Title
Graph Embeddings, Disentanglement, and Algorithm Maps

Author
Qiu, Frank Yuchen

Publication Date
2023

Peer reviewed|Thesis/dissertation

Graph Embeddings, Disentanglement, and Algorithm Maps

by

Frank Qiu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Statistics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Giles Hooker, Co-chair
Professor Bruno Olshausen, Co-chair
Professor Haiyan Huang
Professor Song Mei

Spring 2023

Graph Embeddings, Disentanglement, and Algorithm Maps

Abstract

Graph Embeddings, Disentanglement, and Algorithm Maps

by

Frank Qiu

Doctor of Philosophy in Statistics

University of California, Berkeley

Professor Giles Hooker, Co-chair

Professor Bruno Olshausen, Co-chair

We tackle three disparate topics in this thesis - graph embeddings, disentanglement, and algorithm maps.

In Chapters 2 and 3, we cover graph embeddings: methods which embed graph as vectors in a structure-preserving way. We start at the principle of superposition used by vector-symbolic architectures and derive the tensor product as the canonical binding operation along with a natural random code. Together, they form a novel embedding method, and we list some common graph operations our embeddings are capable of. We give a precise characterization of its statistical behavior, showing it achieves a packing upper bound. and go on to establish a link to adjacency matrices with applications toward their drastic compression. Then, we compare our method to other bind-and-sum methods and showcase a fundamental memory vs. capacity tradeoff.

In Chapter 4, we cover disentanglement, the discovery of semantically meaningful latent factors. We assume that data lives on a low-dimensional manifold and define disentanglement in terms of local charts. Exploring the consequences of this definition, we find that commutativity of the latent factors is an equivalent condition for disentanglement. Under our framework, we also show that sufficiently rich generative models can always be disentangled, and we apply the commutativity condition to learning a dictionary of Lie group operators. We conclude with a discussion of how our definition relates to previous work, suggesting that it is no accident that commutativity - which has been used a computational or conceptual convenience - is so prevalent in these methods.

In Chapter 5, we give some preliminary thoughts on the transfer of algorithms between different contexts. Our key insight is that, rather than focusing on the transfer of a single algorithm, it is more profitable to consider the conditions in which the states and actions of

one setting can be mapped to those of another setting. This in turn induces a natural map of algorithms between contexts, and we give some basic consequences of these definitions.

To Lynda Qiu and Yang Duobao

List of Figures

List of Tables

Contents

Acknowledgments

This thesis would not be possible with the support, time, and friendship of those I've met while at UC Berkeley. First and foremost, I would like to thank my advisors Professor Giles Hooker and Professor Bruno Olshausen. They were always generous with their time and advice, giving me the encouragement to pursue my interest and providing invaluable help in guiding my research.

The members of the Redwood Center of Theoretical Neuroscience were a fundamental part of my graduate experience: their research and discussions planted the seeds of the topics covered in this thesis and broadened my academic horizons. In particular, I would like to thank Yubei Chen, who acted as a mentor to me for many years. His tireless efforts to include me on projects, ask about my well-being, and provide a limitless supply of interesting ideas were fundamental to my success as a graduate student. I would also like to thank Ho Yin Chau, who along with Yubei was a key collaborator on a project that inspired many ideas in this thesis; his insight and helpful discussions sharpened my understanding of key concepts which laid the foundation for some work in this thesis. Others in the Redwood Center I would like to thank include Christian Shewmake, Brian Cheung, and Connor Bybee.

I would also like to thank the staff of the Statistics Department: from cookie time to navigating paperwork, their help was invaluable. In particular, I would like to thank Mary D. Melinn and La Shana Porlaris, who I caused much trouble for and am eternally grateful for their patience. I would like to thank Professor Bin Yu for serving as my faculty mentor: my membership in BAIR and choice to attend Berkeley were largely due to her advice and guidance.

As an instructor, I would like to first thank Professor Merle Behr and Johnny Hong, whose guidance and example were critical in my first semester as a GSI. I would also like to thank Professor Gaston Sanchez, Professor Hank Ibser, Professor Jared Fisher, Professor Aditya Guntuboyina, and Profesor Likun Zhang who I had the pleasure of being a GSI for. I would also like to thank all my fellow GSIs that I worked with, whose company and help were greatly appreciated.

In the Statistics department, I've been the beneficiary of the generosity and warmth of many fellow students. I would like to thank all the students in my cohort; our mutual support and commiseration helped me through the most difficult years of my degree: Olivia Angiuli, Jacob Calvert, Miyabi Ishihara, Tyler Maltba, Mehdi Ouaki, Briton Park, Alexander Tsigler, and Zoe Vernon. In particular, I would like to thank Mehdi Ouaki for the many years of friendship during our time as roommates: movie nights were something I looked forward to every week. I would also like to thank Zsolt Bartha, Billy Fang, Arturo Fernandez, Ella Hiesmayr, Runjing Liu, Benji Lu, Jake Soloff, Dan Soriano, Simon Walter, Jason Wu, and many other peers who helped brighten my day.

Finally, I would like to thank the other members of my committee: Professor Haiyan Huang and Professor Song Mei. Their suggestions after my qualifying exams helped guide the research presented in this thesis, and I am grateful for the time they sacrificed by serving on both my qualifying and thesis committees.

Chapter 1

Introduction

This thesis is divided into three primary topics: Chapters 2 and 3 deal with graph embeddings, Chapter 4 with disentanglement, and Chapter 5 is preliminary work on the transfer of algorithms. In addition, the appendices contain supplementary material on graph hierarchical models as well as experimental disentanglement results.

1.1 Graph Embeddings

Graph embeddings are a family of methods that embed graphs as vectors, so that we may work with them as we would other real-valued data. A good embedding is one that captures the structure of the graph, allowing us to extract information directly from the embedding without accessing the graph itself. Some methods achieve this by embedding a graph's vertices in a way that encodes their connectivity [10] [1]; others seek to factorize a graph's adjacency matrix or tensor to capture relation information [23]. The method in this thesis takes the approach of embedding a graph by embedding its edgeset in a manner that allows for flexible manipulation and graph operations.

To do this, we draw inspiration from the field of vector symbolic architectures (VSAs) [15] [13] [9], which seek to augment vector spaces with the capacity to represent symbolic structures. VSAs do this by equipping a vector space with two primary operators: the binding and superposition operators. The binding operator generates vectors that represent the symbolic binding of two entities, and the superposition operator, usually summation, is used to generate vectors that represent many entities at once. Our method uses the original binding operator - the tensor product - which was first proposed by Smolensky [26] along with summation as the superposition operator.

In Chapter 2, we introduce our method and show its rich representational capacity. We give a statistical characterization of its behavior, demonstrating it achieves a packing upper bound. Moreover, we show our binding operator is the most general method with respect to superposition as summation and that our method is a generalization of adjacency matrices with application towards their compression. In Chapter 3, we compare our method's memory

and capacity to other popular bind-and-sum schemes. We show that compressing the tensor product not only hinders the representational capability of the embeddings but also incurs a proportionate penalty in the capacity, challenging criticisms of the tensor product's memory expense relative to alternative binding operations. Finally, in the appendices we introduce some cursory thoughts on superposition as a computational paradigm and graph hierarchical models.

1.2 Disentanglement

Disentanglement is a term with many interpretations in the machine learning community, but broadly it is the discovery and separation of semantically meaningful latent factors in a dataset. There is a rich literature tackling it from many angles: statistical independence [5] [8] [11], factorized generative models [2] [4] [28], and even group-theoretical approaches [3] [7] [12]. In Chapter 3 , we interpret disentanglement in the context of the manifold hypothesis, which states that high-dimensional data lies on or near a low-dimensional manifold. From this perspective, we define disentanglement as the discovery of an atlas of local charts for the data manifold, where the local coordinates all correspond to semantically meaningful factors. Exploring the consequences of this definition, we find that commutativity between the latent factors is a necessary and sufficient condition for disentanglement. This resulting commutativity criterion is useful for easily checking the validity of a set of candidate factors as well as a guiding principle in the construction of disentangling techniques. Furthermore, under our framework we show how sufficiently rich generative models can always be disentangled in the local chart sense. We conclude with an application of the commutativity criterion to the learning of matrix operators, demonstrating experimentally that a system of matrix exponentials can be well-approximated by a commuting system of exponentials. Included in the appendices are experimental results showing the emergence of local charts in the latent code of a deep autoencoder.

1.3 Transfer of Algorithms

The ability to generalize strategies in one context to another is a primary research topic of interest, and in Chapter 5 we give some preliminary work on a formal framework for describing this transfer. This work was partly inspired by the phenomena of skill transfer between video games and partly by attempts to define an isomorphism of Turing machines [29]. In both cases, the transfer of strategies requires both a map of context states as well as allowable actions. For example, skilled players in one fighting game tend to be skilled at other fighting games, even though the mechanics and controls may differ. Similarly, the rule table of a Turing machine is tied to the symbols the machine can read and write with, and implementing the rule table in another machine requires translating the original symbols to the ones of new machine. We formalize this observation by defining worlds - sets with an

associated set of functions - and players that operate in such worlds. We go on to define world maps that naturally map sets of players between worlds, and we give some basic results concerning these player maps. While this work is far from finished, we hope this section provides some entertainment and food for thought.

Chapter 2

Graph Embeddings via Tensor Products and Orthonormal Codes

2.1 Introduction

Our method specifically represents a graph by embedding its edge set as a real-valued matrix. To do this, we borrow two key ingredients from the VSA approach: the principle of superposition - the embedding of a set is the sum of its constituents' embeddings - and a binding operation that generates edge embeddings. Thus, we represent the edgeset as the sum of its edge embeddings. This comprises the first part of our method, and the second part is the vertex code used to generate the edge embeddings. We shall describe both of these components and analyze them in separate sections.

The general problem of embedding symbolic structures into continuous spaces is a major area of focus in the field of vector symbolic architectures, and this paper draws on key ideas from the field - namely superposition and binding. Indeed, our binding method was first proposed in [26] by Paul Smolensky, who introduced it in the general context of role-filler bindings of arbitrary order. In this work, we study it in the specific context of graphs, where the binding order is fixed at two (two vertices bound into an edge). This fixing of the binding order sidesteps a major issue with the tensor product, whose dimension explodes with the binding order. In fact, in a companion paper we compare the tensor product to other popular binding methods that were specifically proposed as memory-efficient alternatives; in that paper, we find that, at least for graphs, those alternatives offer no memory advantage and sacrifice the tensor product's expressiveness. Similarly, our use of spherical codes was also inspired by previous work by [13], [15],[6], which leveraged the fact that high-dimensional vectors are nearly orthogonal with high probability. Indeed, our spherical code is of the same family as the binary and phasor codes proposed in those papers, where the code vectors are all unit vectors. Furthermore, [22], [20], [25], [14], [24] are examples of graph embedding

approaches that employ the same bind-and-sum approach taken in this paper.

2.2 Notation and Terminology

Our work focuses on embedding objects - vertices, edges, graphs - into vector spaces. Therefore, we shall denote the objects using bolded letters and their embeddings with the corresponding unbolded letter. For example, we denote a graph using $\boldsymbol{G}$ and its embedding using G. In later sections, when there is no confusion we shall drop the distinction between the object and the embedding and just use the unbolded letter.

We also adopt graph terminology that may be non-standard for some readers. There are many names for the two vertices of a directed edge $(d, c) = d \to c$. In this paper, we shall call d the domain of the edge and c the codomain. Unless stated otherwise, all graphs will be directed graphs.

2.3 Method Overview

Given some set $\boldsymbol{V}$, we propose a method for embedding the family of graphs that can be made from $\boldsymbol{V}$. Specifically, for any directed graph $\boldsymbol{G} = (\boldsymbol{V_G}, \boldsymbol{E_G})$ such that $\boldsymbol{V_G} \subseteq \boldsymbol{V}$ and $\boldsymbol{E_G} \subseteq \boldsymbol{V} \times \boldsymbol{V}$, we embed $\boldsymbol{G}$ by embedding its edge-set $\boldsymbol{E_G}$ into a vector space. To this end, we first embed the large vertex set $\boldsymbol{V}$ by assigning each vertex to a d-dimensional unit vector, drawn independently and uniformly from the unit hypersphere $\mathbb{S}^{d-1}$. We then embed each directed edge $(\boldsymbol{d}, \boldsymbol{c})$ in the edgeset by the tensor product of their vertex embeddings:

$$(\boldsymbol{d}, \boldsymbol{c}) \mapsto d \otimes c$$

Fixing the standard basis, the tensor product is the outer product dc^T. Then, the embedding G of the graph $\boldsymbol{G}$ - which we represent by its edgeset $\boldsymbol{E_G}$ - is the sum of its edge embeddings:

$$\boldsymbol{G} \mapsto G = \sum_i d_i \otimes c_i = \sum_i d_i c_i^T$$

Our embedding method can be separated into two parts: the spherical code used to embed vertices and the tensor product used to bind vertices into edges. Later sections will analyze both separately as well compare them to other coding-binding schemes.

Finally, while we represent a graph $\boldsymbol{G}$ by representing just its edgeset $\boldsymbol{E_G}$, we can also embed its vertex set as well by augmenting $\boldsymbol{E_G}$ with self-loops $(\boldsymbol{v}, \boldsymbol{v})$ for each $\boldsymbol{v} \in \boldsymbol{V_G}$. This introduces confusion between a vertex and its self-loop, but for certain graphs,such as directed acyclic graphs, this is not a concern. However, we will focus on the original method of solely embedding the edgeset in this paper.

2.4 Graph Operations

Key Assumption: (Nearly) Orthonormal Vertex Codes

In this section, we will give an overview of some core graph operations possible under the proposed embedding framework. One key property we require is that the vertex codes be (nearly) orthonormal vectors, and this is the primary motivation behind the our use of spherical codes. Later sections will give a more precise analysis of near orthonormality, and for clarity we shall assume exactly orthonormal vertex codes in this section.

Edge Addition/Deletion

Adding/deleting the edge (a, b) to a graph corresponds to simply adding/subtracting ab^T from its graph embedding G.

Outbound/Inbound Vertices

For a given vertex d in G, suppose we wanted to find all the vertices c_i that a vertex d points to: (d, c_i). To do this, we multiply G by d on the left:

$$d^T G = d^T(\sum_i d_i c_i^T) = \sum_i \langle d, d_i \rangle c_i^T = \sum_{d_i = d} c_i^T$$

In the last equality, we use the fact that the vertex embeddings are orthonormal. Since we represent sets as sums, this result is precisely the set of vertices that d points to.

Conversely, say we were interested in finding all vertices d_i that point to a vertex c: (d_i, c). This would analogously correspond to right multiplying by c:

$$Gc = (\sum d_i c_i^T)c = \sum \langle c_i, c \rangle d_i = \sum_{c_i = c} d_i$$

We can generalize this to a set of candidate vertices. For example, we first represent a set of vertices S by the superposition of its constituents $S = s_1 + \cdots + s_n$. Then, to find all vertices in G that are connected to any vertex in S by an inbound edge, we multiply G on the right by S:

$$GS = (\sum d_i c_i^T)(s_1 + \cdots + s_n) = \sum_{c_i \in S} d_i$$

Analogously, to find all vertices in G connected to S by an outbound edge, we multiply G on the left by S^T. All subsequent operations are also multilinear like the vertex query, and so they can all be similarly extended to handle sets of vertices or edges. For simplicity, we will focus on just the singleton case from now on.‘

Edge/Node Queries

Combining the two operations above, checking if a graph G contains the edge (d, c) would correspond to multiplying G by d on the left and c on the right:

$$d^T G c = d^T (\sum d_i c_i^T) c = \sum \langle d, d_i \rangle \langle c, c_i \rangle = 1_{\{(d,c) \in E_G\}}$$

Now suppose G were augmented with self-loops for each vertex, where G contains the loop vv^T for each of its vertices v. One can use the above procedure to detect if G contains a given vertex.

Edge Composition, k-length Paths, and Graph Flow

We can also perform edge composition with the graph embeddings. For example, given edges (a, b) and (b', c), we want to return the composite edge (a, c) only if $b = b'$. This can be done by matrix multiplication of the edge embeddings:

$$(ab^T)(b'c^T) = \langle b, b' \rangle ac^T = 1_{\{b=b'\}} ac^T$$

Therefore, we can compose all edges in a graph G by computing its second matrix power:

$$G^2 = (\sum d_i c_i^T)(\sum d_j c_j^T) = \sum_{i,j} \langle c_i, d_j \rangle d_i c_j^T = \sum_{c_i = d_j} d_i c_j^T$$

Generalizing to paths of length k, the matrix powers of G correspond to the sets of paths of a fixed length:

$$G^k = \{\text{all } k\text{-length paths in } G\}$$

We can combine this with vertex queries to compute graph flows. Say one wanted to know, starting at initial vertex v, all vertices c that are reachable from v by a path of length k. Then one would compute:

$$d^T G^k = d^T (\sum d_i c_i^T)(G)^{k-1} = \sum_{d_i = d} c_i^T (G^{k-1}) = \cdots = \{c \text{ such that there is a } k\text{-length path } d \to c\}$$

An analogous operation exists for determining all vertices d that end up at final vertex c after a path of length k:

$$G^k c = \{d \text{ such that there is a } k\text{-length path } d \to c\}$$

Edge Flipping, Undirected Graphs, and Alternization

Given an edge (a, b) with representation ab^T, the representation of the flipped edge (b, a) is the transpose matrix:

$$ba^T = (ab^T)^T$$

Therefore, the embedding of the dual graph G^{op} - flipping all edges of G- is the transpose of G.

$$G^{op} = G^T$$

One interesting application of this property is to extend our current graph framework to undirected graphs using the matrix symmetrization procedure:

$$G \mapsto G + G^T$$

This procedure may introduce double-counting of certain edges, but for certain graphs, like directed acyclic graphs, this is not an issue.

Subsetting and Subgraphs

In a previous section, we saw that vertex queries can determine the vertices connected to or from a set of vertices $\boldsymbol{S} \subseteq \boldsymbol{V}$. Now, suppose we wanted to know not the vertices but the edges whose domain is in $\boldsymbol{S}$. Abusing notation, let S denote the matrix whose columns are the vertex embeddings $s \in S$, and let $P_S = SS^T$ be the associated projection matrix. To find all edges whose domain is in S, we compute:

$$P_S G = SS^T(\sum d_i c_i^T) = \sum \langle s_j, d_i \rangle s_j c_i^T = \sum 1_{\{s_j = d_i\}} s_j c_i^T = \sum_{d_i \in S} d_i c_i^T$$

Similarly, to find all edges whose codomain is in S, we compute:

$$GP_S = (\sum d_i c_i^T) SS^T = \sum \langle s_j, c_i \rangle d_i s_j^T = \sum 1_{\{s_j = c_i\}} d_i s_j^T = \sum_{c_i \in S} d_i c_i^T$$

Now, say we wanted to extract the full subgraph G_S of G, whose edges are those of G that have both domain and codomain in S. Combining the above two equations, this amounts to conjugating G with P_S:

$$G_S = P_S G P_S$$

Translation between Vertex Codes

Suppose we have two different vertex codes $\phi_1 : V \to V_1$ and $\phi_2 : V \to V_2$. For a graph G, we then have a graph embedding induced from each vertex code, denoted $\phi_1(G)$ and $\phi_2(G)$. There is a natural way to convert $\phi_1(G)$ to $\phi_2(G)$. Let Φ_1 be the matrix who columns are the vertex codes $\{\phi(v_1), \phi(v_2), \cdots\}$ in that order; let Φ_2 be the similarly defined matrix for ϕ_2. Then, we have the transition map between vertex codes $\Psi : V_1 \to V_2$ represented by the matrix $\Phi_2 \Phi_1^T$:

$$\Psi : V_1 \to V_2 \quad ; \quad \phi_1(v_i) \mapsto \phi_2(v_i) \quad ; \quad \Psi = \Phi_2 \Phi_1^T$$

Then, the graph representations $\phi_1(G)$ and $\phi_2(G)$ follow the relation:

$$\phi_2(G) = \Psi \phi_1(G) \Psi^T$$

Counting via the Trace

The trace of a graph embedding G will count the number of self-loops in the graph:

$$tr(G) = tr(\sum d_i c_i^T) = \sum \langle d_i, c_i \rangle = \sum 1_{\{d_i = c_i\}}$$

This property can be used for many interesting graph operations.

Vertex Counting in Augmented Graphs

For a graph G whose edgeset set is augmented with the self-loops vv^T for every v in its vertex set V_G, the trace will naturally return the cardinality of its vertex set:

$$tr(G) = tr(\sum d_i c_i^T) = \sum \langle d_i, c_i \rangle = |V_G|$$

Edge Counting and a Natural Metric

To count the number of edges in a graph $G = \sum d_i c_i^T$, one computes:

$$tr(G^T G) = tr((\sum c_i d_i^T)(\sum d_j c_j^T)) = tr(\sum \langle d_i, d_j \rangle c_i c_j^T) = \sum \langle d_i, d_j \rangle \langle c_i, c_j \rangle = |E_G|$$

This gives a nice relation to the Frobenius norm $||\cdot||_F$ via the identity $||A||_F^2 = tr(A^T A)$. This shows that the squared Frobenius norm of a graph representation is precisely the cardinality of its edge set. Hence, the natural metric on our graph embeddings would be the one induced from the Frobeinus norm:

$$d(G, H) \coloneqq ||G - H||_F = \sqrt{\#\text{ of different edges}}$$

where this metric depends solely on the number of different edges between the two graphs.

Testing Graph Homomorphisms

Finally, one interesting application of the trace is to quantify the 'goodness' of a proposed graph homomorphism. Recall that a graph homomorphism f from graph G to graph H is comprised of a vertex function $f_1 : V_G \to V_H$ and edge function $f_2 : E_G \to E_H$ such that every edge (a, b) is mapped by f_2 to the edge $(f_1(a), f_1(b))$. Hence, every homormorphism is completely determined by its vertex function, and so it suffices to test if a vertex function $f : V_G \to V_H$ induces a graph homomorphism. Therefore, for every proposed vertex function f we assign a quantity called the graph homomorphism coefficient, described in detail below.

Given two graphs $G = \sum d_i c_i^T$ and $H = \sum a_j b_j^T$ with proposed vertex function $f : V_G \to V_H$, the vertex function f induces the following map:

$$G = \sum d_i c_i^T \mapsto \sum f(d_i) f(c_i)^T = f(G)$$

Note that $f(G)$ can be computed using the method mentioned in the previous section. If f is truly a graph homomorphism, then $f(G)$ is a subgraph of H and every edge of $f(G)$ is an edge of H. We therefore compute the number of matching edges between $f(G)$ and H, denoted δ_f:

$$\delta_f = tr(f(G)^T H) = tr((\sum f(c_i)f(d_i)^T)(\sum a_j b_j^T)) = \sum \langle f(d_i), a_j\rangle \langle f(c_i), b_j\rangle$$

Dividing δ_f by $|E_G| = tr(G^T G)$ gives the fraction of edges in $f(G)$ that have matches in H. We call this ratio the graph homomorphism coefficient, denoted as $\Delta_f \coloneqq \frac{\delta_f}{|E_G|}$. Note that $\Delta_f \in [0, 1]$. If f truly induces a graph homomorphism, then every edge has a match and $\Delta_f = 1$; conversely, if $f(G)$ has no matching edges in H, then $f(G)$ is totally different from H and $\Delta_f = 0$. In this sense, the graph homomorphism coefficient gives a measure of how close a vertex function f is to inducing a graph homomorphism.

