Combinatorial Aspects of Feynman Integrals and Causal Set Theory

by

Kimia Shaban

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics & Optimization

Waterloo, Ontario, Canada, 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis consists of two distinct sections, the first highlighting causal set theory (CST) and the second focusing on Feynman period estimation. This work specifically covers how discrete structures from algebraic combinatorics can be applied to problems from physics, and how we can use computational tools and techniques to help solve these problems from a mathematical perspective.

In the first part of the thesis, we introduce CST, an approach to quantum gravity that focuses on how events in spacetime are causally related to one another. In CST, discretized spacetime is given by a locally finite poset. A significant portion of this section focuses on covtree, which allows us to evolve such a discrete spacetime, in a way that does not depend on arbitrary labelling. However recognizing nodes of covtree involve solving a particular downset reconstruction problem. To attack this we define a graph that compares downsets that differ by exactly one element, with a particular focus on the order dimension two case. This section of the thesis sheds light on evolving a spacetime by constructing future elements within the covtree framework. Reconstructing spacetime in this way will allow for researchers to advance our understanding of covtree's structure and improves causal set theory as an approach to quantum gravity.

In the second part, we provide an introduction to Feynman periods, explaining their significance in quantum field theory (QFT) calculations. We will discuss how machine learning models, such as linear and quadratic regression, and graph neural networks, can be applied to Feynman graphs and their properties, to predict the Feynman period. Even simple techniques like linear regression, on graph parameters only, which do not consider the graph structure directly, can make highly accurate predictions of Feynman periods. The work presented in this section, done jointly with Dr. Paul-Hermann Balduf, is published in the Journal of High Energy Physics. This research has significant implications for QFT computations, which can become computationally infeasible at large scales, and facilitates further exploration in particle physics.

Acknowledgements

I would like to first and foremost thank Dr. Karen Yeats, whose support make this possible. They believed in me from the very start, even in times I didn't believe in myself. Thank you for all of your generosity and kindness during the last few years. I greatly enjoyed my Master's, and you are a wonderful supervisor. You are exceptionally kind and were a gem in every step of thesis edits! I am so grateful to have had the opportunity to work with you!

My very first paper was published during my Master's, with Dr. Paul-Hermann Balduf as the main co-author. He came with an amazing project that was inspiring and helped me leap into the applications of artificial intelligence and machine learning. I greatly enjoyed our project, and thank Paul for his patience with me while I learned how to write a paper for the first time. Thank you for all of your help and support!

I want to thank Ian George for always being there to answer my many questions everyday in the office, and for his help with understanding a lot of this content. I would also like to thank Josephine Reynes for her help with IPE, her image in section 5, and for being a supportive friend!

I also want to extend my thanks to Rafael Sorkin and Fay Dowker, whose discussions were invaluable to me. I would additionally like to thank my readers Michael Borinsky, Jonathan Leake.

I want to thank my beloved parents and family, who did everything possible for me to be able to obtain a post-secondary education, which included uprooting their lives and moving to Canada due to the oppression of the Baha'i faith in Iran. To Kiana and Melika and my dog Princess, you are the best siblings I could have ever asked for. Thank you forever for your love on this journey.

I would like to greatly thank Willie Lei for his help on this entire journey. From applying to my first URA, to being accepted to this Master's and your help with fine-tuning the models in this very thesis, it would not have been possible without you. To my dear friend Tiadora Valentina Ruza, you are the best travel partner, seminar partner, and an amazing best friend. I have loved doing math and life with you!

Finally, I want to thank my better half and fiancé Oğuz Üstüntaş, and his loving family, whose support and kindness helped me every step of the way while I was writing this. I couldn't have done this without you!

Dedication

I dedicate this thesis to the Baha'i's of Iran, and to my grandparents whom are not here to read this Ghosronllah Valipour, Ali Shaban, and Malihe Hosseini. I love you, and your warmth and spirit motivated me to finish this project.

Table of Contents

List of Figures

List of Tables

Part 1

Causal Set Theory

Chapter 1

Introduction

This thesis is meant to showcase how mathematics can serve as a tool for understanding the fundamental principles of the physical world, in addition to how hand in hand math and physics go together. In particular, this work focuses on how discrete mathematical structures can be used to create a framework for analyzing problems where continuity assumptions break down, such as in quantum gravity and high energy physics. This thesis explores the application of discrete mathematics to two distinct research areas: causal set theory (CST) and the prediction of Feynman periods.

1.1 Thesis Road map

The first half of this thesis is dedicated to CST. This part of the thesis will provide an overview of partial orderings and how they are used in the CST framework. In this work, we use partially ordered sets (posets) to model spacetime. There are may approaches to quantum gravity, such as holography from string theory and loop quantum gravity, but none are satisfactory [104, 92, 19]. The approach that we will take is CST, which begins with a set of remarkably minimal and physically motivated assumptions: (1) spacetime is fundamentally discrete, and (2) its structure is entirely determined by the causal order of events. These assumptions are grounded in both physical reasoning and are mathematically simple. Notably, CST is an intrinsically Lorentzian theory from the outset, the causal order respects the light-cone structure of relativistic spacetimes. Even if CST does not provide a complete theory of quantum gravity, its conceptual simplicity, deep connection to spacetime causality, and mathematical clarity strongly suggest that it captures a core part of the underlying structure. As such, CST offers not only a compelling framework in

its own right but also a foundation that may prove essential in any eventual synthesis of quantum gravity.

The second half of this thesis is joint work with Paul-Hermann Balduf, focused on Feynman periods, a class of integrals that arise from Feynman integration. Feynman graphs serve as a combinatorial tool for encoding particle interactions in perturbative QFT, and each diagram corresponds to an integral [123]. The period of a Feynman graph is a specific scalar quantity derived from the associated Feynman integral, stripped of external parameters [29, 95, 76]. These periods are arithmetically interesting. However, calculating Feynman periods becomes increasingly difficult as the graphs get larger, with symbolic or numerical integration often becoming intractable due to the complexity of the integral and the combinatorial explosion of possible graphs [8, 123]. This motivates the exploration of new techniques, including the use of machine learning, to predict Feynman periods based on graph-theoretic and combinatorial features, as we explore in this thesis.

Because of this, part two of this thesis explores the use of machine learning models to predict Feynman periods. We use features of our graph such as the Hepp bound, the Martin invariant, and numbers of $6, 8, 10, 12, 14$-edge cuts, which are known to relate to the Feynman period in various ways, so that our models can aim to learn patterns in known period values and extrapolate them to new, unseen graphs. This approach offers a novel perspective on using artificial intelligence for Feynman period prediction [9].

The outline of this thesis is as follows:

- Chapter 2 introduces posets, with a particular emphasis on plane posets, a structure that features two interacting partial orders. This chapter sets the stage for later discussions on their applications in CST.
- Chapter 3 provides an overview of CST, focusing on growth models that we need for chapter four.
- Chapter 4 goes over the thesis' primary result for part 1, regarding the structure of a digraph of certain downsets.
- Chapter 5 introduces a background to Feynman integrals from a predominately graph theoretic perspective, and gives a mathematical motivation to the questions that are being answered with Feynman period predictions.
- Chapter 6 discuses a background to machine learning, providing an introduction to the models that we used to compare for our Feynman period predictions.

- Chapter 7 details my contributions to machine learning approaches for Feynman period estimation, discussing the results of the models used.
- Chapter 8 provides a concluding discussion, reflecting on the connections between discrete mathematics and physics, summarizing my findings, and suggesting directions for future research.

Chapter 2

Poset Background

2.1 Introduction and Motivation

Ordering relationships arise naturally in many mathematical and scientific contexts. There are total orders, where every pair of elements is comparable, such as with the natural numbers under $\leq$. A partially ordered set (or poset) generalizes the more familiar notion of a total order, by allowing some pairs of elements to be incomparable.

Posets are mathematically interesting, because they let you talk about order without needing everything to be comparable. From a combinatorics point of view, they show up in problems like counting linear extensions, computing Möbius functions, or understanding when a structure is a lattice. They are also used to study subsets ordered by inclusion, such as the Boolean lattice of all subsets of an n-element set, which underpins problems in extremal set theory and Sperner theory [46]. They also appear in partition lattices, which structure the ways of breaking a set into non-empty blocks. These are simple to state but can be hard to work out, which makes them interesting to study.

To consider a few further examples, in theoretical computer science, posets model dependency graphs where tasks must be executed in a specific order: for example, in job-shop scheduling, where tasks can only proceed once their prerequisites are complete [100]. In distributed computing, events in a system without a global clock can be partially ordered to represent causal dependencies, as in Lamport's "happened-before" relation [78]. The Bruhat order on elements of a Coxeter group, a highly structured poset, plays a key role in Lie theory and algebraic geometry [63].

This flexibility has made posets central to a wide range of applications across mathematics and computer science [111]. This chapter will provide a background by defining

posets and plane posets, and illustrating their structure with examples, which we will expand on more in chapter 3 and chapter 4.

2.2 Posets

To establish a foundation, we begin with the formal definition of a partial order, followed by several standard definitions of poset related terms from [116].

Definition 2.2.1 (Poset)**.** A partially ordered set (poset) is a set P, equipped with a binary relation $\leq$ satisfying the following properties for all $x, y, z \in P$.

- Reflexivity: $x \leq x$.
- Anti-symmetry: If $x \leq y$ and $y \leq x$, then $x = y$.
- Transitivity: If $x \leq y$ and $y \leq z$, then $x \leq z$.

This definition captures hierarchical structures where elements can be compared according to a partial ordering, though some elements may remain incomparable, which is a crucial component to why posets are so useful in the field of causal set theory (CST). Note that we often talk about posets using standard abuse of notation where we define a poset as $P = (P, \leq)$, where P is both the poset structure and its underlying set of elements.

As an example consider $P = \{ a, b, c, d, e \}$ with partial order relation $\leq$ such that $a \leq b$, $a \leq c$, $a \leq d$, $a \leq e$, $b \leq d$, and $c \leq e$, and reflexivity. We can check that $(P, \leq)$ satisfies the axioms, and so is a poset.

In the definition of poset, we have that every element is comparable to itself; this is referred to as the non-strict case [116]. We will use the notation $<$ to mean for $x, y \in P$ $x < y$ if and only if $x \leq y$ and $x \neq y$.

Remark 2.2.1 (Strict Poset)**.** *There is an equivalent poset definition, where instead of the non-strict partial order relation $\leq$, elements are compared with a strict partial order relation $<$, resulting in a slightly different set of axioms. These two definitions are equivalent, and result in the same underlying structure. For this thesis the non-strict version is used, however for CST often the strict version is used.*

We will now give some standard definitions.

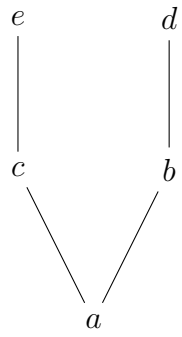


Figure 2.1: Basic Example of Order Diagram

Definition 2.2.2 (Total Order)**.** A total order (or a linear order) is a partial order, but where every two elements of our underlying set S are comparable. For all $x, y, z \in S$

- Reflexivity: $x \leq x$.
- Anti-symmetry: If $x \leq y$ and $y \leq x$, then $x = y$.
- Transitivity: If $x \leq y$ and $y \leq z$, then $x \leq z$.
- Strongly Connected: Either $x \leq y$ or $y \leq x$.

Definition 2.2.3 (Finite Poset)**.** A poset $P = (P, \leq)$ is finite if its underlying set of elements P is finite.

