have been performed on current noisy intermediate-scale devices [44, 38]. However, the general DisCoCat approach can be defined in purely classical terms and implemented on classical hardware, which the setting this project will focus on.

Model overview

In DisCoCat, each word of a sentence is mapped to a tensor in a (real or complex) vector space of a rank and dimensionality depending on its grammatical role. These tensors are then reduced according to the grammatical structure of the sentence to form a vector encoding the semantics of the sentence. This is typically depicted using the diagrammatic notation of monoidal categories (for the categories **FVect** in the case of real tensors and **FHilb** for complex ones), which are essentially tensor network diagrams.

For the purposes of this project, we will be mostly concerned with nouns and transitive verbs. Nouns are represented by rank-one tensors, i.e., vectors of some dimensionality d_n, while verbs correspond to rank-three tensors of dimensionality $d_n \times d_s \times d_n$, where is the dimensionality of the resulting sentence embedding vectors. Other word types can be derived from Lambek's pregroup-based formal grammar [35], but these two suffice for forming simple sentences.

Example 1. The sentence 'Alice likes Bob' is represented by the following diagram:



,

where the thin wires represent tensor contractions involving noun embeddings of size d_n, while the thick wire carries the sentence embedding of size d_s.

In terms of tensors, this shows that we get the sentence embedding vector by reducing the rank-3 tensor for 'likes' with the vectors for 'Alice' and 'Bob' on each side:

$$\overrightarrow{\text{Alice likes Bob}} = \overrightarrow{\text{Alice}} \times \overrightarrow{\text{likes}} \times \overrightarrow{\text{Bob}} \tag{2.1}$$

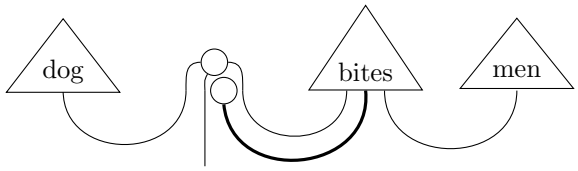
The dimensionality of sentence embedding vectors, and what exactly they should represent, depends on the task at hand. For example, we might be interested in binary sentence classification tasks, such as positive/negative sentiment classification, or judging whether a sentence is true. In these cases, we can set $d_s = 1$ and have positive/negative real numbers correspond to the two classes.

Remark 1. In the case of binary sentence classification with $d_s = 1$, the $d_n \times d_s \times d_n$ tensor for transitive verbs effectively reduces to a $d_n \times d_n$ matrix and the corresponding wire can be removed from the DisCoCat diagrams. This will be important for our derivations in Sections 3.2 and 3.3.

Two approaches to representing words

There are two methods for selecting embeddings for individual words in DisCoCat. For most words, supervised training is proposed, just as in most NLP models. A space of possible vectors/tensors is given – this might be all tensors of the appropriate shape, or some subspace (e.g., for efficient implementation on a quantum computer). Initial word embeddings are chosen at random. A set of training sentences is given with ground-truth labels for some aspect (e.g., classification of truth value, sentiment, topic). Based on these labels, the word embedding parameters are gradually optimised to minimise some error term (e.g., cross-entropy for classification).

For certain hand-selected words, there have been proposals for a different approach: directly engineering the precise inner structure of function words based on their logical meaning [54, 55]. This makes it easier for models to learn with fewer examples, and in certain cases also simplifies calculations. For example, Sadrzadeh et al. suggest the following instantiation for the relative pronoun *"that"* in noun phrases such as "dog that bites men" with desirable logical properties [54]:



where the white dots represent $\sum_{i \in \mathcal{B}_n} |iii\rangle$ and $\sum_{j \in \mathcal{B}_s} |j\rangle$, with $\mathcal{B}_n, \mathcal{B}_s$ standing for the computational bases of the noun and sentence embedding space respectively.

In the case of truth-value classification with $d_s = 1$, this construction reduces to an elementwise product of vectors for the noun "dog" and the phrase "bites men" (note that the transitive verb "bites" is represented by an $d_n \times d_n$ matrix as discussed in Remark 1, which can be applied to the embedding of "men" to produce another d_n-dimensional vector):

$$\overrightarrow{\text{dog that bites men}} = \overrightarrow{\text{dog}} \odot (\overrightarrow{\text{bites}} \times \overrightarrow{\text{men}}) \tag{2.2}$$

The general case with $d_s > 1$ is similar, except we first get a matrix from the rank-3 tensor $\overrightarrow{\text{bites}}$ by reducing it with the white dot in the sentence space, which corresponds to simply summing over the appropriate dimension.

2.2 Knowledge graph embedding

2.2.1 The challenges of knowledge graphs

An important graph-structured domain that has attracted significant attention in the field of graph representation learning is that of *knowledge graphs*: relational datasets storing a wide range of facts about various entities [13, 41, 45].

Formally, a knowledge graph consists of a set of entities $\mathcal{E}$, a set of relations $\mathcal{R}$, and a set of true facts $\mathcal{F} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$. For example, entities could represent people, relations could represent relationships like 'friend of', and the set of facts represents known true statements. The first entity of a triple is often called the *subject*, while the second one is the *object*; alternatively, the terms *head* and *tail* are also used.

The main idea behind *knowledge graph embedding* (KGE) models is to embed knowledge graphs into low-dimensional vector spaces so as to use the resulting representations for various prediction tasks, following a gradient-based optimisation [14, 33]. This is possible thanks to various regularities and patterns in the data that successful KGE models can capitalise on in order to provide accurate predictions.

Link prediction

Real-world knowledge graphs tend to be highly incomplete, with only a fraction of true facts known. Hence, an important task is *knowledge graph completion*, or *link prediction*: given all entities and

relations in a knowledge graph with only a subset of all true facts, the task is to predict further facts that are likely to be true. More specifically, this is usually framed as binary classification task of deciding whether a given (subject, relation, object) triple is a true fact or not. KGE models are usually trained to do this in a supervised manner: they are trained on the known true facts along with randomly sampled false examples and validated on a held-out set. The best-performing model can be used to rank unknown facts and find ones that are considered true with sufficiently high confidence.

In order to do this, we need robust metrics for comparing KGE models. In principle, we could use typical metrics of binary classification tasks, but these are generally not very well-suited to the problem of link prediction. For example, when classifying randomly sampled facts, we can get near-perfect accuracy by always predicting it is not a true fact. This could be mitigated by sampling differently, but then our metric depends heavily on the details of our sampling approach. Instead, the usual metrics are designed specifically for the domain of knowledge graphs.

One standard metric is *hits@k*: for a known true fact (s, r, o), consider completing $(s, r, \cdot)$ with all possible objects. The model is successful on this fact if completing this o is within the k most likely facts according to its ranking. (Importantly, other known true solutions are filtered from the ranking so that the model is not punished for the existence of facts with multiple correct solutions [15].) Variations with facts of the form $(\cdot, r, o)$, where the subject is hidden, are also used. The model's hits@k is then the fraction of true facts in the test set where the model is successful in this sense:

$$H_k = \frac{1}{2|\mathcal{F}^+_{\text{test}}|} \sum_{(s,r,o)\in\mathcal{F}^+_{\text{test}}} \Big(\mathbb{1}[\text{rank}(s \mid \_, r, o) \leq k] + \mathbb{1}[\text{rank}(o \mid s, r, \_) \leq k] \Big), \tag{2.3}$$

where $\mathbb{1}[\cdot]$ is the indicator function, F^+_{test} is the set of held-out true facts, $\text{rank}(s \mid \_, r, o)$ is the rank of the fact (s, r, o) among corrupted negative examples (s', r, o) according to the likelihoods predicted by the model, and $\text{rank}(o \mid s, r, \_)$ is analogous with objects ranked. This is often reported with several different values of k, e.g., $1, 3$ and 10.

Other common metrics include *mean rank* (MR) and *mean reciprocal rank* (MRR). These are based on the same idea of ranking true facts against a filtered list of corrupted variants. MR simply averages the ranks, which is often highly affected by a few bad predictions leading to high ranks; MRR avoids this by averaging their reciprocals:

$$MR = \frac{1}{2|\mathcal{F}^+_{\text{test}}|} \sum_{(s,r,o)\in\mathcal{F}^+_{\text{test}}} \Big(\text{rank}(s|\_, r, o) + \text{rank}(o|s, r, \_) \Big) \tag{2.4}$$

$$MRR = \frac{1}{2|\mathcal{F}^+_{\text{test}}|} \sum_{(s,r,o)\in\mathcal{F}^+_{\text{test}}} \Big(\frac{1}{\text{rank}(s|\_, r, o)} + \frac{1}{\text{rank}(o|s, r, \_)} \Big) \tag{2.5}$$

MR is better if lower, with possible scores in the interval $[1, |\mathcal{E}|]$, while MRR is better if higher, with scores in $(0, 1]$.

Complex query answering

A strength of knowledge graphs as a data structure is the possibility to not just check individual facts, but explore them and reason over several steps.

Example 2. With a comprehensive dataset of professional and social connections, we might be able to search for friends of our friends with experience with a particular technology. This can be expressed as a sentence in first-order logic over the set of entities, using relations of the knowledge graph – we are looking for some person p such that the following holds:

$$\text{experiencedWith}(p, \text{someTech}) \wedge \exists x. \big(\text{friends}(p, x) \wedge \text{friends}(x, \text{me})\big)$$

Complex query answering is the task of finding entities satisfying such queries, typically given in some restricted form of first-order logic depending on the capabilities of each method. Given a complete knowledge graph, it is relatively easy to find solutions to most such queries. However, as we have discussed, these datasets are usually incomplete, so KGE models are often necessary. There is a wide range of approaches: some build on link prediction models and combine their predictions for individual facts to find satisfying entities for a given query [3], while others design methods that jointly consider the entire query without explicitly calculating probabilities of each link holding [28, 25, 52].

For evaluating the performance of complex query answering methods, the same ranking-based metrics are typically used that we discussed in the case of link prediction. By walking over the edges given by known true facts in the training and test set, we can find entities that satisfy a given property, and the models should rank these ground truth answers above others. This lets us define hits@k, MR and MRR just as before.

2.2.2 Link prediction methods

In this section, we review popular methods for the task of link prediction, with a particular focus on tensor-based linear models that will be interesting to compare to the DisCoCat framework.

Current state-of-the-art link prediction methods are largely based on *shallow node/relation embeddings*: for each entity $e \in \mathcal{E}$ and relation $r \in \mathcal{R}$, embedding vectors $\boldsymbol{e}, \boldsymbol{r}$ in some real or complex vector spaces are learned directly by gradient-based optimisation. These are then combined by some *scoring function* that gives some predicted probability $\phi(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o})$ to a fact (s, r, o) being true. There is also work on neural approaches using deep neural networks to produce the embeddings [58] or to learn the scoring functions [22], but so far these tend to overfit and underperform shallow embeddings.

The primary supervised loss minimised during training is typically standard binary cross-

entropy:

$$L = -\frac{1}{\mathcal{F}_{\text{train}}} \sum_{((s,r,o),l)\in F_{\text{train}}} \Big(l \log(\phi(\boldsymbol{s},\boldsymbol{r},\boldsymbol{o})) + (1-l)\log(1-\phi(\boldsymbol{s},\boldsymbol{r},\boldsymbol{o})) \Big) \tag{2.6}$$

where $\mathcal{F}_{\text{train}}$ is the training set including randomly sampled negative examples, with labels $l \in \{0,1\}$ representing the expected predictions. To account for the chance that negative samples might accidentally include unknown true facts, soft labels $l \in [0,1]$ are sometimes used instead of setting them to 0.

In addition to the supervised loss, regularisation methods are often used. This includes, e.g., an L2 regularisation loss term for many models [49, 6].

RESCAL

In the *RESCAL* model, for each relation, a bilinear function of the subject and object embeddings is learned to classify facts [49]. That means that while entities are represented by vectors $\boldsymbol{e} \in \mathbb{R}^d$ for some integer d, relations are represented by entire matrices $\boldsymbol{R} \in \mathbb{R}^{d\times d}$. This allows the scoring function to be defined as follows:

$$\phi(\boldsymbol{s},\boldsymbol{R},\boldsymbol{o}) = \sigma\big(\boldsymbol{s}^T \boldsymbol{R}\boldsymbol{o}\big), \tag{2.7}$$

where $\sigma(\cdot)$ is a sigmoid nonlinearity.

This method achieved state-of-the-art results on benchmarks at the time, and it has desirable properties for exploiting certain common patterns in knowledge graph datasets [29]. However, learning arbitrary matrices for each individual relation comes with some disadvantages: it can allow the model to overfit to the training data, and makes high-dimensional embeddings very memory-intensive for large-scale knowledge graphs. This motivated a line of research into similar, but more restricted, linear models [68, 62, 6].

TuckER

TuckER is a linear model that generalised several popular earlier models, including RESCAL. Instead of representing relations as matrices that combine the entity embedding vectors $\boldsymbol{e} \in \mathbb{R}^{d_e}$, relations are represented by vectors $\boldsymbol{r} \in \mathbb{R}^{d_r}$, and a global rank-3 tensor $\mathbf{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$ is used to combine them:

$$\phi(\boldsymbol{s},\boldsymbol{r},\boldsymbol{o}) = \sigma\big(\big(\mathbf{W} \times_1 \boldsymbol{s} \times_2 \boldsymbol{r} \times_3 \boldsymbol{o}\big) = \sigma\big(\sum_{ijk} \mathcal{W}_{ijk} s_i r_j o_k\big) \tag{2.8}$$

One way to view this is to consider $\mathbf{W}$ as set of d_r slices each representing a RESCAL-style relation matrix. Reducing $\mathbf{W}$ with $\boldsymbol{r}$ first gives $\boldsymbol{M}^{(r)} = \mathbf{W} \times_2 \boldsymbol{r}$ as a linear combination of these slices: $\boldsymbol{r}$ simply gives the weights associated with each slice of $\mathbf{W}$ for the given relation.

This effectively allows weight-sharing between different relations, which is often useful, as the different relations are sometimes based on similar interactions between entities. Furthermore, with $d_r \ll d_e \times d_e$, it mitigates both the overfitting and the memory scaling issues of RESCAL. Combined with the smart use of regularisation methods such as dropout and batch normalisation, this allowed TuckER to achieve state-of-the-art results on several benchmarks.

2.2.3 Complex query answering methods

In this section, we review the *Graph Query Embedding* (GQE) framework for complex query answering proposed by Hamilton et al. [28], which will be of particular importance for us in Section 3.3. We then briefly discuss the some of the primary challenges involved, and more recent approaches to solving them.

Graph Query Embedding

Graph Query Embedding (GQE) is one of the first complex query answering methods on knowledge graphs [28]. It can predict entities satisfying a certain class of logical queries that we will call *tree conjunctive queries*, which can be defined recursively as follows:

$$\begin{aligned} q(x) := \; & r_i(x, e_j) \\ & \mid \exists y \; . \; r_i(x, y) \wedge q'(y) \\ & \mid \bigwedge_k q'_k(x) \end{aligned} \tag{2.9}$$

where $r(s, o)$ holds if (s, r, o) is a fact; r_i are fixed relations and e_j are fixed entities; $q'(\cdot)$ are arbitrary subqueries of the same form.

These queries can be seen as trees where the leaves are anchored to known entities. The inner nodes should be mapped to some unknown entities (not necessarily uniquely) in a way that respects the edge relations, and our task is to find nodes that can likely stand as the root in such a structure.

Example 3. Consider the following query:

$$q(x) = \exists y. r_0(x, y) \wedge \big(r_1(y, e_1) \wedge r_2(y, e_2) \tag{2.10}$$

This is built up by using all cases of the above definition. $q(e^*)$ holds for some entity e^* if it is related by r_0 to some other entity that is related by r_1 to e_1 and by r_2 to e_2.

Remark 2. Our presentation differs slightly from that of Hamilton et al. Notably, they allow the structure given by the query relations to be a directed acyclic graph (DAG) rather than a tree; however, this can be represented in our tree conjunctive query form as well by duplicating nodes with multiple incoming edges. Implementing this in a naive way can be very inefficient

compared to a DAG-based representation for large queries, but the two approaches are equivalent on the conceptual level, and the tree-based perspective will be more useful for our discussions. Furthermore, all queries involved in their experiments were in fact tree conjunctive queries.

The GQE method is based on recursively defining embeddings $[\![q(\cdot)]\!] \in \mathbb{R}^d$ for queries in the same space as the entity embeddings $\boldsymbol{e}_i \in \mathbb{R}^d$, representing where the model expects to find embeddings of entities that satisfy the query. Whether an entity e_i satisfies a query q is then predicted based on the cosine similarity of their embeddings $\boldsymbol{e}_i, [\![q(\cdot)]\!]$.

All that remains is to give an interpretation to the query embeddings $[\![q(\cdot)]\!]$. Based on linear methods such as RESCAL, Hamilton et al. represent each relation with a matrix $\boldsymbol{R}^{(i)} \in \mathbb{R}$ and define the following:

$$[\![r_i(\cdot, e_j)]\!] = \boldsymbol{R}^{(i)} \boldsymbol{e}_j \tag{2.11}$$

$$[\![\exists y \,.\, r_i(\cdot, y) \wedge q'(y)]\!] = \boldsymbol{R}^{(i)} [\![q'(\cdot)]\!] \tag{2.12}$$

Finally, the interpretations of conjunctions was chosen to be a very general neural aggregator inspired by recent deep learning work on sets [70]:

$$[\![\bigwedge_k q'_k(\cdot)]\!] = \boldsymbol{W} \Psi\big(\{\mathrm{NN}([\![q'_k(\cdot)]\!])\}\big), \tag{2.13}$$

where NN is a multi-layer perceptron, Ψ is a simple symmetric aggregation function such as elementwise sum, mean, min or max, and $\boldsymbol{W}$ is an additional linear transformation.

Hamilton et al. also showed that GQE has desirable theoretical properties. Given a set $\mathcal{F}$ of known training facts, let $\mathcal{E}_{\mathcal{F},q}$ be the set of entities that a query q verifiably holds for based on facts in $\mathcal{F}$. With sufficiently high embedding dimensions, for any $\mathcal{F}$, there is a configuration of all embeddings such the cosine similarity between $[\![q(\cdot)]\!]$ and any entity embedding $\boldsymbol{e}_i$ is positive if and only if $e_i \in E_{\mathcal{F},q}$.