Vertex Degree and Gram Matrices

In a directed graph G, consider a fixed vertex v. Then, the in-degree of v, $in(v)$, is the number of edges that go into v - ie. have codomain v. Similarly, the out-degree of v, $out(v)$, is the number of edges that go out of v - ie. have domain v. Generalizing these definitions, we define the joint in-degree of two vertices v and w, $in(v, w)$, as the number of vertices d_i that have both an edge into v and an edge into w. The joint out-degree of two vertices v and w, $out(v, w)$, is similarly defined. Note that using the generalized definitions, the in/out-degree of a vertex with itself coincides with the single-case definition. Using our graph embeddings, there is a natural way to compute both of these degrees.

We first handle the in-degree of a vertex. Recall that Gv is the superposition, or sum, of all vertices d_i that go into v with an edge (d_i, v). Since vertex codes are orthonormal, we can use the squared Euclidean norm to count the number of vertices in superposition:

$$||Gv||^2 = (Gv)^T Gv = v^T (G^T G)v = in(v)$$

In fact, for two vertices v and w, we again appeal to orthonormality of the vertex codes to compute their joint in-degree:

$$(Gv)^T Gw = v^T (G^T G)w = in(v, w)$$

Thus, we see that the Gram matrix $G^T G$ represents a bilinear form that computes the joint in-degree of two vertices:

$$G^T G : V \times V \to \mathbb{R} \qquad ; \qquad v \times w \mapsto v^T (G^T G)w = in(v, w)$$

We shall called the gram matrix $G^T G$ the in-degree matrix.

A similar line of reasoning shows that the Gram matrix GG^T is a bilinear form that computes the joint out-degree between two vertices. Hence, we shall call GG^T the out-degree matrix. Immediately, one natural question when working with Gram matrices are their spectral properties. Indeed, there are natural links to properties of graph connectivity, which we discuss in the next section.

Graph Connectivity

We embed a graph G as a matrix $\mathbb{R}^{d\times d} \cong V \bigotimes V$, and so we may view each graph embedding as a linear operator on the vertex code space V. This perspective yields some interesting links to the connectivity properties of graphs.

Invariant Subspaces and Connected Components of Undirected Graphs

Given a linear operator $T : V \to V$, a subspace $W \subseteq V$ is an invariant subspace of T if $T(W) \subseteq W$. An invariant subspace W is irreducible if it contains no non-trivial invariant subspace. Now, consider a directed acyclic graph G, its dual G^T, and a subset of its vertices $\{v_1, \cdots, v_n\} \subseteq V_G$. Let $W = span(\{v_1, \cdots, v_n\})$, and let $U(G) = G + G^T$ be its induced undirected graph.

Suppose W were an invariant subspace of G, or equivalently $Gv_i \in W$ for all v_i. As Gv_i is the sum of all vertices d_j that connect to v_i via an outbound edge, this means that the only in-bound connections to $\{v_1, \cdots, v_n\}$ are from its members. Similarly, the condition that W be an invariant subspace of G^T is equivalent with the condition the only out-bound to $\{v_1, \cdots, v_n\}$ are from its members. Thus, if we consider the induced undirected graph $U(G) = G+G^T$, W is an invariant subspace iff all paths starting at any v_i terminate at some other v_j. Hence, W is an invariant subspace iff the underlying set of vertices $\{v_1, \cdots, v_n\}$ is a union of connected components of $U(G)$. Immediately, $\{v_1, \cdots, v_n\}$ is a connected component iff W is an irreducible invariant subspace. These are the weakly connected components of the directed graph G.

2.5 Theoretical Derivation of the Embedding Method

In this section, we derive the tensor product binding from principle of superposition and show that it is the most 'general' embedding method. We also explore how orthonormal codes and the tensor product are naturally derived when considering suitability for certain graph operations.

Superposition and the Tensor Product

We shall derive the tensor product from the principle of superposition. Indeed, along the way we shall establish the following connection between the principle of superposition and the tensor product:

Theorem 2.5.1. *For any fixed vertex code $\phi : \boldsymbol{V} \to V$, let $\psi : V \times V \to \mathbb{R}^n$ be any binding operation that respects superposition. Then the resulting bound code $\psi(V, V)$ has a unique linear derivation from the tensor product $V \bigotimes V$. In this sense, the tensor product is the most general binding operation.*

Proof. Assume we are given a vertex code $\phi : \boldsymbol{V} \to V$. Given a directed graph $G = (V_G, E_G)$ in V, our task is to represent G by embedding edge set E_G via a superposition of its edges. First, suppose G has multiple edges from a common domain $\{d\}$ to multiple codomains $\{c_1, \ldots, c_k\}$:

$$\{(d, c_1), \cdots, (d, c_k)\} = \bigcup_i \{(d, c_i)\}$$

Simiarly, consider the reverse situation to multiple domains with a single codomain. Then, any edge embedding $\psi : V \times V \to \mathbb{R}^n$ that respects superposition must satisfy the following two equations:

$$\psi(\{(d, c_1), \cdots, (d, c_k)\}) = \sum \psi(\{(d, c_i)\})$$
$$\psi(\{(d_1, c), \cdots, (d_{k'}, c)\}) = \sum \psi(\{(d_j, c)\})$$

Thus, ψ induces a bilinear function $\tilde{\psi}$ on the vertex code via:

$$\tilde{\psi} : V \times V \to \mathbb{R}^n$$
$$(\phi(v), \phi(w)) \mapsto \psi(v, w)$$

By the universality of the tensor product ([16]), the bilinear $\tilde{\psi}$ map has a corresponding unique linear map $\psi^* : V \bigotimes V \to \mathbb{R}^n$, and so the tensor product uniquely maps into every edge embedding that respects superposition. □

For example, three common alternative binding operations - Hadamard product, convolution, and circular correlation - all are linearly induced from the tensor product. Representing the tensor product as the outer product matrix, the Hadamard product is the diagonal of the matrix, while the convolution and circular correlation are sums along pairs of diagonals. Therefore, given a vertex code ϕ we may as well consider its natural induced edge embedding under the tensor product. Fixing a basis, this can be expressed as the outer product of the vertex codes:

$$\tilde{\phi} : V \times V \to V \bigotimes V \quad ; \quad (d, c) \mapsto d \otimes c = dc^T$$

Derivation from Graph Operations

In this section, we shall show that the tensor product and orthonormal codes can be naturally derived when considering suitability toward graph operations. While we can analyze many possible graph operations and still get the same result, for brevity we shall focus on the vertex query function Q:

$$Q : (V \times G) \to V$$

that takes as input a vertex-graph pair (v, G) and returns all the vertices that v points to or is connected to. We shall consider a general edge-binding operation ψ, and denote edges

under this binding operation as $\psi(v, w)$. As usual, we assume that both Q and ψ respect superposition.

Since Q respects superposition and hence is a multilinear function, we can completely characterize it by considering its actions on some basis set $b_i \times \psi(b_j, b_k)$. Let us assume some subset of vertex embeddings form a basis for V, and let us examine Q applied to a single edge: $Q(u, \psi(v, w))$. For notational simplicity, we will drop the ψ and just represent edges as tuples (v, w). Ideally, we want:

$$Q(u, (v, w)) = \begin{cases} w & u = v \\ 0 & u \neq v \end{cases}$$

However, this condition makes Q a non-linear function. Instead, the 'closer' u is to v, we want the output of Q to be closer to w; on the other hand, if u is totally 'different' from v, we want the query function to return nothing: the zero vector. In fact, let us fix w, so now the function has just two arguments:

$$Q(-, (-, w)) = Q_w(-, -)$$

By fixing w, in light of the previous discussion we see that Q_w interpolates between w and 0 depending on the similarity of its two arguments. Equivalently, this means we can factorize Q_w into a similarity function S multiplying w, where the output of S is in $[0, 1]$.

$$Q_w(-, -) = S(-, -)w$$

The similarity function S checks for agreement between query vertex u and source vertex v, and the closer they are the closer the output is to w. Ideally, we want S to return 1 (and hence an output of w) whenever $u = v$:

$$S(u, v) = 1 \quad \text{if } u = v$$

There is one more thing we need to consider: the orthogonal projection. Consider the orthogonal decomposition of v onto u: $v = v_u + v_u^{\perp}$. By linearity, our similarity function becomes:

$$S(u, v) = S(u, v_u) + S(u, v_u^{\perp}) = \frac{\langle u, v \rangle}{\langle u, u \rangle} S(u, u) + S(u, v_u^{\perp})$$

Consider the second term $S(u, v_u^{\perp})$. It is natural that this term be 0, since these two vectors are not equal/are orthogonal. As we agreed earlier that $S(u, u) = 1$, our similarity function becomes:

$$S(u, v) = \frac{\langle u, v \rangle}{\langle u, u \rangle}$$

We can arbitrarily scale S by some constant and shrink u by that same constant without affecting the outcome, so for convenience let us now assume all vertices are unit vectors. Then, our similarity function becomes:

$$S(u, v) = \langle u, v \rangle$$

Picking any unit basis $\{b_i\}$ of V and applying the same analysis to all pairs of basis vectors, we come to the general equation:

$$S(b_i, b_j) = \langle b_i, b_j \rangle$$

By linearity this completely determines S, and so S is just the usual dot product.

Having determined the form of S, our query function Q can now be expressed as:

$$Q(u, (v, w)) = \langle u, v \rangle w$$

This holds for any w, and so by multilinearity our query function takes the form:

$$Q(\sum u_i, \sum_j (v_j, w_j)) = \sum_i (\sum_j \langle u_i, v_j \rangle w_j) = \sum_i u_i^T (\sum_j v_j w_j^T)$$

Focusing on the graph term $\sum_j v_j w_j^T$, we see this is the graph embedding using the tensor product as a binding operation. To summarize, we derived the form the vertex query Q should take using just multilinearity and some natural conditions, and this natural form is a linear computation involving our proposed embedding method. Hence, regardless of the binding method ψ, the query function Q can be equivalently computed using the graph embedding under the tensor product. Moreover, the natural similarity function, after normalization, is the dot product, and this suggests orthogonality is needed to distinguish the vertex codes.

Considering just the vertex query function, we naturally derive our graph embedding scheme. Indeed, this derivation is connected to superposition and multilinearity. Any graph operation that respects superposition must be linear in each argument; as all multilinear functions filter uniquely through the tensor product, all natural graph operations can be derived from corresponding operations on the tensor product. Thus, we could similarly derive the tensor product when considering any other natural graph operation.

2.6 Spherical Codes and Approximate Orthonormality

In previous sections, we assumed our vertex code to be exactly orthonormal. However, this orthonormality requirement is not efficient in the number of codes we can pack into a space, as we can pack at most d orthonormal vectors into $\mathbb{R}^d$. Instead, we can relax strict orthonormality to pack a much larger number of vertex codes into the same space, and so we pass to nearly orthonormal codes via spherical codes. Two natural questions arise: how many nearly orthogonal vectors can one pack into $\mathbb{R}^d$, and how close do spherical codes come to achieving this limit? In this section we shall answer these two questions and give an account of approximate orthonormality in high dimensions.

Packing Upper Bounds

First, we give an upper bound on the number of approximately orthornormal vectors one can pack in $\mathbb{R}^d$. To make approximate orthonormality precise, we say the unit vectors u, v are ϵ-*orthogonal* if $|\langle u, v\rangle| < \epsilon$. Using the Johnson-Lindenstrauss Theorem, we derive the upper bound as a corollary:

Theorem 2.6.1 (Johnson-Lindenstrauss)**.** *Let $x_1, \cdots, x_m \in \mathbb{R}^N$ and $\epsilon \in (0,1)$. If $d > \frac{8ln(m)}{\epsilon^2}$, then there exists a linear map $f : \mathbb{R}^N \to \mathbb{R}^d$ such that for every x_i, x_j:*

$$(1-\epsilon)||x_i - x_j||^2 \leq ||f(x_i) - f(x_j)||^2 \leq (1+\epsilon)||x_i - x_j||^2$$

Corollary 2.6.1.1. *For any $\epsilon \in (0, \frac{1}{2})$, one can pack at most $O(e^{Cd\epsilon^2})$ ϵ-orthogonal unit vectors in $\mathbb{R}^d$ for some universal constant C.*

Proof. Let m be any integer such that $d > \frac{8ln(m+1)}{\epsilon^2}$. Consider the set of $(m+1)$ vectors $\{e_1, \cdots, e_m, 0\}$ in $\mathbb{R}^m$. These satisfy the conditions of the Johnson-Lindenstrauss Theorem, and let f be the JL map. Then,

$$(1-\epsilon) = (1-\epsilon)||x_i||^2 \leq ||f(x_i) - 0||^2 = ||f(x_i)||^2 \leq (1+\epsilon)$$

Hence, since $||e_i - e_j||^2 = 2$ for $i \neq j$:

$$2(1-\epsilon) \leq ||f(e_i) - f(e_j)||^2 = ||f(e_i)||^2 + ||f(e_j)||^2 + 2\langle f(e_i), f(e_j)\rangle \leq 2(1+\epsilon)$$

Plugging the first equation to the second, we have:

$$-2\epsilon \leq \langle f(e_i), f(e_j)\rangle \leq 2\epsilon$$

Thus, using $u_i = \frac{f(e_i)}{||f(e_i)||}$, the first equation, and the fact that $\epsilon < \frac{1}{2}$:

$$-4\epsilon \leq -\frac{2\epsilon}{(1-\epsilon)} \leq \langle u_i, u_j\rangle \leq \frac{2\epsilon}{1-\epsilon} \leq 4\epsilon$$

Therefore we have:

$$D > 8ln(m+1)/\epsilon^2 \implies m = O(e^{Cd\epsilon^2})$$

for some constant C. □

This upper bound is in fact shown to be tight [17].

Spherical Codes

In practice, we use spherical codes - sampling uniformly from the d-dimensional unit hypersphere $\mathbb{S}^{d-1}$ - to generate nearly orthornormal vertex codes. In this section, we give the exact distribution of the dot product between spherical codes and show that they achieve the previous packing upper bound.

Theorem 2.6.2. *If u, v are uniformly distributed over $\mathbb{S}^{d-1}$, let $X = \langle u, v\rangle$ and $Y = \frac{X+1}{2}$. Then, Y follows a Beta($\frac{d-1}{2}, \frac{d-1}{2}$) distribution.*

Proof. Since both u and v are uniformly distributed over $\mathbb{S}^{d-1}$, by symmetry we may fix v as any unit vector without changing the distribution of X. Hence, let v be the first coordinate vector e_1, and so $X = \langle u, e_1\rangle$. The set of vectors u such that $\langle u, e_1\rangle = x$ form a spherical section of $\mathbb{S}^{d-1}$: a $(d-1)$-sphere of radius $\sqrt{1-x^2}$. Hence, the set of vectors with dot product $x \le X \le x+\delta$ corresponds to a d-dimensional belt on the sphere. Since the probability of a set is proportional to its surface area, $P(x \le X \le x+\delta)$ is proportional to the surface area of the belt. The area of a d-dimensional sphere of radius r is Cr^{d-1} for some constant C, and if $\theta = \cos^{-1}(X)$ we have:

$$\begin{aligned} P(x \le X \le x+\delta) &\propto \int_{\cos^{-1}(x)}^{\cos^{-1}(x+\delta)} (\sqrt{1-cos^2(\theta)})^{d-2} d\theta \\ &= \int_x^{x+\delta} (\sqrt{1-t^2})^{d-2} d(\cos^{-1} t) \\ &\propto \int_x^{x+\delta} (1-t^2)^{\frac{d-3}{2}} dt \end{aligned}$$

Hence, if f_X is the density of X, $f_X \propto (1-x^2)^{\frac{d-3}{2}}$. Then, letting $X+1 = 2Y$, we can simplify:

$$(1-x^2)^{\frac{d-3}{2}} = (1-x)^{\frac{d-3}{2}}(1+x)^{\frac{d-3}{2}} = (2-2y)^{\frac{d-3}{2}}(2y)^{\frac{d-3}{2}} \propto (1-y)^{\frac{d-3}{2}} y^{\frac{d-3}{2}}$$

Thus, $Y = \frac{X+1}{2}$ follows a Beta($\frac{d-1}{2}, \frac{d-1}{2}$) distribution □

Corollary 2.6.2.1. *The dot product X between u, v, uniformly distributed over $\mathbb{S}^{d-1}$, has $E(X) = 0$ and $Var(X) = \frac{1}{d}$.*

Now, given a set of k spherical codes, we want to ensure that all of them are nearly orthogonal. Surprisingly, we shall see that for a fixed probability of violating ϵ-orthonormality, the number of spherical codes k will be $O(e^{Cd\epsilon^2})$ for some constant C, matching our upper bound: spherical codes are optimal.

Theorem 2.6.3. *For k vectors $x_1, \cdots, x_k$ sampled iid from the uniform distribution on $\mathbb{S}^{d-1}$ and for any $\epsilon > 0$, we have:*

$$\max_{i,j} |\langle x_i, x_j, |\rangle < \epsilon$$

with probability at least $1 - 2\binom{k}{2} e^{-\frac{d}{2}\epsilon^2}$

Proof. From Corollary 2.6.2.1, the dot product X is a random variable bounded absolutely by 1 and with variance $\frac{1}{d}$. Hence, Bernstein's inequality gives:

$$\begin{aligned} P(|X| > \epsilon) &= 2P(X > \epsilon) && X \text{ is symmetrically distributed about 0} \\ &\leq 2e^{\epsilon^2/(\frac{2}{d}+\frac{2\epsilon}{3})} && \text{Bernstein's inequality} \\ &\leq 2e^{\epsilon^2/\frac{2}{d}} \\ &= 2e^{\frac{d}{2}\epsilon^2} \end{aligned}$$

Using a union bound over all $\binom{k}{2}$ pairs gives the result. □

Hence, for a fixed error threshold $P(\max_{i,j} |\langle x_i, x_j \rangle| > \epsilon) \leq T$, we can choose the optimal number of vectors k that still put us under the threshold:

$$T \approx C_2 \binom{k}{2} e^{-C_1 d\epsilon^2} \implies k \approx O(e^{C_3 d\epsilon^2})$$

for universal constants C_1, C_2, C_3. This matches the upper bound given by Theorem 2.6.1, and choosing random unit vectors is optimal.

2.7 Relationship to Adjacency Matrices

A few of the proposed graph operations in the previous section might seem familiar to the operations one can do with adjacency matrices. In fact, in the ideal case of exactly orthonormal vertex codes, our proposed method is a generalization of them. In this section, we shall discuss this connection and explore its implications.

Generalization of Adjacency Matrices

Given a vertex set $\boldsymbol{V}$, suppose we encode each vertex as a unique coordinate vector. Then, for any graph G in $\boldsymbol{V}$, its graph embedding is its adjacency matrix. Moreover, given any other orthonormal code $\{v_i\}$, there is always an orthogonal change of basis, represented by orthogonal matrix P such that:

$$\{v_1, \cdots, v_n\} = P\{e_1, \cdots, e_n\}$$

Therefore, if A is the adjacency of graph G we have the following relationship to its graph embeddings:

$$G = PAP^T$$

We see that in the exact orthonormal case, the graph embedding is equal to the adjacency matrix up to an orthogonal change of basis.

2.8 Compressing Adjacency Matrices

Our graph embeddings are generalizations of adjacency matrices in the idealized case of exact orthonormal codes, but in the case of approximately orthonormal codes we can drastically reduce the dimensionality while retaining all of their functionality. Hence, our method may also be viewed as a way to compress adjacency matrices while still retaining much of the functionality toward graph operations.

Usually, the dimension of the adjacency matrix grows quadratically with the size of the vertex set d of a graph. For sparse matrices, where the number of edges k is much lower than number of total possible connections d^2, this is very inefficient from a memory perspective since we are using d^2 parameters to represent $k << d^2$ edges. However, we can view adjacency matrices as special cases of our graph embeddings, and instead the problem becomes representing a superposition of k edges. In the next chapter we show that the dimension of our graph embeddings only needs to scale as k in order to preserve accurate graph operations. Hence, rather than using d^2 parameters to represent a k-sparse graph, we instead use the intuitively correct scaling of k parameters.

In the meantime, we establish that for a graph G, the graph embedding generated by our method is close to its adjacency matrix, up to some orthogonal change of basis.

Theorem 2.8.1. *For a d-dimensional, ϵ-orthonormal code $V = \{v_1, \cdots, v_m\}$ where $m \leq d$, let G be any graph embedding using V with n distinct edges and A be the corresponding adjacency matrix induced from this vertex ordering. Then, there exists an orthogonal change of basis P:*

$$||G - PAP^T||_F = ||P^T GP - A||_F < O(\sqrt{n}m\epsilon)$$

Proof. We use the Gram-Schmidt process to compute an orthogonal set of vectors u_i from the vertex code $\{v_i\}$:

$$u_i = v_i - \sum_{j=1}^{i-1} \frac{\langle v_i, u_j \rangle}{\langle u_j, u_j \rangle} u_j = v_i - \sum_{j=1}^{i-1} \langle v_i, \overline{u}_j \rangle \overline{u}_j$$

where $\overline{u}_j$ is the unit-length version of u_j. By ϵ-orthonormality, we know that $|\langle v_i, v_j \rangle| < \epsilon$ for all i, j. We know that $u_1 = \overline{u}_1 = v_1$. Then, for u_2:

$$1 - \epsilon \leq ||u_2|| = ||v_2 - \langle v_2, \overline{u}_1 \rangle \overline{u}_1|| \leq 1 + \epsilon$$

and

$$||v_2 - u_2|| = ||\langle v_2, \overline{u}_1 \rangle \overline{u}_1|| < \epsilon$$

Hence,

$$||v_2 - \overline{u}_2|| \leq ||v_2 - u_2|| + ||u_2 - \overline{u}_2|| \leq 2\epsilon$$

Similarly, for $||u_3|| = ||v_3 - \langle v_3, \overline{u}_1 \rangle \overline{u}_1 - \langle v_3, \overline{u}_2 \rangle \overline{u}_2||$, let us unravel the term $\langle v_3, \overline{u}_2 \rangle$:

$$|\langle v_3, u_2 \rangle| = |\langle v_3, v_2 \rangle - \langle v_2, \overline{u}_1 \rangle \langle v_3, \overline{u}_1 \rangle| < \epsilon + \epsilon^2$$

This gives:

$$1 - O(2\epsilon) = 1 - \frac{O(2\epsilon)}{1-\epsilon} \leq ||u_3|| \leq 1 + \frac{O(2\epsilon)}{1-\epsilon} = 1 + O(2\epsilon)$$

and so:

$$||v_3 - \overline{u}_3|| \leq O(2\epsilon)$$

Repeating a similar analysis, we see that for m ϵ-orthogonal codes $\{v_1, \cdot, v_m\}$, there exists an orthonormal basis $\{\overline{u}_1, \cdots, \overline{u}_m\}$ such that $||v_i - \overline{u}_i|| < O(m^2\epsilon)$. Hence, let G_V be any graph embedding using the vertex codes $\{v_i\}$ and G_U be the corresponding matrix derived from swapping v_i with $\overline{u}_i$. We then have:

$$||G_V - G_U||_F = ||\sum_{k=1}^{n} v_{i_k} v_{j_k}^T - \overline{u}_{i_k} \overline{u}_{j_k}^T||_F \leq O(\sqrt{n} m^2 \epsilon)$$

As G_U is a sum of the outer products between the orthonormal $\overline{u}_i$, we see that $U^T G_U U$ is an adjacency matrix. Since the Frobenius norm is preserved under orthogonal basis change, we see that G_V is $O(\sqrt{n} m^\epsilon)$ close to an adjacency matrix with respect to the Frobenius norm. □

2.9 Experiments

In this section, we look at how well our methods performs with respect to two basic graph operations - edge query and edge composition - as we increase the number of edges in superposition. This ties into the previous connection of our method as a compression of adjacency matrices. The two graph operations because they are examples of a first and second order linear operations. We shall see that, especially in the case of the first order edge query, our graph embeddings still retain accurate operations even when the number of edges greatly exceeds the graph embedding dimension.