Definition 2.2.4 (Covering Relation ([116], chapter 2))**.** Let $(P, \leq)$ be a poset. We say that y covers x, written $x \lessdot y$, if $x < y$ and there is no $z \in P$ such that $x < z < y$.

Definition 2.2.5 (Order Diagram)**.** An order diagram (also known as a Hasse diagram) is a digraph representation of a finite poset, where for all $x, y \in P$, there will be a directed edge from x to y if and only if $x \lessdot y$.

An order diagram provides a visual representation of a poset that gives an intuitive way to visualize its structure, where by convention the directed edges are drawn using the vertical placement of elements in the Euclidean plane to indicate the order, rather than using arrows. if for $x, y \in P$, $x \leq y$ then x is drawn lower than y (has a lower height coordinate) and edges are only drawn between elements that are covering relations (such that $x \lessdot y$, or $y \lessdot x$). This is the standard visual representation of posets, and is shown in figure 2.1.

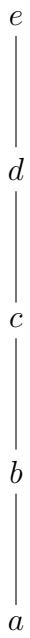


Figure 2.2: Total Order Order Diagram Example

Note that order diagrams for total orders result in straight lines. For example, consider the set $P = \{ a, b, c, d, e \}$ and the total order relation $\leq$ where $a \leq b \leq c \leq d \leq e$. The order diagram for $(P, \leq)$ is drawn in figure 2.2

As another example, consider the set $P = \{ 1, 2, 3, 4, 5, 6 \}$, with the partial order given by the divisibility relation $|$ where $a \leq b$ if a divides b. It's order diagram is shown in figure 2.3.

We will give some additional definitions that will be useful to us.

Definition 2.2.6 (Graded Poset)**.** A poset $(P, \leq)$ is graded if there exists a rank function $r : P \to \mathbb{N}$ such that:

- If $x \in P$ is a minimal element then $r(x) = 0$.
- For $x, y \in P$, if y covers x, then $r(y) = r(x) + 1$.

As an example of a graded poset, consider the Boolean lattice B_n, the poset formed by the power set of a finite set of n elements ordered by set inclusion. The rank of each element is given by its cardinality, that is $r(p) = |p|$ for any p in the power set. The order

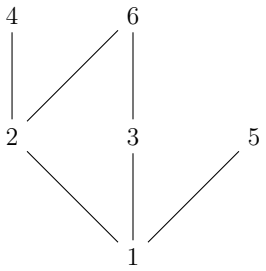


Figure 2.3: Order Diagram Example

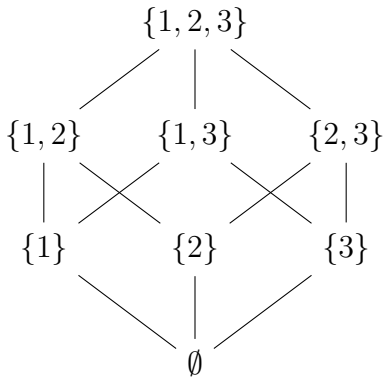


Figure 2.4: Illustration of the B_3 Boolean Lattice

diagram of B_n visually organizes the subsets into levels according to their size, with edges representing cover relations (i.e., adding or removing one element). An example of B_3 is shown in figure 2.4. An example of a non graded poset is shown in figure 2.5.

Definition 2.2.7 (Chain)**.** Let $P = (P, \leq)$ be a poset. A chain is a subset $C \subseteq P$ such that for all $x, y \in C$, either $x \leq y$, or $y \leq x$, so that all elements in the chain are comparable.

A chain C of a poset P, can also be considered a subposet that is a total order [90].

Definition 2.2.8 (Antichain)**.** Let $P = (P, \leq)$ be a poset. An antichain is a subset $A \subseteq P$ such that for all $x, y \in A$, neither $x \leq y$, nor $y \leq x$, so that all elements in the antichain are incomparable.

Definition 2.2.9 (Height of a Poset)**.** The height of a poset P is the length of the longest chain in P.

Definition 2.2.10 (Width of a Poset)**.** The width of a poset P is the size of the largest antichain.

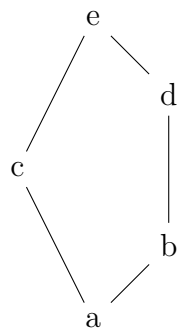


Figure 2.5: Non Graded Poset Example

Definition 2.2.11 (Maximal Chain)**.** A maximal chain in a poset is a chain that cannot be extended further by adding more elements while maintaining the comparability condition.

While some posets of interest are finite (definition 2.2.3), many important examples, both in CST and in combinatorics, are infinite but still satisfy a weaker condition known as local finiteness.

Definition 2.2.12 (Locally Finite)**.** A poset $(P, \leq)$ is locally finite if for all $x, y \in P$ the interval $[x, y]$ is finite where:

$$[x, y] = \{\, z \in P \mid x \leq z \leq y \,\} \tag{2.1}$$

Definition 2.2.13 (Poset Dual)**.** Let $(P, \leq)$ be a poset. The the dual poset, P^d, is the poset $P^d = (P, \leq_d)$ where for all $x, y \in P$, $x \leq_d y$ if and only if $y \leq x$.

2.3 Downsets

In many combinatorial and order-theoretic problems, it is useful to study subsets of a poset that are closed downward—meaning that if an element belongs to the subset, then all smaller elements related to it must also belong. Such subsets are called downsets (or order ideals) and play a fundamental role in analyzing the structure of posets. We begin with the formal definition of a downset in a poset.

Definition 2.3.1 (Downset)**.** Let $(P, \leq)$ be a poset. A *downset* (or order ideal) of P is a subset $D \subseteq P$ such that if $x \in D$ and $y \leq x$ in P then $y \in D$.

Dually, one can consider subsets that are closed upward, meaning that if an element is in the subset, then so are all greater elements related to it. These are called upsets (or order filters).

Definition 2.3.2 (Upset)**.** Let $(P, \leq)$ be a poset. An *upset* (or order filter) of P is a subset $U \subseteq P$ such that if $x \in U$ and $x \leq y$ in P then $y \in U$.

Consider the poset from Figure 2.3, given by $P = \{1, 2, 3, 4, 5, 6\}$ with the partial order as the divisibility relation $|$. The downsets of P of size 3 are $\{\,1, 2, 3\,\}$, $\{\,1, 2, 4\,\}$, $\{\,1, 2, 5\,\}$, and $\{\,1, 3, 5\,\}$.

It is also possible to consider the downsets and upsets generated by a set of elements on a poset. We say that the downset generated by A on a poset P is the subset $\wedge(A) = \{\, y \in P \mid \exists\, x \in A \; s.t. \; y \leq x \,\}$ [48]. In CST, we would say that $\wedge(A)$ is the past of A. The upset generated by A on a poset is the subset $\vee(A) = \{\, y \in P \mid \exists\, x \in A \; s.t. \; y \leq x \,\}$, and similarly in CST this would similarly be known as the future of A.

We will use downsets for a particular reconstruction problem, inspired by CST, which is a nice purely combinatorial question.

2.4 Order Dimension Two Case

Definition 2.4.1 (Intersection of Total Orders, ([116], chapter 4))**.** Let P be a set, and let $\leq_1, \leq_2$ be two total orders defined on P. The *intersection* of these total orders is the partial order $\leq$ on the set P defined such that $x \leq y$ if and only if $x \leq_1 y$ and $x \leq_2 y$.

For instance, the intersection of the total orders $(a \leq b \leq c \leq d \leq e)$ and $(e \leq d \leq c \leq b \leq a)$ is the antichain on the set $\{\, a, b, c, d, e \,\}$. The intersection between the total orders $(a \leq b \leq c \leq d \leq e)$ and $(a \leq b \leq c \leq e \leq d)$ is the poset given by the set $P = \{a, b, c, d, e\}$ and the order relations $a \leq b$, $b \leq c$, $c \leq d$ and $c \leq e$ as these are the relations that are consistent amongst both total orders.

Definition 2.4.2 (Extension)**.** Let $P = (X, \leq_P)$ be a poset. Then $Q = (X, \leq_Q)$ is an *extension* of P if for all $x, y \in X$, $x \leq_P y$ implies $x \leq_Q y$.

Definition 2.4.3 (Linear Extension)**.** Let $P = (P, \leq)$ be a poset. Q is a linear extension of P, if Q is an extension of P and Q is a total order.

Theorem 2.4.1 ([45], lemma 2.31)**.** *Let $(P, \leq)$ be a poset with $x, y \in P$ such that x and y are incomparable. Then there exists linear extensions $\leq_1$ and $\leq_2$ on the same underlying set P such that $x \leq_1 y$ and $y \leq_2 x$.*

Theorem 2.4.2. *Let $(P, \leq)$ be a poset. Then there is a collection of total orders on P whose intersection is $\leq$.*

The proof of this theorem can be sketched as follows, either all elements are comparable, or for any two elements that are not comparable, we use the pairs of total orders from theorem 2.4.1 to construct the poset, so that those elements would not be comparable in the intersection, and do it likewise for any pair of elements that are not comparable. For a complete proof of this theorem, see theorem 2.31 of [45].

Definition 2.4.4 (Order Dimension)**.** The *order dimension* of a poset is the smallest number of total orders whose intersection is the partial order. A family of total orders whose intersection corresponds to the partial ordering is called a *realizer* of the poset [45].

Order dimension has been widely studied in combinatorics and computer science, particularly in scheduling and sorting problems. In CST, a different notion of dimension is central, and the two notions happen to agree in order dimension two, which makes this case special [90].

A poset is said to be planar if its order diagram is planar. Schnyder characterized planar graphs in terms of order dimension and showed that the order dimension of a planar poset is at most three, a result known as Schnyder's theorem [108]. Felsner extended these results to higher-dimensional order structures, providing bounds on the order dimension of posets arising from geometric graphs [49]. As an example, the figure in **??** has order dimension two but is non-planar. The figure in figure 2.7 is planar, and has order dimension three, as shown in [117].

Definition 2.4.5 (Order-Preserving Function)**.** Let $P = (X_P, \leq_P)$ and $Q = (X_Q, \leq_Q)$ be two posets. We say that a map $f : X_P \to X_Q$ is order-preserving if for $x_1, x_2 \in X_P$ $x_1 \leq_P x_2$ implies that $f(x_1) \leq_Q f(x_2)$. [116]

Definition 2.4.6 (Order-Isomorphism)**.** Let $P = (X_P, \leq_P)$ and $Q = (X_Q, \leq_Q)$ be two posets. An order isomorphism from P to Q is a bijective function $f : X_P \to X_Q$ such that for all $x_1, x_2 \in P$, $x_1 \leq_P x_2$ if and only if $f(x_1) \leq_Q f(x_2)$. If such a function exists, we say that P and Q are order-isomorphic and write that $P \cong Q$.

For $\mathbb{R}^2$, we will use the product order, where for $x, y \in \mathbb{R}^2$, we will say $x \leq y \iff \pi_1(x) \leq \pi_1(y)$ and $\pi_2(x) \leq \pi_2(y)$, where π_i is the projection of the i^{th} coordinate.