A brief outline of their proof: by setting d to be the number of entities, the entity embeddings $\boldsymbol{e}_i$ can be set to be computational basis vectors. The relation matrices $\boldsymbol{R}^{(i)}$ can then be boolean adjacency matrices: $\boldsymbol{R}^{(i)}_{jk} = 1$ if $r_i(j, k)$ holds, 0 otherwise. The conjunction aggregation should be a simple min function. It is then straightforward to recursively show that all query embeddings $[\![q(\cdot)]\!]$ will be boolean vectors directly representing the set $E_{\mathcal{F},q}$, and these have the expected cosine similarities with each $\boldsymbol{e}_i$.

Other query answering methods

Since Hamilton et al.'s work on complex query answering, many new complex query answering approaches have been proposed. They all looked for ways to overcome the limitations of vector spaces for representing the logical structure of queries – since they can be combined with logical

operators, a good embedding method for a large class of queries would have to implement operations such as negation and disjunction in the space of query embeddings in addition to conjunctions. We briefly review some noteworthy methods.

Embed2Reason [25] is inspired by *quantum logic* [12], an unusual formal logic that represents logical clauses as vector spaces corresponding to the structure of quantum observables. Note that although this is also a quantum-inspired method, their approach is quite different from what we explore in this project. In Embed2Reason, queries are mapped to axis-aligned linear subspaces going through the origin, and entities satisfying a query are expected to be inside the corresponding subspace. These subspaces are learned via vectors with a loss term that is minimised for binary vectors: a '1' at some position denotes that the subspace extends along that dimension, while a '0' means that it does not. Logical operations can be effectively implemented bitwise on such vectors. This gets us something close to an intuitive logical structure, but it does differ from normal classical logic: for example, the disjunction of two subspaces A, B is the smallest linear subspace containing both of them, which includes vectors that are not inside either of A and B.

Query2Box is based on embedding queries as high-dimensional 'boxes', or hyper-rectangles, and expecting entities satisfying a query to be inside the corresonding box [52]. This has desirable properties for handling logical constructions: conjunctions can be represented as intersections, and top-level disjunctions can seen as multiple queries, each with their own hyper-rectangle. This made it possible to effectively represent a large class of queries in disjunctive normal form, and provide great predictions.

Arakelyan et al. work around the problem of finding a query embedding space suitable for encoding logical structure by avoiding query embeddings entirely. Instead, they propose *Complex Query Decomposition* [3]: they take link predictors trained for predicting individual edges, and use these to score potential embeddings for the query variable and all existentially quantified variables. Optimisation methods are used to find embeddings that the link predictors consider likely to satisfy all the required relations. They propose two variations of this idea. Firstly, a continuous optimisation method where gradient descent is used to find an optimal vectors in the entity embedding space, then they look up entities that are closed to the embedding for the query variable found through gradient descent. Secondly, they consider a discrete combinatorial optimisation approach, where a heuristic beam search method is used for finding good assignments of entities for each variable.

Chapter 3

DisCoCat as a Knowledge Graph Model

In this chapter, we will discuss the application of DisCoCat methods to the knowledge graph embedding tasks of link prediction and complex query answering.

By mapping facts in a knowledge graph to sentences in a restricted subset of natural language, we can apply natural language processing methods to KGE tasks. Unlike most NLP models, DisCoCat's grammar-based approach ensures that we do not lose the structure of the KGE context through this reduction.

This mapping leads to new derivations of existing methods: we find that applying DisCoCat to the task of link prediction is equivalent to the RESCAL model discussed in Section 2.2.2, and discuss how other linear models can also be seen from this perspective. This also gives a new perspective on some recently proposed quantum computing methods for link prediction [39].

For the task of complex query answering, we give a class of queries that can be analysed using a DisCoCat encoding of relative pronouns [54]. This correspondence leads to a variant of the Graph Query Embedding model discussed in Section 2.2.3.

3.1 Embedding knowledge graphs in natural language

It is very natural to represent facts in a knowledge graph as sentences: if entity s represents Alice, o is Bob, and r is the relation 'likes', then the fact (s, r, o) means "Alice likes Bob". More generally, we can represent entities as nouns or noun phrases, relations as verbs or verb phrases, and write facts as simple sentences of the form "[subject] [verb]s [object]". Complex queries can also be represented using more complex sentences, as we will discuss in Section 3.3.1.

This suggests a reduction of knowledge graph tasks to natural language tasks. Given a natural language learning algorithm $\mathcal{A}$ that can be trained to classify sentences, we can build a model for link prediction. We map the entities to nouns and the relations to verbs, turn all facts into simple

sentences, and train $\mathcal{A}$ to classify them as true or false. At test time, the held-out facts can be similarly encoded as sentences and run through the model.

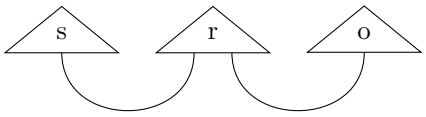
Of course, just because we can represent a task in natural language, that does not mean that NLP methods are well-equipped to handle them. However, using DisCoCat to classify sentences generated by this reduction has desirable properties: due to the grammar-based nature of DisCoCat, and the restricted grammatical forms of our sentences, this mapping results in algorithms that interact well with the structure of the knowledge graph, and are not inherently linked to the unstructured domain of natural languages. In fact, this mapping naturally leads to derivations of well-known successful KGE approaches as we will see in the following sections.

3.2 DisCoCat and link prediction

In the following sections, we will be using DisCoCat for binary classification of truthiness, so our sentence embedding space will have dimensionality $d_s = 1$. As noted in Remark 1, this means we can treat verb embeddings as $d_n \times d_n$ matrices, where d_n is the noun embedding dimensionality, and remove the sentence embedding wires from our diagrams.

3.2.1 Linear knowledge graph embeddings as a special case of DisCoCat

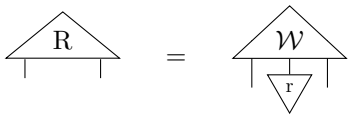
If we apply DisCoCat to the simple sentences that we get from encoding knowledge graph facts, we get sentence diagrams of the following form:



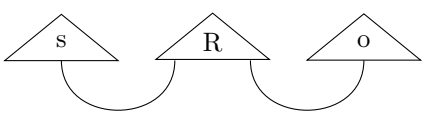
If our word embeddings are arbitrary real-valued vectors/matrices of the appropriate shape, this gives us exactly the same calculation that the RESCAL model would do, as discussed in Section 2.2.2! This can be seen as DisCoCat generalising later knowledge graph embedding methods.

Other linear models from a diagrammatic perspective

Other linear KGE models can be seen as different proposals for the 'inner wiring' of word embeddings. For example, the TuckER method is neatly summarised by the following diagram:



$\phi(s, r, o)$ =



3.2.2 Potential for quantum execution

One of the motivations behind DisCoCat is the possibility of execution on quantum computers, which can have benefits for large tensor computations. In the case of the sentences we get from the KGE task, this corresponds exactly to the *fully parameterised Quantum Circuit Embedding* model proposed by Ma et al. independently of DisCoCat literature [39].

In this method, each entity e is represented as a state $|\psi_e\rangle$ and each relation r as a unitary $\boldsymbol{U}_r$, all produced by *parameterised quantum circuits* (circuits with a fixed structure involving parameterised gates such as Z-rotations, such that optimising those parameters can lead to effective embeddings). To classify a fact (s, r, o), we need to estimate the inner product $\langle\psi_o| \boldsymbol{U}_r |\psi_s\rangle$, which can be done efficiently by the *swap test* [17].

Since Ma et al. propose that the circuit depth should scale linearly with the number of qubits q, the cost of training and running the model scales logarithmically with the dimensionality 2^q of the state space. This makes it possible to train and execute this RESCAL-like model efficiently with a seemingly larger state space than in the case of classical methods. However, the dimensionality of the subspace *actually representable* by the shallow parameterised circuits scales with the number of circuit parameters, which is limited. It might be the case that such a subspace of the exponentially large Hilbert space has beneficial theoretical or practical properties – perhaps if any vector in the entire space can be approximated to a suitable degree, the result will behave similarly to classical models with an exponentially large embedding space, despite the reachable subspace being much more limited. However, it is currently not clear whether there would be such advantages.

3.3 DisCoCat and complex query answering

In this section, we will derive a method for answering *tree conjunctive queries* as defined in Equation 2.9 using DisCoCat. We map conjunctive queries to verb phrases in natural language, discuss suitable logical representations of the function words in these phrases, and describe the resulting algorithm. The result corresponds to a special case of the Graph Query Embedding method discussed in Section 2.2.3, with a conjunctions handled via a certain fixed operator in place of the learnable aggregation function in Equation 2.13. Importantly, the desirable theoretical properties discussed in Section 2.2.3 for GQE still hold for this special case.

Note that there is existing work on representing conjunctive queries in DisCoCat models: Felice et al. considered closely related problems about answering conjunctive queries [23]. However, they focused on exact calculation of satisfactory entities based on a known set of facts rather than approximate learning-based approaches for incomplete knowledge graphs, and looked at a much less restricted class of queries that they showed to be NP-complete.

3.3.1 Conjunctive queries as sentences

To define natural language phrases for any tree conjunctive query, we define a linguistic interpretation mapping each relation r to a transitive verb $[\![r]\!]_L$ and each entity e to a noun $[\![e]\!]_L$ as before, and furthermore we will map each query $q(\cdot)$ to an intransitive verb phrase $[\![q(\cdot)]\!]_L$ as follows:

$$\begin{aligned} [\![r_i(\cdot, e_j)]\!]_L &= \text{“}[\![r_i]\!]_L\text{s } [\![e_j]\!]_L'' \\ [\![\exists y \,.\, r_i(\cdot, y) \wedge q'(y)]\!]_L &= \text{“}[\![r_i]\!]_L\text{s somebody that } [\![q'(\cdot)]\!]_L'' \\ [\![\bigwedge_k q_k'(\cdot)]\!]_L &= \text{“}[\![q_1'(\cdot)]\!]_L \text{ and } \ldots \text{ and } [\![q_n'(\cdot)]\!]_L'' \end{aligned} \tag{3.1}$$

Example 4. Consider the query

$$q(x) = \exists y \,.\, L(x, y) \wedge (K(y, a) \wedge H(y, b)), \tag{3.2}$$

where a, b are entities and L, K, H are relations with

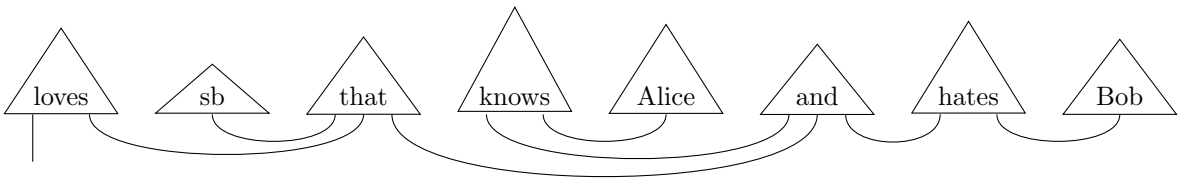
$$[\![a]\!]_L = \text{“Alice”}, \quad [\![b]\!]_L = \text{“Bob”}, \tag{3.3}$$

$$[\![L]\!]_L = \text{“loves”}, \quad [\![K]\!]_L = \text{“knows”}, \quad [\![H]\!]_L = \text{“hates”}. \tag{3.4}$$

Then $[\![q(\cdot)]\!]_L =$ “loves somebody that (knows Alice and hates Bob)”, with brackets only added for disambiguation.

Remark 3. Some sentences, such as the above example, are grammatically ambiguous without parentheses. We will not be concerned with this, sentences will implicitly be treated as appropriate parse trees rather than sequences of tokens.

Answering queries then corresponds to answering “Who $[\![q(\cdot)]\!]_L?''$. As discussed in multiple DisCoCat works [23, 19], if sentence embeddings represent truthiness, we can simply answer this by finding the entity e that maximises the inner product $\langle e|q\rangle$ where $|e\rangle, |q\rangle$ are the embeddings of the entity and the verb phrase respectively. So we need to calculate the embedding $|q\rangle$. E.g., for our Example 4, this is given by the following diagram:



3.3.2 Encoding function words

Besides the words representing entities and relations, the sentences we consider contain the words *“that”, “and”*, and *“somebody”*, which we consider function words. In this section, we will propose specific representations for these words based on prior work that have desirable logical properties.

Encoding *"that"*

For the relative pronoun *"that"*, we follow the proposal of Sadrzadeh et al. [54]. In the case of a 1-dimensional sentence space, their construction reduces to $\sum_{i \in \mathbb{B}_n} |iii\rangle$, where $\mathbb{B}_n$ is the set of computational basis vectors for the space of noun/entity embeddings. This is typically represented with a white dot with three wires in DisCoCat diagrams.

As described by Equation 2.2, this results in the embedding for a phrase of the form "[noun] that [verb]s"to be given by the elementwise multiplication $\boldsymbol{n} \odot \boldsymbol{v}$, where $\boldsymbol{n}$ is the embedding of the noun and $\boldsymbol{v}$ is that of the verb.

Encoding *"somebody"*

To encode *"somebody"*, we observe that in the truth-theoretic setting, the embedding for "somebody that [verb]s" should be fully aligned with "[verb]s", because their dot product gives the truth value of a tautology. For example, the sentence "Somebody that likes Alice likes Alice" is a tautology. Hence, given we want to satisfy the following for any $\boldsymbol{v}$:

$$\overrightarrow{\text{somebody}} \odot \boldsymbol{v} = \boldsymbol{v}, \tag{3.5}$$

which implies $\overrightarrow{\text{somebody}} = \sum_{i \in \mathbb{B}_n} |i\rangle$. This is typically represented as a white dot with a single wire.

Encoding *"and"*

Finally, we need to choose a representation for *"and"*. To motivate this, consider the following two phrases:

1. "(somebody that $[\![q_1(\cdot)]\!]_L$s) that $[\![q_2(\cdot)]\!]_L$s"
2. "somebody that ($[\![q_1(\cdot)]\!]_L$s and $[\![q_2(\cdot)]\!]_L$s)"

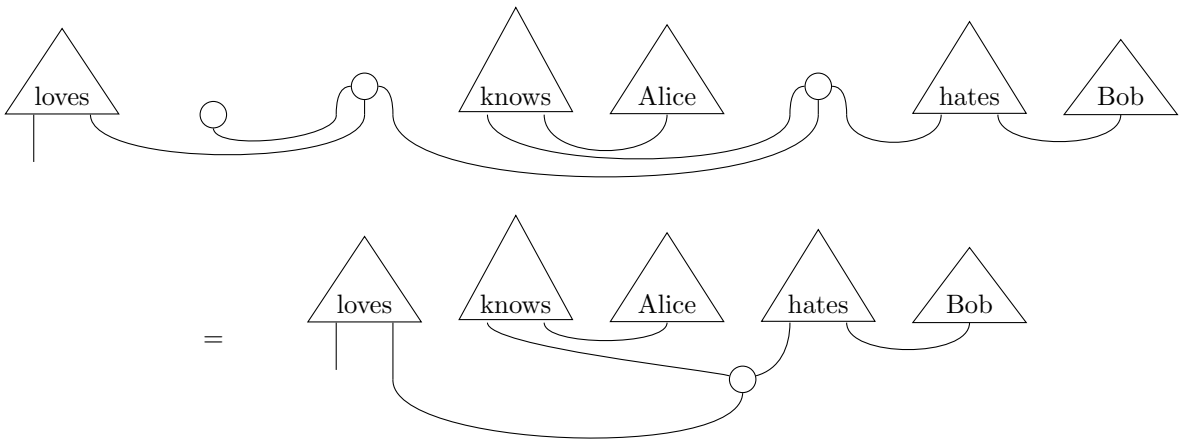
Based on the representations of *"somebody"* and *"that"*, the first one evaluates to $\overrightarrow{[\![q_1(\cdot)]\!]_L} \odot \overrightarrow{[\![q_2(\cdot)]\!]_L}$, and the second one is equal to $\overrightarrow{[\![q_1(\cdot)]\!]_L} \times \overrightarrow{\text{and}} \times \overrightarrow{[\![q_2(\cdot)]\!]_L}$, which is what we are essentially seeking to define here.

Since these two are intuitively semantically equivalent, we should expect them to be equal for any query representations (or other verb phrases). This is only achieved if *"and"* is also represented by $\sum_{i \in \mathbb{B}_n} |iii\rangle$, just like *"that"*. This also has the desired property of associativity: $\overrightarrow{\text{A and (B and C)}} = \overrightarrow{\text{(A and B) and C}} = \overrightarrow{\text{A}} \odot \overrightarrow{\text{B}} \odot \overrightarrow{\text{C}}$.

Note that our choices of function word representations rely on our restricted context: computing logical meanings over a limited grammar. In general natural language, words such as *"and"* can have more complex behaviour that is much harder to capture.

3.3.3 Deriving a special case of Graph Query Embedding

We can substitute and simplify our diagrams using these constructions. Consider Example 4:



(3.6)

We can also directly express how this method builds up a query embedding $[\![q(\cdot)]\!] \in \mathbb{R}^d$ from the learned entity embeddings $[\![e_i]\!] = \boldsymbol{e}_i \in \mathbb{R}^d$ and relation embeddings $[\![r_i]\!] = \boldsymbol{R}^{(i)} \in \mathbb{R}^{d \times d}$, where d is the noun embedding dimension of our DisCoCat model:

$$\begin{aligned} [\![r_i(\cdot, e_j)]\!] &= \boldsymbol{R}^{(i)} \boldsymbol{e}_j \\ [\![\exists y \,.\, r_i(\cdot, y) \wedge q'(y)]\!] &= \boldsymbol{R}^{(i)} [\![q'(\cdot)]\!] \\ [\![\bigwedge_k q'_k(\cdot)]\!] &= \bigodot_k ([\![q'_k(\cdot)]\!]). \end{aligned} \tag{3.7}$$

The first two of these equations are exactly the same as Equations 2.11 and 2.12 defining GQE, the only difference is the third one, where we have a fixed aggregator rather than the very general neural set function in Equation 2.13.

Furthermore, recall the theoretical properties of GQE discussed in Section 2.2.3: for any set of training facts, an appropriate parameterisation can ensure that exactly the facts derivable from the known relations will be predicted a positive score [28]. This still holds for our restricted model – our elementwise product aggregator has the same properties as the min function used by Hamilton et al. in their construction, that it simulates a bitwise and for boolean vectors. The rest of the proof summarised at the end of Section 2.2.3 follows without any changes.