Vertex Query and Edge Composition Accuracy

In all experiments, our vertex codebook had 64 codes, each vertex code a 16-dimensional vector. We used spherical codes, generating our codebook by randomly sampling a unit vector from $\mathbb{R}^{16}$. Finally, we looked at performance over a range of edges in superposition: 8, 16, 24, 32, 48, 64, 80, 128, 240, 400, 540, 800.

For the edge query, we additionally generated two new vertex codes that would serve as the signal edge. For a given edge capacity k, we generated k edges by randomly sampling from the vertex codebook and computing their tensor product. We added them together, as well as the signal edge, to get a final graph embedding. Then, we queried the graph embedding about the presence of the signal edge. For each edge capacity k, we repeated the above procedure 100 times and averaged the query scores. Similarly, we also tested for spurious edge detection by repeating the above steps with one modification: we excluded the

signal edge from the sum so the final graph embedding didn't have it. We then proceeded as above, querying for the now nonexistent signal edge and repeating this 100 times for edge edge capacity k.

For the edge composition, we generated three new vertex codes v_1, v_2, v_3. For each edge capacity k, we first generated a graph embedding with k random edges from the codebook and then added the two edges $v_1 v_2^T$ and $v_2 v_3^T$. We then performed edge composition and queried for the existence of the composite edge $v_1 v_3^T$. We again repeated this for 100 trials and averages the query scores. We also tested for spurious edge composition by omitting the two edges $v_1 v_2^T$ and $v_2 v_3^T$ and proceeding as above. The results are shown in Figure 3.1.

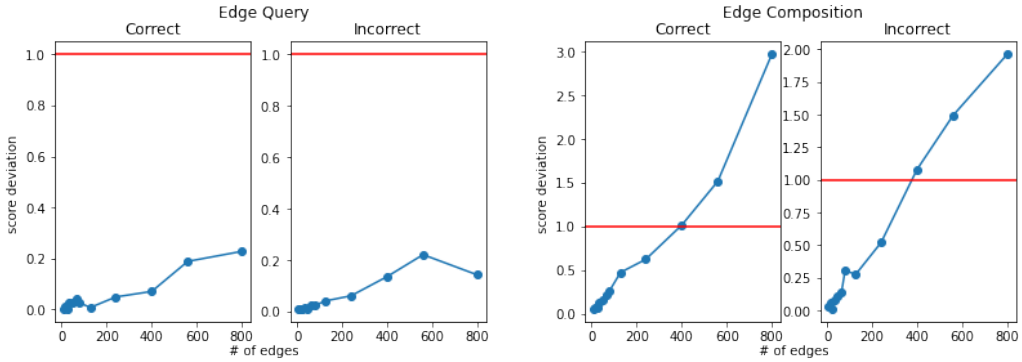


Figure 2.1: Values are the absolute deviations of the average query scores from the ideal score in each experiment. The number of edges is increased while holding the graph embedding dimension constant.

Effect of Varying Codebook Size

Earlier, we claimed that the dimension of our graph embeddings scales with the number of edges in superposition, independent of the number of vertices. We confirm this in simulation by repeating the above edge query experiment, except this time we fix the number of edges to 64 and instead vary the number of vertices from 64 to 2048.

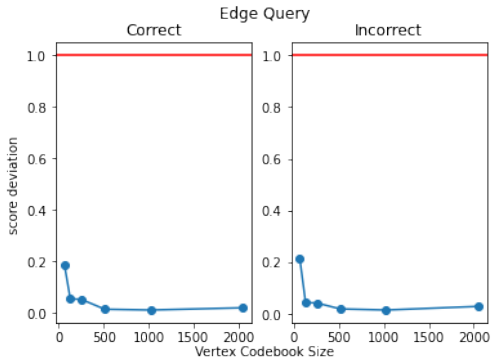


Figure 2.2: Effect of varying codebook size. We looked at edge query performance while varying the number of possible vertices. The results show that performance does not degrade with increasing codebook size.

The results in Figure 3.2 support our claim that the size of graph embeddings scales with the number of edges rather than the number of vertices, and in the next chapter we prove this result.

Chapter 3

Memory and Capacity of Graph Embedding Methods

3.1 Introduction

In this chapter, we analyze the graph embedding method proposed in Chapter 2. In particular, we focus on its memory - the number of parameters used- and its capacity - the size of the graphs it can accurately store in superposition, and we compare it to a comparable graph embedding methods using popular-binding combinations [14] [25] [24]. Previous works have also analyzed the capacity of general vector symbolic architectures [27] [13], and in this work we specifically focus on the contexts of graphs.

3.2 Alternative Binding Operations

Our graph embedding method uses the tensor product to bind the vertex codes together into an edge. However, this means that the graph embedding space scales quadratically with the vertex code dimension, and in response various memory efficient alternatives have been proposed. In this and the subsequent section, we shall primarily analyze three prominent alternative binding operations: the Hadamard product, convolution, and circular correlation.

Hadamard Product

The Hadamard product of two vectors a and b, denoted $a \odot b$, is their element-wise multiplication:

$$[a \odot b]_k = a_k b_k$$

It preserves the dimension of the vertex code and scales linearly with the code dimension. The most common vertex codes when using the Hadamard product are phasor codes and binary codes: vectors whose entries are the complex and real units respectively. These codes

have nice unbinding operations with respect to the Hadamard product, where multiplication by the conjugate of a vertex code will remove that code from the binding:

$$\overline{a} \odot (a \odot b) = b$$

In the next section, we also briefly touch on other codes. However, we shall see that since unbinding the Hadamard product requires element-wise division, many choices of random codes are not suitable for accurate graph operations.

Convolution

The convolution of two vectors a and b, denoted $a * b$, is defined as:

$$[a * b]_k = \sum_i a_i b_{k-i}$$

If $\mathcal{F}$ denotes the Fourier transform, then the convolution can also be expressed as:

$$a * b = \mathcal{F}^{-1}(\mathcal{F}(a) \odot \mathcal{F}(b))$$

Hence, convolution and the Hadamard product are equivalent up to a Hermitian change of basis, and so it is sufficient to analyze just the Hadamard product. Indeed, common codes used with convolution are those whose Fourier transforms are phasor and binary codes, and there is a bijection between the codes used for convolution and codes used for the Hadamard product.

Circular Correlation

The circular correlation of two vectors a and b, denoted $a \star b$, is defined as:

$$[a \star b]_k = \sum_i a_i b_{k+i}$$

Using the Fourier transform, the circular correlation also can be expressed as:

$$a \star b = \mathcal{F}^{-1}(\overline{\mathcal{F}(a)} \odot \mathcal{F}(b))$$

Note that if a is a real vector, then $\overline{\mathcal{F}(a)}$ is the Fourier transform of the flipped vector: let y_k be the k^{th} Fourier coefficient of a:

$$y_k = \sum_{s=0}^{n-1} \exp\left(\frac{2\pi i(-ks)}{n}\right) a_s$$

Taking the conjugate, we get:

$$\overline{y_k} = \sum_{s=0}^{n-1} \exp\left(\frac{2\pi i(ks)}{n}\right) a_s = \sum_{s'=0}^{n-1} \exp\left(\frac{2\pi i(-ks')}{n}\right) a_{-s'}$$

Hence, $\mathcal{F}^{-1}(\overline{\mathcal{F}a}) = Pa$, where P is the permutation that flips indices, and so the circular correlation is convolution augmented with flipping the first argument. This is a special case of using permutation to induce ordered bindings, which we shall cover in the next subsection. Since the circular correlation is a permuted convolution and convolution is equivalent to the Hadamard product, it is sufficient to study the Hadamard product with permutations.

Alternative Bindings as Compressions of the Tensor Product

As noted in our derivation of the tensor product from superposition, the tensor product is a universal construction. That is, every binding method that respects superposition - bilinearity - is linearly induced from the tensor product. The three alternative binding operations - the Hadamard product, convolution, circular correlation- are no exception.

More specifically, fixing the standard basis the tensor product $v \otimes w$ is the outer product vw^T, a $d \times d$ matrix. The Hadamard product is the main diagonal of this matrix. Similarly, the entries of the circular correlation are sums along pairs of diagonals of vw^T, spaced by an interval of $2(d-1)$. For convolution, we analogously take sums along pairs of the reverse diagonals (bottom left to top right).

Hence, all three operations can be seen as compressions of the tensor product, where we compress a $d \times d$ matrix into a d-dimensional vector. While on the surface this might seem to save much in terms of memory, in the next subsection we shall see these compressions are unable to perform some fundamental graph operations. Moreover, in the subsequent section we shall see that these binding alternatives do not actually save on memory, because they suffer from proportional decrease in their representational capacity.

Hadamard Product: Graph Functionality

As established in the preceding subsections, all three operations are, up to a Hermitian change of basis, equivalent to the Hadamard product with possibly a permutation applied to one of its arguments. Hence, in this section we shall analyze the Hadamard product and its suitability for graph embeddings.

Firstly, the Hadamard product can perform edge composition when used in conjunction with binary codes. This is because it has easy unbinding operations, where multiplication by a code will remove that code from the binding. Using the binary/Hadamard scheme, we can perform edge composition by taking the Hadamard product of two edges:

$$(a \odot b) \odot (b \odot c) = a \odot c$$

Moreover, note that in the case of a mismatch between the vertices, the resulting edge is:

$$(a \odot b) \odot (b' \odot c) = a \odot c \odot n$$

where n is a noisy binary code. Importantly, there is no destruction of mismatched edges during edge composition. This will impact its representation capacity, which we shall cover in the next section. Indeed, one can ask if there is any alternative edge composition function that can improve on using the Hadamard product. Assuming this edge composition function respects superposition, one can show that the natural edge composition function under the Hadamard product is again the Hadamard product. We give a sketch of the argument: firstly, the desired composition function respects superposition, so it is a multilinear function and is completely determined by its action on some basis $\{b_i\}$. Fixing the standard basis, we impose the natural constraint:

$$(e_i \odot e_j) \times (e_k \odot e_l) \mapsto \begin{cases} e_i \odot e_l & j = k \\ 0 & j \neq k \end{cases}$$

Since these constraints are satisfied by the Hadamard product, our natural edge composition function must be the Hadamard product, and the defect of mismatched edges not interfering destructively during edge composition remains.

Furthermore, the Hadamard product is unable to represent directed edges because it is symmetric: $a \odot b = b \odot a$. A common fix is to permute one of its arguments before binding, so now we bind as $Pa \odot b$ where P in some permutation matrix. Then our augmented binding scheme becomes:

$$(a, b) \mapsto Pa \odot b$$

This augmented scheme is certainly able to represent directed edges, but it is now unable to perform edge composition. Indeed, disregarding oracle operations where one already knows the bound vertices, using the Hadamard product to perform edge composition results in:

$$(Pa \odot b) \odot (Pb \odot c) = a \odot c \odot (b \odot Pb)$$

The noise term $(b \odot Pb)$ will not cancel unless P is the identity, the non-permuted case. A similar question arises whether there is a better edge composition funciton for the permutated Hadamard: assuming this edge composition function respects superposition, by multilinearity one can show no such operation exists. Intuitively, one would like to send the pair (Pa, a) to the vector of all ones, but the symmetry in the Hadamard product and its compression means such a map is not possible without breaking multilinearity.

In summary, we see that in the binary/Hadamard scheme one sacrifices some core graph functionality: with the regular Hadamard product, one can perform edge composition but cannot represent directed edges; in the permuted case, one can represent directed edges but cannot perform edge composition. Indeed, one can show that other core graph operations, like subsetting and graph homomorphisms, are also impossible under such schemes. Thus, in the binary/Hadamard case we see that compression hurts the representational power of the embedding method.

Hadamard Product: Phasor Codes

Phasor codes, which generalize binary codes, are even less suitable than binary codes. The natural unbinding operation for phasor codes is to multiply by the conjugate codes. However, this already makes it unsuitable for edge composition in both the non-permuted and permuted case. In the non-permuted case:

$$\overline{(a \odot b)} \odot (b \odot c) = \bar{a} \odot c \neq a \odot c$$

The permuted case has a similar deficiency. Again due to the compression of the Hadamard product, there is no multilinear function that conjugates just one argument of the bound edge $a \odot b$. Intuitively, this is due to the symmetry of the Hadamard product, since it unable to distinguish which particular vertex to conjugate. Edge composition using phasor codes is impossible, and any code which can be derived from the phasor code suffers a similar defect.

Hadamard Product: Continuous Codes

Similarly, since the unbinding the Hadamard product requires element-wise division, many choices of random continuous codes are numerically unstable. In fact, the next section we shall see that specific cases of continuous codes all suffer from having infinite moments, making accurate graph operations impossible since the noise terms will overwhelm the signal. Morever, operations like edge composition are also impossible for similar reasons as the phasor code, where one need to apply the unbinding operation to a specific vertex of the bound edge. Due the Hadamard product's compression, such a multilinear map does not exist.

3.3 Random Codes

First, we state some facts about the different coding schemes under consideration. We shall assume all codes have common dimension d.

Spherical Codes

We generate spherical codes by sampling iid from the d-dimensional unit hypersphere $\mathcal{S}^{d-1}$. They have the following properties:

Theorem 3.3.1 (Spherical Code Properties)**.** *Let X denote the dot product between two spherical codes. Then, the following statements hold:*

1. $\frac{X+1}{2} \sim Beta(\frac{d-1}{2}, \frac{d-1}{2})$
2. $E(X) = 0$ *and* $Var(X) = \frac{1}{d}$
3. $|X| \propto \frac{1}{\sqrt{d}}$ *with high probability.*

Proof. The first two claims follow from the results of [**Qiu˙Recipe**]. The final claim follows from either a standard Bernstein bound or a Gaussian approximation. □

Rademacher Codes

Rademacher codes are vectors v where each entry is an iid Rademacher random variable: $v_i = \pm 1$ with probability $\frac{1}{2}$. They have the following properties.

Theorem 3.3.2 (Rademacher Code Properties)**.** *Let X denote the dot product of two Rademacher vectors. The following statements hold:*

1. $\frac{X+d}{2} \sim Binom(d, \frac{1}{2})$
2. $EX = 0$ *and* $Var(X) = d$
3. $\frac{1}{d}X$ *is approximately* $N(0, \frac{1}{d})$.
4. $X \propto \sqrt{d}$ *with high probability.*

Proof. The first two claims follow from the fact that a sum of n iid Rademacher variables is Binomial$(n, \frac{1}{2})$.. The third claim follows either from the Central Limit Theorem or from the Gaussian approximation to the binomial. The fourth claim follows from this same Gaussian approximation or a standard Chernoff inequality. □

Note that all Rademacher codes have norm d. Thus, we may scale them by $\frac{1}{\sqrt{d}}$ to normalize them, and in this sense they are special cases of spherical codes. In fact, by a similar argument as the spherical codes case, one can show that for a fixed error threshold of violating ϵ-orthogonality, normalized Rademacher codes achieve the Johnson-Lindenstrauss upper bound. However, one can have at most 2^d unique codes, while any finite number of spherical codes have probability zero of having a repeat. Rademacher codes still have a hard packing limit, but the trade-off is cleaner unbinding with respect to the Hadamard product.

Other Continuous Codes

Here, we will briefly describe three common continuous codes: Gaussian, Cauchy, and uniform codes. Gaussian codes are generated by having the component of each vector be drawn iid from some Gaussian - for now, let us assume the standard Gaussian. Cauchy and uniform codes are analogously generated - for now, let us assume the standard Cauchy and uniformly on the unit interval $[0, 1]$. The main problem with these codes is that the Hadamard unbinding operation requires element-wise division: the resulting ratios random variables will have infinite moments. This makes them unsuitable for accurate graph operations, since they will result in ill-controlled noise terms.

Theorem 3.3.3. *For t, u iid Gaussian, Cauchy, or uniform. Let $Y = \frac{t}{u}$. Then, for any of the three distributions, all moments of Y are undefined.*

Proof. In the Gaussian case, the ratio of two independent standard Gaussians is a Cauchy random variable, which is known to have infinite moments. In the Cauchy case, the ratio of two independent standard Cauchy rv's has the density:

$$f_{Y_c}(y) \propto \frac{1}{(y^2-1)} ln(y^2)$$

Then, comparing integrals:

$$\infty = \frac{1}{2}\int_0^c |ln(y^2)| \leq \int_0^c y f_{Y_c}(y) \leq \int_0^\infty y f_{Y_c}(y)$$

we see that the first moment is also undefined (for some sufficiently small constant c). In the uniform case, the ratio of two independent $U[0,1]$ rv's is:

$$f_{Y_u}(y) = \begin{cases} \frac{1}{2} & 0 < y < 1 \\ \frac{1}{2z^2} & y \geq 1 \\ 0 & y \leq 0 \end{cases}$$

A similar comparison test also shows that the first moment is undefined. Thus, the first moment, and hence all moments, are undefined for all three choices of distribution. □

3.4 Binding Comparison Overview

In a previous section, we showed that two other alternative binding methods - convolution and circular correlation - are special cases of the Hadamard product; similarly, we found that, of the codes considered, the binary code was the one that had the most representational power with respect to graph operations. Hence, we shall primarily analyze the memory and capacity of our graph embedding method relative to the Hadamard/Rademacher scheme. which uses random binary codes. We also briefly consider other continuous coding schemes paired with the Hadamard product, but we shall see that they are too noisy for accurate graph operations.

We shall look at the memory vs. capacity tradeoff of different edge binding methods with respect to certain graph functions. Firstly, by the superposition principle we assume that the considered graph functions are multilinear functions. Secondly, we focus on two types of graph operations: first and second order operations. First order operations f_1 are any operations that involve graph embeddings once: $f_1(*, G)$ where $*$ represents non-graph arguments. Second order operations f_2 involve graph embeddings twice: $f_2(*, G, G')$. We analyze one representative graph operation from each type: vertex queries as a first-order operation and edge composition as a second-order operation. For both, we analyze the magnitude of the error term as well as the probability of error.

3.5 Vertex Queries

We denote the edge binding operation as ψ, which can be either the Hadamard product or the tensor product. Let us work with the following fixed graph:

$$G = \psi(v, u) + \sum_{i=1}^{k} \psi(q_i, r_i)$$

where all the q, r's are distinct from u, v. We will perform an edge query that seeks to find the vertices in G that vertex v points to (or is connected to in the undirected case).

Hadamard Product and Rademacher Codes

We first look at the Rademacher-Hadamard scheme, analyzing both the magnitude of the error term as well as bouding the probability of retrieving the correct vertex.

Error Norms

In this case, our vertex query is of the form:

$$Q(v, G) = v \odot (v \odot u + \sum_{i=1}^{k} q_i \odot r_i) = u + \sum_{i=1}^{k} q_i \odot r_i \odot v$$

Since the product of Rademcher's is still Rademacher, we see that each term $q_i \odot r_i \odot u = s_i$ is still a Rademacher random vector. We can express the output as:

$$Q(v, G) = u + \sum_{i=1}^{k} s_i$$

Thus, the result of our query can be split into the correct signal u and a noise term ϵ, which is a sum of k independent Rademachers. We then have the following result on their expected magnitudes.

Theorem 3.5.1 (Hadamard/Rademacher Signal-to-Noise)**.** *When performing a vertex query with a single correct vertex u, under the Hadamard product and Rademacher codes of dimension d we have:*

1. *The squared norm of the signal $E||u||^2$ is d.*
2. *The squared norm of the noise $E||\sum_{i=1}^{k} s_i||^2$ is kd*
3. *The signal-to-noise ratio is $\frac{1}{k}$*

Proof. Let us consider the norm of the noise term $\sum_{i=1}^{k} r_i$ relative to the signal u. Then, using the results from Section 3.3 and independence, we see that:

$$E||u||^2 = d \qquad ; \qquad E||\sum_{i=1}^{k} r_i||^2 = \sum_{i} E||r_i||^2 + \sum_{j \neq k} E\langle r_j, r_k \rangle = kd$$

Hence, their ratio is $\frac{1}{k}$ □

Note that we assumed that all the edges were generated by independently sampling Rademacher codes, and hence precludes the possibility of a vertex participating in more than one edge. However, note that we can write the graph embedding into a sum of subgraphs such that each subgraph has edges whose vertices are all distinct:

$$G = \sum_{i}^{l} G_i$$

Note that the number of subgraphs is upper bounded by the maximum node connectivity of the graph G. That is, if every vertex in G is connected to at most L other vertices, then it is possible to express G as a sum of at most $2L$ subgraphs. Hence, we have the following corollary.

Corollary 3.5.1.1. *If a graph G has maximum connectivity L, then:*

1. *The squared norm of the signal $E||u||^2$ is d.*
2. *The squared norm of the noise $E||\sum_{i=1}^{k} s_i||^2$ is $4L^2kd$*
3. *The signal-to-noise ratio is $\frac{1}{4L^2kd}$*

Note that this is a very loose result, since we count each of the k edges L times.

Statistical Error

Now, at this point our query function returns a superposition of the answer u with a noise term $\sum_{i=1}^{k} s_i$. Hence, we would like to perform a look-up operation to recover u by seeing which of the vertex embeddings the output $u + \sum_{i=1}^{k} s_i$ is most similar to. In this case, we will use the dot product to measure similarity. Due to the noise term, we might be concerned with the possibilities of recovering the wrong vertex. In particular, Theorem 3.5 suggests that as the number of edges increases, the probability of recovering an incorrect edge increases.

First, consider the dot product of the true answer u with the output:

$$T = \langle u, u + \sum_{i=1}^{k} s_i \rangle = d + \sum_{i=1}^{k} \langle u, s_i \rangle = d + \epsilon$$

Thus, the dot product of the true answer T will have $ET = d$ and $Var(T) = kd$. The noise term ϵ is a sum of kd Rademacher random variables, and so $\frac{\epsilon+kd}{2} \sim Binom(kd, \frac{1}{2})$. The variance of ϵ is kd, and using the normal approximation to the binomial it is of order $\sqrt{kd}$ with high probability. Similarly, consider the dot product of a false vertex v (that does not equal s_i or u) with the output:

$$F = \langle v, u + \sum_{i=1}^{k} s_i \rangle = \langle v, u \rangle + \sum_{i=1}^{k} \langle u, s_i \rangle$$

Here, F is the sum of $(k+1)d$ Rademacher random variables, with mean 0 and variance $(k+1)d$; moreover, F is of order $\sqrt{(k+1)d}$ with high probability.

Now, we first approximate the probability that F exceeds $d = ET$. We will use the CLT/Gaussian approximation to Binomial to approximate the sum of n iid Rademachers X as $N(0, n)$. Then, we can use these Gaussian bounds:

$$\frac{C}{t} e^{-\frac{t^2}{2}} \leq P(X > t\sqrt{n}) \leq C e^{-\frac{t^2}{2}}$$

where C is some constant. Hence, applying this to F:

$$C\sqrt{\frac{(k+1)}{d}} e^{-\frac{d}{2(k+1)}} \leq P(F > d) = P(F > \sqrt{\frac{d}{(k+1)}}\sqrt{(k+1)d}) \leq e^{-\frac{d}{2(k+1)}}$$

Thus, this suggests the limit of edges we can store in superposition and still have accurate recovery is $O(d)$. A similar computation shows the probability of T being less than 0 scales in a similar manner:

$$C\sqrt{\frac{k}{d}} e^{-\frac{d}{2k}} \leq P(T < 0) \leq e^{-\frac{d}{2k}}$$

In fact, we can be a bit more precise and compute a lower bound on the probability of correct recovery.