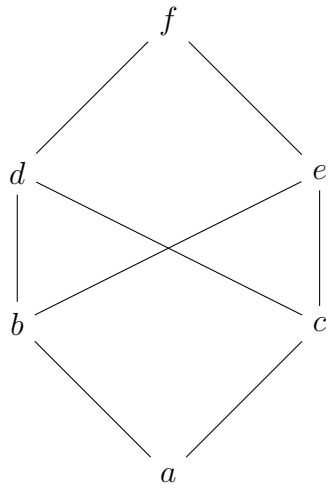


Figure 2.6: Example of Poset with Order Dimension Two That is Non-Planar

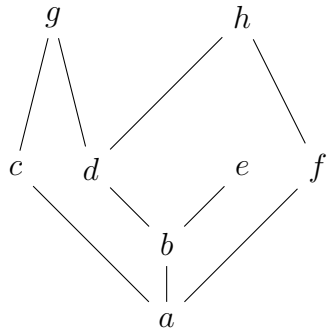


Figure 2.7: Example of Poset with Order Dimension Three That is Planar from [117]

Lemma 2.4.3. *A finite poset P has order dimension two, if and only if there exists an order-preserving function $f : P \to \mathbb{R}^2$, such that $f(P)$ is order-isomorphic to P.*

Proof. Let P be a poset. Suppose $f : P \to \mathbb{R}^2$ is an order-preserving function, such that $f(P)$ is order-isomorphic to P. Let $\leq_1$ and $\leq_2$ be the two total orders on P defined by comparing the first and second coordinates of f, respectively, these are both total orders since they are subposets of the set of real numbers. Since P is order-isomorphic to $f(P)$, it holds that the intersection of these two total orders is the partial ordering on P, which by definition makes P an order dimension two poset.

For the reverse direction, we assume that the order dimension of P is 2, and thus P is the intersection of two total orders $\leq_1$ and $\leq_2$. Define $f : P \to \mathbb{R}^2$ as follows, if $x \in P$ is the i^{th} smallest element under the $\leq_1$ order, and the j^{th} smallest element under the $\leq_2$ order, then let $f(x) = (i, j)$. Then this defines an order-preserving function into $\mathbb{R}^2$ whose image is order-isomorphic. □

Lemma 2.4.3 does extend to higher dimensions as well, that is, a finite poset P has order dimension k if and only if there exists an order-preserving function $f : P \to \mathbb{R}^k$, such that $f(P)$ is order-isomorphic to P, but that fact will not be used in this thesis [122].

2.5 Plane Posets

Double posets were first introduced in [82].

Definition 2.5.1 (Double Poset)**.** Let P be a set of elements, with two partial order relations $\leq_x$ and $\leq_y$ on the set P. Then $(P, \leq_x, \leq_y)$ is a double poset.

For a special subcase of double posets, we consider plane posets from [50].

Definition 2.5.2 (Plane Poset)**.** A *plane poset* $(P, \leq_h, \leq_r)$ is a set P with two partial order relations $\leq_h$ and $\leq_r$ such that for all $x, y \in P$, where $x \neq y$, then either x and y are comparable with respect to $\leq_h$ or, or they are comparable with respect to $\leq_r$, but not both [50].

This is different to our definition of planar poset, as shown in the examples above.

The *height order* $(\leq_h)$ is illustrated in the manner of the order diagram for $(P, \leq_h)$, with additionally the *rightward order* $(\leq_r)$ illustrated as a horizontal precedence, meaning that if $x \leq_r y$ then x appears to the left of y.

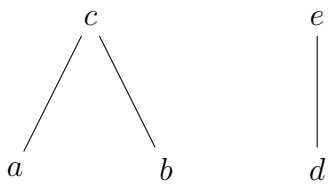


Figure 2.8: Plane Poset Example

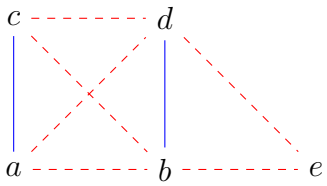


Figure 2.10: A Second Plane Poset Example

Within figure 2.8, it follows that $a \leq_r b \leq_r d$, $c \leq_r d$, $c \leq_r e$, and that $b \leq_r e$ based on spatial positioning. To give an explicit example. We temporarily use the following colouring scheme red and dashed lines for $\leq_r$ and blue for $\leq_h$, to draw our plane poset.

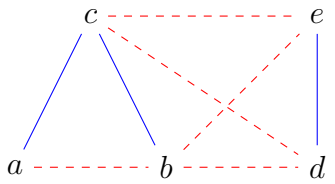


Figure 2.9: Poset extended to plane poset

As another example of a plane poset consider a set $P = \{a, b, c, d, e\}$ with the following relations that extend transitively as well:

- Height order: $(\leq_h)$: $a \leq_h c$ and $b \leq_h d$
- Rightward order: $(\leq_r)$: $a \leq_r b$, $a \leq_r d$, $b \leq_r e$, $c \leq_r b$, $c \leq_r d$, and $d \leq_r e$.

This is illustrated in figure 2.10.

Lemma 2.5.1. *Given two total orders $\leq_1$ and $\leq_2$ on the same underlying set of elements P, if we define $\leq_h$ to be the intersection of $\leq_1$ and $\leq_2$ and $\leq_r$ as the intersection of $\leq_1$ and the dual of $\leq_2$, then this gives a plane poset $(P, \leq_h, \leq_r)$.*

Proof. Suppose we are given two total orders $(P, \leq_1)$ and $(P, \leq_2)$ on the same underlying set P. For any two elements $a, b \in P$, if $a \leq_1 b$ and $a \leq_2 b$, then we will say they are comparable with the height order so that $a \leq_h b$. Otherwise, if $a \leq_1 b$ but $b \leq_2 a$ then we will say $a \leq_r b$. If neither condition holds then they will be comparable with the right order such that $b \leq_r a$, and hence any two elements will be comparable with respect to one of the ordering relations, and the properties necessary on the two partial order relations of a plane poset hold.

Lemma 2.5.2. *Given a plane poset $(P, \leq_h, \leq_r)$, we can construct two total orders $\leq_1$ and $\leq_2$ on the same underlying set of elements P, such that the construction in lemma 2.5.1 gives the plane poset $(P, \leq_h, \leq_r)$.*

Proof. Suppose we are given a plane poset $(P, \leq_h, \leq_r)$. Then we can construct two total orders $(P, \leq_1)$ and $(P, \leq_2)$ as follows. For any $a, b \in P$ such that $a \leq_h b$ we set $a \leq_1 b$ and $a \leq_2 b$. Similarly, if $a \leq_r b$ then we will set $a \leq_1 b$ in L_1 but set $b \leq_2 a$ in L_2. We must now prove that $\leq_1$ and $\leq_2$ are total orders.

For $a \neq b$, either a and b are comparable in $\leq_h$ or in $\leq_r$, but not in both. Since they are not comparable in both, this construction is well defined. Additionally, since they are comparable in at least one of the two order relations, it follows that a and b will be comparable in $\leq_1$, and thus the construction we have given creates a strongly connected total order for $\leq_1$. The same argument holds for $\leq_2$.

It holds trivially that each element is comparable to itself in $\leq_1$ and $\leq_2$, satisfying reflexivity. We now assume that we have an element such that $a \leq_1 b$ and $b \leq_1 a$. If $a \neq b$ then a and b are comparable with exactly one of $\leq_h$ and $\leq_r$. So for it to hold that $a \leq_1 b$ and $b \leq_1 a$, then either $a \leq_h b$ and $b \leq_h a$, or $a \leq_r b$ and $b \leq_r a$. Hence by anti-symmetry $a = b$ after all. A similar argument holds for $\leq_2$.

We wish to show that our construction satisfies transitivity. Suppose we have $a \leq_1 b$ and $b \leq_1 c$. Then either $a \leq_h b$ or $a \leq_r b$ and similarly either $b \leq_h c$ or $b \leq_r c$. If both are comparable with the same order in the plane poset, then transitivity immediately follows from the requirements of a partial order on our plane poset. If they are not both comparable with the same order, consider the case where $a \leq_h b$, and $b \leq_r c$. If $c \leq_h a$, then by transitivity of $\leq_h$, $c \leq_h b$ which is a contradiction. If $c \leq_r a$, then by transitivity of $\leq_r$, $b \leq_r a$, which is a contradiction. Since C is comparable in a in one of the orders, we must have $a \leq_h c$, or $a \leq_r c$ and then it holds that $a \leq_1 c$. Now we consider the case where $a \leq_r b$, and $b \leq_h c$. If $c \leq_h a$, then $b \leq_h a$, which is a contradiction. Similarly if $c \leq_r a$, then $c \leq_r b$, which is also a contradiction. So either $a \leq_h c$ or $a \leq_r c$, and then it holds that $a \leq_1 c$. A similar argument holds for $\leq_2$.

By construction, these two total orders $\leq_1$ and $\leq_2$ will return the given plane poset by lemma 2.5.1. □

Theorem 2.5.3. *There is a bijection between the set of plane posets, and the set of posets that have order dimension less than or equal to two.*

Proof. Immediate from lemma 2.5.1 and lemma 2.5.2.

Definition 2.5.3 (Downset of a Plane Poset)**.** A *downset* of a plane poset is a subset D of a plane poset $(P, \leq_h, \leq_r)$ is a downset of the $\leq_h$ order.

Note that D itself in the above definition is itself a plane poset. For an example consider the plane poset in figure 2.9, the downsets of size 3 are given by $\{a, b, c\}, \{a, b, d\}, \{a, d, e\}, \{b, d, e\}$. We will make more use of these downsets in section 4.2

Remark 2.5.4. *For a set of elements P, any order preserving-function $f : P \to \mathbb{R}^2$, such that $f(P)$ is order-isomorphic to P can be used to plot the plane poset, in a way that is helpful and can show its structure.*

A takeaway here is that this construction that we have shown, where you can take two total orders on the same underlying set of elements, makes it very simple to generate plane posets. It is possible to generate two random lists, and use their intersection to obtain the height and rightward order as shown in lemma 2.5.1. The process of generating a plane poset by selecting two total orders uniformly at random and applying the construction in lemma 2.5.1 is guaranteed to produce a valid plane poset. However, this mapping is not injective: some plane posets arise from many different total order pairs, while others arise from fewer. As a result, the induced distribution on the set of plane posets is non-uniform, and sampling in this way does not produce plane posets with equal probability. Uniform generation of plane posets would require a different sampling method that accounts for the size of each preimage.

In the next chapter, we will see the key role that posets play in causal set theory.

Chapter 3

Causal Set Theory Background

3.1 Motivation

One of the central challenges in modern theoretical physics is the development of a framework for quantum gravity that unifies general relativity (GR) and quantum field theory (QFT). Such a theory is expected to describe physical phenomena where both gravitational and quantum effects are significant, including black holes and the early universe [103]. GR models spacetime as a smooth, four-dimensional Lorentzian manifold, where gravity arises from the curvature of geometry. In contrast, quantum mechanics and QFT describe the fundamental forces and particles in terms of quantized fields. Tensions between the two frameworks become apparent in extreme regimes: GR predicts singularities where curvature diverges, yet quantum effects are expected to modify this behaviour; meanwhile, applying QFT techniques to gravity naively leads to a non-renormalizable theory, producing divergences that cannot be resolved within the standard perturbative framework [92, 104, 103]. As a result, there is motivation to find a quantum theory of gravity.

Traditional approaches to unifying GR and QFT, such as string theory and loop quantum gravity, provide valuable insights but also face significant unresolved challenges. String theory, for instance, requires additional spatial dimensions and remains empirically unverified, while loop quantum gravity encounters difficulties in recovering smooth classical spacetime at large scales [92, 104]. Despite these issues, both frameworks have contributed important conceptual tools and motivated deeper questions about the nature of space, time, and quantization.