Chapter 4

Summary and Outlook

We have shown how the quantum-inspired NLP method DisCoCat can adapted for solving knowledge graph embedding tasks. The algorithms derived this way turned out to be equivalent to existing KGE methods – so although this did not lead any proposals for new methods in this project, it gives a novel perspective on existing work. It is also a surprising observation, since DisCoCat was proposed earlier [21].

This correspondence could be further explored in future work, cross-pollinating both fields with new ideas. For example, the category theoretic perspective of the DisCoCat community could be particularly useful for finding suitable spaces for embedding the logical structures of complex queries – as discussed in the second half of Section 2.2.3, many existing complex query embedding methods can be seen as ways to overcome the limitations of vector spaces for representing logic. The diagrammatic representations of monoidal category theory are also a useful tool for understanding linear KGE models and potentially exploring further alternatives. It even allows us to reason about higher-rank tensors, which is useful for embedding *knowledge hypergraphs*, where some relations might involve more than two entities – a setting that has attracted an increasing amount of attention in recent years [1, 37].

The connection between DisCoCat and KGE methods also lets us bring ideas from the field of relational learning to the DisCoCat context. For example, TuckER has achieved great results in the KG domain, which could motivate an analogous 'inner wiring' of DisCoCat word representations. This could reduce the number of parameters used for high-rank tensors, which is important for scaling DisCoCat models. Furthermore, the performance of TuckER has been shown to be partly thanks to smart regularisation methods, such as dropout and batch regularisation used in the process of tensor reductions [53] – these methods could also be applied to DisCoCat when executed on classical hardware.

Part II

Quantum Graph Neural Networks

Chapter 5

Background and Related Work

5.1 Graph neural networks

5.1.1 Learning over graph-structured data

In the last decades, we have seen the development of a wide range of neural network architectures designed to operate on different kinds of data. For example, convolutional networks rely on the pixel grids of images, while recurrent neural networks are used to process sequential data. There is also a wealth of data with graph-like structure, including knowledge graphs [14], social networks [71], traffic patterns [26] and molecules [66], which motivated a new class of deep neural networks: graph neural networks.

It is well-known that the generalization performance of machine learning models heavily depends on their *inductive bias*: architectural decisions that allow a learning algorithm to prioritize solutions satisfying certain desired criteria, independent of the observed data [46]. For example, for processing image data, one desirable property is the so-called translation-invariance which ensures that the patterns detected will remain valid to potential shifts. Even regularisation terms can be seen as a form of inductive bias [30].

In the context of graphs, the desirable property is invariance: the ordering of nodes and edges in the input should be considered a representational detail that does not affect our predictions [8]. We will formalize the properties to ensure such invariances for different graph representation learning tasks, which can be grouped under two categories of graph-level and node-level tasks.

Graph-based tasks include supervised *graph classification* or *graph regression*, where a large set of individual graphs are given, usually with feature vectors associated with each node, defining *node features*. A ground-truth label is known for each graph, which could be a Boolean or other discrete value for classification tasks, or a real number for regression tasks. For example, we might want to classify molecular graphs by whether they are toxic (true) or not (false), an example of

binary classification [66]. After training on a subset of the available data, we evaluate the model on a held-out test set to see how well it generalises.

In graph classification/regression, we want to restrict our models to learning *invariant functions*, which provide the same output upon permuting the nodes of a graph. Formally, given a graph with adjacency matrix $\boldsymbol{A}$, we say a function f is *invariant* if for any permutation matrix $\boldsymbol{P}$ we have:

$$f(\boldsymbol{A}) = f(\boldsymbol{P}^T \boldsymbol{A} \boldsymbol{P}). \tag{5.1}$$

In principle, a less structured model that can represent arbitrary functions (e.g. multi-layer perceptrons [32, 31]) could potentially learn this invariance property from the data it's given. However, this would typically require far more training data and make it harder to generalise to unseen graphs, so it is better to build this inductive bias into the architecture itself [8, 29].

In node-based tasks, we need to predict some property of each individual node of a given graph. Similarly to graphs, we can distinguish between *node classification*, where the output is a Boolean or one of a small finite set of possible labels; and *node regression*, where we predict real numbers. In most common benchmarks, one large graph is given, but only a subset of node labels are known, and the task is to predict the missing labels. We can further split these problems into *transductive* and *inductive* settings: in a transductive setting, the unlabeled nodes are present during training, while in the inductive case they are only added at test time [69, 29].

In node classification/regression, we want our models to learn *equivariant functions*, where a permutation of the input nodes results in the same permutation of output predictions, so for any permutation we have:

$$\boldsymbol{P}^T f(\boldsymbol{A}) = f(\boldsymbol{P}^T \boldsymbol{A} \boldsymbol{P}). \tag{5.2}$$

5.1.2 Message-passing neural networks

Early work approached the problems of effective graph representation learning from many directions, with perspectives from dynamical systems [57], signal processing [16], and convolutional neural networks [34] among others. Eventually the field arrived at a framework of highly effective methods called *Message-Passing Neural Networks* (MPNNs) [27] that iteratively update the representations of each node based on their local neighbourhoods.

In an MPNN, each node v is assigned some initial state vector $\boldsymbol{h}_v^{(0)}$ based on its features. This is repeatedly updated based on the current state of its neighbours $\mathcal{N}(v)$ and its own state. A function $\textsc{aggregate}(\cdot)$ combines the multiset of states from the neighbours into a *message vector* $\boldsymbol{m}_{\mathcal{N}(v)}$ passed to the node v, then an $\textsc{update}(\cdot)$ function combines the message with the current state to get the next one. This is repeated over several layers, usually involving different learned

functions $\text{AGGREGATE}^{(k)}, \text{UPDATE}^{(k)}$:

$$\boldsymbol{h}_v^{(k+1)} = \text{UPDATE}^{(k)}\Big(\boldsymbol{h}_v^{(k)}, \text{AGGREGATE}^{(k)}\big(\{\!\{\boldsymbol{h}_u^{(k)} \mid u \in \mathcal{N}(v)\}\!\}\big)\Big) \tag{5.3}$$

$$= \text{UPDATE}^{(k)}\Big(\boldsymbol{h}_v^{(k)}, \boldsymbol{m}_{\mathcal{N}(v)}\Big), \tag{5.4}$$

where the double braces $\{\!\{\cdot\}\!\}$ refer to a multiset.

Observe that each MPNN layer is equivariant by construction: we map the feature vectors to a new vector for each node, and at no point are we referring to the ordering of nodes or edges – we treat them as sets.

The specific form of the aggregate and update functions varies across approaches, but it is always a differentiable function with learnable parameters that can be trained like other neural network architectures. After several such layers have been applied, the final node embeddings might be used for predicting individual node-level properties, or they might be pooled to form a graph embedding vector to predict properties of entire graphs. The pooling often takes the form of simple averaging, summing or elementwise maximums, but neural methods such as deep sets [70] are also used: the only requirement is that it has to be an invariant function.

A wide range of highly effective models have been proposed within this framework, making different trade-offs and bringing ideas from various neural methods: Graph Convolutional Networks were inspired by convolutional neural networks [34], Graph Attention Networks brought attention mechanisms to the graph domain [63], and Gated Graph Neural Networks used recurrent neural networks to construct their update function [36], among many others. While the choice of aggregate and update functions leads to important differences between these architectures, the shared underlying structure also creates similarities in their expressivity as we will see later.

5.1.3 Invariant and equivariant graph networks

Maron et al. took a different approach to graph-based learning [42]: they considered information about a graph encoded in arbitrary k-order tensors indexed by node identifiers along each dimension. An adjacency matrix is then a simple example in the $k = 2$ case, while individual node features can be described with $k = 1$.

They generalised the invariance/equivariance conditions of Equations 5.1 and 5.2 to higher-order tensors $\mathbf{A} \in \mathbb{R}^{n^k}$, where n is the number of nodes in the graph. They defined a reordering operator: $\boldsymbol{P} \star \mathbf{A}$ is defined to be the tensor that results from renumbering the nodes according to the permutation given by $\boldsymbol{P} \in \mathbb{B}^{n^2}$. A function f is then *invariant* if $f(\boldsymbol{P} \star \mathbf{A}) = f(\mathbf{A})$ for any $\boldsymbol{P}, \mathbf{A}$, and *equivariant* if $f(\boldsymbol{P} \star \mathbf{A}) = \boldsymbol{P} \star f(\mathbf{A})$.

To define their model, they used *invariant/equivariant linear layers*: linear transformations of the form $\mathbf{L}_i \in \mathbb{R}^{n^{k_i} \times n^{k_{i+1}}}$ that satisfy the respective conditions. The architecture was composed

as follows:

$$F = \text{MLP} \circ \mathbf{I} \circ \mathbf{E}_d \circ \sigma \circ \cdots \circ \sigma \circ \mathbf{E}_2 \circ \sigma \circ \mathbf{E}_1, \tag{5.5}$$

where $\mathbf{E}_i$ are equivariant linear layers for each $1 \leq i \leq d$, $\mathbf{I}$ is an invariant linear layer, σ represents an elementwise nonlinearity, and MLP is a multi-layer perceptron to process the aggregated result.

It is not obvious how to enforce the invariance/equivariance equations over linear maps, so this became their main focus. They showed that this can be achieved by having each weight depend only on the *equality pattern* of the indices identifying the specific tensor element, and that such a parameterisation can represent any invariant/equivariant linear layer. Intuitively, this ensures that applying a permutation to all nodes leaves them seeing the same set of weights, since the equality patterns are unchanged.

Example 5. Consider tensors of order $k = 1$ in both the input and the output (which are simply vectors), in which case our linear layer is a matrix $\boldsymbol{M}$, and we want to characterise the subclass of equivariant matrices. Each parameter $\boldsymbol{M}_{ij}$ is identified by two indices i, j, which can have just two equality patterns: $i = j$ or $i \neq j$. So the space of such equivariant linear layers are just two-dimensional: for all i, j, $\boldsymbol{M}_{ij} = \alpha_1$ if $i = j$, and $\boldsymbol{M}_{ij} = \alpha_2$ for $i \neq j$, for some α_1, α_2.

In general, this result implies that once the size of the graph is greater than the order k of the tensors, the dimensionality of the space of equivariant/invariant linear layers does not increase: it is bounded by the number of equality patterns over the indices of the tensors defining each layer. This is the same as the number of partitions of a finite set, also known as the Bell numbers $b(k)$. Invariant layers from an input tensor of order k to a real output are given by tensors of rank k, so the dimensionality of the space of such invariant layers is $b(k)$; equivariant layers between input and output tensors of rank k are given by tensors of rank $2k$, so the dimensionality of their space is $b(2k)$.

5.2 Expressive power of graph neural networks

5.2.1 The Weisfeiler-Lehman isomorphism test and GNNs

Despite their successes, MPNNs and many other GNN architectures have been shown to have limitations: there are functions over graphs that they can never represent at any finite width and depth, no matter how we set their parameters. This was shown by relating the expressive power of MPNNs to the distinguishing power of the Weisfeiler-Lehman (WL) test for graph isomorphism [67, 43].

The WL test [65] was originally developed as an efficient heuristic algorithm for determining whether two graphs are isomorphic; this is a hard problem with no known polynomial-time solution (see e.g. Babai for related breakthrough result [5]), so efficient methods that work for large classes of graphs are useful. In its simplest, 1-dimensional form (1-WL test), the algorithm iteratively updates labels assigned to nodes. Given two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, initially, node labels are determined by their features in the input; then, at each step, every node label gets updated via an injective hash function based on its current label and its neighbours' labels. Note the similarity to Equation 5.3:

$$h_v^{(k+1)} = \textsc{hash}\Big(h_v^{(k)}, \{\!\{\boldsymbol{h}_u^{(k)} \mid u \in \mathcal{N}(v)\}\!\}\Big) \tag{5.6}$$

Once this iteration stops refining the node states further, i.e., the number of distinct labels stops increasing for both graphs, the algorithm halts. (This can be shown to always happen in at most n steps for a graph with n nodes.) If there is some label occurring with a different frequency in $\mathcal{G}_1$ or $\mathcal{G}_2$ at this point, they are certainly not isomorphic; otherwise the 1-WL test cannot distinguish these graphs, and they are plausibly isomorphic.

As the simplest example of graphs that are not 1-WL distinguishable, consider Figure 5.1: $\mathcal{G}_1$ consisting of two 3-cycles, and $\mathcal{G}_2$ being a 6-cycle, with all nodes in both graphs having the same features. Initially they are all assigned some label l_0. All nodes have the same label and they all have exactly two neighbours also sharing this label, so they are all updated to some $\textsc{hash}(l_0, \{\!\{l_0, l_0\}\!\}) = l_1$, and still all the labels are the same. The algorithm would stop here since no refinement happened; but even if it continued, further iterations could never break this sameness. $\mathcal{G}_1$ and $\mathcal{G}_2$ will end with the same 6 labels in each graph, so they cannot be distinguished.

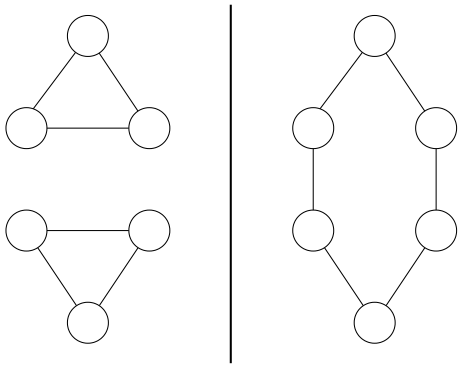


Figure 5.1: The simplest example of two graphs indistinguishable by the 1-WL test.

Note the similarity between this algorithm and the update mechanism of MPNNs. In the best-case scenario, the aggregate and update functions of an MPNN might combine to form an injective function, in which case it can give different predictions for any two graphs that are 1-WL distinguishable (assuming it also has sufficiently many layers). But it can never go beyond this: if two graphs appear the *same* to the 1-WL test, any MPNN will predict the exact *same* outputs for them regardless of how it was trained. For example, due to the example above, an MPNN can never learn to classify graphs by whether they contain a 3-cycle or not.

This result inspired many new methods. Xu et al. proposed the Graph Isomorphism Net-

work [67] (GIN), an MPNN designed to learn injective aggregate-update functions to maximise expressivity. The GIN aggregate-update functions are defined as follows:

$$\boldsymbol{h}_v^{(k)} = \mathrm{MLP}^{(k)}\Big(\big(1+\epsilon^{(k)}\big)\boldsymbol{h}_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} \boldsymbol{h}_u\Big) \tag{5.7}$$

They showed that injective functions of (element, multiset) pairs can be given in this form for many suitable choices of $\epsilon^{(k)}$ for bounded-size multisets from a countable domain. Hence by the universal approximation theorem of MLPs, GINs can learn injective aggregate-update functions over bounded-degree graphs with countable node feature spaces, reaching the full 1-WL expressivity achievable by any MPNN.

Others worked on methods to go beyond 1-WL: Morris et al. proposed higher-order GNNs inspired by higher-dimensional variants of the WL test, which are more expressive but prohibitively computationally expensive on most datasets [48]. Their later work proposed variants leveraging the sparsity found in many real-world domains, making them practical in the regime of small graphs, e.g., molecular learning, and showed great empirical results [47]. Maron et al. showed that their invariant and equivariant graph networks (discussed in detail in Section 5.1.3) also have expressivity tied to 1-WL and its higher-order variants (depending on the order of the tensors involved), and proposed *Provably Powerful Graph Networks*, a relatively efficient higher-order GNN based on the expressivity of matrix multiplication [43] with 2-WL expressivity, but their $O(n^3)$ time complexity and $O(n^2)$ space complexity is still fairly limiting. So despite this promising line of research, the most popular methods for learning on graphs are still MPNNs.

5.2.2 Node classification from a logical perspective

Barceló et al. improve on the WL results about MPNNs by considering what *node classifiers* MPNNs can represent and relating this to sentences in appropriately chosen logics [7].

We can use logics (such as first-order logic) to describe graphs by taking the domain elements to be vertices, and using a binary relation $E(\cdot,\cdot)$ to represent the edges. Then, sentences with exactly one free variable can be viewed as *logical node classifiers*:

Example 6. The sentence

$$\phi(x) = \exists y.\big(y \neq x \wedge E(x,y) \wedge \exists z.(z \neq y \wedge z \neq x \wedge E(y,z) \wedge E(z,x))\big) \tag{5.8}$$

holds for a vertex v (to be instantiated in place of the variable x) if and only if it is in a triangle.

We can formally relate the expressive power of MPNNs to logical classifiers. We say an MPNN M making binary predictions $M(G)_v$ for a node v of a graph G *captures a logical classifier* $\phi(x)$ if for all vertices v of the graph, $M(G)_v = \phi(v)$. And a class C of models (e.g. MPNNs) *captures a*

logic L if for any logical classifier $\phi(x)$ defined in that logic, there exists some model $M \in C$ that captures $\phi(x)$.

The primary result of Barceló et al. concerns the logic C defined by adding counting quantifiers $\exists^{\geq k} x$ to first-order logic, where the sentence $\exists^{\geq k} x.\phi(x)$ is true if there exist at least k different elements x such that $\phi(x)$ holds. For example, the sentence $\exists^{\geq 5} y.E(x,y)$ holds for nodes x with degree at least 5.

More specifically, they focused on C^2, a restriction of C where only two variables can be used (so e.g., the triangle example involving x, y and z is not in C^2 – in fact, such a classifier cannot be defined there). This logic is known to be closely linked to the 1-WL test: the 1-WL test assigns different labels to two nodes in a graph if and only if those nodes can be distinguished by some C^2-classifier [18]. So based on previous results, it is clear that MPNNs (of the standard form given in Equation 5.3) cannot capture any logical classifiers that cannot be represented in C^2.

However, the reverse does not follow: although sufficiently powerful MPNNs can reproduce the labels of the WL test, they can not reproduce certain labeling schemes that are refined by the algorithm.

Example 7. Consider the classifier $\phi(x) = \exists y.\neg\exists x.E(x,y)$. Note that x is only used within a quantified scope, so the free variable x is unused and this logical classifier assigns the same thing to all nodes: whether there is an isolated vertex in the graph.

However, MPNNs cannot propagate the information about its existence, since they only pass information along edges. In fact, Barceló et al. show that the logical classifiers captured by standard MPNNs are exactly those of *graded modal logic*, a strict subset of C^2 where every existential quantifier has to be 'guarded' by a relation asserting the existence of an edge.