Theorem 3.5.2. *Under the above setup, let A be the correct recovery event given M erroneous choices: the event where the correct vertex u is most similar to the output of the vertex query relative to M other wrong candidate vertices. Then, for some constant C we have:*

$$P(A) \geq 1 - M e^{-\frac{d}{2(2k+1)}}$$

Proof. Now, let us first compute the lower bound. Note that correct recovery is precisely the event where the similarity of the correct vertex $T = \langle u, u \rangle$ is larger than the similarities $F_1 \cdots, F_M$ of the M erroneous vertices, where $F_i = \langle v_i, u \rangle$ for the erroneous vertex v_i. Then,

$$P(T > \max(F_1, \cdots, F_M)) = P(\cap\{T > F_i\}) = 1 - P(\cup\{T \leq F_i\})$$

By construction, the F_i's are iid. Letting ϵ denote the error term:

$$\begin{aligned} P(\cup\{T \leq F_i\}) &\leq \sum_{i=1}^{M} P(T \leq F_i) \\ &= MP(T \leq F_1) \\ &= MP(d + \epsilon \leq F_1) \\ &= MP(F_1 - \epsilon \geq d) \\ &\leq Me^{-\frac{d}{2(2k+1)}} \end{aligned}$$

Hence

$$P(T > \max(F_1, \cdots, F_M)) \geq 1 - Me^{-\frac{d}{2(2k+1)}}$$

We used the fact that a difference of Rademacher sums is still a Rademacher sum, so $F_1 - \epsilon$ is a sum of $(k+1)d + kd = (2k+1)d$ Rademachers. □

This theorem confirms the informal analysis of this section: the number of edges k that be stored in superposition cannot be more than $O(d)$ without seriously compromising the accuracy of the vertex query.

Hadamard Product and Continuous Codes

Now, let us suppose the we were working with any continuous code (Gaussian, Cauchy, Uniform). Our vertex query would now be unbinding the graph by the reciprocal of the query vertex u:

$$Q(u, G) = u^{-1} \odot (u \odot v + \sum_{i=1}^{k} q_i \odot s_i) = v + \sum_{i=1}^{k} u^{-1} \odot q_i \odot s_i$$

In the noise term, note that we now have a sum of vector whose entries are ratios: $\frac{q_i}{u^{-1}} s_i$. In section 3.3, we saw that the entries will have undefined moments: they follow heavy-tailed distribution. Thus, it is very likely that the noise overwhelms the true answer v regardless of how many edges k are in superposition. This makes such continuous codes infeasible for vertex queries.

Tensor Product and Spherical Codes

Here, we look at the same error quantities for the tensor-spherical scheme: the error norms and the probability of retrieving the correct vertex.

Error Norms

Now, our vertex query is of the form:

$$Q(v, G) = v^T(vu^T + \sum_{i=1}^{k} q_i r_i^T) = u^T + \sum_{i=1}^{k} \langle v, q_i \rangle r_i^T = u^T + \sum_{i=1}^{k} s_i^T$$

We have a corresponding result on the average squared norms and the signal-to-noise ratio.

Theorem 3.5.3 (Tensor/Spherical Signal-to-Noise)**.** *When performing a vertex query with a single correct vertex u, under the tensor product and spherical codes of dimension d we have:*

1. *The squared norm of the signal $E||u||^2$ is 1.*
2. *The squared norm of the noise $E||\sum_{i=1}^{k} s_i||^2$ is $\frac{k}{d}$*
3. *The signal-to-noise ratio is $\frac{d}{k}$*

Proof. The first claim holds since spherical codes have norm 1. The squared norm of the nuisance term $\sum_{i=1}^{k} \langle v, q_i \rangle s_i^T$ is:

$$E||\sum_{i=1}^{k} \langle v, q_i \rangle s_i^T||^2 = \sum_i E(\langle v, q_i \rangle)^2 + 2 \sum_{j \neq k} E\langle v, r_j \rangle \langle v, r_k \rangle \langle r_j, r_k \rangle = \frac{k}{d}$$

Hence, the ratio of the answer-noise average norms is $\frac{d}{k}$

□

Statistical Error

Now, we again want to recover the answer u by finding which vertex embedding the query output is most similar to, and we will again use the dot product to measure similarity. First, the dot product of the true answer u with the query output:

$$T = u^T(u + \sum_{i=1}^{k} \langle v, q_i \rangle r_i) = 1 + \sum_{i=1}^{k} \langle v, q_i \rangle \langle u, r_i \rangle = 1 + \epsilon$$

The noise term ϵ is a sum of k terms of the form $e_i = \langle v, q_i \rangle \langle u, r_i \rangle$, and using independence and the Cauchy-Schwarz inequality:

$$Ee_i = 0 \quad ; \quad Ee_i^2 = \frac{1}{d^2} \quad ; \quad E|e_i| \leq \frac{1}{d}$$

Hence, the variance of ϵ is $\frac{k}{d^2}$, and so for accurate retrieval we see that $k \leq O(d^2)$ or else ϵ will be of the same magnitude as the signal 1. Similarly, the dot product of a false vertex t (not matching any of the v_i's) is:

$$F = \langle t, u \rangle + \sum_{i=1}^{k} \langle v, q_i \rangle \langle t, r_i \rangle = \langle t, u \rangle + \epsilon$$

Hence, let us first calculate the probabilty that F exceeds the $ET = 1$. However, it is a sum of random variables with a different distribution, so we will make one further simplification. As in section 3.3, the term $\langle t, u \rangle$ will be of the order $\frac{1}{\sqrt{d}}$ with high probability. Hence, we assume conservatively that $|\langle t, u \rangle| = O(\frac{1}{\sqrt{d}})$. Thus, for large d we can make the following simplification:

$$P(F > 1) \lessapprox P(\epsilon > 1 - O(\frac{1}{\sqrt{d}})) \approx P(\epsilon > 1)$$

Hence, as $\epsilon = \sum^k e_i$ where e_i's are independent with $Ee_i^2 = \frac{1}{d^2}$, then Bernstein's inequality gives:

$$P(F > 1) \approx P(\epsilon > 1) \leq e^{-1/[2(\frac{k}{d^2} + \frac{1}{3})]} \approx e^{-\frac{d^2}{2k}}$$

A similar computation. using the fact that ϵ is symmetrically distributed, gives an upper bound of $e^{-\frac{d^2}{k}}$ for $P(T < 0)$. Hence, both suggest that the limit of edges we can store in superposition an still have accurate recovery is $O(d^2)$.

As in the Hadamard/Rademacher case, we have corresponding bound on the probability of accurate recovery for the tensor/spherical scheme.

Theorem 3.5.4. *Under the Hadamard/Rademacher scheme, let A be the correct recovery event given M erroneous choices: the event where the correct vertex u is most similar to the output of the vertex query relative to M other wrong candidate vertices. Then, for some constant C we have:*

$$P(A) \gtrapprox 1 - Me^{-\frac{d^2}{k}}$$

Proof. First, we compute the lower bound.. We have:

$$P(T > \max(F_1, \cdots, F_M)) = 1 - P(\cup\{T \leq F_i\})$$

We can make the same simplifying conservative assumption of $|\langle t, u \rangle| = O(\frac{1}{\sqrt{d}})$ as above to

get:

$$\begin{aligned} P(\cup T \leq F_i) &\leq \sum P(T \leq F_i) \\ &= MP(T \leq F_1) \\ &= MP(1 + \epsilon_1 \leq \langle t, u \rangle + \epsilon_2) \\ &\lessapprox MP(\epsilon_2 - \epsilon_1 \geq 1 + O(\frac{1}{\sqrt{d}})) \\ &\leq Me^{-\frac{d^2}{k}} \end{aligned}$$

Thus,

$$P(T > \max(F_1, \cdots, F_M)) \gtrapprox 1 - Me^{-\frac{d^2}{k}}$$

□

Thus, this also confirms that when the vertex code dimension is d, we cannot store more than d^2 edges using the tensor/spherical scheme without compromising the accuracy of the vertex query.

Memory and Capacity

As a reminder, the vertex code has dimension d. Then, using the Hadamard product with Rademacher codes, the graph embedding space is also dimension d; the previous analysis suggests that we can store at most $k = O(d)$ edges in superposition without seriously affecting the accurate retrieval of the answer. On the other hand, using the tensor product with spherical codes, the graph embedding space is d^2, and the previous analysis shows that we can store at most $k = O(d^2)$ edge without affecting accuracy. Hence, in both cases the number of edges we can store in superposition vs. the dimension of the graph embedding space have the same ratio.

3.6 Edge Composition

Now, let us work with the following fixed graph:

$$G = \psi(u, v) + \psi(v, w) + \sum_{i=1}^{k-1} \psi(q_i, r_i)$$

where all the q, r's are distinct from u, v, w and ψ denotes the binding operation. We will look at edge composition, checking specifically for the correct composition of the two composable edges:

$$(u, v) \circ (v, w) \mapsto (u, w)$$

To check for the presence of the correct edge, we will perform an edge query and analyze both the error norms and probability of successfully retrieving the correct edge.

Hadamard Product and Rademacher Codes

In this section, we analyze edge composition in the Hadamard/Rademacher scheme.

Error Norms

We want to do edge composition with G, which in this case represents just the binding of G with itself:

$$G \odot G = (u \odot v + v \odot w + \sum_{i=1}^{k-1} q_i \odot r_i) \odot (u \odot v + v \odot w + \sum_{i=1}^{k-1} q_i \odot r_i)$$

After distributing, we will get $(k+1)^2$ total terms:

$$G \odot G = u \odot w + \sum_{i=1}^{(k+1)^2-1} e_i = u \odot w + R$$

We assumed that all every vertex was distinct, so each e_i is a Hadamard product of either two or three vertices. Since a product of Rademachers is still Rademacher, each e_i is a Rademacher vector and ϵ is a sum of $(k+1)^2 - 1 = k^2 + 2k$ independent Rademacher vectors.

Theorem 3.6.1 (Hadamard/Rademacher Signal-to-Noise)**.** *When performing a edge query with a single correct edge (u, v), under the Hadamard product and Rademacher codes of dimension d we have:*

1. *The squared norm of the signal $E||u \odot w||^2$ is d.*
2. *The squared norm of the noise $E||R||^2$ is $(k^2 + 2k)d$*
3. *The signal-to-noise ratio is $\frac{1}{k^2}$*

Proof. By construction there is only one correct composable edge in G - $u \odot w$ - and all other terms are noise. Using the results from section 3.3, we can characterize the signal-to-noise ratio.

$$E||u \odot w||^2 = d \qquad E||R|| = \sum_i E||e_i||^2 + \sum_{j \neq k} E\langle e_j, e_k \rangle = (k^2 + 2k)d$$

□

Statistical Error

After performing edge composition, we have a superposition of the single composed edge in G - $u \odot w$ - along with noise ϵ. Now, say we want to recover exactly which edges were composable in G. To this end, we can do two things: we can either unbind by one vertex

and compute a dot product with the other, or we can unbind by the given edge and then sum all the entries together. Both approaches give the same result, so we shall focus on the latter for simplicity.

Hence, we shall detect the (non)existence of a candidate edge $s \odot t$ by first unbdinding and then summing the entries:

$$G \odot G \mapsto (s \odot t) \odot (G \odot G) \mapsto sum[(s \odot t) \odot (G \odot G)]$$

Now, let us first consider checking the true edge $u \odot w$:

$$(u \odot w) \odot (u \odot w + E) = (1 + E')$$

Unbinding by the true edge will generate a vector of 1's and, as a product of Rademachers is still Rademacher, a new error term ϵ' that, like ϵ, is a sum of $k^2 + k$ independent Rademacher vectors. Now, we then sum up the entries (or equivalently compute the dot product with the vectors of 1's) and we get:

$$T = sum(1 + E') = d + \sum^{d(k^2-2k)} r_i = d + \epsilon$$

where each r_i is a Rademacher random variable. Therefore, we use the same arguments as the vertex query section to get $E(T) = d$ and $Var(T) = d(k^2 - 2k)$. The noise term ϵ has variance $d(k^2-2k)$, and so it is of order $\sqrt{d(k^2 - 2k)} \approx k\sqrt{d}$. This suggests that for accurate retrieval, the number of edge in superposition $k \leq O(\sqrt{d})$.

Similarly, we now do the same procedure for a false random edge $s \odot t$, and since it does not match then we will get sum of $d(k^2 - 2k + 1)$ Rademachers. Thus, the output F of any false edge is:

$$F = \sum^{d(k+1)^2} r_i$$

We conclude that $EF = 0$ and $Var(EF) = d(k + 1)^2$.

Repeating the same analysis as in the vertex query section, we get the following bounds on the probability that F exceeds $d = ET$:

$$C\sqrt{\frac{(k+1)^2}{d}} e^{-\frac{d}{2(k+1)^2}} \leq P(F > d) \leq e^{-\frac{d}{2(k+1)^2}}$$

Similarly,

$$C\sqrt{\frac{(k^2 - 2k)}{d}} e^{-\frac{d}{2k}} \leq P(T < 0) \leq e^{-\frac{d}{2(k^2-2k)}}$$

Finally, for M false edges we have the following result using the same techniques as in the vertex query case:

Theorem 3.6.2. *Under the Hadamard/Rademacher scheme, let A be the correct recovery event given M erroneous choices: the event where the correct edge $u \odot v$ is most similar to the output of the edge query relative to M other wrong candidate edges. Then, for some constant C we have:*

$$P(A) \geq 1 - Me^{-\frac{d}{4(k+1)^2-2}}$$

Thus, these all suggest that for accurate edge composition, the number of edges in superposition can be at most $\sqrt{d}$.

Tensor Product and Spherical Codes

Error Norm

Using the tensor product, our graph G is:

$$G = uv^T + vw^T + \sum_{i=1}^{k-1} q_i r_i^T$$

and we do edge composition by a matrix multiplication of the graph embedding:

$$G^2 = uw^T + \sum_{i=1}^{(k+1)^2-1} \langle a_i, b_i \rangle d_i c_i^T = uw^T + R$$

where a, b, c, d are all iid uniform from the d-dimensional hypersphere.

Theorem 3.6.3 (Tensor/Spherical Signal-to-Noise)**.** *When performing a edge query with a single correct edge (u, v), under the tensor product and spherical codes of dimension d we have:*

1. *The squared Frobenius norm of the signal $E||uw^T||_F^2$ is 1.*
2. *The squared Frobenius norm of the noise $E||R||_F^2$ is $\frac{k^2-2k}{d}$*
3. *The signal-to-noise ratio is approximately $\frac{d}{k^2}$*

Proof. The signal, as the outer product of two orthonormal vectors, has Frobenius norm 1. Similarly, the nuisance term E has the following expected squared Frobenius norm:

$$||E(R^T R)||_F^2 = tr[E(R^T R)] = \sum_i E(\langle a_i, b_i, \rangle)^2 = \frac{k^2 - 2k}{d}$$

Hence, the ratio of the answer-noise average norms is $\frac{d}{k^2-2k} \approx \frac{d}{k^2}$. □

Statistical Error

Again, we query the edge composition G^2 for the (non)existence of a candidate edge. In the tensor product case, the natural edge query operation for a query edge (s, t) is:

$$s^T G^2 t$$

Hence, let us first examine the result for the only true edge (u, w):

$$T = u^T(G^2)w = u^T(uw^T + R)w = 1 + u^T Rw = 1 + \epsilon$$

Expanding the error term ϵ:

$$\epsilon = u^T Rw = u^T(\sum^{k^2-2k} \langle a_i, b_i\rangle d_i c_i^T)w = \sum^{k^2-2k} \langle a_i, b_i\rangle\langle u, d_i\rangle\langle c_i, w\rangle$$

we see it is a product of $k^2 - 2k$ iid terms, each of which is the product of three independent dot products. Hence, we have $E(\epsilon) = 0$ and $Var(\epsilon) = \frac{k^2-2k}{d^3} \approx \frac{k^2}{d^3}$, and similarly $ET = 1$ and $Var(T) = \frac{k^2-2k}{d^3}$. This suggests that $k \leq O(d^{\frac{3}{2}})$ to have accurate edge composition.

Similarly, let us considering querying by any non-existent edge $s \to t$:

$$F = s^T G^2 t = \langle s, u\rangle\langle w, t\rangle + \sum^{k^2-2k} \langle a_i, b_i\rangle\langle s, d_i\rangle\langle c_i, t\rangle = \langle s, u\rangle\langle w, t\rangle + \epsilon$$

The first term is a product of independent dot products, so it has mean 0 and variance $\frac{1}{d^2}$. The second term has the exact same distribution as the error term in the previous paragraph. We have $EF = 0$ and $Var(F) = \frac{1}{d^2} + \frac{k^2-2k}{d^3}$

Now, we first compute the probability that F exceeds $ET = 1$. As in the edge binding section, using a Bernstein concentraation inequality the first term of F - $\langle s, u\rangle\langle w, t\rangle$ - has magnitude at most $\frac{2}{d}$ with high probability. Hence, we work with the conservative assumption that $|\langle s, u\rangle\langle w, t\rangle| = O(\frac{1}{d})$. Hence, for large d we can make the simplification:

$$P(F > 1) \lessapprox P(\epsilon > 1 - O(\frac{1}{d})) \approx P(\epsilon > 1)$$

Then, using a Bernstein inequality gives:

$$P(F > 1) \lessapprox P(\epsilon > 1) \leq e^{-1/[2(\frac{k^2+2k)}{d^3} + \frac{1}{3})]} \approx e^{-\frac{d^3}{2k^2}}$$

A similar computation for T, using the fact that ϵ is symmetrically distributed, gives:

$$P(T < 0) \lessapprox e^{-\frac{d^3}{2k^2}}$$

These both suggest that we can store at most $O(d^{\frac{3}{2}})$ edges while retaining accurate recovery.

Finally, as in other sections we compute a lower bound on the probability of getting a correct answer when testing both the true edge (t, w) and M false edges.

Theorem 3.6.4. *Under the tensor/spherical scheme, let A be the correct recovery event given M erroneous choices: the event where the correct edge uv^T is most similar to the output of the edge query relative to M other wrong candidate edges. Then, for some constant C we have:*

$$P(A) \geq 1 - Me^{-\frac{d^3}{k^2}}$$

Proof. Again, we make the same conservative assumption of $|\langle s, u\rangle\langle w, t\rangle| = O(\frac{1}{d})$. We have:

$$\begin{aligned} P(\cup\{T \leq F_i\}) &\leq \sum P(T \leq F_i) \\ &= MP(T \leq F_i) \\ &= MP(1 + \epsilon_1 \leq \langle s, u\rangle\langle w, t\rangle + \epsilon_2) \\ &\lessapprox MP(\epsilon_2 - \epsilon_1 \geq 1) \\ &\leq Me^{-\frac{d^3}{k^2}} \end{aligned}$$

Again, this suggests the number of edges k in superposition must have order less than $d^{\frac{3}{2}}$ for accurate recovery.

□

Memory and Capacity

The graph embedding dimension under Hadamard product with Rademacher codes is d, and the above analysis gives a limit of $\sqrt{d}$ edges that can be stored in superposition. On the other hand, the tensor product with spherical codes has dimension d^2 and it can store at most $d^{\frac{3}{2}}$ edges in superposition. In both cases, the edge-dimension ratio is $\sqrt{d}$. Again, the Hadamard product with Rademacher codes offers no concrete memory advantages over the tensor product. Any savings we have in memory are offset by a corresponding reduction in capacity.

3.7 Binding Comparison Summary

General Memory vs. Capacity Ratio

In general, for the Hadamard/Rademacher scheme a n-order operation on a graph with k edges will create k^n nuisance terms. Hence, a similar argument as the above two sections will give a limit of at most $d^{\frac{1}{n}}$ edges that can be stored in superposition. Thus, the general capacity-memory ratio for the Hadamard/Rademacher scheme is:

$$\frac{d^{1/n}}{d} = d^{\frac{1}{n}-1} = d^{-\frac{n-1}{n}}$$

On the other hand, for the tensor/spherical scheme an n-order operation will also create k^n edges, but each will be weighted by a random coefficient with mean 0 and variance $d^{-(n-1)}$.

During edge recovery, we will then have k^n error terms with mean 0 and variance $d^{-(n+1)}$. Thus, our capacity-memory ratio for the tensor/spherical scheme is:

$$\frac{d^{-(n+1)/n}}{d^2} = d^{-\frac{2n-(n+1)}{n}} = d^{-\frac{n-1}{n}}$$

In summary, we see that the Hadamard/Rademacher offers no relative memory advantages, since it suffers from proportional hit to its capacity.

Compression and Expressivity

In the previous two sections, we saw that the Hadamard/Rademacher scheme not only falls short of the tensor/spherical scheme in terms of graph functionality, it also provides no meaningful savings in relative memory efficiency. Indeed, while we analyze just the specific Hadamard/Rademacher case, we can extend it to cover the other two alternative binding operations: these two alternatives are special cases of the Hadamard product, so we may as well analyze the Hadamard product and its possible codes; of the possible codes, the binary and phasor codes do not suffer from numerical and accuracy issues stemming from element-wise division as other codes do; of these two codes, the binary code is the only one capable of edge composition. Hence, from a graph functionality standpoint the Hadamard/Rademacher scheme is the closest to matching the tensor/spherical scheme, and so it is the natural scheme for comparison. However, we see that the Hadamard/Rademacher scheme still falls short in representational power and only matches in relative representational capacity of the tensor/spherical scheme.

A final point might be made about the effect of higher order tensors and their impact on memory, since the dimensionality of an n-order tensor is d^n. While this is general is a defect of the tensor product that the other alternative binding methods do not suffer from, in the specific context of graph embeddings the tensor order is always small; usually, it is at most 3 when working with typed graphs. Hence, concerns about memory stemming from higher order tensors do not apply to graph embeddings. In summary, from a memory standpoint the tensor product performs just as well as other alternative binding methods.

3.8 Simulations

In this section, we perform some simulations that confirm out theoretical results. For the spherical/tensor and Rademacher/Hadamard schemes, we looked at performance in the edge query and edge composition, comparing accuracy in detecting both the presence and absence of a signal edge. In all cases, we looked at accuracy when varying the number of edges in superposition from 8 to 500; at each edge capacity, we generated a new graph and performed the one of the graph operations, repeating this for 200 trials and averaging the results. Each graph was generated by independently generating vertex codes, binding them, and summing them together. Shown below are the results of these experiments for the spherical/tensor

case. The red lines denotes the ideal values: a value of 1 when the target edge was present and 0 when the target edge was absent.

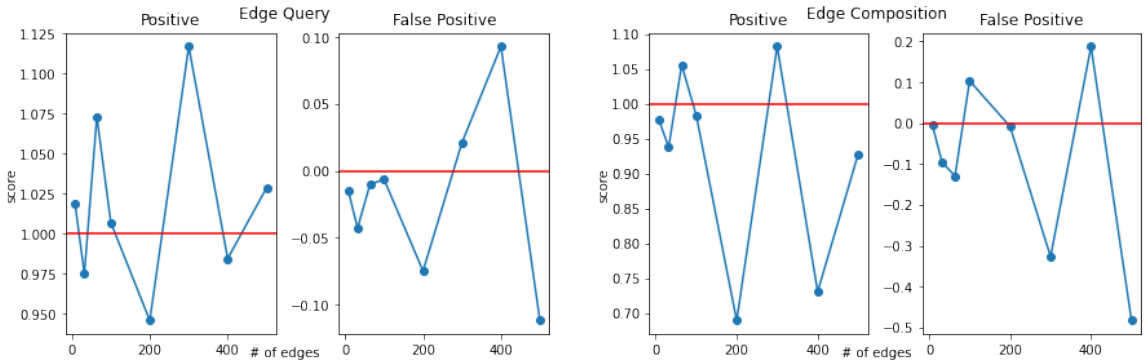


Figure 3.1: Spherical/tensor scheme. Results are shown for the edge query and edge composition operations. For each graph operation, we tested performance in both detecting a test edge and ignoring a spurious edge. The red line indicate the ideal values in each case - 1 for the positive case and 0 for false positive case.