Causal set theory (CST), begins from a minimal set of physically motivated axioms: discreteness, and a partial order reflecting causal structure. These axioms directly encode

the causal relationships between events, and they do so without requiring any geometric or topological background. Even if CST is ultimately not the full answer to quantum gravity, its conceptual simplicity and physical grounding suggest that it captures a foundational layer of reality. The poset structure alone, stripped of continuous geometry, turns out to be remarkably rich: it is sufficient to encode volume, causal relations, and even topological and geometric approximations in suitable limits. While it remains an open question whether this information is enough to reconstruct all of spacetime, the causal set hypothesis provides a compelling demonstration of how much can be obtained from simple, fundamentally discrete assumptions.

3.2 Foundation

In general relativity, spacetime is modelled as a smooth, four-dimensional Lorentzian manifold, typically denoted by (M, g) where M is the underlying differentiable manifold and g g is a Lorentzian metric tensor defined on M. The manifold M encodes the topological and differentiable structure of spacetime, while the metric g determines both geometric distances and causal relationships between events [103]. While this continuous model has been remarkably successful at macroscopic scales, it poses significant challenges when one attempts to quantize it. Of interest to us, is the line of investigation in quantum gravity that seeks to replace the continuous manifold with a fundamentally discrete structure, from which continuum properties can be recovered in appropriate limits.

Returning to the continuous case for motivation, among the earliest proposals to use causality as the central organizing principle of spacetime were those of Lorentz, Weyl, and Robb, who suggested that causal order might be more fundamental than the metric itself [113]. This idea gained strong mathematical footing in the 1970s through the work of Hawking, King, McCarthy, and Malament. Their results, now known collectively as the HKMM theorem, are simply stated as "Causal Structure + Volume Element = Lorentzian Geometry" [113]. This forms the basis for the well-known CST slogan: "Order + Number $\sim$ Lorentzian Geometry."

CST postulates that spacetime is a discrete set of events, partially ordered by causality. CST can thus be described by the following proposal due to Bombelli, Lee, Meyer, and Sorkin [113, 19]:

- Quantum gravity is a quantum theory of locally finite posets, which are called causal sets (causets) in this context.

- A continuum spacetime (M, g) is an approximation of an underlying causet C such that $C \sim (M, g)$ where order $\sim$ causal order and number $\sim$ spacetime volume.

3.2.1 The Beginning of Causal Set Theory

The causet approach states that spacetime at the deepest level is a finite or countable set of elementary events equipped with a partial order that encodes their causal relationships. In this way we have that a causet is an underlying set C with a partial order relation $\leq$. In standard CST literature, many use the $\prec$ to denote the strict partial ordering. In CST, there is one more requirement on the causet, *locally finiteness* (definition 2.2.12) [113, 43].

An important question in CST concerns how well a causet can approximate a continuum manifold. Consider a causet C and a Lorentzian manifold (M, g). Then an embedding $\Phi : C \hookrightarrow (M, g)$ is a map such that C is order-isomorphic with its image, that is an injective map such $x \leq_C y \iff \Phi(x) \leq_M \Phi(y)$ where $\leq_C$ and $\leq_M$ are the ordering relations on C and M respectively, for $x, y \in C$. Furthermore Φ is faithful at density ρ if $\Phi(C)$ is uniformly distributed in (M, g) with density ρ, see equation 7 of [113]. This then leads this leads to a central open question in CST, sometimes referred to as the hauptvermutung.

Hauptvermutung 3.2.1. *A causet C can be faithfully embedded at density ρ into two distinct spacetimes if and only if they are approximately isometric. [113]*

There is not a precise definition of "approximately isometric" that is suitable in all contexts, but the central motivation is that faithful embeddings should be rare and geometrically constrained. The hauptvermutung thus expresses the idea that the causal and volumetric structure encoded in the causet should, in principle, determine the continuum geometry up to small deformations. The hauptvermutung thus lies at the heart of whether CST can reproduce continuum spacetimes uniquely in an appropriate limit. [113].

3.2.2 General Covariance

One challenge in CST is formulating dynamics in a way that is independent of relabelling (which in CST is called general covariance, the principle that the laws of physics should be independent of the labelling). In the continuum, general covariance is a foundational feature of general relativity (GR), ensuring that physical predictions do not depend on arbitrary coordinate choices. For discrete spacetimes, where the continuum manifold is replaced by a locally finite partially ordered set, reproducing this principle is nontrivial.

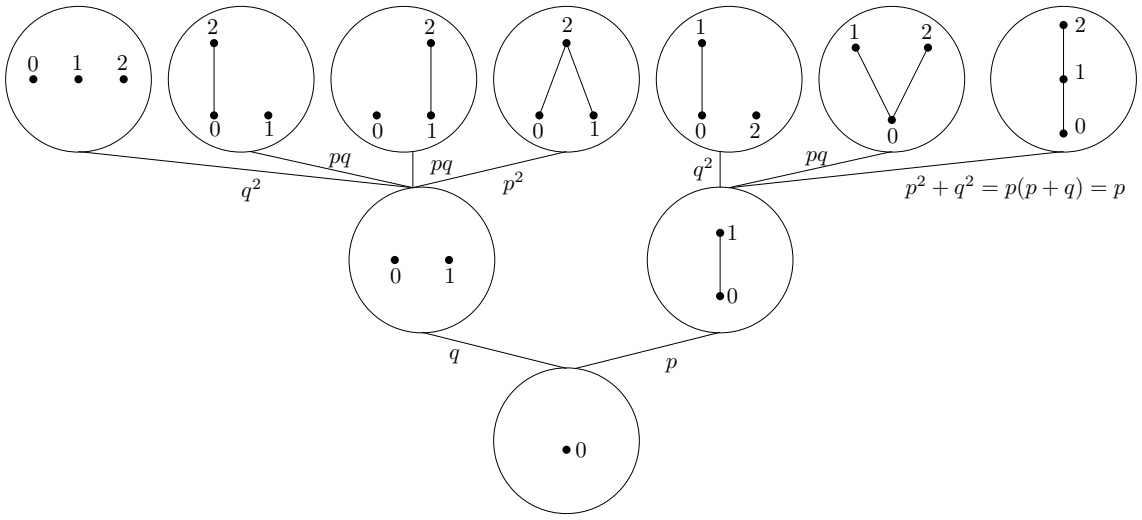


Figure 3.1: The First Three Levels

Broadly speaking, there are two modes of analysis in CST. The first is kinematical: one considers the structure and properties of causal sets generated via a random sprinkling into a continuum spacetime. This approach does not define a dynamics, but rather focuses on the statistical or geometric characterization of causets and their continuum approximations. The second is dynamical: causal sets are constructed incrementally via growth models or other rule-based evolution processes. These models aim to capture an intrinsic, discrete notion of time evolution that respects causal structure.

In the kinematical approach, one common method is to generate a causal set by performing a Poisson sprinkling of points into a fixed Lorentzian manifold. Here, points are selected randomly with uniform density ρ, and the causal relations among them are inherited from the background spacetime. This process is approximately covariant in the sense that the probability distribution is invariant under Lorentz transformations, and the resulting causal set inherits many properties of the continuum spacetime [19, 18]. However, because it relies on embedding the causal set into a pre-existing spacetime, the sprinkling procedure is not suitable as a fundamental dynamical law.

To model intrinsic, background-independent dynamics, one turns to growth models, where elements of the causal set are added one at a time according to a stochastic rule. An early example is transitive percolation, in which each new element selects predecessors independently with fixed probability, and the causal set is formed by taking the transitive closure [18]. For instance, suppose we begin with an empty causet and add elements one

by one. The first element is added with no predecessors, as seen in figure 3.1. The second element selects the first as a predecessor with probability p otherwise, it is unrelated with probability $q = 1 - p$. The third element independently decides whether to link to each of the two existing elements, and so on. After all links are chosen, any implied causal relations are added (for example, if $a \leq b$ and $b \leq c$, then $a \leq c$ are added via transitive closure). One interesting case is the last case on the third level of figure 3.1, where one can see that the probability simplifies to p. If the element with label 2 was to be related to 1, or both 0 and 2, this would occur with probability $pq + p^2 = p(q + p) = p$. Thinking about this process a different way, this model can be rephrased as random graph orders [51]

A generalization are the classical sequential growth (CSG) models introduced by Rideout and Sorkin [101]. CSG models are particularly nice because they are exactly the set of growth models satisfying a list of physical axioms, and they include transitive percolation. Like transitive percolation, all of these growth models build a tree of possibilities, but the nodes of these trees remain labelled posets, and thus each step is label-dependent even though the completed structure is not. Thus, while observables are covariant, the underlying dynamics are not manifestly so [113].

One way to move to label independence (i.e. to move to unlabelled posets) would be to simply take the structure of our labelled growth model up to isomorphism. This will give the poset of posets, ordered by inclusion as a downset. In order to obtain the general covariance, we lose the tree structure.

To address this, the covariant tree (covtree) formalism was introduced by Dowker, Imambaccus, Owens, Sorkin, and Zalel [42]. Covtree reformulates causal set growth in a way that is manifestly covariant from the outset, while remaining a tree. Instead of treating each node as a labelled causal set, it defines each node as a covariant event, an equivalence class of labelled causal sets under relabelling. Growth then proceeds through a branching tree in which each node corresponds to a set of unlabelled posets, and each path through the tree represents a covariant history of causet evolution. This framework ensures that all intermediate stages of growth are covariant. As such, covtree provides a clean, fully covariant foundation for describing dynamics in CST [126]. The first few levels of covtree are illustrated in figure 3.2.

More precisely, a node of covtree is a set of unlabelled posets, which is exactly the set of downsets of size n up to isomorphism, for some n, of some poset P. One node of covtree $\{ Q_1, \ldots, Q_k \}$ is the parent of another if $\{ Q_1, \ldots, Q_k \}$ is exactly the set of $\{ R_1, \ldots, R_\ell \}$ downsets of size $n-1$ of the R_i where $n = |R_1| = |R_2| = \cdots = |R_\ell|$. Note that in figure 3.2, children are drawn above their parents.

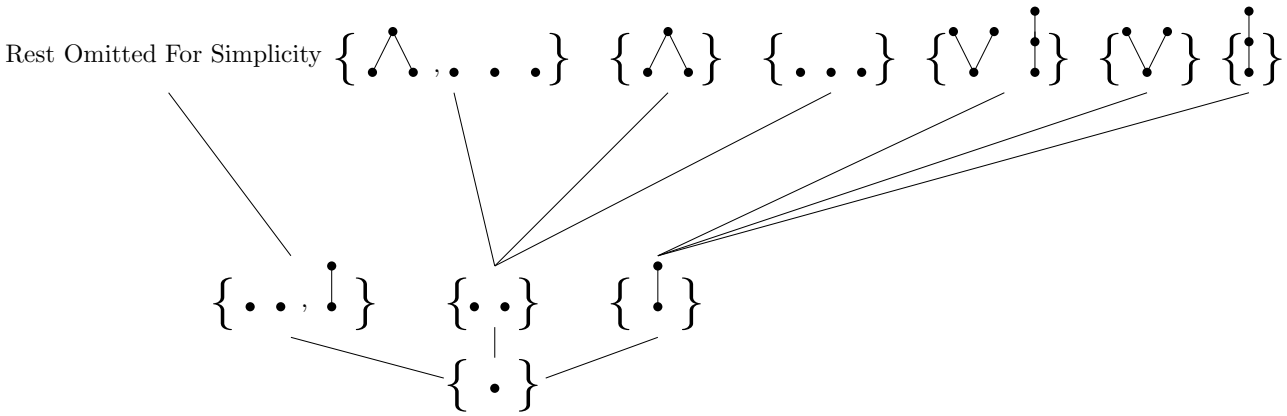


Figure 3.2: Structure of the first 3 levels of the covtree, adapted from [42] . For space purposes, only the right two of the second layer have their third layer illustrated.