To improve on this, Barceló et al. propose MPNNs *with global readout*, or ACR-GNNs in their nomenclature (with ACR standing for Aggregate-Combine with Readout). These extend the standard MPNN equation as follows:

$$\boldsymbol{h}_v^{(k+1)} = \textsc{update}^{(k)}\Big(\boldsymbol{h}_v^{(k)}, \textsc{aggregate}^{(k)}\big(\{\!\!\{\boldsymbol{h}_u^{(k)} \mid u \in \mathcal{N}(v)\}\!\!\}\big), \textsc{readout}^{(k)}\big(\{\!\!\{\boldsymbol{h}_u^{(k)} \mid u \in \mathcal{V}\}\!\!\}\big)\Big) \tag{5.9}$$

At each step, an invariant readout function aggregates the current state of all nodes and the result of this aggregation is passed to all nodes. They show that this extension allows MPNNs to capture all C^2-classifiers. They provide a construction based on marking whether each node satisfies certain subexpressions of the classifier in their state, aggregating the total number of nodes satisfying clauses in existential quantifiers into the readout, and using this to relay information about non-adjacent nodes. In each layer, subexpressions with increasing quantifier depths are

effectively 'evaluated' at each node, so we get the required result after a number of layers equal to the quantifier depth.

Finally, they show that it even suffices to have the readout calculation only in the final layer. To do this, they use the non-final layers to collect all the possible information about each node's neighbourhood into its state, using integer functions defined in terms of prime factorisations to squeeze all this information into a single dimension of the state, essentially calculating a Gödel encoding of its neighbourhood[1]. The final layer's readout then aggregates how many times any substructure up to a certain size occurs in the graph, which is combined with local information at each node to do the classification. The size of the neighbourhoods needed is still a function the quantifier depth of the logical quantifier in question.

5.2.3 The power of randomisation

A recent approach to increasing the expressivity of GNNs relies on randomly initialising some or all node features [56, 2]. Although this makes the model outputs nondeterministic, we can talk about their capacity to approximate deterministic functions.

For some deterministic Boolean function f over arbitrary graphs, a configuration $\hat{f}$ of a randomised model is a δ-*approximation* of f if for any graph $\mathcal{G}$, $\hat{f}(\mathcal{G}) = f(\mathcal{G})$ with probability at least $1 - \delta$. If such a configuration exists for arbitrarily small δ, we say that the randomised model can represent f. We can also talk about real-valued functions by requiring the output $\hat{f}(\mathcal{G})$ to be within some ϵ with probability at least $1 - \delta$.

Sato et al. showed that by randomly initialising some features of MPNNs, sufficiently large models can represent any substructure-counting function with high probability [56]. This is significant because deterministic MPNNs cannot even tell whether a graph contains a triangle, as discussed above. Abboud et al. improved on this result in the case of MPNNs with a readout layer (as in Equation 5.9): they showed that such MPNNs can approximate *any* Boolean or real-valued function over graphs with arbitrary precision with high probability [2].

Both of these results are intuitively based on the idea that random node initialisation with sufficiently many parameters will assign unique initial states to different nodes with high probability. This can act as a sort of 'identifier' for each node: for example, after 3 rounds of message-passing, a node might be able to tell that its own identifier was seen at a 3-hop distance, so it must be in a triangle [56]. For arbitrary functions over graphs, Abboud et al. give a construction using the C^2 logic analysed by Barceló et al., based on the fact that any graph with all nodes having unique

[1] Although this can indeed be approximated by neural networks *in principle*, it also relies on arbitrary-precision arithmetic for arbitrary-size graphs. To fit such an encoding into a fixed amount of bits, a restriction to degree-bounded graphs is needed.

random features can be identified by a C^2-sentence.

One disadvantage of these approaches is that they make it harder to train the models. The model needs to see many different random labelings to eventually become robust to this variation and stop it from influencing the predictions in undesirable ways. The extent of this effect can be mitigated by using fewer randomised features, but this comes at the cost of making it more difficult to discern properties that need this randomisation [2].

5.3 Quantum machine learning

In recent years, the field of quantum computing has made significant steps towards practical usefulness, which has sparked increasing interest in potential applications. Although fault-tolerant quantum computers will likely not be available soon, there is a lot of research into more near-term applications of the *noisy intermediate-scale quantum computers* (NISQ) available today and in the foreseeable future.

One area where many people hope for the NISQ devices to have an impact is machine learning [51, 10]. Although the current state of the art of quantum hardware is not sufficient to gain an advantage over classical methods (despite recent achievements of quantum supremacy for carefully selected tasks [4]), there is already plenty of research based on small-scale experiments done by either classical simulation or on the currently available small-scale noisy hardware.

Most of these quantum machine learning proposals rely on *parameterised quantum circuits*: circuits where the structure is fixed, but individual gates include free parameters such as $Z(\alpha)$-rotations that can be tuned for a specific task [59, 10]. These can be used in a training loop very similarly to classical machine learning architectures: training samples are encoded into the input state, then the circuit is run, measurements are made and compared with the ground-truth labels. Lacking the possibility of standard backpropagation, there are alternative ways of calculating gradients [60, 11], and gradient-free optimisation methods are also used [50].

5.3.1 Quantum alternatives to GNNs

Within the field of quantum ML, a few methods have already been proposed for learning on graph-structured data. We briefly survey existing proposals, and in Section 6.2 we will relate them to the framework we introduce.

Beer et al. train quantum models for node prediction that are run individually on each node, with a graph-based loss term enforcing that adjacent nodes should get similar predictions [9]. Their framework works with any kind of quantum machine learning architecture, so this could be a promising way to enforce some regularity over the space of inputs of any model, and might

be applicable to semi-supervised node classification tasks where only some of the node labels are available. However, individual predictions are not influenced by adjacent nodes at inference time, which seriously limits expressivity, and the method cannot be applied for graph-level tasks.

Verdon et al. propose an intuitive analogue of classical GNNs [64]. They consider the total Hilbert space of their model to be a product of subspaces for each node, i.e., each node is assigned a set of qubits in case of typical discrete-variable quantum computing. They parameterise their circuits indirectly through a Hamiltonian defined as the sum of one and two-node operators that get applied based on the adjacency matrix of the given graph.

Although their general model includes the ability to learn individual weights for each node, for our project we are primarily interested in their *Quantum Graph Convolutional Neural Network* (QGCNN) variant, where the operators applied at each node are given by the same parameters. Combined with the fact that edge operators are applied to different qubits appropriately if we reorder the nodes of the graph, this makes their model equivariant. We discuss their setup in more detail in Section 6.2.2 where we also relate it to other possible models.

Finally, Zheng et al. very recently proposed a quantum graph convolutional neural network based on a similar premise, but they directly parameterise two-node unitaries to apply for each edge [72]. This is similar to approaches we discuss in Section 6.2.3, with the caveat that they use arbitrary unitaries which might not commute in general, and this breaks equivariance.

All of these models come with small-scale experiments, but little discussion of possible design choices for such models or theoretical considerations. In the following chapters, we consider possible equivariant quantum models and analyse the expressivity of interesting subclasses.

Chapter 6

Quantum Graph Architectures

6.1 Equivariant Quantum Graph Circuits

6.1.1 Model setup

Similarly to Verdon et al. [64] and Zheng et al. [72], we represent nodes in a quantum circuit as a joint system given by the tensor product of subsystems for each node. This lets us model complex interactions via entanglement between the node states in the exponentially large Hilbert space of the joint system. For each input graph, we prepare an initial state representing all node features and want to learn a quantum procedure that maps these to useful predictions.

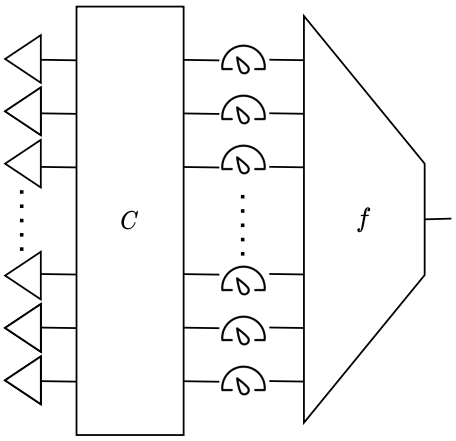


Figure 6.1: Overview of our model setup. A product state is prepared based on individual nodes, then an EQGC $\boldsymbol{C}$ is applied, the result is measured and aggregated by some classical function f to produce the result.

We will therefore set the initial state to be a product of node states encoding each node's features: for a graph of n nodes, each with features encoded in a vector $|v_i\rangle \in \mathbb{C}^s$ for some fixed integer s, the joint state is then $|\psi\rangle = \bigotimes_{i=1}^{n} |v_i\rangle \in \mathbb{C}^{s^n}$.

We restrict ourselves to making measurements at the end of the process, so we wish to learn unitary transformations that map this input to a useful output state where we can measure each node. Since we focus on graph classification/regression, the results from the measuring all nodes will be classically aggregated to produce a Boolean or real-valued prediction.

Importantly, we will allow this unitary to depend on the adjacency matrix of the input graph:

for example, our model might perform some two-node unitary in the positions corresponding to each edge. We only enforce that the procedure respects the graph structure by behaving equivariantly: permuting the nodes of a graph should permute the output the same way.

This brings us to our definition of *Equivariant Quantum Graph Circuits* (EQGCs):

Definition 1. An EQGC is an arbitrary function $\boldsymbol{C}(\cdot)$ mapping an adjacency matrix $\boldsymbol{A} \in \mathbb{B}^{n\times n}$ to a unitary $\boldsymbol{C}(\boldsymbol{A}) \in \mathbb{C}^{s^n \times s^n}$ for any n that behaves equivariantly in the following sense:

For any permutation p over n elements, consider its usual representation by $\boldsymbol{P} \in \mathbb{B}^{n\times n}$ as well as a larger matrix $\tilde{\boldsymbol{P}} \in \mathbb{B}^{s^n \times s^n}$ that reorders the tensor product, mapping any $|v_1\rangle |v_2\rangle \dots |v_n\rangle$ with $|v_i\rangle \in \mathbb{C}^s$ to $|v_{p(1)}\rangle |v_{p(2)}\rangle \dots |v_{p(n)}\rangle$. Then $\boldsymbol{C}$ must satisfy:

$$\boldsymbol{C}(\boldsymbol{A}) = \tilde{\boldsymbol{P}}^T \boldsymbol{C}(\boldsymbol{P}^T \boldsymbol{A} \boldsymbol{P}) \tilde{\boldsymbol{P}} \tag{6.1}$$

Remark 4 (A note on directedness)**.** Unlike many works on GNNs, our definition of EQGCs allows us to consider *directed* graphs naturally, and this will also be true of interesting subclasses we consider later. Of course, we can still easily operate on undirected data by either adding edges in both directions, or placing extra restrictions on our models. For the purposes of expressivity, we will still focus on classifying graphs in the undirected case, as this is better explored in previous works on classical methods.

Measurement and classical aggregation: To predict graph-level properties, we need to perform some permutation-invariant aggregation over the nodes. For the purposes of this project, we limited our analysis to a simple approach: measure each individual node state and perform invariant aggregation classically. The invariance condition restricts us to only rely on how many nodes we measured in each of their s possible states, so we assemble this data into a vector in $\mathbb{R}^s$ and apply a learned MLP to a predict a Boolean property. [1]

In the rest of this chapter, we focus on exploring and characterising the possible choices of such functions $\boldsymbol{C}(\cdot)$: Is the dependence on $\boldsymbol{A}$ necessary? What subclasses might we implement in practice? Can we parameterise them all uniformly, with a fixed number of parameters for unbounded n?

6.2 Special classes of equivariant quantum graph circuits

6.2.1 Circuits with arbitrary dependence on adjacency matrix

Our definition of EQGCs was very general, allowing the unitary to depend on the adjacency matrix of a graph via an arbitrary function as long as it acts equivariantly. In principle, this allows the

[1] Another option would be to do this in the quantum circuit, e.g. by bringing in fresh readout qubits and entangling them with the state of each node identically; we leave this to future work.

unitaries given for non-isomorphic graphs to be chosen completely independently. While this gives us a flexible framework to discuss possible approaches, we clearly need to choose some further restricted subclass if we hope to inductively generalise to unseen graphs, and to represent models with a finite number of parameters. [2]

The following section discusses some interesting EQGC subclasses that might be good candidates for training graph models, as well as their relations to each other.

6.2.2 Circuits parameterised via Hamiltonians

Unitaries are tricky to parameterise directly: for example, a linear combination of unitaries is not unitary in most cases. One alternative is to use the fact that any unitary $\boldsymbol{U}$ can be expressed using its Hamiltonian $\boldsymbol{H}$, a Hermitian matrix of the same size such that $\boldsymbol{U} = \exp(-i\boldsymbol{H})$. We can let the Hamiltonian depend linearly on the adjacency matrix, with Hermitian operators applied based on the structure of the graph:

Definition 2. An *Equivariant Hamiltonian Quantum Graph Circuit* (EH-QGC) is an EQGC given by a composition of finitely many layers $\boldsymbol{C} = \boldsymbol{L}_k \circ \cdots \circ \boldsymbol{L}_1$, with each $\boldsymbol{L}_i$ for $1 \leq i \leq k$ given as:

$$\boldsymbol{L}_i = \exp\Big(-i\big(\sum_{(j,k)\in\mathcal{E}} \boldsymbol{H}_{j,k}^{(\text{edge})} + \sum_{v\in\mathcal{V}} \boldsymbol{H}_v^{(\text{node})} \big) \Big), \tag{6.2}$$

where $\mathcal{V}, \mathcal{E}$ are the sets of nodes and edges respectively, $\boldsymbol{H}^{(\text{edge})}$ and $\boldsymbol{H}^{(\text{node})}$ are learnable Hermitian matrices over one and two-node state spaces, and the indexing $\boldsymbol{H}_{j,k}^{(\text{edge})}, \boldsymbol{H}_v^{(\text{node})}$ refers to the operators applied at the specified node(s). For example, $\boldsymbol{H}_3^{(\text{node})} = \boldsymbol{I} \otimes \boldsymbol{I} \otimes \hat{\boldsymbol{H}}^{(\text{node})} \otimes \boldsymbol{I}$ in the case of $n = 4$ nodes. This means that if the graph is permuted, the operators will be applied at changed positions appropriately, hence making the model equivariant.

Remark 5. We could in principle consider more general forms of equivariant Hamiltonians: for example, we could allow any Hermitian matrix that is also an equivariant layer as explored in Section 6.3, or add three-node terms for each triangle in the graph based on the adjacency matrix (or look at even more complex patterns). However, the EH-QGC construction is already very flexible while limiting multi-node interactions to scale linearly with the number of edges, allowing for efficient implementation on a quantum computer.

This is closely related to the approach taken by Verdon et al. [64], and their *Quantum Graph Convolutional Neural Network* (QGCNN) model. They define the Hamiltonian for a QGCNN layer as follows:

$$\boldsymbol{H} = \sum_{(j,k)\in\mathcal{E}} \sum_{r\in\mathcal{I}_{\text{edge}}} w_r \hat{\boldsymbol{O}}_j^{(r)} \otimes \hat{\boldsymbol{P}}_k^{(r)} + \sum_{v\in\mathcal{V}} \sum_{r\in\mathcal{I}_{\text{node}}} z_r \hat{\boldsymbol{R}}_v^{(r)}, \tag{6.3}$$

[2] In fact, we show in Section 6.3.3 that we cannot represent all unitaries over arbitrarily large graphs with a finite number of parameters even without a dependence on the adjacency matrix.

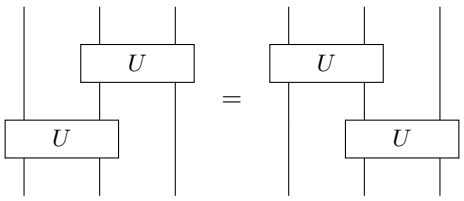
where the w_r, z_r are learnable parameters; the Hamiltonians $\hat{\boldsymbol{O}}, \hat{\boldsymbol{P}}, \hat{\boldsymbol{R}}$ are predefined fixed operators indexed by the fixed sets $\mathcal{I}_{\text{edge}}, \mathcal{I}_{\text{node}}$. So a fixed set of node and edge operators is chosen, and the training process determines their respective weights. Different fixed operators for any given experiment, and specific instantiations were chosen so that they commute in order to make the matrix exponential separable.

Using such a more restricted Hamiltonian makes circuit compilation easier and makes it more feasible to execute such models on today's noisy intermediate-stage quantum computers. But in principle, this is not necessary: approximating the more general EH-QGC Hamiltonian with a circuit would be more complicated and expensive, but would still be doable on a fault-tolerant quantum computer. We can even construct QGCNNs following Equation 6.3 with a large set of fixed operators spanning the entire space of one and two-node Hermitian matrices, essentially recovering Equation 6.2. Therefore for the purposes of our theoretical analysis, we will consider the general EH-QGC model, but our expressivity results will also apply to sufficiently powerful QGCNN instantiations as defined by Verdon et al.

6.2.3 Circuits parameterised via commuting unitaries

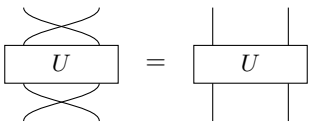
A similar, but more direct approach would be to consider two-node unitaries instead of Hamiltonians and apply the same learned unitary for each edge of the graph. The locality of two-node unitaries also has the same benefits as the two-node Hamiltonians mentioned previously: the number of operations scales linearly with the number of edges in a graph. This is also the approach taken by Zheng et al., but we need to add extra conditions that they do not consider to ensure equivariance [72].