Similarly, shown below are the same experiments for the Rademacher/Hademacher scheme. We see that the values deviate from the correct values much faster than the tensor spherical case.

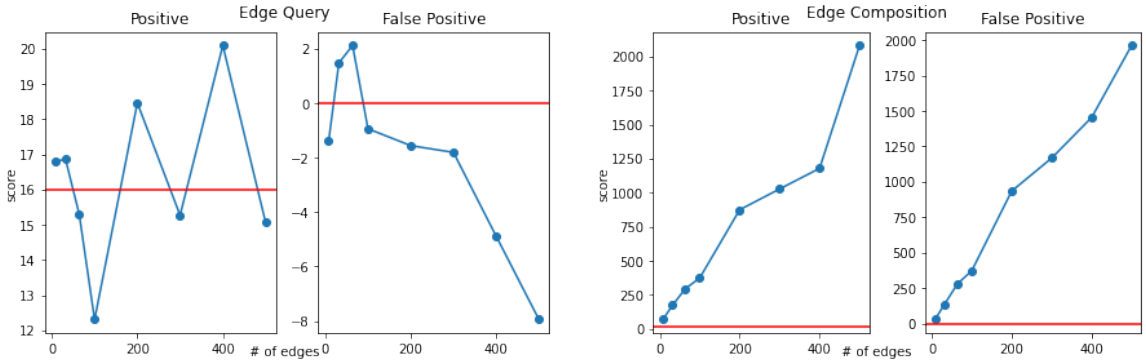


Figure 3.2: Rademacher/Hadmard scheme. We repeated the same tests from the tensor/spherical scheme. In this case, the ideal values for the positive and false positive cases are 16 and 0 respectively, denoted by red lines.

We note that our theoretical analysis assumed that the graph embeddings had distinct edges, with no repeats. However, in these experiments we generated edges by binding together two vertex codes randomly sampled from a fixed codebook. While this codebook was randomly generated, this procedure does not exclude the possibility of vertex codes participating in multiple edges, which explains the monotonic behavior of the edge query/ edge composition tests. We chose this experimental setup because it is how graph embeddings are generated in practice, and the empirical results still corroborate our theoretical results: the capacity of the Rademacher/Hadamard scheme is much lower relative to the spherical/tensor scheme.

Chapter 4

Disentanglement from the Manifold Perspective

Introduction

Most approaches to disentanglement involve learning a generative function $f(\mathcal{X}_1, \cdots, \mathcal{X}_n)$ such that each argument $\mathcal{X}_i$ corresponds to semantically meaningful variations in the data. That is, given a dataset P, one learns a generative function f such that:

$$f(x_1, \cdots, x_n) \approx p \quad ; \quad \forall p \in P$$

The above requirement might be unrealistic for real datasets where the data can vary in many distinct ways. For example, consider a collection of headshots of multiple people, taken under varying lighting conditions and poses. It would difficult to learn a global function $f(x_1, x_2)$ where x_1 controls lighting and x_2 controls pose, since there is so much variability of facial features between subjects. Instead, it might be more realistic to learn a local function f_p that controls lighting and pose for each person p.

The above process of breaking up the generative function f can be further extended: for many or all data points p_i we learn a local generative function $f_{p_i}(x_1, \cdots, x_n)$ such that

$$f_{p_i}(x_1, \cdots, x_n) \approx p'$$

for p' close to p_i. We've gone from the strong requirement of a global generative function f to the weaker requirement of a collection of local functions $f_{p_i}(x_1, \cdots, x_n)$. Intuitively, we can interpret these latent variables $x_1, \cdots, x_n$ as the intrinsic coordinates of the data. For example, going back to our dataset of headshots, variations in lighting, pose, and other features are intrinsic coordinates that describe our data.

Defining Disentanglement

The manifold hypothesis posits that high-dimensional data drawn from the real world lies on or near a low-dimensional manifold. More informally, high-dimensional data has a low-dimensional description. This description is given by the manifold's local charts, which map

patches of the manifold to Euclidean coordinates. Returning to our disentangling generative function $f(x_1, \cdots, x_n)$, let us assume the latents x_i are all real numbers. As previously discussed, we can interpret these disentangled latent variables as coordinates with which to describe our data. Therefore, we formalize this coordinate intuition in the language of manifolds:

Definition 1. *Data lies on a manifold, and* ***disentanglement*** *is the discovery of an atlas of local charts for the data manifold.*

4.1 Smooth Manifolds: Technical Preliminaries

In this section, we give a brief technical overview of smooth manifolds. If the reader is already familiar with the topic, they are encouraged to skip ahead.

Smooth Manifolds

A manifold is a space that locally looks like Euclidean space, much like how a sphere locally looks like a plane. More formally, a n-dimensional manifold M is a second-countable Hausdorff space equipped with a collection of local charts $\phi_i : V_i \to U_i$. Here, the V_i form an open cover of M, and each local chart is a homeomorphism between an open set in the manifold $V_i \subseteq M$ and an open set in Euclidean space $U_i \subseteq \mathbb{R}^n$. Unpacking this definition, our manifold is some space with a collection of small patches that tile it, and each patch is associated with its own chart map that gives a bijective mapping between the manifold patch and a corresponding patch of Euclidean n-space. If we consider each dimension of Euclidean space as a spatial coordinate, the chart map ϕ_i gives the n local coordinates of the patch V_i. Hence, manifolds are locally equivalent to patches of Euclidean space.

A manifold is smooth if the transition maps between its local coordinates are smooth maps with smooth inverse. That is, consider the neighborhood $V_1 \cap V_2$, which is the intersection between the local charts $\phi_1 : V_1 \to U_1$ and $\phi_2 : V_2 \to U_2$. These give two coordinate systems describing $V_1 \cap V_2$, and we require that reparametrizing from coordinates U_1 to coordinates U_2 be a smooth function $\phi_1 \circ \phi_2^{-1} : U_1 \to U_2$ with smooth inverse. Note that our transition map is just a map between open subsets of Euclidean space, and so we can interpret smoothness here in the usual sense for Euclidean spaces. In summary, a smooth manifold is just a manifold where reparametrization of local coordinates is smooth.

Finally, we define smooth maps between smooth manifolds. We say a map $f : M \to N$ is smooth if for each $p \in M$ there exist open patches $U \subseteq M$ and $V \subseteq N$ such that: $p \in U$, $f(p) \in f(U) \subset V$ and the maps between their coordinates is smooth. That is, breaking the map f into its action on local patches, the map f is a smooth map if it is smooth in local coordinates.

Tangent Spaces, the Tangent Bundle, and Vector Fields

Each point p on the smooth manifold M has an associated vector space called the tangent space, denoted T_pM. Intuitively, the tangent space T_pM is the set of directional derivatives along the manifold at p, and we can interpret an element of the tangent space $v \in T_pM$ as the derivative along velocity v. For example, consider the unit sphere. The tangent space at a point p on the sphere is isomorphic as a vector space to $\mathbb{R}^2$, and visually it's represented by the tangent plane to the sphere at p. Every vector in this plane represents a possible direction we could move on that sphere. Note that the vector perpendicular to this plane does not lie in the tangent space - it represents a direction that takes us off the sphere.

Consider a smooth function between smooth manifolds $f : M \to N$. f induces a linear map between the tangent spaces, and we call this induced map the differential df. Representing f in local coordinates, the differential df coincides with the Jacobian of f. Just as the Jacobian maps directional derivatives in one space to directional derivatives in another, the differential gives a linear map between the tangent spaces T_pM and $T_{f(p)}N$.

Now we define vector fields over a manifold M. In the Euclidean case, a vector field assigns a directional derivative to each point, and analogously a vector field on a manifold M assigns a direction of change to each point $p \in M$. In light of the above discussion, this amounts to assigning each point p to a vector in its tangent space $v_p \in T_pM$.

At each point p of the smooth manifold we have its associated tangent space T_pM. If we take the disjoint union of these tangent spaces to create a set, this set has a natural atlas of charts induced from M; this makes it into another manifold called the tangent bundle TM. Returning to vector fields, we define smooth vector fields as smooth functions $X : M \to TM$ such that $X(p) \in T_pM \subseteq TM$; a smooth vector field is a vector field that is a smooth map between the manifolds M and TM.

4.2 Flows over the Data Manifold

In this section, we introduce the technical concept of smooth flows over a smooth manifold. We then interpret this in the context of the data manifold, showing how flows correspond to our natural intuition of varying a latent factor. Throughout this and subsequent sections, we assume our data lies on a smooth manifold M.

Data Flows: Intuition

Given our smooth data manifold M, let us concentrate on a single data point $p_0 \in M$. It is contained in some open set V with local chart $\phi : V \to U \subseteq \mathbb{R}^n$, and we can describe V using the local coordinates U. Let $u_0 = \phi^{-1}(p_0)$ denote the coordinates of p_0. Starting at u_0, suppose we moved in coordinate space along the first dimension via the smooth path $\gamma_1(t) = u_0 + te_1$. This then induces a smooth path on our data manifold by mapping this path onto the manifold through the inverse local chart map: $\tilde{\gamma}_1(t) = \phi^{-1} \circ \gamma_1(t)$. Since we are

manipulating a single dimension in coordinate space, on the data manifold this corresponds to manipulating the data by varying a single latent factor. For example, in a dataset of shapes at different locations and orientations, varying a single coordinate might correspond to smoothly translating each shape in one direction. Now, at any point p within that same local chart, let consider a family of paths like γ_1. In fact, treating the start point as another variable, we get a function of two variables: $\theta_1(t, p) = p + te_1$. This function θ_1 describes a time-dependent evolution of the manifold patch $V \subseteq M$, where after time t we have displaced each point t units along the first coordinate axis.

Curves, Flows, and Vector Fields

To formalize the preceding discussion, we define a smooth flow on a manifold M as a smooth function $\theta : D \times M \to M$, where for every p the slice $D \times p \subseteq \mathbb{R}$ is some open interval containing 0, such that:

$$\theta(t, \theta(s, p)) = \theta(t + s.p) \qquad ; \qquad \theta(0, p) = p$$

Intuitively, we think of the argument D as the time parameter, and a smooth flow describes a path over the manifold starting at point p after time t. We may also restrict the flow to be defined over some local patch $V \subset M$ rather than the entire manifold. Indeed, for a majority of this paper we shall be primarily concerned with flows over a local patch.

Let us return to the example of the first coordinate flow θ_1 in the previous section. There is a connection between θ_1 and the first coordinate vector field X_1:

$$X_1 : U \to TU \quad ; \quad u \mapsto e_1|_u = \frac{\partial}{\partial x_1}|_u$$

Thinking of vector fields as describing dynamics on U, say we begin at some point $u \in U$. We then follow the dynamics of X_1 to traverse U, and in fact this gives rise to the flow function $\theta_1(t, u)$. Conversely, if we were to take the time derivative of θ_1 at each point u, this in fact would return the vector field X_1. The flow and vector field generate each other, and we formally call X_1 the infinitesimal generator of θ_1:

$$\theta_1(t, u) : \mathbb{R} \times U \to U \quad ; \quad \theta_1(t, u) = u + te_1 \quad ; \quad \frac{\partial}{\partial t}\theta_1(t, u) = X_1(u)$$

Indeed, the smooth curve $\gamma_1(t) = u_0 + e_1 t$ previously mentioned is a curve of θ_1, where we follow the flow starting at point u_0:

$$\gamma_1(t) = \theta_1(t, u_0)$$

and we see that taking the derivative at any point of γ coincides with X_1:

$$\frac{d}{dt}\gamma_1|_s = X_1(\gamma_1(s))$$

Formally, γ_1 is called an integral curve of X_1, and we see that in fact it is part of the flow θ_1 generated from X_1. All three constructions - curve, flow, vector field - are related.

Using the inverse local chart ϕ^{-1} to map from coordinate space back to the data manifold, all of the above machinery have analogues on the manifold. There is a corresponding vector field $\tilde{X}_1$:

$$\tilde{X}_1 = d\phi^{-1} \circ X_1 \circ \phi : M \to TM$$

with a corresponding flow on the data manifold $\tilde{\theta_1}(t, p)$. Going back to the face example, the flow $\tilde{\theta_1}(t, p)$ could correspond to varying the lighting condition of various faces, and starting at a face p the traced path would define a curve $\tilde{\gamma}_1(t)$ on our data manifold:

$$\tilde{\gamma}_1(t) = \tilde{\theta_1}(t, p_0)$$

This flow in turns corresponds to the vector field $\tilde{X}_1$, and in our example this vector field would describe the dynamics of changing the lighting condition.

Coordinate Flows to Data Flows

The above discussion focused on the curves, flow, and vector field associated with the first coordinate, and each of the n coordinates have analogous constructions. Each coordinate has its constant vector field $X_i(u) = \frac{\partial}{\partial x_i}|_u$ which assigns the i^{th} coordinate derivative to each point in coordinate space, and each coordinate vector field has its associated coordinate flow function θ_i. Using the inverse local chart ϕ^{-1} to map back onto the manifold, we have the corresponding vector fields $\tilde{X}_i$ and flows $\tilde{\theta}_i$, which correspond to changing the data by varying a single latent factor. For example, starting at an image of triangle, smoothly varying the first coordinate might correspond to smoothly translating this triangle. The local chart ϕ disentangles a patch of the data manifold into semantically meaningful coordinates, and varying the coordinates corresponds to traversing the data manifold by following the induced flows of these coordinate vector fields.

4.3 Commutativity

In this section, we first establish formal definitions of commutativity for smooth flows and vector fields before giving examples where checking commutativity is intuitive and easy. We then establish that commutativity is an equivalent condition for disentanglement and examine its implications.

Intuition

One observation is that in coordinate space, the flows of the coordinate vector fields commute. Suppose we flow along the i^{th} coordinate for time t and along the j^{th} coordinate for time s. If we were to reverse their order, we would still end up in the same place - the order of the flows

is interchangeable. Since each manifold patch is diffeomorphic to a coordinate patch, the corresponding manifold flows must also commute. This simple observation has important consequences on what constitutes a valid coordinate system and in fact is a necessary and sufficient for disentanglement.

Commutativity: Technical Definitions

Flows

First, we shall establish formal definitions of commutativity for the two pieces of manifold machinery mentioned so far: vector fields and curves. As in the above motivating example, we say that two flows commute if their order is interchangeable:

$$\theta_i(t, \theta_j(s, u)) = \theta_j(s, \theta_i(t, p)) \quad \forall s, t, p$$

where we appropriately restrict the flow domains s, t so everything is well-defined.

Vector Fields

As for vector fields, we can define commutativity via the Lie bracket. For two smooth vector fields V, W on smooth manifold M, their Lie bracket $[V, W]$ is another a smooth vector field on M. While it has many interpretations, one characterization of the Lie bracket $[V, W]$ is that it coincides with the Lie derivative of V with respect to W: $L_W V$. In local coordinates, $L_W V$ is the directional derivative of V along the flow of W, and intuitively it captures how much V changes when moving along W. Now, we say the vector fields V and W commute if $[V, W] = 0$. From the Lie derivative interpretation this means that V doesn't change when moving along the flow of W. Since the Lie bracket is antisymmetric ($[V, W] = -[W, V]$) this also means that W doesn't change along the flow V. The dynamics of V and W don't change when following the other's flow; they are independent of each other. The following results summarize the discussion so far:

Theorem 4.3.1. *(Thm 9.42 of [18]) For smooth vector fields V and W, The following are equivalent:*

1. *V and W commute*
2. *W is invariant under the flow of V*
3. *V is invariant under the flow of W*

A Practical Interpretation of Commuting Vector Fields

Consider a dataset of images of a single triangle at different locations and orientations. The three relevant factors here would be the triangle's orientation and location on the plane. Intuitively, the triangle rotates "in the same way" regardless of its location, and vice versa

for moving the triangle regardless of its orientation. If we denote the location of a triangle by a center point c, this can be formalized by the following rotation and translation operators that act on the plane:

$$\theta_{rot}(t,(p,c)) = [\exp\left(\begin{bmatrix}0 & -t\\ t & 0\end{bmatrix}\right)(p-c), c] \qquad ; \qquad \theta_{trans}((u,v),(p,c)) = (p+(u,v), c+(u,v))$$

where p is a point on the plane. One can show that these two operators commute in the formal sense, and this coincides with our intuition that rotations and translations are independent operations. Therefore, we can interpret the commutativity of vector fields as their dynamics being decoupled from each other, and in the context of disentanglement this means our latent factors' variations are independent of one another.

Commutativity Equivalence and Practical Implications

Now that we've established definitions of commutativity for flows and vector fields, it turns out that there is an equivalence between their commutativity:

Theorem 4.3.2. *(Thm 9.44 of [18]) Smooth vector fields commutes iff their flows commute.*

From the previous section, checking the commutativity of vector fields requires computing their Lie bracket. However, the above result means we can also determine commutativity by checking if their associated flows commute, which can be a much simpler, intuitive method. For example, going back to the translation/rotation operators mentioned in the triangle example, one could certainly check commutativity by computing their Lie bracket. However, it is clear that rotating the triangle about its center and translating it are interchangeable, and so the associated vector fields must commute. In practice, usually it is much easier to check commutativity in the flows than in the vector fields, and their equivalence greatly simplifies checking commutativity.

The Commutativity Criterion

Local Charts and Commutativity

Let us concentrate on a local patch V of the data manifold with local chart $\phi : V \to U$. As mentioned previously, the coordinate vector fields in U induce vector fields $\tilde{X}_i$ on the data manifold. By the naturality of the Lie bracket, the induced vector fields also commute:

$$[\tilde{X}_i, \tilde{X}_j] = 0 \qquad i \neq j$$

Theorem 4.3.2 tells us that their flows must also commute. Any local chart induces commuting flows on the data manifold, and commutativity is a necessary condition. It turns out the reverse is also true:

Theorem 4.3.3. *(Thm 9.46 of [18]) Let $\tilde{X}_1, \cdots, \tilde{X}_k$ be linearly independent commuting vector fields on a open subset V of a smooth manifold M. For every $p \in V$, there exists a local chart $\phi_p : V_p \to U_p$ containing p such that $\tilde{X}_i$ correspond to the coordinate vector fields X_i of that chart for $1 \leq i \leq k$.*

The extra condition of linear independence between the vector fields precludes trivial cases: the Lie bracket is anti-symmetric which implies that $[V, V] = 0$; every vector field commutes with itself; therefore, if $\{X_i\}$ is a system of commuting vector fields, then adding any linear combination to that set still results in a commuting set. In summary, the coordinate fields of every local chart form a commuting system, and a commuting system of linearly independent vector fields are the coordinate fields of some local chart.

Disentanglement and Commutativity

From the disentanglement perspective, the above result gives a necessary and sufficient condition for disentanglement. Any system of latent factors must commute in order to form a disentangled description of the data, and if they commute they there exist a local chart such that each factor corresponds to a coordinate. Note that Theorem 4.3.3 does not require that the number of latent factors k be equal to the dimension n of the data manifold - we can still apply the commutativity criterion to a set of latent factors that we believe only partially describe our data. Hence, disentanglement and commutativity are equivalent, and learning a system of commuting factors automatically gives a disentanglement description of our data. We summarize the discussion so far in the following statement:

Theorem 4.3.4 (Commutativity Criterion)**.** *Let $\{X_i\}$ be any set of potential latent factors. Assuming linear independence, commutativity, in either their flows or dynamics, is a necessary and sufficient condition for their disentanglement.*

Example: Rotations and Translations

The above result implies that it is impossible for a system of non-commuting factors to be jointly disentangled. Let us return to the image dataset of triangles under different orientations and locations. One well known example of non-commutative data transformations are translations and rotations of images, where rotation happens about the center of the image grid (rather than about the triangle's center). Disentangling this dataset would amount to learning some local chart

$$\phi(x_1, x_2, x_3) = p$$

such that x_1 corresponds to rotations and x_2, x_3 correspond to translations. However, rotations about a fixed point and translations don't commute, so no such chart can exist by Theorem 4.3.4. It is impossible to jointly disentangle them.

There is a fix to this problem: rather than rotations about a fixed point of the image grid, x_1 can instead parametrize rotations about a center point of the triangle. Object-centric rotations do commute with translations, which means their underlying vector fields must

commute. By Theorem 4.3.3, this guarantees the existence of local chart that disentangles them, and indeed we previously gave explicit operators that commuted with each other:

$$\theta_{rot}(t,(p,c)) = [\exp\left(\begin{bmatrix}0 & -t\\ t & 0\end{bmatrix}\right)(p-c), c] \qquad ; \qquad \theta_{trans}((u,v),(p,c)) = (p+(u,v), c+(u,v))$$

These operators act on both point p in the plane and a center point c of the triangle. Regarding images as functions on a subset of the plane, these operators induce changes in images by acting on their domains. Since these operators and hence their corresponding vector fields commute, these factors can be disentangled into a local chart.

Even though we could have generated the data using rotations about the image grid's center and translation, we are unable to disentangle them. What went wrong? Theorem 4.3.4 insists their joint disentanglement is impossible, but this might seem counter-intuitive since we generated the data using them. For example, we can certainly come up with a generative function

$$f(x_1,x_2,x_3) = R(x_1)T(x_2,x_3)p_0$$

where $R(x_1)$ rotates the triangle and $T(x_2,x_3)$ translates it. Indeed, let us distinguish one corner of the triangle to break its natural symmetry, so each image of the triangle is uniquely determined by its orientation and spatial location. In fact, the dataset can be identified with the manifold $\mathbb{S}^1 \times \mathbb{R}^2$ with local coordinates (s,u), where the s indicates the triangle's orientation and u its spatial position. Assuming the original triangle p_0 has orientation 0 and is located at the origin, the generative function f in local coordinates would be:

$$(t,v) \mapsto (t, R(t)v)$$

Under this map, the three coordinate vector fields are sent to:

$$\frac{\partial}{\partial t} \mapsto \frac{\partial}{\partial t} + (-v_1\sin t - v_2\cos t)\frac{\partial}{\partial v_1} + (v_1\cos t - v_2\sin t)\frac{\partial}{\partial v_2}$$

$$\frac{\partial}{\partial v_1} \mapsto \cos t\frac{\partial}{\partial v_1} + \sin t\frac{\partial}{\partial v_2}$$

$$\frac{\partial}{\partial v_2} \mapsto -\sin t\frac{\partial}{\partial v_1} + \cos t\frac{\partial}{\partial v_2}$$

Note that all three vector fields involve the angle of rotation t. In particular, the translation vector fields $\frac{\partial}{\partial v_1}$ and $\frac{\partial}{\partial v_2}$ depend on the rotation operator, meaning that they are jointly entangled. While the generative function is a diffeomorphism, the coordinate fields it gives rise to do not coincide with the operations of grid-centric rotation and translation, and these transformations cannot be realized as a local chart.

While the above generative function f is rich enough to generate the entire dataset, its latent variables are entangled. One cannot find a coordinate description of the data such that the coordinates correspond to grid-centric rotation and translation. However, we were able to find another parametrization that could be realized as a coordinate chart, and the question arises if this is possible for any generative model. In the next section we tackle this question.