Figure 3.3: Example of a Set That Cannot be a Node of Covtree

3.3 Witness Reconstruction

An important challenge of covtree, is that it is not obvious when a set of posets is a node of covtree, that is when the set is exactly the set of downsets of size n, up to isomorphism, for some n of some poset P. As an example, consider . It does not satisfy being a node of covtree because it is missing the case where two nodes are connected by a line, followed by a node, as shown in . Thus, given a set of downsets of size n, we ask whether there exists a poset whose set of downsets of size n matches this set, and hence witnesses that the original set is a node in covtree (we call this larger poset a witness).

This witness reconstruction problem will bear on the question of how much of a causet's global structure can be recovered from its local features. Being able to identify witnesses

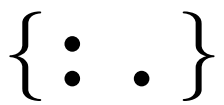


Figure 3.4: Illustration of Missing case

from a set of downsets would further our understanding of covtree, and shed light on the discrete dynamics by which causal sets evolve under covariant growth models. In the next chapter, we return to downsets in the special case of two-dimensional plane posets, which may serve as a simplified model for studying witness reconstruction in the dimension two case.

Chapter 4

The Downset Graph

4.1 Introduction

This chapter studies a graph that encodes certain relationships among a set of downsets of a fixed size. Starting with the set of all downsets of size n of a fixed poset, we prove that this graph will be connected. Additionally, we will expand this to the plane poset case, and show that the corresponding digraph notion will be acyclic, and prove a local structural property of it.

4.2 Comparing Downsets

The fundamental concepts of posets, downsets, plane posets, and total orders were introduced in chapter 2. In this section, we will focus on establishing a notion of comparison of downsets for general posets. The next section will describe the process of comparing downsets from a given collection of downsets of the same size on a poset. For this section we will assume that our posets are finite.

Definition 4.2.1 (Set of Downsets of Size n)**.** Given a poset $(P, \leq)$, we define $\mathcal{D}_n(P)$ to be the set of all downsets of P of size n.

As an example, consider the poset in figure 4.1. Then $D_3(P) = \{\{a, b, c\}, \{a, b, d\}, \{a, d, e\}, \{a, d, f\}, \{d, e, f\}, \{d, e, g\}\}$. In our case, we are interested in pairs of downsets that differ by exactly one element. Specifically, downsets A and B differ by exactly one

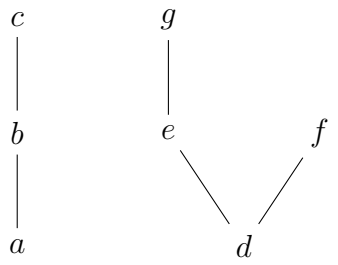


Figure 4.1: Poset with Seven Elements

element if there exists a set X, such that $A = X \cup \{ a \}$ and $B = X \cup \{ b \}$, for some a, b in A and B respectively. As an example, consider the downsets $\{ a, b, c \}$ and $\{ a, b, d \}$ in the example above. We will introduce a way to connect such elements with a graph structure.

Lemma 4.2.1. *Let $(P, \leq)$ be a poset. If two downsets $A, B \in \mathcal{D}_n(P)$ differ by exactly one element, then the element they differ by is a maximal element in both of their respective downsets.*

Proof. Suppose that $A, B \in \mathcal{D}_n(P)$ for a given poset $(P, \leq)$ are two downsets which differ by exactly one element. Thus we can write $A = X \cup \{ a \}$, and $B = X \cup \{ b \}$, for some subset $X \subset P$ and distinct elements $a, b \in P$. To show that a and b are maximal elements in A and B respectively, we will assume for sake of contradiction that a or b is not a maximal element in its respective downset. Without loss of generality, we assume that a is not a maximal element in A and that there is some $w \in A$ such that $a \neq w$ and $a \leq w$. However this raises a contradiction as $w \in X$, and thus $w \in B$, and as $a \notin B$, B cannot be a valid downset. Thus, a and b are maximal elements in their respective downsets.

□

Definition 4.2.2 (Graph of Set of Downsets of Size n)**.** The graph of downsets of size n, $G_n(P)$, of a poset $(P, \leq)$ is a graph $G_n = (V, E)$ where $V = \mathcal{D}_n(P)$, and for $X, Y \in V$ there is an edge from X to Y if X and Y are downsets that differ by exactly one element.

This graph is a kind of exchange graph. Consider the poset in figure 4.2. Then we have that the downsets of size 3 are given by: $D_1 = \{a, b, c\}, D_2 = \{a, b, d\}$ and $D_3 = \{b, d, e\}$, and $G_n(P)$ is the graph in figure 4.3.

Theorem 4.2.2 (Connected Graph Theorem)**.** *For any given poset $(P, \leq)$ the graph $G_n(P)$ is connected for all $n \in \mathbb{N}$.*

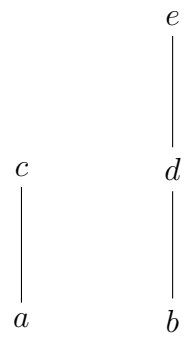


Figure 4.2: Poset Example

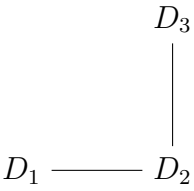


Figure 4.3: Graph Example

Proof. Let $(P, \leq)$ be a poset, and fix $n \in \mathbb{N}$. Let $G_n(P)$ be as in definition 4.2.2. Let A, B be any two downsets in $\mathcal{D}_n(P)$. We wish to prove that there exists an A, B path in $G_n(P)$ by the principal of mathematical induction.

Consider $|A \setminus (A \cap B)|$. If $|A \setminus (A \cap B)| = 1$, then A and B differ by exactly one element, and thus they are adjacent in $G_n(P)$. We now assume for our inductive hypothesis that there exists an A, B-path for any $A, B \in \mathcal{D}_n(P)$ such that $|A \setminus (A \cap B)| < k$, for $k > 1$, $k \in \mathbb{N}$. Let A, B be two downsets such that $|A \setminus (A \cap B)| = k$.

Let $X = A \cap B$. Then we can consider $A = X \cup \{ a_1, \ldots, a_k \}$ and $B = X \cup \{ b_1, \ldots, b_k \}$. The elements $a_1, \ldots a_k$ and $b_1, \ldots, b_k$ must all be distinct. Without loss of generality, let a_k be a maximal element in $\{ a_1, \ldots a_k \}$, hence also maximal in A. Without loss of generality, let b_k be a minimal element in $\{ b_1, \ldots, b_k \}$, hence $X \cup \{ b_k \}$ is also a downset. Let $X' = X \cup \{ b_k \}$, and let $A' = X' \cup \{ a_1, \ldots, a_{k-1} \}$. Then A' is a downset of P because we removed a maximal element of $\{ a_1, \ldots, a_k \}$, and A' is adjacent to A. Then $|A' \setminus (A' \cap B)| = k-1$, and hence there is an A', B path in $G_n(P)$, as $B = X' \cup \{ b_1, \ldots, b_{k-1} \}$. Thus there exist an A, B path as well, satisfying that $G_n(P)$ is connected. □

This proof not only gives connectivity but also a bound (in terms of k) on the length of the path between any two vertices. This will be pursued further in future work.

4.3 Comparing Downsets of Plane Posets

Order dimension two is useful for CST, because as noted before, it where order dimension agrees agrees with the other dimension notion that is used in CST, even though those two notions differ for higher dimensions. Additionally, order dimension two is nice as it is where we have our plane poset formulation.

Remark 4.3.1. *Note that our definition of $\mathcal{D}_n(P)$ does not change for plane posets. If we had a poset $(P, \leq)$ and a plane poset $(P, \leq_h, \leq_r)$, on the same underlying set P where $\leq$ was the same partial ordering as $\leq_h$, then they would share the same set of downsets of a given size, $\mathcal{D}_n(P)$, for all n, but all the downsets inherit a plane structure themselves.*

Lemma 4.3.2. *Let $(P, \leq_h, \leq_r)$ be a plane poset of size n. If two downsets $X, Y \in \mathcal{D}_n(P)$ for some $n \in \mathbb{N}$ differ by exactly one element, then those elements must be comparable with respect to the rightwards order.*

Proof. Suppose for sake of contradiction that two downsets $X, Y \in \mathcal{D}_n(P)$ of our plane poset differ by one element and are comparable by the height order. Let $x, y \in P$ be the two maximal elements with respect to their downsets is $x \in X$, $y \in Y$ that our downsets differ by. Without loss of generality we have that $x \leq_h y$. Then in this case Y is not a valid downset. Thus we have reached a contradiction and the elements that the downsets differ by must be comparable with the rightward order. □

We consider the downsets of figure 4.1. We can extend the poset to be a plane poset in this case, using the rightward ordering given in the figure. For example, the downsets $\{a, b, c\}$ and $\{a, b, d\}$ each differ by exactly one element, and $c \leq_r d$. We can extend the notion of $G_n(P)$ to create a digraph, by taking the edge that would correspond to the two downsets above, and creating a directed edge that would point towards the latter downset.

Definition 4.3.1 (Digraph of Set of Downsets of Size n)**.** The digraph of downsets of size n, $\overrightarrow{G_n}(P)$, of a plane poset $(P, \leq_h, \leq_r)$ is a digraph $\overrightarrow{G_n} = (V, E)$ where $V = \mathcal{D}_n(P)$, and for $X, Y \in V$ there is a directed edge from X to Y if X and Y are differ by one element $x \in X$ and $y \in Y$ such that $x \leq_r y$, which we denote as $X \to Y$.

As an example, consider the poset from example figure 4.2, which we extend to be a plane poset, using a rightward ordering in figure figure 4.4.

Then we have that the downsets of size 3 are still given by: $D_1 = \{a, b, c\}, D_2 = \{a, b, d\}$ and $D_3 = \{b, d, e\}$, and $\overrightarrow{G_n}(P)$ is the digraph in figure 4.5.

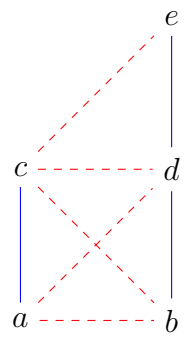


Figure 4.4: Plane Poset Example

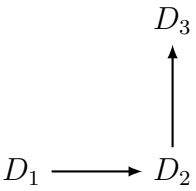


Figure 4.5: Digraph Example

Theorem 4.3.3 (Digraph Theorem)**.** *For any given plane poset* $(P, \leq_h, \leq_r)$ *the digraph* $\overrightarrow{G_n}(P)$ *is acyclic for all* $n \in \mathbb{N}$.

Proof. For a plane poset $(P, \leq_h, \leq_r)$, suppose that for sake of contradiction that the digraph $\overrightarrow{G_n}(P)$ contains a directed cycle. Then there exists a sequence of downsets $D_1, D_2, \ldots, D_\ell \in \mathcal{D}_n(P)$ such that $D_1 \to D_2 \to \cdots \to D_\ell \to D_1$ is a directed cycle in $\overrightarrow{G_n}(P)$. Each directed edge $D_i \to D_{i+1}$ corresponds to a replacement of a maximal element $b_i \in D_i$ with an element $a_i \notin D_i$ such that $a_i \leq_r b_i$, and $D_{i+1} = (D_i \setminus \{ b_i \}) \cup \{ a_i \}$. Since the cycle is a sequence that must return to D_1, the set of all elements removed, and the set of all elements added must be equal, and thus $\{ a_1, \ldots, a_\ell \} = \{ b_1, \ldots, b_\ell \}$.