Specifically, we need to enforce that the order we apply these unitaries does not matter. This gives us the following commutativity condition for a unitary $\boldsymbol{U}$:



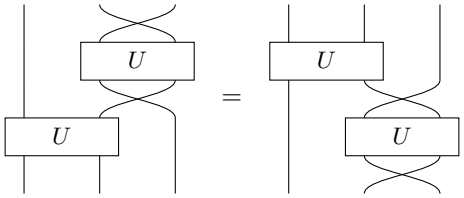
(6.4)

Furthermore, note that we might be working with directed graphs or undirected graphs as mentioned in Remark 4. If the graphs are undirected, we should ensure the following to make sure the direction of the edge representation does not affect our predictions:

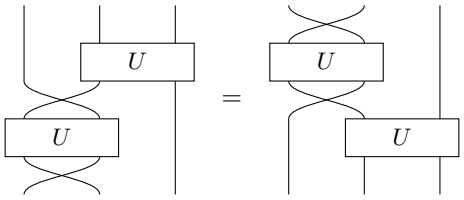


(6.5)

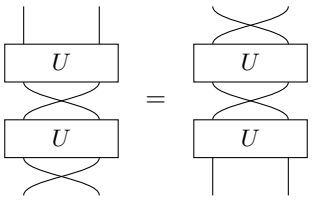
In the case of directed graphs, we need to satsify the following conditions instead:



(6.6)

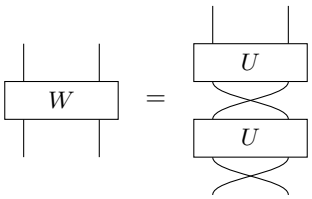


(6.7)

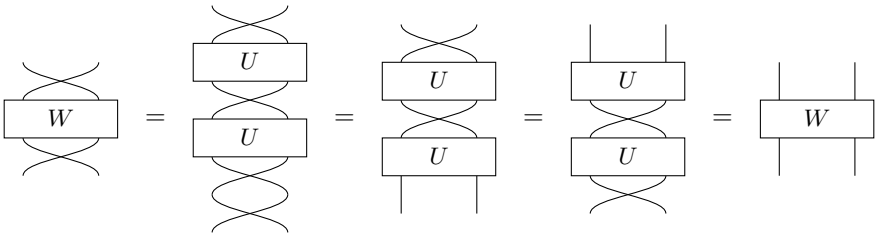


(6.8)

Of course, such a directed unitary can also be used for directed graphs by applying it in both directions: in fact, if Equation 6.8 is satisfied, this composition itself satisfies the undirected Equation 6.5:



(6.9)



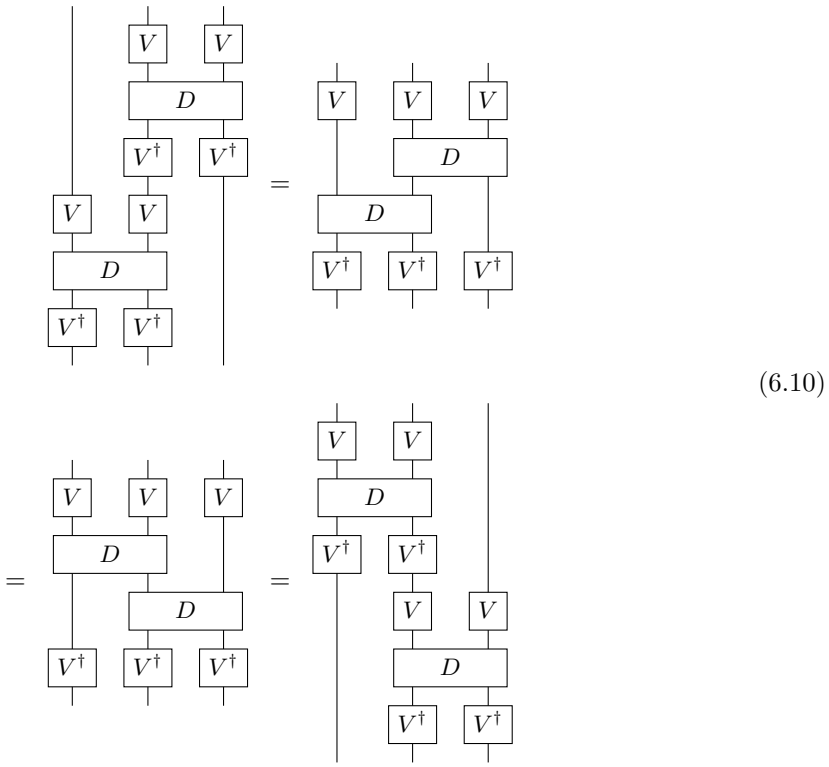
Equivariantly Diagonalizable Unitary Quantum Graph Circuits

It is not clear whether we can nicely parameterise the space of all such commuting unitaries, but we can focus on a further subclass.

Definition 3. An *equivariantly diagonalisable unitary* is a unitary that can be expressed in the form $\boldsymbol{U} = (\boldsymbol{V}^\dagger \otimes \boldsymbol{V}^\dagger)\boldsymbol{D}(\boldsymbol{V} \otimes \boldsymbol{V})$ for a unitary $\boldsymbol{V} \in \mathbb{C}^{s \times s}$ and diagonal unitary $\boldsymbol{D} \in \mathbb{C}^{s^2 \times s^2}$.

Note that all unitaries can be diagonalised in the form $\boldsymbol{U} = \boldsymbol{P}^\dagger \boldsymbol{D} \boldsymbol{P}$ for some other unitary $\boldsymbol{P}$ and diagonal unitary $\boldsymbol{D}$. The above is simply the case when $\boldsymbol{P}$ decomposes as $\boldsymbol{V} \otimes \boldsymbol{V}$ for one single-node unitary $\boldsymbol{V}$.

Using the facts that $\boldsymbol{P}^\dagger = \boldsymbol{V}^\dagger \otimes \boldsymbol{V}^\dagger$, that $\boldsymbol{I} \otimes \boldsymbol{D}$ is still a diagonal matrix and that diagonal matrices commute, we can see that equivariantly diagonalisable unitaries satisfy Equation 6.4:



(6.10)

The directed versions (Equations 6.6, 6.7, 6.8) are similar, since $\boldsymbol{V} \otimes \boldsymbol{V}$ and $\boldsymbol{V}^\dagger \otimes \boldsymbol{V}^\dagger$ commute with the swap, and then analogous derivations apply.

Note also that a square matrix is unitary if and only if all of its eigenvalues (the diagonal elements of $\boldsymbol{D}$) have absolute value 1. We can therefore parameterise these unitaries by combining arbitrary single-node unitaries $\boldsymbol{V}$ with diagonal matrices $\boldsymbol{D}$ of unit modulus numbers. (If we want to further add the inductive bias of undirected graphs, we can set $\boldsymbol{D}\left|e_1 e_2\right\rangle = \boldsymbol{D}\left|e_2 e_1\right\rangle$ for any computational basis vectors $\left|e_1\right\rangle, \left|e_2\right\rangle$.)

Definition 4. An *Equivariantly Diagonalisable Unitary Quantum Graph Circuit* (EDU-QGC) is an EQGC expressed as a composition of *node layers* $\boldsymbol{L}_{\text{node}}$ and *edge layers* $\boldsymbol{L}_{\text{edge}}$ given as follows on a graph with node and edge sets $(\mathcal{V}, \mathcal{E})$:

$$\boldsymbol{L}_{\text{node}} = \bigotimes_{v \in \mathcal{V}} \boldsymbol{V}_i = \boldsymbol{V}^{\otimes |\mathcal{V}|} \tag{6.11}$$

$$\boldsymbol{L}_{\text{edge}} = \prod_{(j,k) \in \mathcal{E}} \boldsymbol{U}_{jk} \tag{6.12}$$

In short, we either apply the same single-node unitary to all nodes, or we apply the same equivariantly diagonalisable unitary in the appropriate position for each edge. Since both types of layers are equivariant by construction, so is their composition, hence EDU-QGCs are a valid EQGC class.

Relation to EH-QGCs

It can be shown that EDU-QGCs are a subclass of the Hamiltonian-based EH-QGCs discussed in Section 6.2.2. This is particularly useful for investigating questions of expressivity: we also get a result about the expressivity of EH-QGCs by showing the existence of EDU-QGC constructions representing some function.

Theorem 1. *Any EDU-QGC can be expressed as an EH-QGC.*

To prove this, we consider node layers and edge layers separately and show that both can be represented by one or more EH-QGC layers. 1.1 proves the case for node layers; 1.2 for a special subclass of edge layers; finally, 1.3 builds on these two to prove the case for all edge layers.

Lemma 1.1. *Any node layer $\boldsymbol{L}_{node} = \boldsymbol{V}^{\otimes|\mathcal{V}|}$ (as defined in Equation 6.11) can be expressed as an EH-QGC layer.*

Proof. Let $|\mathcal{V}| = n$ and let $\boldsymbol{R}$ be the Hamiltonian for $\boldsymbol{V}$, then $\boldsymbol{H} = \boldsymbol{R}^{\otimes n} = \sum_{v\in\mathcal{V}} \boldsymbol{R}_v$ is an appropriate EH-QGC Hamiltonian (of the form defined in Equation 6.3). We can easily show $\boldsymbol{H}$ is then the Hamiltonian for the EDU-QGC layer $\boldsymbol{V}^{\otimes n}$:

$$\begin{aligned}
\exp(-i\boldsymbol{H}) &= \sum_{k=0}^{\infty} \frac{(-i\boldsymbol{H})^k}{k!} \\
&= \sum_{k=0}^{\infty} \frac{(-i)^k(\boldsymbol{R}^{\otimes n})^k}{k!} \\
&= \sum_{k=0}^{\infty} \frac{((-i\boldsymbol{R})^k)^{\otimes n}}{k!} \\
&= \Big(\sum_{k=0}^{\infty} \frac{((-i\boldsymbol{R})^k)}{k!}\Big)^{\otimes n} \\
&= \exp(-i\boldsymbol{R})^{\otimes n} \\
&= \boldsymbol{V}^{\otimes n}
\end{aligned}$$

□

Lemma 1.2. *Any diagonal edge layer $\boldsymbol{L}_{diag} = \prod_{(j,k)\in\mathcal{E}} \boldsymbol{D}_{jk}$, with a* diagonal *unitary applied for each edge, can be expressed as an EH-QGC layer.*

Proof. A diagonal unitary $\boldsymbol{D}$ has a diagonal Hamiltonian $\boldsymbol{R}$, where $\boldsymbol{D}_{jj} = \exp(-i\boldsymbol{R}_{jj})$. Using the fact that $\exp(\boldsymbol{A})\exp(\boldsymbol{B}) = \exp(\boldsymbol{A}+\boldsymbol{B})$ for commuting matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, and that all diagonal matrices commute, we will derive that applying the Hamiltonian $\boldsymbol{R}$ for each edge at the same time has the effect of applying the unitary $\boldsymbol{D}$ for each.

Consider two edges $\{(v_1, u_1), (v_2, u_2)\}$. The overall unitary we apply, with implicit identities on all other nodes, is

$$\begin{aligned}\boldsymbol{D}_{v_2u_2}\boldsymbol{D}_{v_1u_1} &= \exp(-i\boldsymbol{R}_{v_2u_2})\exp(-i\boldsymbol{R}_{v_1u_1}) \\ &= \exp(-i(\boldsymbol{R}_{v_2u_2} + \boldsymbol{R}_{v_1u_1}))\end{aligned}$$

This generalises easily to n nodes: the Hamiltonian of the overall unitary is $\sum_{(j,k)\in\mathcal{E}} \boldsymbol{R}_{jk}$ as required. □

Lemma 1.3. *Any edge layer $\boldsymbol{L}_{edge} = \prod_{(j,k)\in\mathcal{E}} \boldsymbol{U}_{jk}$ (as defined in Equation 6.12), with any equivariantly diagonisable unitary $\boldsymbol{U}$, can be expressed as an EH-QGC layer.*

Proof. This relies on Lemmas 1.1 and 1.2. We can show that a layer of equivariantly diagonalisable unitaries can expressed as a layer of diagonal unitaries sandwiched between two layers of single-node unitaries. Each of these can be represented as an EH-QGC layer by the previous lemmas, therefore giving us a 3-layer EH-QGC construction for this statement.

Consider an equivariantly diagonalisable unitary $\boldsymbol{U} = (mV^\dagger \otimes \boldsymbol{V}^\dagger)\boldsymbol{D}(\boldsymbol{V} \otimes \boldsymbol{V})$ applied for each edge in a layer $\prod_{(j,k)\in\mathcal{E}} \boldsymbol{U}_{jk}$. From the perspective of each node involved in edges, this decomposes as follows:

- a single-node unitary $\boldsymbol{V}$
- some number of two-node diagonal matrices separated by $\boldsymbol{V}^\dagger \times \boldsymbol{V} = \boldsymbol{I}$, which can be ignored
- a single-node unitary $\boldsymbol{V}^\dagger$

For nodes that are not part of any edge, we have the identity matrix that can be written as $\boldsymbol{V}^\dagger \times \boldsymbol{V}$. So we can rewrite the layer:

$$\prod_{(j,k)\in\mathcal{E}} \boldsymbol{U}_{jk} = \Big((\boldsymbol{V}^\dagger)^{\otimes n}\Big)\Big(\prod_{(j,k)\in\mathcal{E}} \boldsymbol{D}_{jk}\Big)\Big(\boldsymbol{V}^{\otimes n}\Big) \tag{6.13}$$

This is of the 3-layer form we discussed, proving the lemma. □

Proof of Theorem 1. Putting together Lemma 1.1 and 1.3 completes the proof: both types of EDU-QGC layers given by Equations 6.11 and 6.12 can be represented by one or more EH-QGC layers, so a sequence of EH-QGC layers can represent any EDU-QGC. □

6.3 Equivariant quantum set circuits

In this section, we consider restricting EQGCs so that they cannot depend on the adjacency matrix of the input graph. Since the connectivity information is not part of the input anymore,

these models can be considered as learning equivariant functions on sets, a different but also useful inductive bias. They could also serve as building blocks for EQGCs if the adjacency information is included in some other form, such as preparing an entangled initial state based on the input graph. Understanding their capabilities and limitations is therefore useful in multiple ways.

Definition 5. An *Equivariant Quantum Set Circuit* (EQSC) an arbitrary function $\boldsymbol{C}(\cdot)$ mapping an *integer* n representing the size of a set to a unitary $\boldsymbol{C}(\boldsymbol{A}) \in \mathbb{C}^{s^n \times s^n}$ for any n that behaves equivariantly as described in equation 6.1.

It is clear that EQSCs are a subclass of EQGCs, since they consist of unitaries depending on strictly less information than EQGCs.

We would like to characterise the space of all EQSCs, similarly to what Maron et al. did for equivariant linear layers with regards to a different tensor-based encoding of graph data [42] (see next section for detailed comparison). However, the unitarity constraint does not lend itself to such analysis very easily, so we instead characterise upper and lower bounds for the space of equivariant unitaries.

First, we look at a larger set by dropping the unitarity constraint and considering all equivariant linear layers instead. In the case of using a single qubit per node, we give a detailed characterisation and calculate the exact dimensionality of the space of linear layers for sets of size n. We also demonstrate how specific circuits fit into this description, and discuss how this generalises to larger node states.

Secondly, we further restrict fixed EQGCs to be *diagonal* unitaries, and show that the space of such matrices still grows in dimensionality as we increase the size of the input set.

6.3.1 Comparison with classical invariant/equivariant graph networks

In their paper on invariant and equivariant graph networks [42], Maron et al. ask similar questions to characterise and implement classical equivariant/invariant models operating on tensors representing relational data. While the questions we investigate were partly inspired by them, and our data can also be seen as high-order tensors, there are significant differences in our setting.

Most importantly, the order of k the tensors they dealt with was fixed and independent of the size n of the input graph, while the size of the tensors along each of those k dimensions depended n. For example, their input included the adjacency matrix, a tensor in $\mathbb{R}^{n^2}$. For EQGCs and EQSCs, this is the other way around. Adding more nodes means working with a larger tensor product, but each dimension is of a fixed size s. For example, with a single qubit per node, our state is in $\mathbb{C}^{2^n}$. This matters for the notion of equivariance/invariance: applying a permutation p brings the element at an index $(i_1, i_2, \ldots, i_n)$ to $(i_{p(1)}, i_{p(2)}, \ldots, i_{p(n)})$ for us, instead of $(p(i_1), p(i_2), \ldots, p(i_n))$

as in the previous work.

Finally, there are a few more obvious differences: due to the quantum context, we are working with complex numbers rather than reals, and we are interested in the extra condition of unitarity rather than arbitrary linear layers.

6.3.2 An upper bound: equivariant linear layers

The unitarity constraint is tricky to analyse, so in this section we will focus on a superset of EQSCs with simpler structure.

Definition 6. Let an *n-element equivariant set linear layer* (n-ELL) be a single complex matrix $\boldsymbol{L} \in \mathbb{C}^{s^n \times s^n}$ that satisfies Equation 6.1. In other words, an n-ELL is what $\boldsymbol{C}(n)$ can be for an EQSC $\boldsymbol{C}$ without the unitarity constraint.

First let us consider the case when $s = 2$, so each node is assigned a single qubit which is in a superposition of $|0\rangle$ and $|1\rangle$, and the action of an n-ELL $\boldsymbol{L}$ can be represented as mapping bitstrings of length n (i.e., computational basis vectors in $\mathbb{C}^{2^n}$) to linear combinations of such bitstrings. The general case for $s > 2$ is conceptually analogous, but this is easier to state and prove clearly.

Theorem 2. *A matrix $\boldsymbol{L} \in \mathbb{C}^{2^n \times 2^n}$ is an n-ELL if and only if it can be expressed by weights $w_{ijk} \in \mathbb{C}$ for $0 \leq i \leq n$, $0 \leq j \leq i$ and $0 \leq k \leq n - i$ as follows: for computational basis states $|\psi\rangle, |\theta\rangle$, $\langle\theta| \boldsymbol{L} |\psi\rangle = w_{ijk}$ if the bitstring representing $|\psi\rangle$ contains $|1\rangle$s in i different positions, the bitstring representing $|\theta\rangle$ contains j $|1\rangle$s at positions where $|\psi\rangle$ had $|1\rangle$s and k $|1\rangle$s at positions where $|\psi\rangle$ had $|0\rangle$s.*

Example 8. For $n = 3$,

$$\begin{aligned} \boldsymbol{L}|100\rangle = & w_{100}|000\rangle + w_{101}\big(|001\rangle + |010\rangle\big) + w_{102}|011\rangle + \\ & w_{110}|100\rangle + w_{111}\big(|101\rangle + |110\rangle\big) + w_{112}|111\rangle \end{aligned} \tag{6.14}$$

This shows that $\langle 001|\boldsymbol{L}|100\rangle = \langle 010|\boldsymbol{L}|100\rangle = w_{101}$ since $\langle 001|$ and $\langle 010|$ both contain one $\langle 1|$ in a position where where $|100\rangle$ has a $|0\rangle$, and no $\langle 1|$ where $|100\rangle$ has a $|1\rangle$; and $\langle 101|\boldsymbol{L}|100\rangle = \langle 110|\boldsymbol{L}|100\rangle = w_{111}$, because they both contain one $\langle 1|$ for the $|0\rangle$s in $|100\rangle$ and one $\langle 1|$ for the single $|1\rangle$ in $|100\rangle$. The other inner products involving $|100\rangle$ all differ in how many $\langle 1|$s meet $|1\rangle$s and $|0\rangle$s, so they can be chosen independently of each other. (Note however that they are not independent of other values of the matrix $\boldsymbol{L}$ such as those in the vectors $\boldsymbol{L}|001\rangle$ and $\boldsymbol{L}|010\rangle$, as we will see in the proof.)