4.4 Disentangling Generative Models

Overview

So far, we assumed our data lies on a manifold and framed disentanglement as learning the local charts of this data manifold. How do generative models fit within this framework? Suppose given dataset P, we learned some generative model:

$$f(x_1, \cdots, x_m) = p \quad ; \quad p \in P$$

We assume that f is a smooth function like most popular generative models. Now, consider the latent code $x_p = (x_1, \cdots, x_m)$ that gives rise to the data point p: $f(x_p) = p'$. The generative model should ideally capture all of the variation in the data, so any sufficiently small variation in p should be captured by small variations about the latent code x_p. We shall see that this condition is enough to guarantee the existence of a local chart, up to a reparametrization and subsetting of the latents. In other words, the latent space of a sufficiently rich generative model can be compressed and locally disentangled into a coordinate map of the data.

The Rank Theorem

First, we establish some technical machinery that allows us to frame the preceding discussion. We say a smooth map between two manifolds $g : M \to N$ has constant rank r if, at each point $p \in M$, the image $dg(T_pM)$ has dimension r (recall that dg is map between vector spaces). We say that g is a smooth submersion if its rank equals the dimension of N and a smooth immersion if its rank equals the dimension of M. Returning to our generative model f, we assumed that around each data point p our model is rich enough to capture all local variation around p by a corresponding variation in its latent variables. Equating small variations with derivatives, this means that the generative model is surjective on each data point's tangent space. This surjectivity requirement is the same as requiring f to be a smooth submersion. Now, we state an elementary result of constant rank maps.

Theorem 4.4.1 (Thm 4.12 of [18])**.** *Let M and N be smooth manifolds of dimension m and n respectively, and let $F : M \to N$ be a smooth map of constant rank r. At each $p \in M$ there exists a smooth chart (U, ϕ) containing p and a smooth chart (V, ψ) containing $F(p)$, where $F(U) \subseteq V$, such that F in local coordinates takes the form:*

$$F(x_1, \cdots, x_m) = (x_1, \cdots, x_r, 0 \cdots, 0)$$

In particular, if F is a smooth submersion then:

$$F(x_1, \cdots, x_n, x_{n+1}, \cdots, x_m) = (x_1, \cdots, x_n)$$

Around each point there exists some local coordinate system such that only the first n coordinates of the latent space M are relevant, and in local coordinates F is merely a projection map.

Application to Generative Models

Let us apply the above result to our generative model. A new coordinate system is just a reparametrization of the old one, and so the result states that there is a smooth reparametrization of the latent space such that only the first n latents are relevant (n is the dimension of the data manifold). That is, there is exists some reparametrization ϕ of the latents:

$$\phi(y_1, \cdots, y_m) = x_1, \cdots, x_m$$

such that only $y_1, \cdots, y_n$ matter. In fact, this subset gives rise to a local coordinate chart of the data manifold:

Theorem 4.4.2 (Disentanglement of Generative Models)**.** *Suppose the data comprises or lies on an n-dimensional smooth manifold M. Let $f : U \to M$ be a generative function that is a smooth submersion from an open set $U \subseteq \mathbb{R}^n$ onto the data manifold. Then, for each datum p with latent code x_p (ie. $f(x_p) = p$), there exists a local smooth reparametrization ϕ_p such that the first n reparametrized coordinates are the coordinates of a local chart of M containing p.*

Proof. The Rank Theorem guarantees the existence local charts (ϕ_p, V) and (ψ_p, W) containing x_p and p respectively such that f in local coordinates is a projection onto the first n coordinates. More precisely, if π_n denotes the projection onto the first n coordinates then:

$$\psi_p \circ f \circ \phi_p^{-1} = \pi_n$$

Let $\pi(V)$ denote the projection of V onto its first n coordinates, and let $\pi^{-1} : \pi(V) \to V$ denote the injection into the slice $\{x \in V \mid x_k = 0 \ \forall k > n\}$ (we may center V about 0 without affecting anything). Then, the map $\psi_p \circ f \circ \phi_p^{-1} \circ \pi^{-1}$ is the identity. Therefore, $f \circ \phi_p^{-1} \circ \pi^{-1}$ is a diffeomorphism from $\pi(V) \subseteq \mathbb{R}^n$ to a local neighborhood of M containing $f(p)$, and its inverse is a local chart of M containing $f(p)$. The coordinates of this local chart are the first n reparametrized coordinates of the latent space under ϕ_p. □

Locally around each data point p there exists a reparametrization of the latent space that gives a disentangled description. Let us revisit the image dataset of triangles under translations and rotations. We previously noted that we could generate the entire dataset by translations and rotations about the image center, but we were unable create a local chart using these operations. However, the Rank Theorem guarantees the existence of a reparametrization that makes the latent variables into a local chart, and in that case we changed image-centric rotations to object-centric rotations.

Relaxing the Smooth Submersion Condition

So far, our discussion centered on generative functions that, at each data point p, could capture any small variation by a corresponding variation in the latent code. This condition

is strong and might not hold for every single data point, especially at the edges of the data cloud. However, it turns out we can apply the previous machinery locally at data points where the generative function captures all small variations about that point. First, we make use of the following result:

Theorem 4.4.3. *Let $F : M \to N$ be a smooth map between manifolds. If dF_p is surjective (injective), then there exists an open neighborhood U containing p such that $f|_U$ is a smooth submersion (immersion).*

For a generative function f it is very likely there are data points p where f can capture all small variation about p, and df_p is surjective. Then, the above result allows us to restrict our attention to an open neighborhood around the latent code x_p of p and apply the previous subsection's machinery to yield a local chart about p.

On the other hand, sometimes a generative model can only partially describe the data, especially if it can vary in many complex ways. In such cases, it is reasonable to assume that at some points our generative model is nondegenerate, where each latent dimension encodes a different variation in the data. The above result states that in a local neighborhood about such points, the generative model is a smooth immersion. Applying the Rank Theorem to this local neighborhood gives a reparametrization that disentangles the latent space into a subset of the data manifold's coordinates. In fact, using a similar argument as in the previous subsection, this leads to a slice chart, a partial description of the manifold. The discussion up to this point can be summarized in the follow result:

Theorem 4.4.4 (Local Disentanglement of Generative Models)**.** *For a generative model f, let p be a point where either f captures all local variation (dF_p surjective) or each latent variable encodes different variations (dF_p injective). Then, there exists a local neighborhood containing the latent code x_p of p that can be disentangled.*

Maximal Compression of Generative Models

It often arises that the latent space of a generative model f is overcomplete, and one would like to reduce the latent dimension. Ideally, the goal would be to maximally compress the latent dimension without affecting the expressivity of the model. In this section, we consider how to define expressivity in a generative model, and we argue that the disentanglement procedure in the previous section achieves the above goal.

Defining the Expressiveness of a Generative Model

There are many ways to define the expressiveness of a generative model. One natural definition is its range, or the breadth of data a model can generate. However, we shall see that some maps preserve the global range of a function while restricting its local range. This suggests that an appropriate definition of expressiveness should also consider the local variability of a model.

Consider two clocks, each of which has only an hour hand. We can describe these clocks via two dimensions, one for each hand's location, and in fact the positions of the clocks' hands are isomorphic to a torus. Therefore, let our generative function be the natural map $g : \mathbb{R}^2 \to T^2$. Suppose we simultaneously rotated the left clock at rate a and the right clock at rate b. Coupling the rotations reduces the degrees of freedom from two to one, expressed via the map γ:

$$\gamma : \mathbb{R} \to \mathbb{R}^2 \qquad ; \qquad t \mapsto (at, bt)$$

Under this reparametrization, our generative function $g \circ \gamma$ generates clock positions by simultaneous rotation at rates a and b for their respective clocks. If $\frac{a}{b}$ is irrational, then simultaneously rotating both will never lead to a repeat position, and with sufficient rotation $g \circ \gamma$ can approximate any clock position arbitrarily well: we have described our data using just one latent dimension rather two. However, this is unsatisfying because $g \circ \gamma$ is locally limited in its clock positions. Originally, we could arbitrarily rotate either of the clocks' hands under g, but under $g \circ \gamma$ small variations in the latent dimension lead to specific coupled combinations of clock positions.

In light of this, merely preserving a generative function's range does not seem to be a satisfactory definition of expressiveness, and we need to consider local expressivity. One of way of framing local expressivity is by considering all possible directions of change at a point; preserving a generative model's expressiveness would mean preserving all directions of change possible under the generative model. Therefore, we formalize the local expressivity of a generative model under the following definition:

Definition 2. *The **local expressivity** of a generative model $f : X \to M$ at latent code $x \in X$ is quantified by the dimension of the tangent space $df(T_xX)$.*

Disentanglement Provides Maximal Compression

Given a latent reparametrization γ, the tangent space $T_{g\circ\gamma(y)}M$ is a subspace of $T_{g(x)}M$ where $x = \gamma(y)$, and preserving the tangent space is equivalent to preserving its dimension. Therefore, we require that any latent reparametrization γ to preserve the dimension of T_xX. Our machinery applies whenever g is full rank (ie. submersion or immersion), and in local coordinates it results in a full-rank projection map. Therefore, under our proposed definition of local expressivity, the disentangling reparametrization introduced in this section maximally compresses the latent space while preserving a generative model's expressivity since it automatically is a full-rank reparametrization.

4.5 Application to Matrix Exponential Operators

In this section, we apply the concepts covered so far to the task of learning matrix exponential operators that traverse a dataset. We shall see that applying the commutativity criterion justifies a common computational shortcut and vastly simplifies the dictionary of matrix generators to be learned.

Overview

One common approach to learning continuous data operators is through the matrix exponential

$$f(t) = e^{tA}$$

This can be seen as a continuous analog of the linear operator A, applied for time t. In practice, to learn these operators one learns a dictionary of matrix generators A_i. Then, one generates a continuous operator by summing and exponentiating these matrix generators:

$$e^{\sum \alpha_i A_i}$$

Given some starting point p_0, one finds an appropriate matrix exponential that such that any close point p can be approximated by its application to p_0:

$$e^{\sum \alpha_i A_i} p_0 \approx p$$

In this manner, the learned dictionary of matrix generators form operators that can describe any point of the dataset.

Sums of Matrix Generators

Commutativity of Matrix Generators

Computing a matrix exponential is relatively expensive, and a common computational shortcut is to assume the generators A_i are jointly diagonalizable: $A = PD_iP^{-1}$ for some diagonal matrix D_i with common matrix P. Then, the matrix exponential is easy to compute

$$e^{\sum \alpha_i A_i} = e^{\sum \alpha_i P D_i P^{-1}} = P e^{\sum \alpha_i D_i} P^{-1}$$

as the matrix exponential of a diagonal matrix is just the exponentiation of each individual diagonal entry.

Furthermore, one problem when working with sums of matrix generators is that the property

$$e^{sA+tB} = e^{sA} e^{tB}$$

generally holds only when A and B commute. This introduces a spurious ordering in the dictionary elements, where the matrix exponential will be different depending on the order of summation. A common solution is to enforce that the matrix generators commute, and this also has a computational benefit: since commuting matrices are simultaneously diagonalizable, the dictionary of commuting matrices can be put in the form $A_i = PD_iP^{-1}$. We arrive at the diagonalization trick by enforcing commutativity.

Let us examine commutativity of these matrix exponentials from the disentanglement perspective. Locally at each datapoint p_0 we can think of these operators as generative functions ϕ:

$$\phi(\alpha_1, \cdots, \alpha_n) = e^{\sum \alpha_i A_i} p_0 = p$$

where the generators A_i commute. Suppose we varied α_1 while holding all other α_i's constant, and let $p = e^{\sum_2 \alpha_i A_i} p_0$. The resulting function would be a flow function θ_1, as

$$\theta_1(t+s, p) = e^{(s+t)A_1} p = e^{tA_1} e^{sA_1} p = \theta_1(t, \theta_1(s, p))$$

In fact, each of the arguments α_i corresponds to a flow θ_i, and since their sums commute their flows also commute. Their associated vector fields are $X_i(p) = A_i p$, and computing their Lie bracket:

$$[X_i, X_j]_p = J_{X_i}|_p X_j(p) + J_{X_j}|_p X_i(p) = A_i A_j p - A_j A_i p = 0$$

wherre $J_{X_i}|_p$ is the Jacobian of X_i evaluated at p. Their Lie bracket is 0, so the vector fields commute. Indeed, using the fact that $\frac{\partial}{\partial t} e^{tA} = A$, every matrix exponential is the flow of a unique linear vector field $X_A(p) = Ap$. Using the commutativity equivalence between flows and their infinitesimal generators, we confirm that commutativity of matrix exponentials is equivalent with commutativity of their matrix generators.

By the commutativity criterion, our generative function $\phi(\alpha_1, \cdots, \alpha_n)$ is in fact a local chart. Commuting systems of matrix generators are special not only because they sum nicely or are easy to compute: they are precisely the generators that lead to a disentangled description of our data. This also implies that when working with a general system of matrix generators, the systems of non-commuting generators are the coupled systems whose dynamics are entangled. Therefore, this suggests that the correct formulation is the diagonalized case: not only is it computationally cheaper and simpler, it is the formulation that guarantees a disentangled system of linear operators.

Commutative Approximation to a System of Matrix Exponentials

So far, we have covered theoretical reasons for why it is desirable to start with diagonal matrices when working with matrix exponentials. However, suppose we started with an arbitrary system of matrix generators. Then, is there an equivalent or approximating system of diagonal generators?

Assuming our original system is linearly independent, applying the disentangling machinery from the previous section yields reparametrized latents:

$$\beta_1, \cdots, \beta_n = \psi(\alpha_1, \cdots, \alpha_n)$$

that form a local chart. Then, using these reparametrized latents our generative function becomes:

$$\phi(\beta_1, \cdots, \beta_n) = e^{\sum \psi_i^{-1}(\beta) A_i} p_0$$

where the weights $\psi_i^{-1}(\beta)$ are smooth functions of the reparametrized coordinates. At this point, one may ask if there is an equivalent system of matrix exponentials that uses the disentangled coordinates β_i. Since such a system is a local chart and hence must commute,

the associated matrix generators are jointly diagonalizable. Hence, we equivalently ask if there exists diagonal matrices D_i and matrix P such that

$$\psi(\beta_1, \cdots, \beta_n) = Pe^{\sum \beta_i D_i} P^{-1} p_0$$

generates the same range as that of our original generative model ϕ. While we do not have a definitive answer, we perform some experiments that suggest we can approximate our original system with a commuting one.

Experiment Overview

In this section, we empirically test if an arbitrary system of matrix exponentials can be well approximated by a commuting one, at least locally around a point. We are given a system of matrix generators A_i along with a fixed initial point p_0. Together, these describe a small patch around p_0 via the generative function:

$$\phi(\alpha_1, \cdots, \alpha_n) = e^{\sum \alpha_i A_i} p_0 = p$$

Our question is if there exits diagonal matrices D_i with common matrix P such that the function:

$$\psi(\beta_1, \cdots, \beta_n) = Pe^{\sum \beta_i D_i} P^{-1} p_0$$

generates a similar set of points as that of ϕ. To simplify things, we will assume that P is orthogonal and so $P^{-1} = P^T$. To this end, we shall learn D_i and P by optimizing the following objective:

$$L(P, D, \beta, p) = \frac{1}{2} ||Pe^{\sum \beta_i D_i} P^T p_0 - p||^2$$

for p in the range of ϕ.

Gradients

Let $\epsilon = Pe^{\sum \beta_i D_i} P^t p_0 - p$ denote the error; let $D = \sum \beta_i D_i$ the exponent; let P_j denote the j^{th} column of P. Then, the gradients are:

$$\frac{\partial L}{\partial \beta_i} = \epsilon^T (Pe^D P^T p_0)$$

$$\frac{\partial L}{\partial P_j} = D_{jj}[(P_j^T p_0)\epsilon^T + (P_j^T \epsilon) p_0^T]$$

$$\frac{\partial L}{\partial D_{ij}} = \epsilon^T (P_j P_j^T e^D \beta_i) p_0$$

d	n	ARE
8	3	.022
10	4	.098
15	6	.074
30	9	.090
60	18	.142

Table 4.1: d denotes the dimension of the underlying space, and n denotes the number of matrix generators. The Average Relative Error (ARE) is a measure of how well the learned diagonalized system approximates the original matrix exponentials.

Experiment Details and Results

We generate a batch of 50 data points $p = e^{\alpha_i A_i} p_0$, where the α's are randomly sampled. Then, we perform alternating gradient descent to learn P and D_i, and this procedure is performed on 5 batches in total.

After learning the diagonalized system $(P, \{D_i\})$, we test how closely it approximates the original system by testing it on a separately generated batch of 100 points. We then compute the average relative error (ARE):

$$ARE = \frac{1}{N} \sum_i \frac{||\hat{p}_i - p_i||}{||p_i||}$$

where $\hat{p}_i$ is the approximation of p_i using our diagonalized system. We repeat the above procedure for a range of values in both the data dimension d and the number of generators n. The results are summarized in Table 4.1.

We see empirically that commuting systems approximate non-commuting systems, even when the number of generators is small relative to the dimension. In light of the theoretical and computational properties of commuting diagonalized systems, this suggests that using disentangled systems from the start is a good approach.

Extending to a System of Matrix Exponentials

To conclude this discussion, we briefly touch on extending a system of commuting matrix. We are given generators A_i that give a partial, disentangled description of our data, where $A_i = PD_iP^{-1}$ for some common matrix P. Suppose there were a matrix A such that together with A_i's they form a (partial) local chart. The commutativity condition requires that the vector fields represented by A_i commute with that of A, or equivalently that A_i commutes with A as we saw in a previous section. Hence, they can be jointly diagonalized as QD_iQ^{-1}. The question arises: is this new basis Q different from P, and do we have to learn an entirely new basis? The answer is no:

Theorem 4.5.1. *Let A_i be commuting set of matrices, which can simultaneously diagonalized as $A_i = PD_iP^{-1}$ for some common matrix P. Let A be a matrix that commutes with every A_i. Then, $A = PDP^{-1}$ for some diagonal matrix D.*

Proof. We know that the set of matrices $\{A, A_1, \cdots, A_n\}$ is a commuting set of matrices. Therefore, they can be simulatenously diagonalized as QE_iQ^{-1} for common matrix Q. Diagonalization of a square matrix C is a special case of its Jordan canonical form $C = JBJ^{-1}$. Here, B is some block diagonal matrix, where the blocks corresponds to the generalized eigenvalues of C. Up to a permutation of the blocks in B and corresponding columns in J, the JCF of C is unique - in particular, J is unique up to a permutation of its columns. Applying this to our system of commuting vectors fields and their associated matrices, we can express A_1 as PDP^{-1} and QD_1Q^{-1}. By uniqueness of the JCF, we see that $P = Q$. Hence, $A = QDQ^{-1} = PDP^{-1}$. □

When extending a system of commuting matrix exponentials, the basis P stays the same. All that needs to be done is to learn a new diagonal matrix D, and extending a disentangled system of matrix exponentials is straightforward.

4.6 Commutativity and Operators

In this section, we briefly touch on interpreting commutativity from the perspective of operators acting on the data. That is, we shall focus on a set of operators $T_{t_1}, \cdots, T_{t_n}$ that generate our data by sequential action on a starting data point p_0:

$$T_{t_n} \circ \cdots \circ T_{t_1} p_0 = p$$

Here, the parameter t_i indexes a family of operators T_{t_i}: for example, we might consider the family of rotation operators, indexed by rotation parameter θ. Indeed, if we further impose the condition that $T_{s_1} \circ T_{t_1} = T_{s_1+t_1}$, the operator family T_{t_i} forms a (possibly non-smooth) flow function.

Coordinate Description via Operators

Recall that in the case of translations and grid-centric rotations of a triangle, these operators did not commute and hence they could not be realized as a coordinate chart. In that case, we generated the data by first translating then rotating a triangle p_0:

$$(t, v) \mapsto R_t \circ T_v p_0$$

where R_t is rotation by angle t and T_v is translation by vector v. In fact, this generative function was a diffeomorphism onto our data manifold and gave rise to a local chart. However, the induced coordinate vector fields under this local chart were not the original operators as the translation vector fields depended on the angle of rotation.

Indeed, the above situation is true for any set of operators. While one can certainly generate data by a sequential application of operators, by the commutativity criterion only the commuting sets can be expressed as coordinates. Otherwise, as we saw in the example of the triangle, the induced coordinate flows will not correspond to the original operators.

Operators and Group Actions

There has been a body of work that frames the data-generating process in terms of group actions: some set of composable, invertible transformations acting on the data to generate the entire dataset. Interestingly, the discussion so far on commutativity has some natural links to various approaches on this topic.

Factorizing Group Actions

In [12], they begin with the setup of a big group G acting on the data X to generate the dataset. In this context, disentanglement is framed as factorizing this group action into a product group ΠG_i acting on a corresponding factorized set ΠX_i, with each factor G_i acts only on its corresponding X_i. Thinking of each G_i as representing a different set of symmetries, we have separated the data into different components X_i that have their own symmetries G_i.

However, recall the example of the triangle: the data was generated by the product group $\mathbb{S}^1 \times \mathbb{R}^2$ - grid-centered rotation and translation - acting on the data, yet we could not give a coordinate map where each coordinate represented the appropriate operators. The main problem was that the two group factors - grid-centric rotation and translation - did not commute. This is a general defect, and we shall see that factorization into a product group requires commutativity.

Firstly, a group action of G on X can be understood as a group homomorphism f of G into $Aut(X)$, the automorphism group of X. Now, suppose we had a set of candidate subgroups G_i that might factorize the action of G, with a corresponding collection of group actions for each subgroup $f_i : G_i \to Aut(X)$. Then, this collection of group actions naturally filters through the categorical coproduct:

$$\begin{array}{ccc} G_i & \longrightarrow & \coprod G_i \\ & \searrow^{f_i} & \downarrow \coprod f_i \\ & & Aut(X) \end{array}$$

However, the group coproduct is the free product, not the product group as proposed in the above definition. This is because the image $f(G) \subseteq Aut(X)$ might not be commutative and could be sensitive to the order in which each group G_i is applied. Thus, while we may certainly consider different sequences of the G_i acting on X, the order matters and may lead to different orbits. For example, suppose we act on a triangle by grid-centered rotation

and vertical translation. If the triangle started at the origin, rotation followed by vertical translation would only generate triangles on the y-axis. On the other hand, translation followed by rotation would generate triangles with non-zero x-coordinates. This sensitivity to the order is incompatible with a factorized group acting on a factorized set: if each G_i acts separately on an X_i, then their order should not matter since they are acting on different components.

On the other hand, supposed we assumed that the entire group action was commutative, or equivalently that $f(G)$ is a commutative subgroup of $Aut(X)$. The categorical coproduct for commutative groups coincides with the direct product for finitely many factors, and hence the natural construction becomes the product group. Moreover, if we relaxed our assumption to just the $f(G_i)$ commuting with each other, that is enough to define a unique, order-agnostic map from the direct product into $Aut(X)$ that makes the following diagram commute

$$\begin{array}{ccc} G_i & \longrightarrow & \prod G_i \\ & \searrow^{f_i} & \downarrow{\scriptstyle f^*} \\ & & Aut(X) \end{array}$$

where $f^*(g_1, \cdots, g_n) = \sum f_i(g_i)$. Indeed, commutativity of the subgroups G_i is a necessary consequence of the factorization $G = \Pi G_i$. Each subgroup G_i in G is of the form $e_1 \times \cdots \times G_i \times e_{i+1} \times \cdots \times e_n$, and so each subgroup must commute with the others as their multiplication involves different components:

$$(g, e) * (e, h) = (g, h) = (e, h) * (g, e)$$

Therefore, since the subgroups G_i commute with each other, it is natural to require that their corresponding actions $f(G_i)$ commute. If we directly consider G to be a subgroup of $Aut(X)$ itself, the above discussion shows that G can be factorized as ΠG_i only if the G_i commute with each other. Factorization implicitly assumes commutativity.