Consider the set P and the rightward order $\leq_r$. Let $i_1 = 1$; then $a_{i_1} \leq_r b_{i_1}$ and b_{i_1} is equal to some a_{i_2}; then $a_{i_2} \leq_r b_{i_2}$ and b_{i_2} is equal to some a_{i_3} and so on: $a_{i_1} \leq_r b_{i_1} = a_{i_2} \leq_r b_{i_2} = \cdots = a_{i_k} \leq_r b_{i_k} = a_{i_1}$, where i_k is the index in the sequence at which we add element a_1 back to the downset in consideration. However, we have created a cycle in a partial ordering, which is not possible. Thus the graph $\overrightarrow{G_n}(P)$ is acyclic.

We conclude this chapter with a structural result on $\overrightarrow{G_n}(P)$. Whenever there is branching, it can always be bridged by at most one other element.

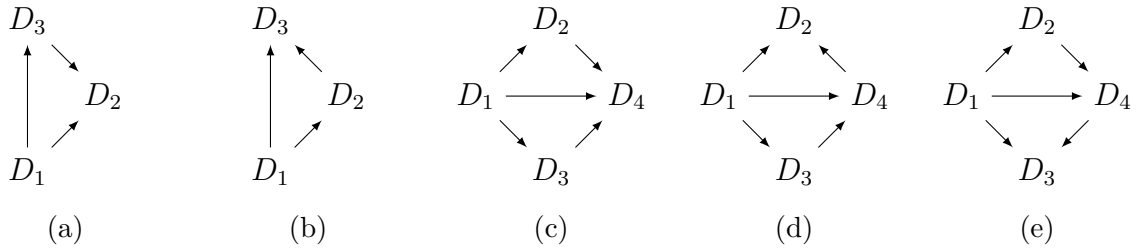


Figure 4.6: All possible local configurations for a branching point in the digraph $\overrightarrow{G_n}(P)$.

Theorem 4.3.4 (Main Theorem)**.** *For a plane poset $(P, \leq_h, \leq_r)$ if there exists downsets $D_1, D_2, D_3 \in \mathcal{D}_n(P)$ such that $D_1 \to D_2$ and $D_1 \to D_3$ in $\overrightarrow{G_n}(P)$, then at least one of the following cases will hold true, where for the latter three cases there exist downsets $D_4 \in \mathcal{D}_n(p)$.*

Proof. Suppose that we have a plane poset $(P, \leq_h, \leq_r)$ with $A, B, C \in \mathcal{D}_n(P)$ for some $n \in \mathbb{N}$, such that $A \to B$ and $A \to C$. Then it holds that $B = (A \cup \{\, b \,\}) \setminus \{\, a_1 \,\}$ for $a_1 \in A$, $b \in B$, and $a_1 \neq b$ such that $a_1 \leq_r b$. Similarly, $C = (A \cup \{\, c \,\}) \setminus \{\, a_2 \,\}$ for $a_2 \in A$, $b \in B$ where $a_2 \neq c$ such that $a_2 \leq_r c$. Note that a_1 and a_2 must be comparable with the rightwards order because they are both maximal elements of A.

If a_1 and a_2 are the same element, then we would have that B and C are also adjacent in $G_n(P)$, and would give either case (a) or (b) of figure 4.6. On the other hand, if b and c were the same element then $B = (C \cup \{\, c \,\}) \setminus \{\, b \,\}$, so B and C would be adjacent in $\overrightarrow{G_n}(P)$ again giving (a) or (b) of figure 4.6. Hence, we assume that a_1, a_2, b and c are all distinct elements in the poset P for the remainder of the proof.

Without loss of generality, we assume that $a_1 \leq_r a_2$. Then by transitivity we have that $a_1 \leq_r a_2 \leq_r c$, and $a_1 \leq_r b$. Suppose $c \leq_r b$, then by transitivity $a_2 \leq_r b$ as well. Consider the downset $D = (D_1 \cup \{\, b \,\}) \setminus \{\, a_2 \,\} = (B \cup \{\, a_1 \,\}) \setminus \{\, a_2 \,\} = (C \cup \{\, b \,\}) \setminus \{\, c \,\}$ of P. This is a downset as we already knew $A \cup \{\, b \,\}$ is a downset, and the rest of the elements are comparable with the rightward order. Thus we have that $A \to D$, $B \to D$, and $D \to C$ giving case (d) of figure 4.6.

Now suppose $b \leq_r c$, then consider the downset $D = (A \cup \{\, c \,\}) \setminus \{\, a_1 \,\} = (B \cup \{\, b \,\}) \setminus \{\, c \,\} = (C \cup \{\, a_1 \,\}) \setminus \{\, a_2 \,\}$, which is a downset because $a_1 \leq_r c$, and we already know the rest of the elements satisfy being a downset. In this case, $a_1 \leq_r c$ by transitivity, so

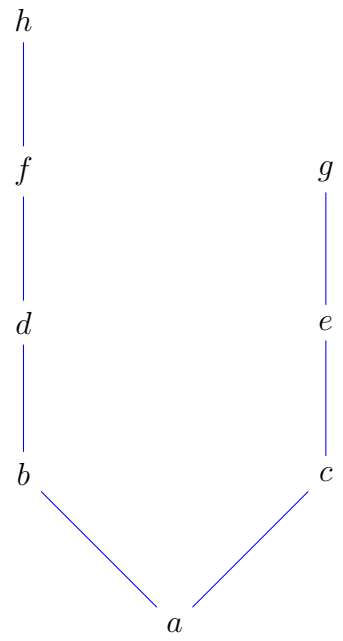


Figure 4.8: Poset example where the rightward order is rightward positioning

$A \to D$, $B \to D$ and $C \to D$, so we end up in case (c) of figure 4.6.

Note that when $a_2 \leq_r a_1$, the argument is analogous with case (d) is replaced by case (e) of figure 4.6, which concludes this result. □

From figure 2.8, we recall the four downsets of our plane poset which we will denote $D_1 = \{ a, b, c \}, D_2 = \{ a, b, d \}, C = \{ a, d, e \}$ and $D = \{ b, d, e \}$. We wish to construct a digraph G_3, where $\mathcal{D}_3(P) = \{ D_1, D_2, D_3, D_4 \}$. Then we have the following digraph:

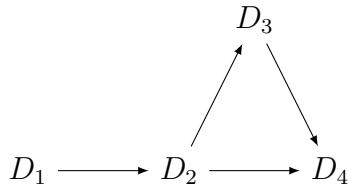


Figure 4.7: Graph of Downsets

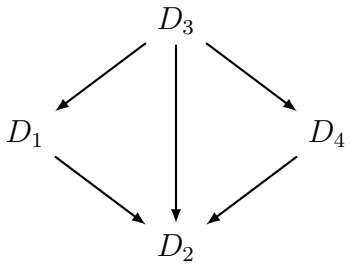


Figure 4.9: Digraph Example

Then the downsets of size 3 are $D_1 = \{4, 2, 3\}, D_2 = \{4, 2, 1\}, D_3 = \{4, 7, 2\}$ and $D_4 = \{4, 7, 1\}$, and $\overrightarrow{G_3}(P)$ is shown in figure 4.9.

4.4 Witness Reconstruction

One natural question in causal set theory that was touched on earlier was the concept of recognizing nodes of covtree. Using notation from this chapter, understanding whether $\mathcal{D}_n(P)$ is a node of covtree would be of interest. Furthermore, another question is how much of the covtree structure can be understood or recovered from local order-theoretic information. In particular, given a set of downsets that might arise from a covariant growth process, can we determine whether they correspond to a valid path through the covtree? And if so, can we reconstruct a causal set that realizes them?

Order dimension two posets provide a natural setting in which to explore this question. These are posets that can be expressed as the intersection of two linear orders, and they form a tractable subclass that still captures nontrivial causal structure. In CST, such posets arise in the study of $1+1$-dimensional spacetimes. The constraint of dimension two reduces combinatorial complexity while preserving enough structure to model meaningful causal behaviour. This makes them an ideal setting for testing ideas about witness reconstruction and covtree navigation.

We hope to use $G_n(P)$ and $\overrightarrow{G_n}(P)$ to better understand the structure of witnesses to the property of being a node in covtree. We hope that this will be useful for proving properties of witnesses, such as their size, and that this will help with the algorithmic concerns in the order dimension two case. For example, consider the set of downsets in figure 4.10.

In this case, the smallest poset that satisfies the above set of downsets of size 5 is shown in figure 4.11. This is a surprising size, given that the expected value in this case based on

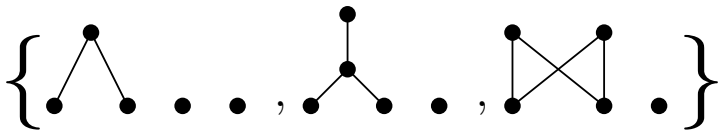


Figure 4.10: Set of Downsets of Size 6

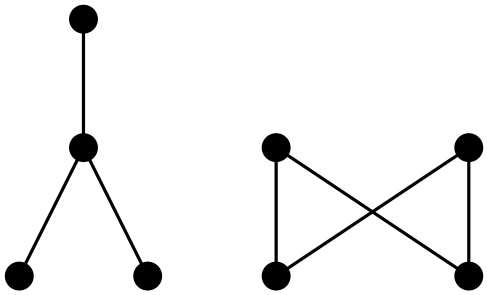


Figure 4.11: Witness for figure 4.10

the number of elements in the set and the size of the downsets would suggest the smallest poset would be of size 7, but the smallest witness has size 8.

This example illustrates that even small collections of downsets can encode non-obvious structure, and recovering a corresponding witness requires careful analysis.

Part 2

Machine Learning

Chapter 5

Introduction to Feynman Periods

This chapter goes over the background needed to understand Feynman periods from a combinatorial point of view. We start with some graph structures that appear in QFT, including 4-regular graphs, loop orders, and the Kirchhoff polynomial. Then we explain how Feynman periods are defined for primitive graphs in ϕ^4 theory. The chapter then proceeds to include a section on symmetries of the Feynman period, and provides a brief motivation for using machine learning to predict Feynman periods. We also go over some features that are useful for studying or estimating periods, such as the Hepp bound, edge cuts, and number of graph automorphisms.

5.1 Graph Polynomials

This chapter will follow [123]. In this section, we introduce several classes of graphs that will be central to our discussion of Feynman periods. These graphs arise naturally in quantum field theory, but we begin here with a purely combinatorial viewpoint. For this thesis, graphs will be finite, undirected, and connected, but can have self-loops and multi-edges. One standard way to set up graphs for Feynman graphs is with half edges, as shown in [123].

Definition 5.1.1 (Graphs via Half-Edges)**.** A graph G is defined by a finite set of half-edges H, together with:

- a set partition of H whose blocks are called *vertices*, denoted $V(G)$,
- a set partition of H of blocks of size less than or equal to two:

– the blocks of size two are *internal edges*, the set of which is denoted $E(G)$,
– the blocks of size one are called *external edges*.