For further clarity, consider representing the following EQSCs with such weights:

Example 9 ($CZ(\alpha)$-gates between all pairs of nodes)**.** Consider a circuit $\boldsymbol{L}$ consisting of controlled Z-rotations with a parameter α applied between each pair of qubits. For computational basis

states $|e_1\rangle, |e_2\rangle$, this only applies phases, therefore we have a diagonal matrix and $\langle e_1| \boldsymbol{L} |e_2\rangle = 0$ if $|e_1\rangle \neq |e_2\rangle$. The phase applied is $e^{-i\alpha}$ for each pair of qubits that are both set to one, so if the input contains i ones then we get a phase of $e^{-i(i-1)\alpha/2}$ in total. Therefore $\boldsymbol{L}$ is represented by $w_{ijk} = e^{-i(i-1)\alpha/2}$ if $j = i, k = 0$ and 0 otherwise.

Example 10 (Arbitrary single-qubit unitaries applied everywhere)**.** Let $\boldsymbol{U} = \left(\begin{smallmatrix} u_{0,0} & u_{0,1} \\ u_{1,0} & u_{1,1} \end{smallmatrix}\right)$. Then, for $x, y \in \{0, 1\}$, we have $\langle x| \boldsymbol{U} |y\rangle = u_{x,y}$. Suppose we apply this unitary to all n qubits. Then, for two computational basis states $|e_1\rangle, |e_2\rangle$, $\langle e_1| \boldsymbol{U}^{\otimes n} |e_2\rangle$ is of the form $u_{0,0}^a \times u_{0,1}^b \times u_{1,0}^c \times u_{1,1}^d$, where a and d are the number of overlapping $|0\rangle$s and $|1\rangle$s respectively in the bitstring representation of $|e_1\rangle, |e_2\rangle$, b is the number of positions where $|e_1\rangle$ contains a $|0\rangle$ and $|e_2\rangle$ contains a $|1\rangle$, and c is the same in the other direction.

This lets us express the w_{ijk} parameters representing $\boldsymbol{U}^{\otimes n}$ from inner products of computational basis states $\langle e_1| \boldsymbol{U} |e_2\rangle$ and expressing a, b, c, d as above:

- d, the number of overlapping ones, is just j.
- c, the number of ones in $\langle e_1|$ meeting zeros in $|e_2\rangle$, is just k.
- We can get b, the number of zeros in $\langle e_1|$ meeting ones in $|e_2\rangle$ as $i - j$, subtracting the overlapping ones from the number of ones in the input.
- We can get a, the number of overlapping zeros as $(n - i) - k$, getting the number of zeros in $|e_2\rangle$ as $(n - i)$ and then subtracting the k positions where $\langle e_1|$ has a one.

So we get that $\boldsymbol{U}^{\otimes n}$ is represented by $w_{ijk} = u_{0,0}^{n-i-k} \times u_{0,1}^{i-j} \times u_{1,0}^k \times u_{1,1}^j$.

We will prove this theorem through two simple lemmas.

Lemma 2.1. *Any n-ELL $\boldsymbol{L}$ is entirely characterised by its output on $|s_0\rangle = |00\ldots00\rangle, |s_1\rangle = |00\ldots01\rangle, \ldots, |s_{n-1}\rangle = |01\ldots11\rangle, |s_n\rangle = |11\ldots11\rangle$.*

Proof. Consider any the computational basis vector $|e\rangle \in \mathbb{C}^{2^n}$. This corresponds to some string of zeros and ones. Then, for the $|s_i\rangle$ containing the same number of zeros and ones, there is some permutation of indices $\tilde{\boldsymbol{P}} \in \mathbb{C}^{2^n \times 2^n}$ such that $|e\rangle = \tilde{\boldsymbol{P}} |s_i\rangle$ and therefore $\boldsymbol{L} |e\rangle = \boldsymbol{L}\tilde{\boldsymbol{P}} |s_i\rangle$. Multiplying by $\tilde{\boldsymbol{P}}^T$ gives $\tilde{\boldsymbol{P}}^T \boldsymbol{L} |e\rangle = \tilde{\boldsymbol{P}}^T \boldsymbol{L}\tilde{\boldsymbol{P}} |s_i\rangle = \boldsymbol{L} |s_i\rangle$ by equivariance, so $\boldsymbol{L} |e\rangle = \tilde{\boldsymbol{P}}\boldsymbol{L} |s_i\rangle$. So knowing $\boldsymbol{L} |s_i\rangle$ for each $|s_i\rangle$ determines $\boldsymbol{L} |e\rangle$ for all computational basis vector, hence determining it entirely. □

Lemma 2.2. *We must have $\langle e_1| \boldsymbol{L} |s_i\rangle = \langle e_2| \boldsymbol{L} |s_i\rangle$ for computational basis vectors $|e_1\rangle, |e_2\rangle$ which can be transformed to each other by permuting over indices that have the same value (0 or 1) in $|s_i\rangle$.*

Proof. Consider the permutation of indices $\tilde{\boldsymbol{P}}$ that turns $|e_1\rangle$ to $|e_2\rangle$. Note that $\tilde{\boldsymbol{P}}|s_i\rangle = |s_i\rangle$ by the given premise, so by equivariance we have $\langle e_1|\boldsymbol{L}|s_i\rangle = \langle e_1|\tilde{\boldsymbol{P}}^T\boldsymbol{L}\tilde{\boldsymbol{P}}|s_i\rangle = \langle e_1|\tilde{\boldsymbol{P}}^T\boldsymbol{L}|s_i\rangle = \langle e_2|\boldsymbol{L}|s_i\rangle$. □

Proof of Theorem 2. Lemma 2.2 showed that $\boldsymbol{L}|s_i\rangle$ expressed in the computational basis will have the same weight for any basis vector with j ones where $|s_i\rangle$ had ones, and k ones where $|s_i\rangle$ had zeros. Denote this weight w_{ijk}. By Lemma 2.1, these parameters uniquely characterise the equivariant linear layer.

This proves the theorem in the forward direction: any n-ELL can be characterised by weights w_{ijk}. Now we show the other direction, that any linear transformation characterised by an arbitrary choice of w_{ijk} satisfies Equation 6.1 and therefore is an n-ELL. Consider an arbitrary $\boldsymbol{L} \in \mathbb{C}^{2^n \times 2^n}$ given in this form. It suffices to show that it behaves correctly with respect to swap permutations and input states in the computational basis: more complex permutations can be built by composing swaps, and more complex states by linear combinations of basis states. For any bitstring input $|e\rangle$, we can have two kinds of swaps:

- In the first case, we swap two indices with the same digit in the bitstring (both $|0\rangle$ or $|1\rangle$). The input to $\boldsymbol{L}$ is unchanged, and equivariance is respected because the same coefficients from w_{ijk} are multiplying pairs of output vectors that should be swapped.
- In the second case, the digits at the two indices differ. The inputs passed to $\boldsymbol{L}$ on the two sides of the equation are different, and equivariance is ensured by the number of overlapping $|1\rangle$s changing in a way that the w_{ijk} coefficients get swapped consistently.

□

As a consequence, we can easily see that the dimensionality of the space of n-ELLs is unbounded in terms of n, as opposed to the equivariant layers studied by Maron et al., so we cannot hope to uniformly parameterise the entire space for unbounded n.

Corollary 2.1. *The dimensionality of the space of n-ELLs with a single qubit per node over n nodes is:*

$$\sum_{i=0}^{n}(i+1)(n-i+1) = \frac{1}{6}n(n+1)(n+5)$$

Proof. The left-hand side follows from the above by considering the number of (i, j, k) triples with $0 \leq i \leq n$, $0 \leq j \leq i$, $0 \leq k \leq n - i$. We get the closed form on the right using the formula for pyramid numbers and simplifying. □

Generalising to larger node states

An analogous result holds for n-ELLs $\boldsymbol{L}$ with larger node states $s > 2$. Say we have s possible node basis states $\{|0\rangle, \ldots, |s-1\rangle\}$. In this case, a single matrix element $\langle\theta|\boldsymbol{L}|\psi\rangle$ for computational basis states $|\psi\rangle, |\theta\rangle$ is depends on the entire set of how many $|i\rangle$ appear in $|\psi\rangle$ in positions where $|\theta\rangle$ contains a $|j\rangle$, for all $i, j \in \{0 \ldots s-1\}$.

To prove this, similarly to Lemma 2.1, we can show that it suffices to specify $\boldsymbol{L}|\psi\rangle$ for each distribution of input node states; and similarly to Lemma 2.2, we can show that $\langle\theta|\boldsymbol{L}|\psi\rangle$ is invariant to changing $\langle\theta|$ in a way that does not change the number of any $\langle i|$ to $|j\rangle$ 'matches' as described above.

Implications for equivariant quantum set circuits

Corollary 6.3.2 has implications for the unitarity-constrained EQSCs – it gives an upper bound for the dimensionality of the space of possible values for $\boldsymbol{C}(n)$ for any EQSC $\boldsymbol{C}$. However, it does not rule out that the size of the unitarity-constrained space might have a fixed bound for arbitrary n. In that case we could still effecively parametrerise EQSCs by using the same weights to specify $\boldsymbol{C}(n)$ for all n, similarly to Maron et al.

6.3.3 A lower bound: diagonal equivariant unitaries

To see whether the size of the space of EQSCs grows with the size of the set, we can investigate a more restricted space as a lower bound: *diagonal* unitaries satisfying the equivariance Equation 6.1.

A general diagonal unitary can apply an arbitrary phase to each computational basis state independently. The equivariance condition restricts us to applying the same phase for inputs that could be transformed to each other by permuting the indices, i.e. inputs that contain the same distribution of node states (the same number of $|0\rangle$s and $|1\rangle$s when using one qubit per node). This gives a lower bound of $n+1$ on the dimensionality of equivariant unitaries over sets of size n using a single qubit per node, which is still unbounded in n. More generally, for n nodes with s possible states each, the lower bound is the number of unique s-tuples of nonnegative integers that sum to n, which is given by $\binom{n+s-1}{s-1}$.

Chapter 7

Expressive Power of Quantum Methods

In this chapter, we analyse the expressivity of the Equivariant Quantum Graph Circuit classes discussed in Section 6.2: the Hamiltonian-based EH-QGCs and the more restricted EDU-QGCs defined using commuting unitaries.

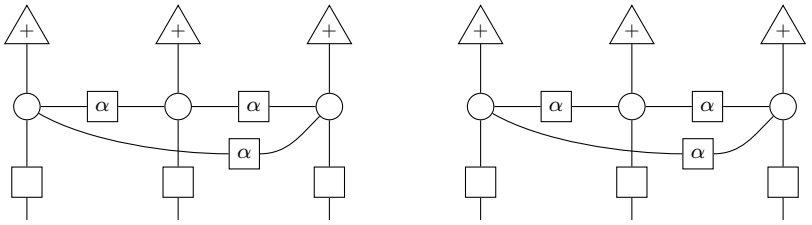
Since quantum circuits operate differently from MPNNs and other popular GNN architectures, one might hope that they are more expressive. Since current classical methods with high expressivity are either computationally expensive (like higher-order GNNs) or require a large number of training samples to converge (like GNNs with random node initialisation), this could in principle lead to a form of quantum advantage with sufficiently large-scale quantum computers.

We first test empirically whether a very simple EDU-QGC configuration goes beyond 1-WL distinguishing power. After finding that it does, we show that EDU-QGCs subsume MPNNs: any MPNN can be 'simulated' by a suitable EDU-QGC configuration. We finally conclude with a proof that they are in fact universal models for arbitrary functions on bounded-size graphs, building on prior results regarding randomised MPNNs.

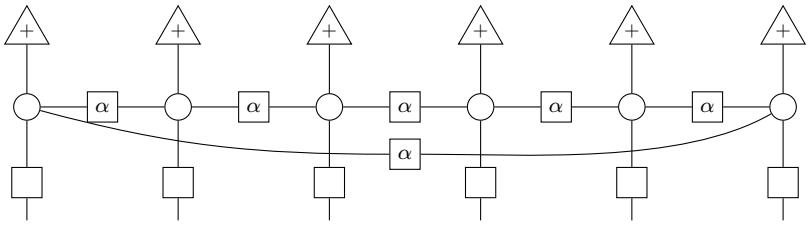
Since we have proven EDU-QGCs to be a subclass of QGCNNs in Theorem 1, the results immediately follow for EH-QGCs as well as the closely related QGCNNs defined by Verdon et al. which we discussed in Section 6.2.2.[1]

Our result does *not* go as far as to convincingly show quantum advantage, since the specific constructions we give could be simply executed classically, but it is an important step towards understanding the capabilities of quantum methods.

[1] And of course, this also applies for the broad class of all EQGCs, but this a fairly vacuous statement due to their extreme generality as noted in Section 6.2.1.



(a) $\boldsymbol{C}$ applied to two triangles



(b) $\boldsymbol{C}$ applied to a 6-cycle

Figure 7.1: The two circuits in the experiment in top-to-bottom ZX-diagram notation, with the α-box between white spiders representing a $CZ(\alpha)$ gate (a standard ZX-calculus shorthand [20]).

7.1 Experiment: beyond the Weisfeiler-Lehman test

7.1.1 Experimental setup

To get a sense of the expressivity of EDU-QGCs, we take a very simple restricted model with a single parameter and apply it to two 1-WL equivalent graphs. If the distribution of the model's outputs differs across these two examples, we know that there are graphs that are indistinguishable by MPNNs but distinguishable by EDU-QGCs with some nontrivial probability (i.e., better than 50%, but maybe not arbitrarily close to 100%).

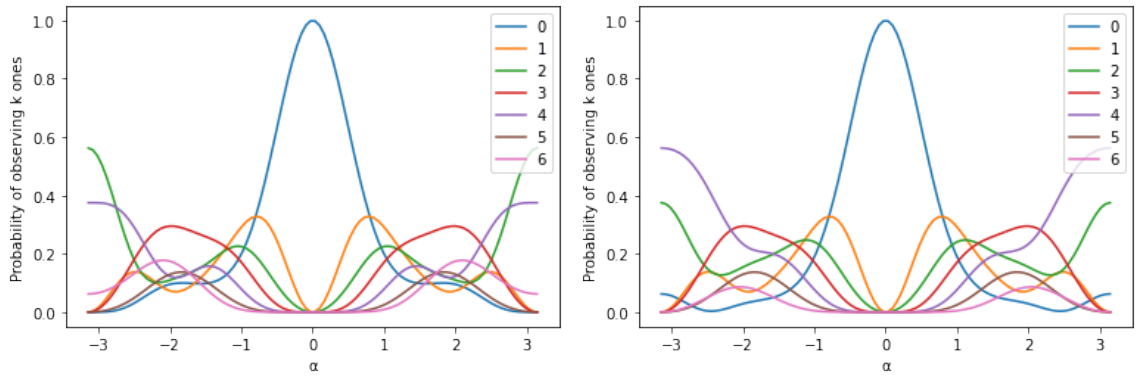
As our model, we use an EDU-QGC with one qubit per node, defined by two layers:

1. First, we apply $CZ(\alpha)$ gates for some learnable parameter α for all pairs of nodes that share an edge. The $CZ(\alpha)$ gate is represented by a diagonal matrix $\mathrm{diag}(1, 1, 1, e^{-i\alpha})$, so by definition it is an equivariantly diagonalised by $\boldsymbol{V} = \boldsymbol{I}$. and applying it for each edge is a valid EDU-QGC layer. It also satisfies the undirected condition from Equation 6.5, so we can use it for undirected graphs without applying it in both directions.

2. Secondly, we apply a Hadamard gate to each node to get interesting measurement outcomes from the phases applied by the $CZ(\alpha)$ gates.

After the two layers, we measure all qubits in the computational basis and count the number of $|1\rangle$s measured.[2] We simulate the circuit classically, so we can exactly calculate the probability

[2]Since their positions are irrelevant for any invariant aggregation, and we are using a single qubit per node, this



(a) Measurement probabilities for two triangles

(b) Measurement probabilities for one 6-cycle

Figure 7.2: Probabilities of observing given number of $|1\rangle$s as a function of $\alpha \in [-\pi, \pi]$ over the two input graphs. The two distributions differ most visibly when α is near $\pm\pi$.

distribution of this value.

As our two samples, we use the smallest 1-WL equivalent pair of graphs from Figure 5.1: $\mathcal{G}_1$ consisting of two disconnected triangles, and $\mathcal{G}_2$ being a single 6-cycle. These are indistinguishable to MPNNs as discussed in Section 5.2.1. We do not assign any node features and simply initialise each node state as the $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ state.

The two circuits we get are shown in Figure 7.1. We run these for different values of α and compare the distribution of the number of $|1\rangle$s measured in their outputs.

7.1.2 Results

We plot the probability of observing $0, 1, \ldots, 6$ ones from each sample in Figure 7.2. We can make several interesting observations:

- Most importantly, the distributions do differ for most values of α! Although it is not a huge difference, the model is also very simple and this is enough to show that we do not face the same limitation as MPNNs.
- The distributions are most obviously different at $\alpha = \pm\pi$. Here, their total variation distance is $\frac{1}{2}$.
- We always measure $|000000\rangle$ if $\alpha = 0$. This is expected, since $CZ(0)$-rotations do nothing, the input uniform superposition can be viewed as Hadamards applied to $|000000\rangle$, and Hadamards are involutory. Therefore the entire circuit reduces to measuring each qubit after initialising them as all zeros.

is all we can use from the output.

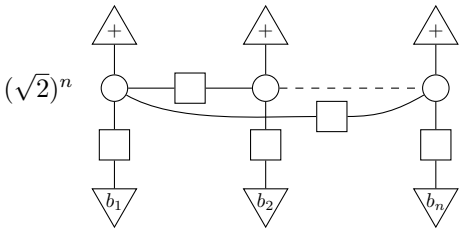
- The probabilities of observing odd numbers of $|1\rangle$s is the same across the two graphs for any given value of α (but varies based on α).
- We never observe any odd values for $\alpha = \pm\pi$. Although is not particularly useful, explaining the phenomenon will serve as a good test of our understanding in the following section.

7.1.3 Understanding the behaviour at $\alpha = \pi$

In an effort to better understand the power of such circuits, we focused on analysing the most well-behaved special case of the above EDU-QGC, with $CZ(\pi)$ rotations.