Commutative Lie Group Actions

On a related note, there has been some work in learning actions of a commutative group on the data. That is, given some commutative group G, we wish to learn an appropriate mapping $f : G \to Aut(X)$ that fully describes our data ([7], [3]). In these works, one assumed that G was a linearly acting on the data via orthogonal transforms, and the goal was learn these operators by learning appropriate combinations of the irreducible representations. To vastly simplify the problem, one further assumed that G was a compact, connected, commutative Lie group, which are all equivalent to torii. This allowed one to express the group action in a simple form:

$$f(x_1, \cdots, x_n) = V e^{D(x_1, \cdots, x_n)} V^T$$

where $D(x_1, \cdots, x_n)$ is a diagonal matrix who entries represented the weighted sums of the irreducible representations. This greatly simplifies the problem and is a general case of

the diagonalization trick for learning matrix exponentials. For both of these shortcuts, the commutativity criterion gives another perspective on why these approaches work so well: the restriction to commutative groups guarantees that the learned map would be a disentangled chart of the data.

Chapter 5

Transfer of Algorithms: Players and Worlds

This chapter constitutes more speculative work and is included for the sake of completeness. We hope that the ideas presented here seem interesting to the reader. The primary contribution of this chapter is the assertion that a player or algorithm cannot be separated from its context, because its actions are tied to the world it lives in. Therefore, rather than focusing on the transfer of a specific player, we focus on structural maps between the worlds the players live in which induces a natural map of players.

5.1 Introduction

A Turing Machine (TM) is an abstract machine equipped with an infinitely long tape of discrete squares. It has a single machine head that scans one square at a time, and depending on the symbol in the scanned square as well as its own internal state (m-configuration in [29]) the TM does some operation(s) and changes its internal state. The TM has a table of rules that governs its actions, and there are finitely many rules in this table. As with any mathematical structure, one natural goal is to define the structure preserving maps. That is, what is a homomorphism of TMs? Purely as a model of computation, we can say two TMs are equivalent if they compute the same sequence. However, the construction of the universal TM, which is able to implement the rule table of any other TM, suggests some finer notion of similarity - that TMs are not only similar if they give the same output, but they also follow "similar" computations to arrive at that output. If we think of a given TM as an algorithm, we are asking if two algorithms are similar in the steps that they take, not just in the end product of their computation.

In this sense, suppose we had some TM M working with some set of symbols L. If we were to swap L bijectively with a new set L' and change the rule table accordingly, this would generate a new TM M'; however, M' is in some loose sense isomorphic to M, because they are doing the same computation but using different symbols. On the other hand, consider

two implementations of the same TM: say one is implemented on a physical machine with a physical tape, while the other is implemented virtually on a computer. Both again are running the same "algorithm", but formally they are different because they use different operations and symbols.

The difficulty here is trying to precisely separate the abstract "algorithm" and the implementation of said algorithm. The algorithm of the TM - its rule table - is tied to a specific set of readable and writable symbols. Hence, rather than trying to lift an algorithm into multiple settings, it might be more profitable to think in terms of "nice" structural maps between the worlds that they are implemented in. Many of the notions in this chapter draw heavily from category theory, and for the uninitiated reader we refer them to the classical text by MacLane [21].

5.2 Basic Definitions

We define a **world** $\mathcal{W}$ as a double $\mathcal{W} = (\Omega, \mathcal{F})$, where Ω is some set and $\mathcal{F}$ is a set of functions $f : \Omega \to \Omega$ that includes the identity function. We then define a **player** $\mathcal{P}$ as a double $\mathcal{P} = (\mathcal{Q}, \Gamma)$. $\mathcal{Q}$ is the set of internal states with a reserved start symbol $q_0 \in \mathcal{Q}$ and stop symbol $q_1 \in \mathcal{Q}$; Γ is the transition function $\Gamma : \mathcal{Q} \times \Omega \to \mathcal{Q} \times \mathcal{F}$. Similar to a Turing Machine, a player has its own internal state, and given some state of the world $\omega \in \Omega$, the player will perform one of the allowable actions $f \in \mathcal{F}$ before transitioning to a new internal state according to Γ. Note that in this setup, the actions of the player $\mathcal{P}$ are inherently tied to the world $\mathcal{W}$ it lives in, and $\mathcal{P}$ can only interact with $\mathcal{W}$ using the allowable functions $\mathcal{F}$.

We can augment the definition of a player to enforce finite running time. A **finite player** is any player that reaches the stop state q_1 after finitely many operations (calls to Γ) given any starting state $\omega_0 \in \Omega$. From now on we assume all players are finite unless otherwise stated. For finite players, we can denote their start and end states as $Start(\mathcal{P}), End(\mathcal{P}) \in \Omega$. Then, we say that two finite players $\mathcal{P}_1, \mathcal{P}_2$ are **homotopic** if

$$Start(\mathcal{P}_1) = Start(\mathcal{P}_2) \implies End(\mathcal{P}_1) = End(\mathcal{P}_2)$$

That is, viewing each finite player as a function $\Omega \to \Omega$, two players are homotopic if they represent the same function.

Having defined players and the worlds they live in, we can now define **world maps** $\Phi : \mathcal{W} \to \mathcal{W}'$: a pair of functions $\Phi = (\phi, \psi)$ that makes the following diagram commute for $f \in \mathcal{F}$.

$$\Phi : \mathcal{W} = (\Omega, \mathcal{F}) \to (\Omega', \mathcal{F}) = \mathcal{W}'$$
$$\phi : \Omega' \to \Omega \quad ; \quad \psi : \mathcal{F} \to \mathcal{F}'$$

$$\begin{array}{ccc} \Omega' & \xrightarrow{\psi(f)} & \Omega' \\ {\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\phi} \\ \Omega & \xrightarrow{f} & \Omega \end{array}$$

Given source world $\mathcal{W}$ and target world $\mathcal{W}'$, the world map Φ maps states of the target into the source and actions of the source into the target.

Maps of Players

If $\mathcal{P} = (\mathcal{Q}, \Gamma)$ is a player in world $\mathcal{W}$, then a world map $\Phi : \mathcal{W} \to \mathcal{W}'$ can be used to translate $\mathcal{P}$ into a player in $\mathcal{W}'$ via the following transformation:

$$\mathcal{Q} \mapsto \mathcal{Q} \qquad ; \qquad \Gamma \mapsto (I_{\mathcal{Q}} \times \psi) \circ \Gamma \circ (I_{\mathcal{Q}} \times \phi) \tag{5.1}$$

In other words, the induced player $\widetilde{\Phi}(\mathcal{P})$ will have the same internal states as $\mathcal{P}$. During its operation, the induced player first maps the current state of $\mathcal{W}'$ into a state of $\mathcal{W}$, whereupon the original player decides on an action f and then selects the appropriate analog $\psi(f)$ for the induced player to perform. Note that the commutative diagram condition enforces consistency between the actions of $\mathcal{P}$ and $\mathcal{P}'$: following the operation of $\mathcal{P}$, at any point we may map the current state ω to the corresponding state ω' and it would be consistent with following the operation of $\mathcal{P}'$.

For world $\mathcal{W}$, let $\mathcal{P}(\mathcal{W})$ be the set of all possible players in $\mathcal{W}$. Then, a world map $\Phi : \mathcal{W} \to \mathcal{W}'$ induces a map of players using equation 5.1.

$$\Phi : \mathcal{W} \to \mathcal{W}'$$
$$\widetilde{\Phi} : \mathcal{P}(\mathcal{W}) \to \mathcal{P}(\mathcal{W}')$$

Example: Homomorphisms of TMs

A TM can be framed as player in a specific world $\mathcal{W} = (\Omega, \mathcal{F})$. Let Ω be set of all functions $f : \mathbb{Z}_{\geq 0} \to \mathcal{L} \times \{0, 1\}$, where $f(i) = (L_i, i_k)$ with $i_k = 1$ if the i^{th} square is the scanned square and 0 otherwise. $\mathcal{F}$ would comprise the two shift operators as well as a print function for each symbol. Then, a homomorphism from TM M to M' would actually be a world map $\Phi : \mathcal{W} \to \mathcal{W}'$ such that $M = \widetilde{\Phi}(M) = M'$. Applying this to the case of TMs running in different languages, the state map would map sequences and print functions in one language to the other. In the case where the languages are bijective, the world map defined above is also bijective and hence the induced player map is also bijective. The makes precise the intuition that bijectively swapping the tape symbols does not change the underlying algorithm: they are related by a bijective, "algorithm-preserving" map of players.

5.3 Basic Properties

In this section, we'll gives some basic properties about world maps and their induced players maps.

Theorem 5.3.1. *The following proerties hold for world maps:*

1. *The composition of world maps is a world map*
2. *The induced player map of the composite is the composite of the induced maps:* $\widetilde{\Phi_1 \circ \Phi_2} = \widetilde{\Phi_1} \circ \widetilde{\Phi_2}$.
3. *For each world, the identity functions on the states* Ω *and functions* $\mathcal{F}$ *comprise the identity world map*

Proof. Let Φ_1 and Φ_2 be two worlds maps, with their associated state maps ϕ_1, ϕ_2 and function maps ψ_1, ψ_2. We defined the composite map via composition of the state and function maps: $\Phi_2 \circ \Phi_1 = (\phi_1 \circ \phi_2, \psi_2 \circ \psi_1)$. Commutativity of the composite map's diagram easily follows. Hence, the composite map is also a world map.

Now, let $\widetilde{\Phi}_i$ be the induced player maps for $i = 1, 2$. Again, the induced player map acts on each players transition function Γ by:

$$\Gamma \mapsto (I_{\mathcal{Q}} \times (\psi_2 \circ \psi_1)) \circ \Gamma \circ (I_{\mathcal{Q}} \times (\phi_1 \circ \phi_2)) \tag{5.2}$$

Using the identification $I_{\mathcal{Q}} \times (\psi_2 \circ \psi_1) = (I_{\mathcal{Q}} \times \psi_1) \circ (I_{\mathcal{Q}} \times \psi_2)$, it is easy to see that the above map is precisely the composite of the induced player maps.

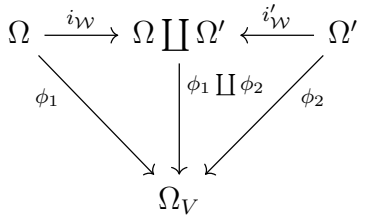
Finally, it is easy to see that the state and function identity maps form a commutative diagram, so together they form the identity world map. □

The previous properties can then be summarized as:

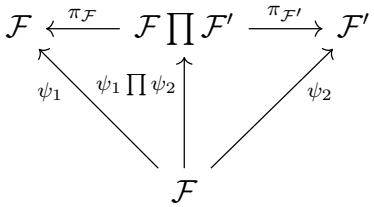
Theorem 5.3.2. *Let* W *be the category of worlds, with objects as worlds and morphisms as world maps. Then, the correspondence between world maps and player maps is a functor from* W *to* SET.

Theorem 5.3.3. *The category of worlds* W *has categorical products.*

Proof. The categorical product is essentially gluing the two worlds in a disjoint fashion. We shall construct the categorical product for worlds $\mathcal{W} = (\Omega, \mathcal{F})$ and $\mathcal{W}' = (\Omega', \mathcal{F}')$. First, note that world map $\mathcal{W} \to \mathcal{W}'$ comprises of a covariant map $\psi : \mathcal{F} \to \mathcal{F}'$ and a contravariant map $\phi : \Omega' \to \Omega$. Hence, we are led to the coproduct of the underlying state space and the product of the underlying function set. Let $\mathcal{W} \times \mathcal{W}' = (\Omega \coprod \Omega', \mathcal{F} \times \mathcal{F}')$. The state space is the disjoint union of Ω and Ω'. The function set $\mathcal{F} \times \mathcal{F}'$ consists of all functions $f \times f'$ that act on Ω by f and act on Ω' by f'. Consider any two world maps $\Phi_1 : \mathcal{V} \to \mathcal{W}$ and $\Phi_2 : \mathcal{V} \to \mathcal{W}'$. By the universality of the coproduct, for the state maps we have the commutative diagram:



Here, $i_{\mathcal{W}}, i'_{\mathcal{W}}$ are the canonical inclusions, and $\widetilde{\phi_1 \coprod \phi_2}$ is the induced map from ϕ_1, ϕ_2. Similarly, for the function set we have the following commutative diagram:



Firstly, the projections $\mathcal{W} \times \mathcal{W}' \xrightarrow{\pi} \mathcal{W}, \mathcal{W}'$ are worlds maps because they're restriction to either Ω or Ω' in the disjoint union with their respective function projections. Now, what remains is to check that the induced product map $\Phi_1 \times \Phi_2 = (\phi_1 \coprod \phi_2, \psi_1 \times \psi_2)$ is a world map. However, because this is essentially the disjoint operation on either Ω or Ω', commutativity follows.

□

Theorem 5.3.4. *The category of worlds has the trivial world $W_0 = (*, id)$ as an initial object.*

Proof. The trivial world is the singleton equipped with just the identity as a function. Every set has a canonical map to the singleton, and the canonical function map is the one that maps on identity function to the other. Taken together, these define a canonical world map from the trivial world to any world. □

Corollary 5.3.4.1. *The category of players has an initial object, and this initial object points to the 'do nothing' player in each player set.*

Theorem 5.3.5. *The category of worlds doesn't have a terminal object.*

Proof. This follows from the fact that the category of sets has no initial, as world maps are contravariant to their set maps. □

Lemma 5.3.1. *Suppose $\Phi = (\phi, \psi)$ is a world map $W \to W'$ such that ϕ, ψ have inverses ϕ^{-1}, ψ^{-1}. Then, Φ has the inverse $\Phi^{-1} = (\phi^{-1}, \psi^{-1})$.*

Proof. Note that as Φ is world map, the following equality holds for every $f \in \mathcal{F}$:

$$f \circ \phi = \phi \circ \psi(f)$$

which implies:

$$f = \phi \circ \psi(f) \circ \phi^{-1}$$

For $f' \in \mathcal{F}'$ we have $f' = \psi(f^*)$ for some $f^* \in \mathcal{F}$ by bijectivity of ψ. Then,

$$\begin{aligned}\phi^{-1}\psi^{-1}(f') &= \phi^{-1}\psi^{-1}\psi(f^*) \\ &= \phi^{-1} f^* \\ &= \phi^{-1}(\phi \circ f' \circ \phi^{-1}) \\ &= f' \circ \phi^{-1}\end{aligned}$$

Hence, the map makes the diagram commute and is a world map. □

Corollary 5.3.5.1. *For world map $\Phi = (\phi, \psi)$, if:*

1. *If ϕ is surjective then ψ must be injective.*
2. *If ϕ is injective then $\tilde{\Phi}$ maps homotopic players to homotopic players.*

Proof. Let use first assume that ϕ is surjective. Suppose $\psi(f) = \psi(f')$. As usual, we have the equality:

$$f \circ \phi = \phi \circ \psi(f)$$

This implies:

$$f \circ \phi = f' \circ \phi$$

Since ϕ is surjective, we have $f = f'$.

Now let use assume that ϕ is injective. Then, let $\mathcal{P}'$ denoted the induced player from $\mathcal{P}$. We have

$$\begin{aligned} Start(\mathcal{P}'_1) = Start(\mathcal{P}'_2) &\implies Start(\mathcal{P}_1) = Start(\mathcal{P}_2) \\ &\implies End(\mathcal{P}_1) = End(\mathcal{P}_2) \qquad \text{since homotopic} \\ &\implies End(\mathcal{P}'_1) = End(\mathcal{P}'_2) \qquad \text{by injectivity}\end{aligned}$$

□

One important question is whether a world map always exists between different worlds. The answer to this question is no.

Theorem 5.3.6. *The set of maps between different worlds $\mathcal{W}$ and $\mathcal{W}'$ may be empty*

Proof. We give a counter example. Let $\mathcal{W}_1 = (2, \Sigma_2)$ be the set of two elements with the flip and identity as its function set. Let $\mathcal{W}_2 = (3, \{Id, Cyc\})$ be the set of three elements with the function set consisting of the identity and a cyclic permutation. Consider any world map from $\mathcal{W}_1 \to \mathcal{W}_2$. Focusing on its function map ψ, it must send the flip operation F on the 2-set to either the cycle or the identity. The state map $\phi : 3 \to 2$ partitions 3 into two disjoint fibers. In order for the diagram to commute, we need a ϕ such that $\psi(F)$ acts like the flip F on the fibers of ϕ. This is impossible, because we cannot partition a set of three elements in two such that the cyclic permutation or identity will flip the partitions. □

5.4 Composition of Players

Recall that a player always starts with the start state q_0 and terminates operation with the end state q_1. Hence, there is a natural way to compose two players: once the first reaches its end state, we begin the operation of the second player. Formally, we can construct the **composition of two players $\boldsymbol{P_2 \circ P_1}$** as follows. The state space will be the disjoint union of the two players' states: $\mathcal{Q} = \mathcal{Q}_1 \coprod \mathcal{Q}_2$; the start symbol $q_0 = q_0^1$ will come from $\mathcal{Q}_1$, and the end synbol $q_1 = q_1^2$ will come from $\mathcal{Q}_2$. The transition function Γ will be defined piece-wise:

$$\Gamma(q, \omega) = \begin{cases} \Gamma_1(q, \omega) & q \in \mathcal{Q}_1 \\ \Gamma_2(q, \omega) & q \in \mathcal{Q}_2 \end{cases}$$

with the identification of the start state of Q_1 and the end state of Q_2: $q_1^1 \cong q_0^2$. If we consider the equivalence classes of players that are the same up to some finite sequence of the identity function, this immediately imposes a monoid structure on the set of players. In fact, world maps preserve this monoid structure.

Theorem 5.4.1. *The category of players is a subcategory of the category of monoids MON. Moreover, the correspondence between world maps and player maps is a functor from W to MON.*

Proof. It is easy to see that the "do nothing" player serves the role as the identity. The preceding discussion shows that, with respect to composition, a set of players (or more correctly their equivalence classes) forms a monoid. The fact that the induced players maps from a world map are a monoid homomorphism follow from the fact that the composition of commutative diagrams is again a commutative diagram. □

Bibliography

[1] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 585–591.

[2] Charles F. Cadieu and Bruno A. Olshausen. "Learning Intermediate-Level Representations of Form and Motion from Natural Movies". In: *Neural Computation* 24 (2012), pp. 827–866.

[3] Ho Yin Chau et al. "Disentangling images with Lie group transformations and sparse coding". In: *Neural Information Processing Systems*. 2022.

[4] Yubei Chen, Dylan M. Paiton, and Bruno A. Olshausen. "The Sparse Manifold Transform". In: *Neural Information Processing Systems*. 2018.

[5] Brian Cheung et al. *Discovering Hidden Factors of Variation in Deep Networks*. 2014.

[6] Brian Cheung et al. "Superposition of many models into one". In: *ArXiv* abs/1902.05522 (2019).

[7] Taco Cohen and Max Welling. "Learning the Irreducible Representations of Commutative Lie Groups". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, II–1755–II–1763.

[8] Patrick Esser, Robin Rombach, and Björn Ommer. "A Disentangling Invertible Interpretation Network for Explaining Latent Representations". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 9220–9229.

[9] Ross W. Gayler and Simon D. Levy. "A Distributed Basis for Analogical Mapping". In: 2009.

[10] Aditya Grover and Jure Leskovec. "Node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 855–864.

[11] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[12] Irina Higgins et al. *Towards a Definition of Disentangled Representations*. 2018.

[13] Pentti Kanerva. *Sparse Distributed Memory*. Cambridge, MA, USA: MIT Press, 1988.

[14] Jaeyoung Kang et al. "RelHD: A Graph-based Learning on FeFET with Hyperdimensional Computing". In: *2022 IEEE 40th International Conference on Computer Design (ICCD)* (2022), pp. 553–560.

[15] Denis Kleyko et al. "Vector Symbolic Architectures as a Computing Framework for Emerging Hardware". In: *Proceedings of the IEEE* 110.10 (2022), pp. 1538–1571.

[16] Serge Lang. *Algebra*. New York, NY: Springer, 2002.

[17] Kasper Green Larsen and Jelani Nelson. "Optimality of the Johnson-Lindenstrauss Lemma". In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. 2017, pp. 633–638.

[18] John M. Lee. *Introduction to Smooth Manifolds*. Springer, 2000.

[19] Tai Sing Lee and David Mumford. "Hierarchical Bayesian inference in the visual cortex." In: *Journal of the Optical Society of America. A, Optics, image science, and vision* 20 7 (2003), pp. 1434–48.

[20] Yunpu Ma et al. "Holistic Representations for Memorization and Inference". In: *UAI*. 2018.

[21] Saunders MacLane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics, Vol. 5. New York: Springer-Verlag, 1971.

[22] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. "Holographic Embeddings of Knowledge Graphs". In: *AAAI*. 2016.

[23] 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*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 809–816.

[24] Igor O. Nunes et al. "GraphHD: Efficient graph classification using hyperdimensional computing". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2022), pp. 1485–1490.

[25] Prathyush Poduval et al. "GrapHD: Graph-Based Hyperdimensional Memorization for Brain-Like Cognitive Learning". In: *Frontiers in Neuroscience* 16 (2022).

[26] P. Smolensky. "Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems". In: *Artif. Intell.* 46.1–2 (Nov. 1990), pp. 159–216. ISSN: 0004-3702.

[27] Anthony Thomas, Sanjoy Dasgupta, and Tajana Simunic. "A Theoretical Perspective on Hyperdimensional Computing". In: *J. Artif. Intell. Res.* 72 (2020), pp. 215–249.

[28] Yao-Hung Hubert Tsai et al. "Learning Factorized Multimodal Representations". In: *ArXiv* abs/1806.06176 (2018).

[29] Alan M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* 2.42 (1936), pp. 230–265.

Appendix A

Superposition and Graph Hierarchical Models

This appendix contains some cursory material on graph hierarchical models as a computation paradigm for hyperdimensional computing, applying the graph embedding method introduced in Chapter 2. The main ingredient starts with the premise of computing using superpositions of states, represented by linear sums of (nearly) orthonormal vectors. This naturally leads to state maps, which can be represented by our graph embedding method. We highlight some of its desirable properties and compare to a traditional Bayesian model. In particular, we hope that its properties make it a good alternative to realizing the ideas presented in Lee and Mumford's paper on modeling the visual processing pathway[19].

A.1 Superposition Again

Taking the concept of superposition a bit more literally, we can start at the notion of a superposition of states:

$$\Psi = \sum a_i \psi_i$$

where the vectors ψ_i form an orthonormal set and $||\Psi||^2 = 1$. When we want to "measure" from the state Ψ, we will observe one of the states ψ_i with corresponding probability a_i^2.

We can work backwards: say we had some finite set $X = \{x_1, \cdots, x_n\}$. We could assign each element x_i to some vector v_i such that $\{v_1, \cdots, v_n\}$ form an orthonormal set. Then, any distribution $P = (p_1, \cdots, p_n)$ on X could be represented by a superposition of the form:

$$\Psi_P = \sum a_i v_i \qquad ; \qquad a_i^2 = p_i$$

Importantly, assuming $a_i \in \mathbb{R}$, we have two choices $\pm a_i$ that satisfy the above constraints. Here, one might object and say that we could also represent the same distribution P by a convex combination of the embedding vectors:

$$P = (p_1, \cdots, p_n) \mapsto \sum p_i v_i$$

This way, we avoid the ambiguity of the sign of the coefficients and get a one-to-one map from distributions to convex combinations. However, as it will become clearer later, the introduction of parity allows for interesting behavior to emerge, since states can add or cancel each other. This idea of fuzzy superposition leads to an interesting question: how can we build hierarchical models using such a framework?