Each half-edge belongs to exactly one vertex and is either paired (forming an internal edge) or unpaired (forming an external edge). In this setup, a vertex is said to be *incident* to an edge if it contains one of the edge's half-edges. Two vertices are said to be *adjacent* if they share an internal edge. This definition generalizes the classical notion of a graph by encoding incidence information at the level of half-edges. In standard graph theory language, each internal edge connects two vertices, and each external edge is a dangling leg attached to a single vertex. For this chapter, we will use usual graph theory language whenever obvious. To see an example of a half-edge graph, consider figure 5.1. While it can be convenient to choose an arbitrary edge and vertex ordering, as well as an orientation for the edges, these choices are not part of the core definitions used in this work.

5.1.1 Valency and 4-Regular Graphs

A vertex $v \in V(G)$ has degree (or valency) $d(v)$ equal to the number of edges incident to it. In the half-edge graph case, the degree of a vertex is the size of the part that defines that vertex. A graph is said to be k-regular if $d(v) = k$ for all vertices in the graph. This is not the standard graph theory sense of a k-regular graph, due to the external edges. 4-regular graphs are the graphs that appear in ϕ^4-theory, a scalar field theory, where interaction vertices involve exactly four scalar fields.

Definition 5.1.2 (Completion and Decompletion)**.** Let K be a 4-regular graph, with no external edges (so a 4-regular graph in the usual graph theory sense). Then we say that K is the completion of G, where G is a 4-point ϕ^4 graph with obtained by removing one vertex of K. We say that G is a decompletion of K [123].

As an example, consider figure 5.2, showing a completion on five vertices. Then, if we were to remove the top vertex, we have the decompletion shown in figure 5.3.

Within ϕ^4-theory, we are specifically looking at 4-point Feynman graphs. This means we have removed exactly one vertex from the standard graph theory definition of a 4-regular graph, leaving exactly 4 vertices to have an edge that doesn't connect to anything else (the external edges) [123, 8]. A visual representation of a 4-point ϕ^4 Feynman graph is shown in figure 5.3.

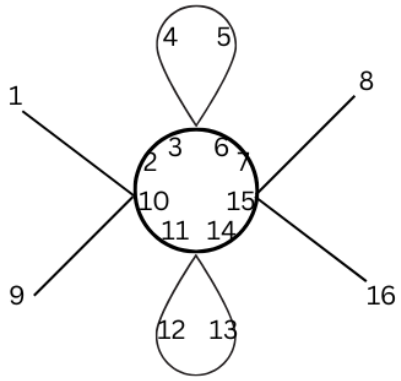


Figure 5.1: Half-Edge Example: Vertices are: $\{1, 2, 9, 10\}$, $\{3, 4, 5, 6\}$, $\{7, 8, 15, 16\}$, $\{11, 12, 13, 14\}$, internal edges are: $\{2, 3\}$, $\{4, 5\}$, $\{6, 7\}$, $\{10, 11\}$, $\{12, 13\}$, $\{14, 15\}$ and external edges are $1, 8, 9, 16$ (following [123]).

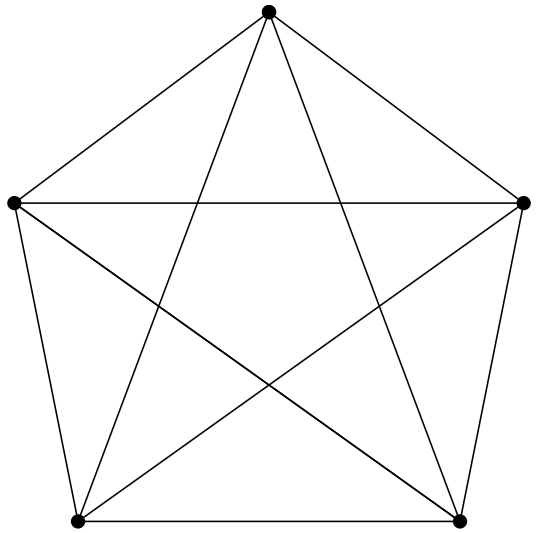


Figure 5.2: Example of a Completion on Five Vertices

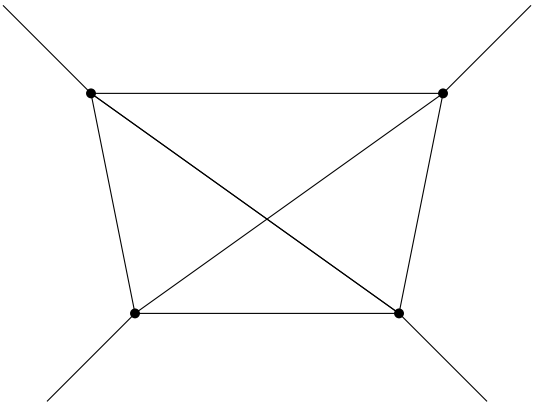


Figure 5.3: Decompletion of figure 5.2

5.1.2 Loop Order and Spanning Trees

The loop order (or first Betti number) of a connected graph G is defined as

$$\ell(G) = |E(G)| - |V(G)| + 1 \tag{5.1}$$

This counts the number of independent cycles in the graph and corresponds to the number of independent loop momenta in the associated Feynman integral in momentum space (see section 5.2). The loop order is the dimension of the cycle space of the graph [123]. In the physics world, loop refers to what a graph theorist would call a cycle, but we will only use it in the fixed phrase loop order.

Definition 5.1.3 (Spanning Tree)**.** A spanning tree of G is a subset of internal edges of G that is connected and contains no cycles, and includes all of the vertices of the graph. The set of all spanning trees of a graph G is denoted by T_G.

5.1.3 Divergences

We follow [62] for some standard definitions.

Definition 5.1.4 (Primitive Log-Divergent Graph)**.** Let G be a ϕ^4 graph. G is said to be primitive log-divergent if $|E(G)| = 2\ell(G)$ (where $|E(G)|$ is the number of edges), and every non-empty proper subgraph G' of G has $|E(G')| > 2\ell(G')$.

The only primitive graphs we will consider are the primitive log-divergent graphs, so we will just call them primitive going forward.

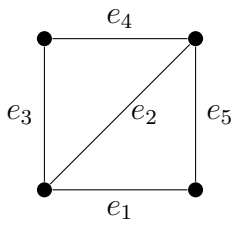


Figure 5.4: Example Graph for the Dual Kirchhoff Polynomial.

5.1.4 Kirchhoff Polynomials

The dual Kirchhoff polynomial, sometimes called the first Symanzik polynomial, plays a key role in defining the Feynman integral. Let G be a connected graph. For each edge $e \in E(G)$, we assign a variable a_e [123, 95].

Definition 5.1.5 (Dual Kirchhoff Polynomial)**.** The dual Kirchhoff polynomial Ψ_G is defined as

$$\Psi_G = \sum_{T \in T_G} \prod_{e \notin T} a_e \tag{5.2}$$

where the sum runs over all of the spanning trees T in G and the product runs over all edges of G that are not in T. Note that to keep notation light we abuse notation slightly and write a_i instead of a_{e_i} whenever convenient.

As an example consider figure 5.4. The spanning trees will be the acylic and connected subgraphs that include all 4 vertices and exactly 3 edges (since $|V| - 1 = 3$) There are 8 such trees in total.

$$\Psi_G = a_1a_2 + a_1a_3 + a_1a_4 + a_2a_3 + a_2a_4 + a_2a_5 + a_3a_5 + a_4a_5$$

Another way to think about the dual Kirchhoff is through a certain determinant. We choose an arbitrary orientation on the edges of G, and an ordering for the vertex and edge sets. The signed incidence matrix (an $V(G) \times E(G)$ matrix), uses the ordering and orientation on the vertex and edge set. An edge from i to j has tail i and head j. Then the ij^{th} entry will be 1 if the edge j has tail i, and a -1 if it has head i, and zero otherwise.

Definition 5.1.6 (Expanded Laplacian)**.** Let G be a graph, with signed incidence matrix E where one row has been removed. If Λ is the diagonal matrix of edge variables, then the

expanded Laplacian is given by:

$$M = \begin{bmatrix} \Lambda & E^T \\ -E & 0 \end{bmatrix}$$

.

It turns out that $\Psi_G = \det(M)$.

Remark 5.1.1. *Another version of the Kirchhoff polynomial is given by taking product over all of the edges that are in the spanning tree, given by*

$$\sum_{T \in T_G} \prod_{e \in T} a_e \tag{5.3}$$

This version is also given by $\det(E\Lambda E^T)$*, where that matrix is the Laplacian of the graph. For more information on the this version of the Kirchhoff polynomial, see [123, 29].*

5.2 Feynman Graphs

To represent these interactions in a computationally tractable way, physicists often use Feynman graphs. Feynman graphs, introduced by Richard Feynman in the 1940s, provide a visual and combinatorial method to compute particle interaction amplitudes in quantum field theories [66].

There is a procedure called Feynman rules, for reading off an integral from each Feynman graph. This is the Feynman integral. There are three forms of the Feynman integral that are of interest to us when studying Feynman graphs: position space, momentum space, and parametric space [123]. In position space, the integral is written in terms of spacetime coordinates at the interaction vertices, and propagators depend on the distances between these points. In momentum space, the variables are the momenta flowing through internal edges, and the integrals are constrained by momentum conservation at each vertex. Finally, in parametric space, the integral is written using the dual Kirchhoff polynomial, and another polynomial, called the second Symanzik polynomial [106].

5.3 Feynman Periods

In QFT, divergent integrals must be regularized and renormalized to extract finite predictions. A key step in this process involves identifying which parts of a graph are responsible

for the divergences [76]. There is a process called renormalization that can be encoded with Hopf algebras, that tells us how to extract finite predictions from Feynman integrals that have divergences. Feynman periods capture the residue of the integral at the divergence and are useful to understanding the transcendental content of the theory, as well as being nice combinatorially [76, 123, 95].

With the Kirchhoff polynomial Ψ_G defined, we now define the Feynman period, a scalar quantity obtained by integrating over the variables we assigned to the edges of G. For a connected ϕ^4 Feynman graph G that has 4 external edges it holds that:

$$|V(G)| = \ell(G) + 1 \tag{5.4}$$

Definition 5.3.1 (Feynman Period)**.** Let e_1 be an edge of a graph G and set $a_{e_1} = 1$. Then the Feynman period (in the parametric space) is the integral:

$$P_G = \int_{a_e \geq 0, e \neq e_1} \frac{\prod_{e \in E(G), e \neq e_1} da_e}{\Psi_G^2|_{a_{e_1}=1}} \tag{5.5}$$

If G is primitive log divergent then P_G converges. For some examples of period calculations, see [106].

Theorem 5.3.1. *If two graphs G and G' are both primitive 4-point ϕ^4 graphs, with the same completion then they will have equal Feynman periods [123].*

For more detail of theorem 5.3.1, see [123].

Feynman periods are interesting number theoretic values. For example, the period of the graph K_4 $P_{K_4} = 6\zeta(3)$, where $\zeta(3)$ is the Riemann Zeta function, evaluated at 3.

Other Feynman periods can be expressed in terms of multiple zeta values, which are a generalization of positive integer evaluations of the Riemann Zeta function. Yet others cannot be expressed in terms of their multiple zeta values but come from more complicated geometries like K_3 surfaces [28]. Others we do not know at all currently. For more examples, see [106].