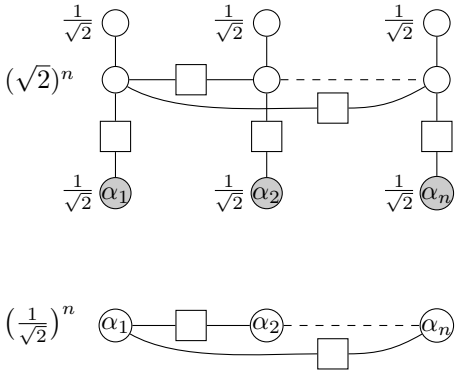
Using the ZX-calculus, we show that applying it to any n-cycle graph results in a uniform distribution over certain measurement outcomes, give a simple algorithm to check for a given n-length bitstring whether it is one of these possible outcomes, and prove that the number of measured $|1\rangle$s always has the same parity as the size n of the graph.

With $\alpha = \pi$, the α-boxes representing the CZ-gates in Figure 7.1 turn into simple Hadamard. So for any specific bitstring $|b_1 \dots b_n\rangle$, we can get the probability of measuring it by simplifying the following scalar:



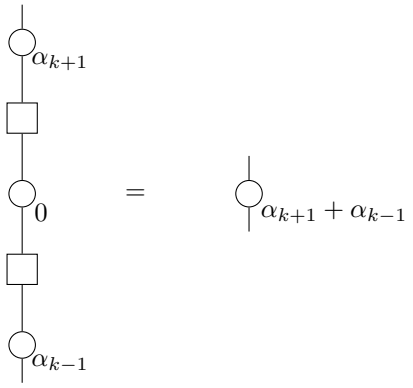
where the numerical term comes from normalising each CZ-gate with a factor of $\sqrt{2}$.

We can substitute the appropriate white and gray spiders for the $|+\rangle, |0\rangle$ and $|1\rangle$ states to apply ZX-calculus techniques [20]: a white spider with phase 0 for the $|+\rangle$ state, and gray spiders with 0 and π phases respectively for $|0\rangle$ and $|1\rangle$. All of these need to be normalised with a factor of $\frac{1}{\sqrt{2}}$. Due to the Hadamard gates, these all turn into white spiders that can be fused together, so this is equal to a simple trace calculation:



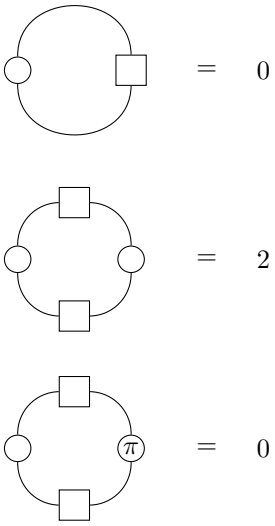
where $\alpha_i = 0$ if $b_i = 0$ and π if $b_i = 1$.

This can be simplified step by step. Firstly, as long as there are any spiders with $\alpha_i = 0$ and two distinct neighbours (i.e., there are at least 3 nodes in total), we can remove them and fuse their neighbours:



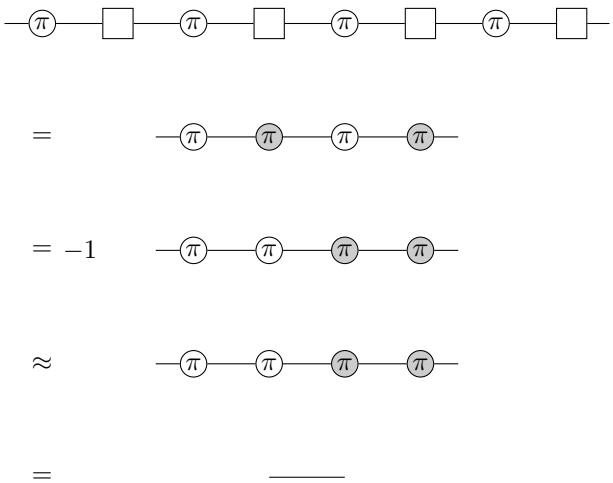
(7.1)

After repeating this, we get one of two outcomes. Firstly, we might end up with one of 3 possible circuits with that still have some $\alpha_i = 0$ but less than 3 nodes, which we can evaluate by direct calculation of their matrices:



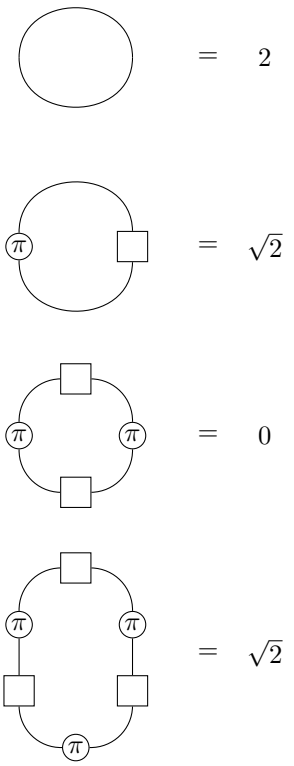
(7.2)

Or all the remaining spiders have $\alpha_i = \pi$, we can repeatedly eliminate them in groups of 4:



(7.3)

On repeating this, we end up with 0 to 3 nodes with $\alpha_i = \pi$, which we can evaluate directly:



(7.4)

Observe that during the simplifications, we only introduced phases with an absolute value of 1, which do not affect measurement probabilities. Furthermore, we always decreased the number of nodes involved by 2 or 4, hence the parity is unchanged. This means for odd n, we will always end up with one of the odd-cycle base cases with a trace of 0 or $\pm\sqrt{2}$, while for even n, we get to the even-cycle base cases with traces of 0 or 2.

Combining with the initial coefficient of $\left(\frac{1}{\sqrt{2}}\right)^n$ and taking squared norms, we get that *for odd n, each bitstring is observed with probability* 0 *or* $\frac{1}{2^{n-1}}$ (so half of all possible bitstrings are observed), while *with even n, each bitstring is observed with probability* 0 *or* $\frac{1}{2^{n-2}}$ (so we see only a quarter of all bitstrings).

Furthermore, to check which bitstrings are observed, we can summarise the ZX-diagram simplification as a simple algorithm acting on cyclic bitstrings (where the first and last bits are considered adjacent):

- As long as there is a 0 in the bitstring and the length of the bitstring is more than 2, remove the zero along with its two neighbors, and replace them with the XOR of the neighbors.
- If you end up with just $|00\rangle$, the state has a positive probability to be observed. If you end up with $|0\rangle$ or $|01\rangle$, it has 0 probability.
- When there are only 1s remaining, if the number of these is 2 mod 4, the input has 0 probability to be observed, otherwise positive.

This shows us why the observed number of $|1\rangle$s always has the same parity as n: at each step, both the parity of $|1\rangle$s and the parity of the bitstring's length is unchanged. The only even-length base case with an odd number of ones is $|01\rangle$, which corresponds to states with 0 probability; and similarly the only odd-length base case with an even number of ones is $|0\rangle$, which has the same outcome.

We can also derive the specific probabilities observed in the experiment. It's easy to see from this that in the case of a triangle, the observable states are $|001\rangle, |010\rangle, |100\rangle, |111\rangle$. This allows us to calculate the probabilities observed for the case of two triangles. For the 6-cycle, the observable states are $|000000\rangle$, six rotations of $|000101\rangle$, six rotations of $|001111\rangle$, and three rotations of $|101101\rangle$, giving the expected probabilities as well.

7.2 Simulating message-passing neural networks

Our experiments gave us clues about the expressivity of EDU-QGCs and EH-QGCs, but they are also very limited due to the small size of the circuits we can feasibly simulate. We know that our models can distinguish some graphs that MPNNs cannot, but this does not even show whether they are strictly more expressive: there might be some graphs that MPNNs can distinguish but our methods cannot.

In this section, we rule out this possibility by showing that any MPNN with sum aggregation can be 'simulated' by an EDU-QGC. Notably, this includes Graph Isomorphism Networks [67] among other popular methods. Since GINs are as expressive as the 1-WL isomorphism test, this shows that EDU-QGCs have the power to distinguish any two graphs that are not 1-WL equivalent with 0 probability of errors.

Before we state our results, a few remarks:

Remark 6. We consider MPNNs node states with real numbers represented in fixed-point arithmetic. Although GNNs tend to be defined with uncountable real vector state spaces, these can be approximated with a finite set if the data is from a bounded set: indeed, physical hardware itself has a finite number of bits. (The difference between floating-point and fixed-point calculations is not an important difference as far as expressivity is concerned.)

Remark 7. There is no practical benefit from putting the specific constructions from our proofs into the quantum setting rather than just running the classical model itself – the significance of the result is in characterising what EDU-QGCs and related quantum models can represent.

The main result of this section is the following:

Theorem 3. *EDU-QGCs can represent any MPNN with sum aggregation: any (Boolean or real) function that can be represented by an MPNN with an aggregation function of the form* $\textsc{aggregate}^{(k)}(\{\!\{\boldsymbol{h}_i\}\!\}) = \sum_i \boldsymbol{h}_i$ *can also be represented an EDU-QGC.*

For an MPNN with k layers with an embedding dimensionality of w, with a fixed-point real representation of b bits per real number, this EDU-QGC needs $(2k+1)wb$ qubits per node.

Since MPNNs with sum aggregation, such as the Graph Isomorphism Networks defined by Equation 5.7, can distinguish any 1-WL distinguishable graphs [67], we immediately obtain the following corollary:

Corollary 3.1. *EDU-QGCs can distinguish any pair of 1-WL distinguishable graphs.*

It remains to prove Theorem 3, which we do by giving an explicit construction to simulate an arbitrary MPNN with sum aggregation. The node states will be conceptually split into registers representing fixed-point real numbers in two's complement in the computational basis. We first need to establish that we can perform addition on these registers using unitary transformations.

Lemma 3.1. *Consider two node states with two registers each, storing unsigned integers: $|a_1, a_2\rangle \otimes |b_1, b_2\rangle$, with $a_i, b_i \in \{0, \ldots, 2^b - 1\}$ for some b. Let $\boldsymbol{U}$ map $|a_1, b_1\rangle \otimes |a_2, b_2\rangle$ to $|a_1, b_1 + a_2\rangle \otimes |a_2, b_2 + a_1\rangle$, with standard overflowing addition. $\boldsymbol{U}$ is an equivariantly diagonalisable unitary and satisfies the undirected symmetry condition in Equation 6.5.*

Proof. Let $\boldsymbol{S}_a$ be a single-node unitary that increments integers encoded in the computational basis by a. Note that $\boldsymbol{S}_a = \boldsymbol{S}_1^a$. Diagonalise $\boldsymbol{S}_1$ as $\boldsymbol{V}^\dagger \boldsymbol{D} \boldsymbol{V}$, then $\boldsymbol{S}_a = (\boldsymbol{V}^\dagger \boldsymbol{D} \boldsymbol{V})^a = \boldsymbol{V}^\dagger \boldsymbol{D}^a \boldsymbol{V}$.

Now $\boldsymbol{U}$ can be represented by applying $\boldsymbol{V}$ to the second register of each node, conditionally applying $\boldsymbol{D}$ to the the second register of each node some number of times depending on the value of the first register, and finally applying $\boldsymbol{V}^\dagger$ to the second registers. The controlled application of a diagonal matrix is still diagonal, so this decomposition diagonalises $\boldsymbol{U}$ equivariantly with $(\boldsymbol{I} \otimes \boldsymbol{V})^{\otimes 2}$.

The undirected symmetry Equation 6.5 can be seen easily from the definition of $\boldsymbol{U}$: swapping a_1 with a_2 and b_1 with b_2 results in swapping the values in the output. □

Lemma 3.2. *Consider two node states with two registers each, storing fixed point unitaries in two's complement: $|a_1, a_2\rangle \otimes |b_1, b_2\rangle$, with $a_i, b_i \in \{(-2^{b-1} + 1) \times 2^{-k}, \ldots, 2^{b-1} \times 2^{-k}\}$ for some b, k. Let $\boldsymbol{U}$ map $|a_1, b_1\rangle \otimes |a_2, b_2\rangle$ to $|a_1, b_1 + a_2\rangle \otimes |a_2, b_2 + a_1\rangle$, with standard overflowing addition. $\boldsymbol{U}$ is an equivariantly diagonalisable unitary.*

Proof. As far as the bit-level operations are concerned, this is exactly the same as Lemma 3.1: with two's complement, standard overflowing addition of unsigned integers can represent addition

of signed integers, and fixed point reals are essentially integers interpreted with a multiplication of 2^{-k}. □

Proof of Theorem 3. Let M be an MPNN with k layers and width w, where the initial states are $\boldsymbol{h}_1 \dots \boldsymbol{h}_n$. We define an EDU-QGC $\boldsymbol{C}$ which computes the same final node embeddings as M, based on M's iterated message-passing and node update procedure.

We conceptually divide the qubits for each node v into $(k+1) \times w$ registers $\boldsymbol{h}_v^{(0,0)}, \dots, \boldsymbol{h}_v^{(k,w-1)}$ of b qubits each, and $k \times w$ registers $\boldsymbol{a}_v^{(1,0)}, \dots, \boldsymbol{a}_v^{(k,w-1)}$ of b qubits each. This is a total of $(k+1)w \times b + kw \times b = (2k+1)wb$ qubits as expected. The $\boldsymbol{s}_v^{(0)}$ registers are initialised to the initial MPNN node states $\boldsymbol{h}_v$, and all other qubits are set to $|0\rangle$. Then, for each MPNN layer, we first simulate its message-passing phase with two-node unitaries for all edges, then we simulate the update functions with single-node unitaries.

Specifically, for the k-th message-passing layer of M, we apply a unitary $U^{(k)}$ for each edge (v, u) that should have the effect of adding the value of $\boldsymbol{h}_v^{(k-1,i)}$ to $\boldsymbol{a}_u^{(k,i)}$ and vice versa for each $i \in \{0 \dots w-1\}$. This results in the $\boldsymbol{a}_v^{(k,\cdot)}$ registers eventually storing the sum of their neighbours' states from the previous layer, which simulates the sum aggregation. This is an equivariantly diagonalisable unitary acting well on undirected graphs by Lemma 3.2, so applying it for each edge is a valid EDU-QGC layer.

For the k-th update layer, a unitary is applied to each node that XORs the result of the MPNN's update function, $\text{UPDATE}^{(k)}(\boldsymbol{h}_v^{(k-1,\cdot)}, \boldsymbol{a}_v^{(k)})$ onto the set of registers$\boldsymbol{h}_v^{(k,\cdot)}$, which are until this point still initialised to all zeros. This is a permutation and therefore a unitary, so applying it for each node is a valid EDU-QGC layer.

At the end of the circuit, we measure all qubits, which will include the final node states $\boldsymbol{h}_v^{(k,\cdot)}$. We can classically aggregate in the same way the MPNN pools its results to give our prediction. This will match the MPNN's output for all inputs with 0 probability of errors. □

7.3 EDU-QGCs are universal for graphs

In the following section, we will build on powerful results about randomisation in classical MPNNs (discussed in Section 5.2.3) to show stronger claims about the expressivity of our quantum models. In short, we will simulate classical models that randomise some part of the node state by putting some qubits into the uniform superposition over all bitstrings, then operating in the computational basis. Note that unlike in the classical case, where this randomisation had to be explicitly added to extend model capacity, we can do this *without modifying our model definition* in the quantum case – our results will apply to EDU-QGCs as given in Definition 4 and their superclasses.

This will allow us to prove the following theorem, which was shown to hold for classical MPNNs with readouts at each layer by Abboud et al.:

Theorem 4. *For any real function f defined over graphs up to size n, and any $\epsilon > 0$, there is an EDU-QGC that calculates $f(\mathcal{G})$ with probability $(1-\epsilon)$ for any graph $\mathcal{G}$.*

7.3.1 From Boolean to real-valued functions

We will prove Theorem 4 by first looking at the case of Boolean-valued functions over graphs, and show that the case for reals follows by the same argument as Abboud et al. used [2].

Lemma 4.1. *For any Boolean function f defined over graphs up to size n, and any $\epsilon > 0$, there is an EDU-QGC that calculates $f(\mathcal{G})$ with probability $(1-\epsilon)$ for any graph $\mathcal{G}$.*

We show that Theorem 4 follows, so it just remains to prove Lemma 4.1:

Proof of Theorem 4 given Lemma 4.1. Consider the outputs of any real-valued function f over graphs of size n expressed in binary decimal form, in the form of zeros and ones assigned to different positions. Since there is a finite number of such graphs, there is a finite number k of different decimal places where the result differs for any two graphs. For each of these, a binary classifier can be represented by EDU-QGCs by Lemma 4.1 that gives the correct prediction with probability $1 - \frac{\epsilon}{k}$.

Say the i-th binary classifier predicts an output $F_i(G) \in \{0, 1\}$ for any bounded-size graph G that represents the bit at position $k_i \in \mathbb{Z}$ of the desired real-number output. Running these 'next to each other' is also a valid EDU-QGC, and their results can then be combined by an MLP to calculate the real output:

$$F(G) = \Big(\sum_i F_i(G) \times 2^{k_i} \Big) + C \tag{7.5}$$

By the union bound, the total probability of any classifier making a mistake is ϵ, so with probability $(1-\epsilon)$ our prediction can be as accurate as allowed by our representation of real numbers. □

7.3.2 Individualising graphs

Abboud et al. prove their results about the power of MPNNs as follows [56, 2]: say a graph is *individualised* if all nodes are extended with unique features. They construct MPNNs that accurately model any function from a large class assuming the input graph is individualised. And for any graph of n nodes and arbitrarily small desired error rate ϵ, if we randomise some node features appropriately, the result will be individualised with probability at least $(1-\epsilon)$.

In the case of EDU-QGCs, if we assume some part of all node states is initialised to all $|0\rangle$s, we can have the first EDU-QGC layer apply a unitary on all nodes consisting of Hadamard gates

on the appropriate qubits. This maps them to the uniform superposition over all bitstrings. If we then use the construction from Theorem 3 that acts classically on the computational basis, and then measure the results, we get the same result as running the MPNN with a randomised initial state. The following lemma bounds the number of qubits required for this:

Lemma 4.2. *Putting n sets of $b \geq 2\log(n) + \log(1/\epsilon)$ qubits each in the uniform superposition and measuring them leads to n unique bitstrings with probability at least $(1-\epsilon)$.*

Proof. We are effectively just randomising b classical bits uniformly. If we randomise b individual bits of node state uniformly at random, each pair of nodes would get the same label with just 2^{-b} probability. This applies for each of the $n(n-1)/2 < n^2$ pairs of nodes, so by the union bound, the total probability of any match is at most $2^{-b}n^2$. This is less than ϵ if $b \geq 2\log(n) + \log(1/\epsilon)$ bits are randomised. □

7.3.3 Achieving universality

Note that we cannot directly rely on the results of either Abboud et al. or Sato et al.: although our theorem is analogous to that of Abboud et al., they used MPNNs *with readouts at each layer* as described in Equation 5.9, which out quantum models cannot simulate [2]. Sato et al. used MPNNs witout readouts, but did not prove such a claim of universality. Therefore, give a novel MPNN construction that is partially inspired by Sato et al., but relies solely on the results of Xu et al. about their Graph Isomorphism Networks [67], and use it to show Lemma 4.1.