A.2 Hierarchical Model Basics

In this section, we detail the basic operation of a hierarchical model, and we specifically consider a hierarchical model with 3 layers - $X \to Y \to Z$ - where each layer is a set of possible states generated by superpositions of their respective singular states. To ground things, we can imagine this as some model of the visual pathway, where X, Y, Z represent low, mid, and high-level stages of visual cognition.

State Maps

The singular low-level state x_1 might suggest multiple mid-level states y_i, which we represent as a superposition:

$$x_1 \mapsto \sum \alpha_{i1} y_i \qquad ; \qquad \sum \alpha_{i1}^2 = 1$$

By linearity, this map can be represented by:

$$(\sum \alpha_{i1} y_i) \otimes x_1 = \sum \alpha_{i1} (y_i \otimes x_1) = \sum \alpha_{i1} (y_i x_1^T)$$

In a similar fashion, every singular state $x_i \in X$ would also be mapped to some state in Y. By the superposition principle, this defines a linear map $\alpha : X \to Y$ which takes the form:

$$\alpha = \sum_i \sum_j \alpha_{ij} (y_i x_j)^T$$

Thus, the state $x = \sum a_j x_j$ would be mapped to:

$$x \mapsto \alpha x = [\sum_{ij} \alpha_{ij} (y_i x_j^T)](\sum a_j x_j) = \sum_{ij} \alpha_{ij} a_j y_i$$

After normalization, we have a valid state of Y. In the context of the visual pathway, a low-level visual percept like as a curve might suggest multiple mid-level percepts like a circle, an oval, or a square. We represent this by mapping the single percept to a superposition of the many percepts it suggests. Note that the proposed state map is precisely the graph embedding method proposed earlier. Indeed, we may pass from an orthonormal basis of singular states to a nearly orthonormal set using spherical codes, allowing for drastic compression of our model's parameters - especially, as we saw earlier, when our state map's matrix is sparse.

So far we have described how states of X map to states of Y: how does this fit within the hierarchical model? Say we have initial states x_0, y_0 of X, Y. The state x_0 then maps to αx_0, and we would like to update the state of Y using this information. There are two natural update rules: we can either replace or add the current state y_0 with αx_0. We defer to addition, because this has a nice interpretation as discretizing the dynamics of the linear vector field α. Thus, a single pass of bottom-up inference will use the information x_0 in X to compute a new suggested state αx_0 of Y. This new state will be combined with the current state to give the updated state $y_1 = y_0 + \alpha x_0$, and after normalization this is again a valid state. The same mechanics apply for bottom-up inference from Y to Z.

Top-Down Inference and Update Rules

Now, we would like to perform top-down inference, where we update X using information from Y. Given the state map $\alpha : X \to Y$, there is a natural choice: the adjoint operator of α. The adjoint operator is a linear map $\alpha^T : Y \to X$ which, as the notation suggests, is just the transpose of α in matrix form. Using the adjoint map α^T, we update X using Y in the same fashion as bottom-up inference: given y_0, we compute $\alpha^T y_0$ and add it to the initial state x_0 to get the updated state $x_1 = \alpha^T y_0 + x_0$. After renormalization, this becomes a valid state of X. The adjoint map α^T is a natural inverse to the flow of information from a state map α: the more the current state y coincides with αx_i, the more weight x_i will be given under α^T.

Another thing to note is that the state map $\alpha : X \to Y$ maps singular states of X to states of Y, so we require the columns of α to be unit norm. Say we relaxed this requirement to the columns of α having norm at most 1. Then, we could consider maps of the form $\tau\alpha$ for some constant $0 < \tau < 1$, and the update rule than becomes $y_1 = y_0 + (\tau\alpha)x_0$. Letting $\tau \to 0$, we see that our hierarchical model, during both bottom-up and top-down inference, becomes a linear dynamical system. In this sense, the update rule of adding together states, rather than replacing them, can be thought as discretizing the dynamics of a linear system.

A.3 Properties and Comparisons

In this section we'll go over some interesting properties of this hierarchical model and compare it to the traditional hierarchical statistical model: the Bayesian hierarchical model.

Modularity

The proposed model is amenable to manipulation: we can not only arbitrarily modify a state map $x_i \mapsto \alpha_j y_j$ independently of all others, we can also easily expand the model to handle new singular states, especially when using an approximately orthonormal set. Modifying the state map for an individual state consists of just adding or subtracting the appropriate edges $\sum_j \alpha_j y_j x_i^T$. Expanding the singular states is similarly simple. Say that we wanted

to add the singular state y' to Y, which would allow for new states that are superpositions involving y'; note than any states of Y remain valid superpositions in this new expanded state space. Going back to modeling the visual pathway, we might want to form a new mid-level concept y'. Naturally, we would define this mid-level concept as a superposition of lower-level concepts, giving rise to the state map $y' \mapsto \sum \alpha'_i x_i$. We then expand the state map $\alpha^T : Y \to X$ to accommodate y' by just adding the appropriate edges: $\alpha^T = \alpha^T + \sum \alpha'_i x_i y'^T$, and this similarly updates the adjoint map α. A similar picture holds for adding singular states to X.

Bayesian Comparison

For comparison, consider a traditional Bayesian model of the form $P(Y|X)$. Adding a new value y' to the support of Y would require generating a fundamentally new likelihood function $P(Y \bigcup y'|X)$. Even if we already have the values $P(y'|x)$ for every $x \in X$, there is no canonical procedure for grafting these values with $P(Y|X)$. Similarly, adding new values x' to X entails creating a new prior distribution $P(X \bigcup x')$. While there are definitely ways to augment the likelihood/prior in both cases, it is not as straightforward as in the superposition model.

Positive and Negative Interference: Explaining Away

As mentioned earlier, working with superpositions of states rather than convex combinations of states allows for states to interact in both a constructive and destructive manner. For example, suppose we had two states of X and two states of Y related by the following map:

$$x_1 \mapsto \frac{1}{\sqrt{2}}(y_1 + y_2) \qquad x_2 \mapsto \frac{1}{\sqrt{2}}(y_1 - y_2)$$

Individually, both x_1 and x_2 map to a state that is an equal superposition of the two singular states y_1 and y_2, and we have an equal chance of observing y_1 or y_2 in either state. By the superposition principle the state $\frac{1}{\sqrt{2}}(x_1 + x_2)$ would map to y_1, as destructive interference cancels out the y_2 term. Translating this to the visual model case, we can say that the low-level percepts x_1 and x_2 individually both involve the percepts y_1 and y_2 equally; however, when we know that both x_1 and x_2 are relevant, information about the relevance of both low-level percepts allows us to pinpoint a single mid-level percept y_1.

Similarly, the adjoint of the above map leads to the equations:

$$y_1 \mapsto \frac{1}{\sqrt{2}}(x_1 + x_2) \qquad y_2 \mapsto \frac{1}{\sqrt{2}}(x_1 - x_2)$$

Using top-down inference, the state $\frac{1}{\sqrt{2}}(y_1 + y_2)$ maps to the singular state x_1. We then update the state of X by adding x_1 and normalizing: we push the state closer to x_1, and increase the probability of observing just x_1. Going back to the visual processing model, mid-level concepts can "explain away" lower-level concepts through destructive interference.

A similar phenomenon occurs when states interact through positive interference. If we consider the maps

$$x_1 \mapsto \frac{1}{\sqrt{2}}(y_1 + y_2) \qquad x_2 \mapsto \frac{1}{\sqrt{2}}(y_2 + y_3)$$

the state $\frac{1}{\sqrt{2}}(x_1 + x_2)$ maps to $\frac{1}{\sqrt{6}}(y_1 + 2y_2 + y_3)$ - we are four times as likely to observe y_2 as the other singular states. Individually, x_1 and x_2 map to states that do not statistically prefer a single one, but a superposition of both leads to a state where one observes y_2 with high probability.

Bayesian Comparison

Such behavior is impossible to implement in a Bayesian model, because when we work with probabilities we are working with mixtures and not superpositions. That is, when dealing with distributions over X, the state of X is always a single outcome x_1 and can never be both x_1 and x_2 at the same time. Therefore, it is impossible to model interactions between the different states. In a Bayesian model, the closest analog to a superposition $a_1x_1 + a_2x_2$ is the corresponding distribution $P = (a_1^2, a_2^2)$. Then, through the likelihood function $P(Y|X)$ this induces a distribution on Y:

$$P(y) = P(y|x_1)p(x_1) + P(y|x_2)p(x_2)$$

Note that since all of the terms on the RHS are positive, one can never have destructive interference of probabilities. The closest behavior one can get is if one state x_1 induces a (near) deterministic state on Y. For example, we have $P(y_1|x_1) = 1$. Then, we can push Y toward the state y_1 by pushing the distribution on X towards x_1. However, this is unlike the behavior of the superposition model: with superposition, we had a maximum entropy state $\frac{1}{\sqrt{2}}(x_1 + x_2)$ induce a zero entropy state y_1, even though each individual zero entropy state x_i induces a max entropy state $\frac{1}{\sqrt{2}}(y_1 \pm y_2)$.

Model Transfer

One central question concerning any model is how to transfer a given model to a new context. For the superposition model, there is a natural way to do this: suppose we had two state spaces X, X'. What is a suitable definition for how these two spaces are related? One way would be to define a state map $\rho : X \to X'$, and by the superposition principle such a map is completely determined by its action on the singular states of X. Hence, our state map is:

$$\rho : X \to X' \qquad ; \qquad \rho = \sum_j (\sum \rho_{ij} x_i') x_j^T$$

Going to our visual pathway model, our model might originally be attuned to processing images of people, where mid-level percepts are noses, eyes, arm, etc. Now, say we wanted to

transfer this model to handle images of lions. Then, a nose might correspond to a snout, a hand to a paw, hair to a mane, etc.

Now, say we wanted to retain the high-level percepts of a face, a lower and upper body, running motion, etc. for lions as for people, so we have the identity map $I : Z \to Z'$ on the space of high-level percepts (technically not the identity map but an isomorphism). Through the adjoint map $\beta^T : Z \to Y$, we can describe each high level percept $z \in Z$ as a superposition of mid-level human percepts. For example, a human face z_1 could be described as a superposition of a nose y_1, a mouth y_2, eyes y_3, and hair y_4:

$$z_1 = \sum_i \beta^T_{i1} y_i$$

The map of mid-level percepts, from human to lion, would then create a corresponding description for a lion face: a lion face z'_1 would be described as a superposition of a snout $\rho(y_1)$, a mouth $\rho(y_2)$, eyes $\rho(y_3)$, and a mane $\rho(y_4)$:

$$z'_1 = \sum_i \beta^T_{i1} \rho(y_i)$$

Hence, given any superposition model $\cdots \to X_i \to X_{i+1} \to \cdots$ defined by the maps $\alpha_i : X_i \to X_{i+1}$, we can transfer it to a model on a new context $\cdots \to X'_i \to X'_{i+1} \to \cdots$ via state maps on the corresponding state spaces $\rho_i : X_i \to X'_i$. Then, a map $\alpha'_i : X'_i \to X'_{i+1}$ would be described via the composition:

$$\alpha'_i : X'_i \xrightarrow{\rho_i^T} X_i \xrightarrow{\alpha_i} X_{i+1} \xrightarrow{\rho_{i+1}} X'_{i+1} \qquad \alpha'_i = \rho_{i+1} \alpha \rho_i^T$$

For example, in the example of transfer the human, the adjoint map $\beta^T : Z \to Y$ from high to mid-level percepts take the form:

$$\beta^T = \rho \beta^T I^T = \rho \beta^T$$

Bayesian Comparison

There are many restrictions when working the Bayesian hierarchical model. For example, say we wanted to transfer a Bayesian model $X \to Y$ to one on different spaces $X' \to Y'$. Conceptually, this would be working with the pull-back measures given measurable maps $f : X \to X'$ and $g : Y \to Y'$. However, we run into practical issues: Bayesian models are based on densities - $P(Y|X), P(X)$ - so how do we compute the densities of pull-back measures?

Our best and possibly only tool for computing such a density is to use the change-of-variables formula. However, at the very least this requires the maps f, g to be invertible. On the other hand, degenerate maps can arise in many context: two different outcomes x_1 and x_2 might be most similar to the same outcome x'_1. Without the change-of-variables formula, it is unclear how to compute such densities.

Moreover, the fact that we are working with mixtures and not superposition doesn't allow the maps f, g to have the same nuance as those of the superposition model. For example, in the human-lion example, there is no clear human analog for the percept of a tail y_1'. Hence, instead we can map the tail percept y_1' to a superposition of the two closest human percepts: an arm y_1 and a leg y_2. Hence, we have the map:

$$y_1' \mapsto \rho_1 y_1 + \rho_2 y_2$$

However, a map of probability spaces maps each outcome to a single outcome; it would be impossible to represent such ambiguity when working with the Bayesian model.

Skip Connections

In our three layer hierarchical model $X \to Y \to Z$, we considered both bottom-up and top-down inference between immediate layers: $X \to Y$ and $Y \to Z$. However, one might want to consider the effects of X on Z without going through the mediating step of updating Y. Given the state maps $\alpha : X \to Y$ and $\beta : Y \to Z$, by superposition (ie. linearity) we can compute the resulting state map $\gamma : X \to Z$ via:

$$\gamma : X \to Z \qquad \gamma = \beta\alpha$$

Note that we do need to normalize the columns of γ to ensure each singular state is sent to a valid state of Z. Thus, to directly pass information from X to Z, to calculate the state map we just need to do a matrix multiply.

Bayesian Comparison

Assuming we are working with a finite (or at least countable) state spaces, a Markov bayesian model $X \to Y \to Z$ consists of two likelihood matrices $P(Y|X)$ and $P(Z|X)$. To compute the corresponding likelihood matrix $P(Z|X)$, we would also just do a matrix multiply. Assuming the Bayesian state space is finite/countable, both models are equally easy to work with in this aspect.

However, the picture becomes different when dealing with continuous state space for the Bayesian. In this case, to compute the likelihood function $P(Z|X)$ we need to compute the integral:

$$p(z|x) = \int p(z|y)p(y|x)dy$$

This is a much more intensive computation than a matrix multiply. One might argue that a continuous state space for the Bayesian model is the more natural comparison to the superposition model: the superposition state space is always continuous, since each normalized superposition is a distinct space.

Simulateneous Inference

In our three layer model, $X \to Y \to Z$, how we do simultaneous inference of Y given X, Z? We are looking for some state function $\phi : X \times Z \to Y$ that respects superposition and appropriately updates Y. Fixing the Z argument to a singular state z_1, we have an induced state map $\phi_{z_1}(x) = \phi(x, z_1) : X \to Y$. Since such a state map must obey the superposition principle, we see ϕ must be linear in the first argument. A similar argument shows ϕ must be linear in the second argument, and so ϕ is a bilinear function $X \times Z \to Y$. By the universality of the tensor product, ϕ has a unique corresponding linear map $\tilde{\phi} : X \otimes Z \to Y$. Thus, we can bind together states of X and Z using the tensor product into a single state space $X \otimes Z$, and our state function ϕ becomes the unique linear map $\tilde{\phi} : X \otimes Z \to Y$. Both problem formulations are equivalent: we either look for a bilinear map on the product space $X \times Z$ or a linear map on the tensor product $X \otimes Z$.

Now, we are looking for some bilinear map induced from the linear maps $\alpha : X \to Y$ and $\beta^T : Z \to Y$. The pair of linear maps (α, β^T) naturally induce a linear map: $\alpha \otimes \beta^T : X \otimes Z \to Y \otimes Y$, where:

$$x_i \otimes z_j \mapsto (\alpha x_i) \otimes (\beta^T z_j)$$

This construction is universal in the sense that the set of pairs of linear maps (α, β^T) is bijective with the set of linear maps $\alpha \otimes \beta^T$. That is, there is a natural bijection (in fact, a natural isomorphism in the language of category theory):

$$[L(X, Y) \times L(Z, Y)] \cong L(X \otimes Z, Y \otimes Y)$$

We can go from the specific problem of finding a bilinear map for the pair (α, β^T) to the general problem of finding a "nice" function F that associates an appropriate linear map $F(f, g) : X \otimes Z \to Y$ to every pair of maps (f, g). That is, we are looking for some nice map F:

$$L(X \otimes Z, Y \otimes Y) \xrightarrow{F} L(X \otimes Z, Y)$$

Hence, the problem reduces to finding a fixed linear map $F : Y \otimes Y \to Y$. One canonical map is the diagonal map D, which maps $y_i \otimes y_i \mapsto y_i$ and 0 otherwise. Thus, using $F = D$, we have the resulting state function: $\phi = D \circ (\alpha \otimes \beta^T)$. Specifically, for $\alpha x_i = \sum \alpha_s y_i$ and $\beta^T z_j = \sum_t \beta_t y_t$, then $\phi(x_i \otimes z_j) \mapsto \sum \alpha_s \beta_s y_s$.

The states x_i, z_j map to the states $\alpha x_i, \beta^T z_j \in Y$, and for simultaneous inference we just compute the element-wise product the two vectors $\alpha x_i, \beta^T z_j$ - their agreement. Practically, to compute the resulting state from the pair (x, z), we would first compute $\alpha x, \beta z$ and then element-wise multiply the two vectors together. Then, inference would proceed as normal: add it to the current state of Y and normalize to get a new state. In summary, there is a natural and compuationally easy method to implement simultaneous inference.

Bayesian Comparison

The analog in a Markov Bayesian model $X \to Y \to Z$ would be computing the likelihood of Y given X, Z:

$$P(Y|X,Z) = \frac{P(Z|Y)P(Y|Z)}{Z_Y}$$

where Z_Y is the normalization constant. Much difficulty comes from computing the normalizing constant Z_Y. Some numerical approaches like MCMC are used to sample from the distribution $P(Y|X,Z)$; other variation approaches that try to approximate the likelihood. There are probably many other classes of methods for approximating the likelihood, but all of these methods approximate the likelihood and are usually computationally expensive.

Priors

In our model $\alpha : X \to Y$, we update Y using value of X by adding αx to the current state y. Thus, the initial states x_0, y_0 are natural analogs to priors in the usual Bayesian setup, and we can seed them with an appropriate state.

Bayesian Comparison

Note that unlike a prior, the initial state y_0 can have a negatively weighted singular state y_i in its superposition. Upon updating from X via αx, this negative term could cancel and destroy y_i from consideration. If we interpret a negative sign as meaning the belief in the exclusion of that state, the Bayesian analog would be having a prior probability of 0: $p(y_i) = 0$. In a Bayesian model, we update the prior to the posterior via:

$$p(y_i|X)\frac{P(X|y_i)P(y_i)}{P(X)}$$

Hence, if the prior is 0 then the posterior will always be 0, which is one major weakness of Bayesian models: certainty of exclusion in the prior persists. On the other hand, the superposition model has no such limitation.

Another issue in Bayesian models is the choice of an uninformative prior. Sometimes, the uniform prior is used and justified as a max entropy distribution. However, it is not invariant to reparametrizations of the sample space and so expresses a statement for a particular parametrization. On the hand, the Jeffrey's prior is another choice of uninformative prior that is invariant to reparametrizations. It is uninformative in the sense of depending solely on the likelihood function $P(X|Y)$, but it is perhaps antithetical to a Bayesian approach because the Fisher information is an expectation over all possible data X rather than the observed data. In any case, there is no canoncial uniformative prior. However, in the statemodel case the uninformative prior state would just be initialization at 0 - no state.

A.4 Booleans to States

So far, we have discussed the superposition of states x_i with the strict condition of orthonormality. However, in the real world we often deal with states that are "related" some manner, and we want the geometry of state space to reflect these relations: "related" things should be parallel while "unrelated" things should be orthogonal. In this section, we'll consider how to reflect statistical relations in the geometry of state space.

Norm-Preserving Maps

Say we have some set of singular states x_i. The state space X consists of superpositions of these states, and when we measure from a state $x \in X$ we will always observe one of the singular states. Over some period of repeated observation we begin to notice statistical relations between the observed states x_i, and we want the geometry of state space to respect these statistical relations.

Here is the formalism. At any fixed point of time, the observation of state x_i is a Boolean random variable B_i, with value 1 if we observe x_i and 0 otherwise. Thus, to each state x_i is the associated Boolean B_i, and together they live in the L^2 space of some probability space P: $L^2(P)$. Modulo the set of functions that are 0 a.s., this is an inner product space under the inner product:

$$< f, g >= Efg$$

Rather than working with the Boolean B_i, we will instead work with the mean-zero, unit-variance versions: $B_i' = \frac{B_i - EB_i}{\sqrt{Var(B_i)}} = \frac{B_i - p_i}{\sqrt{p_i(1-p_i)}}$. Note that the inner product between the B_i''s is equal to the correlation:

$$< B_i', B_j' >= Corr(B_i, B_j)$$

Since we want the geometry of the singular states x_i to reflect their statistical relations, we want the following equation to hold for all i, j:

$$Corr(B_i, B_j) =< x_i, x_j >$$

Properties

Correlation, Negation, and Independence

We have the following result:

Theorem A.4.1. *For any two Boolean functions B_i and B_j:*

1. *$Corr(B_i, B_j) = 1$ iff $B_j = B_i$.*
2. *$Corr(B_i, B_j) = -1$ iff $B_j = 1 - B_i$.*
3. *$Corr(B_i, B_j) = 0$ iff B_i and B_j are independent.*

Proof. We'll start with proving (1) and (2). For the Boolean B_i with its scaled/centered version $B_i' = \frac{B_i - p_i}{\sqrt{p_i(1-p_i)}}$, the negation $1 - B_i$ has the corresponding scaled/centered version:

$$(1 - B_i)' = \frac{1 - B_i - (1 - p_i)}{\sqrt{(1-p_i)p_i}} = -B_i'$$

Hence, the correlation between a Boolean and its negation is -1. On the hand, for an inner product space the Cauchy-Schwarz inequality is exact iff one of the vectors is a multiple of the other:

$$< u, v >= ||u||||v|| \iff u = cv$$

If the correlation between two Boolean B_i, B_j has absolute value 1, then $B_i' = cB_j'$. Since they are Booleans, this implies $c = \pm 1$, so B_j is equivalent to either B_i or its negation. This establishes (1) and (2).

Now for (3). The correlation between two Boolean is 0 iff their covariance is 0. For two Booleans, this means:

$$\begin{aligned} &Cov(B_i, B_j) = E(B_iB_j) - E(B_i)E(B_j) = 0 \\ &\implies P(\{B_i = 1\} \cap \{B_j = 1\}) = P(B_i = 1)P(B_j = 1) \end{aligned}$$

From the above equation, we deduce that two Boolean B_i, B_j are independent iff they have 0 correlation.

□

Hence if we embed vectors such that:

$$< x_i, x_j >= Corr(x_i, x_j)$$

then independent states map to orthogonal vectors and logical opposites map to the reverse vector.

Implementation

This structure can be learned on the fly, where over the course of observation we push the vectors of correlated states together and orthogonalize independent states. For example, we can keep a running total $\frac{N_{ij}}{N}$ over some fixed window of time T, where N_{ij} is the number of times the states x_i, x_j occur close together and N is the total number of observations over that window. Then, we would push the two vectors together

$$x_i \mapsto (\eta \frac{N_{ij}}{N} x_j + x_i)/||\eta \frac{N_{ij}}{N} + x_i||$$

We could easily update the state maps $\alpha : X \to Y$ via the projection operator. If x_i' is the new state vector and x_i is the old, the projection operator would be $P = x_i' x_i^T$. Then, we can update α to αP^T. A similar procedure works for updating the codomain Y.