We will rely on the following about Graph Isomorphism Networks which follows directly from Corollary 6 of Xu et al. [67]:

Lemma 4.3. *Let $\mathcal{X}$ be a countable set of vectors, and let $\mathcal{P}_k(\mathcal{X})$ be the set of multisets of elements of $\mathcal{X}$ with size at most k. The aggregate-update function of GINs given in Equation 5.7 applied to inputs from $(\mathcal{X} \times \mathcal{P}_k(\mathcal{X}))$ (representing a node's previous state and the multiset of its neighbours' previous states) can learn injective functions over such an input space.*

From this result, we build up to MPNNs that can injectively encode the connected subgraph of each node into their final states if the initial features are unique. To formalise this, we need the following auxiliary definition:

Definition 7. For a graph G with initial node features $\boldsymbol{h}_v$ for each node v, a node u in G and $k \in \mathbb{Z}^+$, define

$$T(G,u,l) = \begin{cases} \{\!\{\boldsymbol{h}_u\}\!\} & \text{if } k = 0 \\ \left(\boldsymbol{h}_u, \{\!\{T(G,v,k-1) \mid v \in \mathcal{N}(u)\}\!\}\right) & \text{if } k > 0 \end{cases} \tag{7.6}$$

where $\mathcal{N}(u)$ repersents the set of neighbours of a node u.

Following Sato et al., we call this a *level-k tree*, and it represents total information propagated to a node in k message-passing steps.

We show that GINs with k layers can injectively encode the level-k tree of a node:

Lemma 4.4. *Let $GIN_{\boldsymbol{\theta}}(G)_v$ represent the final node features of node v in a graph G after applying a Graph Isomorphism Network with parameters $\boldsymbol{\theta}$. There is some configuration $\boldsymbol{\theta}^*$ of a k-layer GIN such that for any nodes v_1, v_2 in degree-bounded graphs G_1, G_2 respectively, with initial node features chosen from a countable space, if $T(G_1, v_1, k) \neq T(G_2, v_2, k)$ then $GIN_{\boldsymbol{\theta}^*}(G_1)_{v_1} \neq GIN_{\boldsymbol{\theta}^*}(G_2)_{v_2}$.*

Proof. By induction. The base case $k = 1$ follows directly from Lemma 4.3. The inductive step follows from the same claim, since the outputs of a GIN layer applied to a countable input space still form a countable space: the set of bounded-size multisets from a countable space is still countable, and so is any image of this set under some function. □

Furthermore, we show that a the level-n tree of a node in a graph of n nodes identifies the isomorphism class of the node's connected component:

Lemma 4.5. *Let G_1, G_2 be two connected graphs with node sets V_1, V_2 of size n with node feature vectors $\boldsymbol{h}_v$ unique within each graph, and take any $v_1 \in V_1, v_2 \in V_2$. If G_1 and G_2 are not isomorphic, then $T(G_1, v_1, n) \neq T(G_2, v_2, n)$. Furthermore, if G_1 and G_2 are not connected, then the same result holds but with isomorphism class of the connected component of v_1 and v_2 instead of G_1 and G_2 itself.*

Proof. Let $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ be the unique node feature vectors in G_1. Note that all of these will appear in $T(G_1, v_1, n)$, because the features of any nodes at distance d from v_1 will appear in $T(G_1, v_1, d)$ by induction, and a connected graph of n nodes has a diameter at most $(n-1)$. Therefore if G_2 contains a different set of unique node features, we get $T(G_1, v_1, n) \neq T(G_2, v_2, n)$ immediately.

Otherwise for each i, we can denote $v_i^{(1)}$ as the node in G_1 with feature vector $\boldsymbol{v}_i$, and $v_i^{(2)}$ as the node in G_2 with the same vector. (These are unique by the uniqueness of feature vectors.) From $T(G_1, v_1, n)$, we can extract the sets $\mathcal{N}_1(\boldsymbol{v}_i) = \{\boldsymbol{h}_u \mid u \in \mathcal{N}(v_i^{(1)})\}$, i.e., the features of nodes adjacent to the node with the feature vector $\boldsymbol{v}_i$. This also follows by induction: $T(G_1, v_1, k)$ recursively includes a tuple $(\boldsymbol{h}_u, \{\!\{T(G_1, w, k-d-1) \mid w \in \mathcal{N}(u)\}\!\})$ for any u at $d \leq k-1$ steps from v_1, and $T(G_1, w, k-d-1)$ gives us $\boldsymbol{h}_w$ for any k, d. Similarly, from $T(G_2, v_2, n)$, we can extract $\mathcal{N}_2(\boldsymbol{v}_i) = \{\boldsymbol{h}_u \mid u \in \mathcal{N}(v_i^{(2)})\}$. If $T(G_1, v_1, n) = T(G_1, v_2, n)$, then $\mathcal{N}_1(\boldsymbol{v}_i) = \mathcal{N}_2(\boldsymbol{v}_i)$ for all i, which gives an isomorphism between G_1 and G_2: the nodes $v_i^{(1)}$ and $v_i^{(2)}$ are in correspondence.

The case for not connected graphs holds because $T(G, v, n) = T(C, v, n)$ for a any graph G with a node v in a connected component C, and then the same derivation applies. □

These results finally allow us to prove Lemma 4.1 and thereby also complete the proof of Theorem 4:

Proof of Lemma 4.1. We start by initialising a sufficient number of qubits of each node to $|+\rangle$ such that with probability $(1-\epsilon)$, observing all n initial node states leads to n unique measurements. By Lemma 4.2, $\lceil 2\log(n) + \log(1/\epsilon)$ quits suffice. We apply an n-layer GIN to this input, which our EDU-QGC can simulate by Theorem 3. By combining Lemmas 4.4 and 4.5, with appropriate parameterisation the GIN, the final node states will be an injective function of the node's connected component.

Since there is a finite number of such graphs, the set of the GIN's outputs is bounded, so an MLP applied to the node state can turn this into a vector of indicator variables for each isomorphism class within some required accuracy: let an indicator $I_C^{(v)}$, part of the node state for node v, be between $1 - \frac{1}{3n}$ and 1 if the v's component is isomorphic to a graph C (without regard for the random features) and between 0 and $\frac{1}{3n}$ otherwise. Since the update function in the GIN architecture is an MLP, this computation can be built into its final layer, which our EDU-QGC can simulate.

We can then pool the node states by summing them into graph-level indicators: for each isomorphism class C of at most n-node graphs, the pooled embedding will contain a summed value N_C encoding the number of nodes whose connected component is in that isomorphism class. For each I_C, the total error is at most $\frac{1}{3}$, so graphs with a different multiset of connected components will be mapped to different vectors. Since the set of graphs of size n is finite, the space of these vectors is bounded, and we can apply an MLP to these values to learn any Boolean function over bounded graphs. If we construct an MLP with accuracy 0.4, the output is always more than 0.6 if the correct answer is 1 and always less than 0.4 if the correct answer is 0. This can be mapped to discrete values in $\{0, 1\}$ with perfect accuracy via a continuous function easily representable by further MLP layers. Therefore the output of the model will be exactly correct as long as observing the $|+\rangle$ states leads to a unique initial state for each node, which has probability at least $(1-\epsilon)$ as required. □

Chapter 8

Summary and Outlook

8.1 Summary of results

In this project, we proposed Equivariant Quantum Graph Circuits, a general framework of quantum machine learning architectures for graph-based machine learning, and explored possible designs within that framework.

Several interesting subclasses were discussed and it was shown that most prior work can be seen as a special case of this framework. EQSCs, corresponding to functions on sets, were characterised in detail, with upper and lower bounds on the dimensionality of such layers over a graph of size n. Two other subclasses, QGCNNs and EDU-QGCs, were proven to have desirable theoretical properties: they are universal for functions defined up to a fixed graph size, just like randomised MPNNs. Our experiments were small-scale due to the computational difficulties of simulating quantum computers classically, but they did confirm that the distinguishing power of our quantum methods exceeds that of deterministic MPNNs.

8.2 Open questions

By defining the framework of EQGCs and their subclasses, many questions can be raised that we did not explore in this project. We showed upper and lower bounds on the dimensionality of the space of fixed equivariant unitaries over n nodes in Section 6.3 – what is the exact value? We showed that EDU-QGCs are a subset of QGCNNs in Section 6.2.3 – but are they a strict subset? And what other subclasses would make sense? Using arbitrary node-level Hamiltonians or unitaries led to good expressivity results as we showed in Chapter 7, but it is not feasible to scale to a large number of qubits per node, since the space of parameters grows exponentially. Perhaps a small number of qubits will already turn out to be useful, but EQGC classes with better scalability to

large node states should also be investigated.

There are also design choices beyond the EQGC framework that might be interesting. We assumed measurement happens at the end of the circuit, but it might be useful to measure some earlier and condition operations on others classically: MacCormack et al. proposed such methods [40], although not in the domain of graphs. More broadly, mixed-state quantum computing offers possibilities that we have not analysed.

Ultimately, the biggest questions in the field of quantum computing are about quantum advantage: what useful tasks can we expect quantum computers to speed up, and what kind of hardware do these applications require? More work into the theoretical capabilities of quantum machine learning architectures might be able contribute to this: if we can find a class of useful functions that are significantly more costly to learn classically than in the quantum setting, perhaps related to the behaviour of molecules or other quantum systems, that would be very relevant for practical applications.

Bibliography

[1] R Abboud et al. "BoxE: A box embedding model for knowledge base completion". In: *NeurIPS Proceedings* 33 (2020).

[2] Ralph Abboud et al. "The Surprising Power of Graph Neural Networks with Random Node Initialization". In: *IJCAI*. 2021.

[3] Erik Arakelyan et al. "Complex Query Answering with Neural Link Predictors". In: *International Conference on Learning Representations*. 2021.

[4] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510.

[5] László Babai. "Graph isomorphism in quasipolynomial time". In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pp. 684–697.

[6] Ivana Balažević, Carl Allen, and Timothy Hospedales. "TuckER: Tensor Factorization for Knowledge Graph Completion". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 5185–5194.

[7] Pablo Barceló et al. "The Logical Expressiveness of Graph Neural Networks". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[8] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[9] Kerstin Beer et al. "Quantum machine learning of graph-structured data". In: *arXiv preprint arXiv:2103.10837* (2021).

[10] Marcello Benedetti et al. "Parameterized quantum circuits as machine learning models". In: *Quantum Science and Technology* 4.4 (2019), p. 043001.

[11] Ville Bergholm et al. "Pennylane: Automatic differentiation of hybrid quantum-classical computations". In: *arXiv preprint arXiv:1811.04968* (2018).

[12] G. Birkhoff and J. Neumann. "The Logic of Quantum Mechanics". In: *Annals of Mathematics* 37 (1936), pp. 1–26.

[13] Kurt Bollacker et al. "Freebase: a collaboratively created graph database for structuring human knowledge". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008, pp. 1247–1250.

[14] Antoine Bordes et al. "Learning structured embeddings of knowledge bases". In: *Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011.

[15] Antoine Bordes et al. "Translating Embeddings for Modeling Multi-relational Data". In: *NIPS*. 2013.

[16] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*. 2014.

[17] Harry Buhrman et al. "Quantum fingerprinting". In: *Physical Review Letters* 87.16 (2001), p. 167902.

[18] Jin-Yi Cai, Martin Fürer, and Neil Immerman. "An optimal lower bound on the number of variables for graph identification". In: *Combinatorica* 12.4 (1992), pp. 389–410.

[19] B. Coecke et al. "Foundations for Near-Term Quantum Natural Language Processing". In: *ArXiv* abs/2012.03755 (2020).

[20] Bob Coecke and Aleks Kissinger. "Picturing quantum processes". In: *International Conference on Theory and Application of Diagrams*. Springer. 2018, pp. 28–31.

[21] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen J Clark. "Mathematical foundations for a compositional distributional model of meaning". In: *Linguistic Analysis* 36.1 (2010), pp. 345–384.

[22] Tim Dettmers et al. "Convolutional 2d knowledge graph embeddings". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[23] Giovanni Felice, Konstantinos Meichanetzidis, and Alexis Toumi. "Functorial Question Answering". In: *Electronic Proceedings in Theoretical Computer Science* 323 (2020), pp. 84–94.

[24] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6 (1982), pp. 467–488.

[25] Dinesh Garg et al. "Quantum Embedding of Knowledge for Reasoning". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[26] Xu Geng et al. “Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 3656–3663.

[27] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1263–1272.

[28] Will Hamilton et al. “Embedding Logical Queries on Knowledge Graphs”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 2026–2037.

[29] William L. Hamilton. “Graph Representation Learning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), pp. 1–159.

[30] David P Helmbold and Philip M Long. “On the inductive bias of dropout”. In: *The Journal of Machine Learning Research* 16.1 (2015), pp. 3403–3454.

[31] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257.

[32] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366.

[33] Shaoxiong Ji et al. “A Survey on Knowledge Graphs: Representation, Acquisition, and Applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–21.

[34] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[35] J Lambek. “Type Grammar Revisited”. In: *Logical Aspects of Computational Linguistics* (1999), p. 1.

[36] Yujia Li et al. “Gated Graph Sequence Neural Networks”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016.

[37] Yu Liu, Quanming Yao, and Yong Li. “Generalizing tensor decomposition for n-ary relational knowledge bases”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 1104–1114.

[38] Robin Lorenz et al. “QNLP in Practice: Running Compositional Models of Meaning on a Quantum Computer”. In: *ArXiv* abs/2102.12846 (2021).

[39] Yunpu Ma et al. “Variational quantum circuit model for knowledge graph embedding”. In: *Advanced Quantum Technologies* 2.7-8 (2019), p. 1800078.

[40] Ian MacCormack et al. *Branching Quantum Convolutional Neural Networks*. 2020. arXiv: 2012.14439 [quant-ph].

[41] Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. “Yago3: A knowledge base from multilingual wikipedias”. In: *7th biennial conference on innovative data systems research*. CIDR Conference. 2014.

[42] Haggai Maron et al. “Invariant and Equivariant Graph Networks”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[43] Haggai Maron et al. “Provably Powerful Graph Networks”. In: *NeurIPS*. 2019.

[44] K. Meichanetzidis et al. “Grammar-Aware Question-Answering on Quantum Computers”. In: *ArXiv* abs/2012.03756 (2020).

[45] George A Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[46] Tom M. Mitchell. *The Need for Biases in Learning Generalizations*. Tech. rep. Rutgers University, 1980.

[47] Christopher Morris, Gaurav Rattan, and Petra Mutzel. “Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[48] Christopher Morris et al. “Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks”. In: *AAAI*. 2019.

[49] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. “A three-way model for collective learning on multi-relational data”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. 2011, pp. 809–816.

[50] Mateusz Ostaszewski, Edward Grant, and Marcello Benedetti. “Structure optimization for parameterized quantum circuits”. In: *Quantum* 5 (2021), p. 391.

[51] Alejandro Perdomo-Ortiz et al. “Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers”. In: *Quantum Science and Technology* 3.3 (2018), p. 030502.

[52] Hongyu Ren, Weihua Hu, and Jure Leskovec. “Query2box: Reasoning over Knowledge Graphs in Vector Space using Box Embeddings”. In: *International Conference on Learning Representations*. 2020.

[53] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. "You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings". In: *International Conference on Learning Representations*. 2020.

[54] Mehrnoosh Sadrzadeh, Stephen Clark, and Bob Coecke. "The Frobenius anatomy of word meanings I: Subject and object relative pronouns". In: *Journal of Logic and Computation* 23 (2014).

[55] Mehrnoosh Sadrzadeh, Stephen Clark, and Bob Coecke. "The Frobenius anatomy of word meanings II: possessive relative pronouns". In: *Journal of Logic and Computation* 26.2 (2014), pp. 785–815.

[56] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. "Random features strengthen graph neural networks". In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM. 2021, pp. 333–341.

[57] Franco Scarselli et al. "The Graph Neural Network Model". In: *Trans. Neur. Netw.* 20.1 (2009), pp. 61–80.

[58] Michael Schlichtkrull et al. "Modeling relational data with graph convolutional networks". In: *European semantic web conference*. Springer. 2018, pp. 593–607.

[59] Maria Schuld et al. "Circuit-centric quantum classifiers". In: *Physical Review A* 101.3 (2020), p. 032308.

[60] Maria Schuld et al. "Evaluating analytic gradients on quantum hardware". In: *Physical Review A* 99.3 (2019), p. 032331.

[61] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[62] Théo Trouillon et al. "Knowledge Graph Completion via Complex Tensor Factorization". In: *Journal of Machine Learning Research* 18 (2017), pp. 1–38.

[63] Petar Velickovic et al. "Graph Attention Networks". In: *CoRR* abs/1710.10903 (2017).

[64] Guillaume Verdon et al. "Quantum graph neural networks". In: *arXiv preprint arXiv:1909.12264* (2019).

[65] Boris Weisfeiler and Andrei Leman. "The reduction of a graph to canonical form and the algebra which appears therein". In: *NTI, Series* 2.9 (1968), pp. 12–16.

[66] Zhenqin Wu et al. "MoleculeNet: a benchmark for molecular machine learning". In: *Chemical Science* 9.2 (2018), pp. 513–530. ISSN: 2041-6539. DOI: 10.1039/c7sc02664a. URL: http://dx.doi.org/10.1039/c7sc02664a.

[67] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[68] Bishan Yang et al. “Embedding Entities and Relations for Learning and Inference in Knowledge Bases”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[69] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. “Revisiting semi-supervised learning with graph embeddings”. In: *International conference on machine learning*. PMLR. 2016, pp. 40–48.

[70] M. Zaheer et al. “Deep Sets”. In: *NIPS*. 2017.

[71] Muhan Zhang and Yixin Chen. “Link prediction based on graph neural networks”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 5165–5175.

[72] Jin Zheng et al. “Quantum Graph Convolutional Neural Networks”. In: *arXiv preprint arXiv:2107.03257* (2021).