5.4 Symmetries

There are certain graph theoretic symmetries which the Feynman period satisfies. These help with period computation and are a hint at which graph features may be useful in

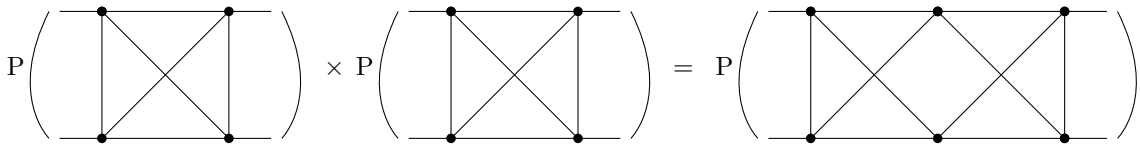


Figure 5.5: Illustration of Product Identity

period prediction. This section follows [62].

5.4.1 Products

The graph product we are interested in is the 2-sum, which merges two graphs over a shared edge. The period is multiplicative for this graph product. This property facilitates the decomposition of complex graphs into simpler components, and provides us with the product identity from [123].

Theorem 5.4.1 (Product Identity)**.** *Let G be a primitive 4-point ϕ^4 graph with a two-vertex cut that separates the external edges into two pairs. Let v_1 and v_2 denote the cut vertices, with G_1 and G_2 being the two subgraphs formed by splitting at v_1 and v_2. Then define $\overline{G_1}$ and $\overline{G_2}$ to be the graphs G_1 and G_2 respectively, with an additional edge between v_1 and v_2. Then the period satisfies:*

$$P_G = P_{\overline{G_1}} \cdot P_{\overline{G_2}} \tag{5.6}$$

For an example of the product identity, see figure 5.5. As stated above, we know that $P(K_4) = 6\zeta(3)$, and thus the period of the final graph in the figure is given by $36\zeta(3)^2$.

5.4.2 Duality

For G a planar graph, let $G^\star$ be its planar dual.

Theorem 5.4.2 (Dual Symmetry)**.** *For G a planar, primitive 4-point ϕ^4 graph. Suppose $G^\star$ is also a primitive 4-point ϕ^4 graph. Then $P_G = P_{G^\star}$.*

This symmetry allows computations to be transferred between dual graphs, potentially simplifying the analysis. The proof of this dual symmetry centrally involves the Fourier transform. As a result, sometimes the dual symmetry is called the Fourier symmetry [62, 53].

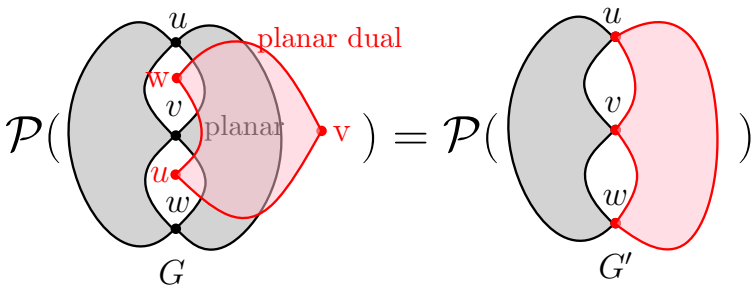


Figure 5.6: Illustration of Fourier Split Identity by Josephine Reynes

5.4.3 Fourier Split

The Fourier split is a kind of half dual where first if a graph has 3-vertex separation, and one side is planar than we can take the planar dual of that side and glue it back on to the other side, resulting in a graph with the same period [95].

Definition 5.4.1 (Terminal Dual ([62], definition 2.6))**.** Let G be a graph that has 3 marked vertices, that we label $0, 1$ and k. We define a new graph G_e to be G in addition to the edges $(0, 1)$, $(0, k)$ and $(1, k)$ and suppose G_e is planar. The *terminal dual* of G is defined by taking the dual of G_e, labelling the dual vertex created by edge $(0, 1)$ with label k, the dual vertex created by the edge $(0, k)$ labelled by 1, the dual vertex created by the edge $(1, k)$ labelled by 0, and removing the star created by the dual edges of $(0, 1), (0, k)$ and $(1, k)$.

Theorem 5.4.3 (Fourier Split Identity ([62], theorem 2.8))**.** *Let G be a primitive 4-point ϕ^4-graph with a 3-vertex separation. Label the 3 vertices by $0, 1$ and k, and further suppose one side of the separation has the property that we can take its terminal dual. Let H be the result of taking the terminal dual of this side, and re-attaching it to the other side, so the labels $0, 1$ and k match, Then $P_H = P_G$.*

This identity is illustration in figure 5.6.

5.4.4 Completion

Recall the definition of completion from definition 5.1.2. The period is completion invariant in the following sense.

Theorem 5.4.4 (Completion Identity)**.** *If G_1 and G_2 are two primitive ϕ^4 graphs with the same completion, then:*

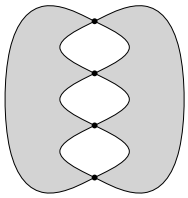
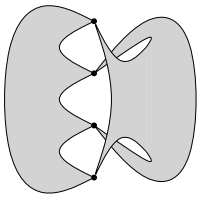
$$P_{G_1} = P_{G_2}.$$

In view of this theorem, we can think of the period as being attached to the completion of the graph, since the choice of decompletion doesn't matter. This is why in the table illustrated in [106], the period is associated with the completion of the graphs.

5.4.5 Twist

The twist identity is best seen at the completed level.

Theorem 5.4.5 (Twist Identity)**.** *Let*

$K_1 =$  *and* $K_2 =$  (5.7)

both of which are completed primitive graphs. Let G_i be any decompletion of K_i. Then $P_{G_1} = P_{G_2}$.

5.5 Useful Features

We will now discuss the features that we will use in our machine learning models in chapter 7.

5.5.1 Features with Symmetries

Some of the most interesting features are the ones that have the same symmetries as the Feynman period. This suggests they should have a close algebraic relation.

Hepp Bound

The Hepp bound, is an upper bound for the Feynman period, that is helpful for its prediction.

Definition 5.5.1 (Hepp Bound)**.** As explained by [95], instead of using the dual Kirchhoff polynomial in the equation, the Hepp bound uses a maximal monomial at each point in the integration domain.

$$\mathcal{H}(G) = \int_{a_i \geq 0} \frac{\prod_{i=1}^{|E(G)-1|} da_i}{(\max_{T \in T_G} \{\prod_{e \notin T} a_e\})^2 \mid_{a_{|E(G)|=1}}} \tag{5.8}$$

Strictly as a bound, the Hepp bound is quite poor, however with a certain scaling it correlates surprisingly well with the Feynman period [95, 123, 8]. Empirically, $\frac{\log P}{\ell - 1}$ and $\frac{\log (H/2)}{\ell - 1}$ have a near linear relationship as shown in [95]. With a 5-parameter fit function, Balduf was able to show that the Hepp bound could predict the Feynman period for a given loop order with less than 1% error [8].

The Hepp bound satisfies all five symmetry theorems from section 5.4 [95].

Further, it is conjectured to be a perfect invariant in the following sense.

Conjecture 5.5.1 (Conjecture 1.2 ([95]))**.** *Two primitive 4-point ϕ^4 graphs G_1 and G_2 have the same Hepp bound if and only if they have the same Feynman period.*

In chapter 7, the difference between machine learning models with and without the Hepp bound will be investigated, in a different way showing how good of a predictor for the Feynman period it is.

Martin Invariant

The Martin polynomial is a kind of circuit partition polynomial.

Definition 5.5.2 (Transition System)**.** At a given vertex v of a graph G, we say that a transition is a perfect matching of the edges, where we match the half edges at that vertex. Then a transition system P is a fixed choice of transition per vertex [96].

We can see P as a partition of G into one or more circuits. We are going to write $|P|$ for the number of circuits in P, thought of as a decomposition. There are some minor technicalities that occur when there are loops, see [96].

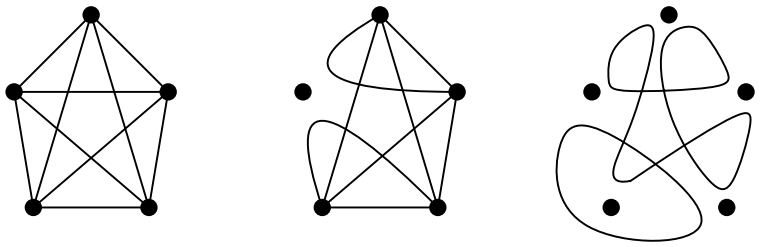


Figure 5.7: Transition System Example. A special thank you to Josephine Reynes for her help with lines!

Figure 5.7 is an example of a transition system on K_5 illustrating how this particular transition system gives a decomposition into two circuits.

Definition 5.5.3 (Martin Polynomial)**.** The Martin polynomial of a graph G with all vertices of even degree, is given by taking the sum over all transition systems P [96].

$$m(G,x) = \sum_{P} (x-2)^{|P|-1} \in \mathbb{Z}[x] \tag{5.9}$$

Definition 5.5.4 (Martin Invariant)**.** For a $2k$-regular graph G where $k \geq 1$, the Martin invariant is the derivative with respect to k.

$$M(G) = \frac{4(-1)^k}{(k-2)!(2k)!} m'(G, 4-2k) \in \mathbb{Q} \tag{5.10}$$

We are interested in the 4-regular case, where the graph is a completion of a primitive 4-point ϕ^4-graph. In this case, $k = 2$ so $M(G) = \frac{1}{6}m'(G,0)$.

The Martin invariant satisfies all five symmetry theorems from section 5.4 [95].

Edge Cuts

Definition 5.5.5 (Edge Cut)**.** A cut induced by a vertex set $X \subset V(G)$ is all of the edges that have exactly one end in X (that is, edges that incident to exactly only vertex in X).

The number of 6-edge cuts satisfies all five symmetry theorems from section 5.4 with suitable normalization [95].

The number and nature of such cuts give insight into the graph's structure and can influence the behaviour of its Feynman integral. We explore several of these cuts towards using machine learning for predicting the Feynman period in chapter 7. In particular we used the number of n-edge cuts for $n = 6, 8, 10, 12, 14$.

Although the 8-edge cuts do not respect the symmetries of the Feynman period, their inclusion was helpful for predicting the Feynman period and was seen to improve the accuracy of the prediction. Adjusting the higher edge cuts to create further invariants satisfying the symmetries remains a question of active research.

Vertex Cuts

The usefulness of edge cuts suggests also considering the number of vertex cuts of various sizes, although they do not satisfy the symmetries. We consider n-vertex cuts for $n = 3, 4, 5$. A cut induced by a vertex set $C \subset V(G)$ is called a vertex cut of a connected graph if the removal of C from G results in a disconnected graph.

5.5.2 Global Counting

Graph Automorphisms

Inspecting graphs with particularly large period, we tend to find very symmetric graphs, such as circulants and similar. So, we might guess that the size of the automorphism group is a useful indicator [8]. Write $\mathrm{Aut}(G)$ for the group of automorphisms of a graph G [8].

Additionally, in the perturbative series we care about in QFT each graph's period is weighted with a symmetry factor of $|\mathrm{Aut}(G)|^{-1}$. So, the period and $|\mathrm{Aut}(G)|$ come together physically, giving another reason why it might be a useful period predictor.

As shown in [8], the symmetry factors of completions and decompletions are also related by:

$$(\ell(K) + 2)\frac{1}{|\mathrm{Aut}(K)|} = \sum_{G \text{ decompletion of } K} \frac{1}{|\mathrm{Aut}(G)|}.$$

Non-Isomorphic Decompletions

Following from the observations about automorphisms, then the interplay between completion and decompletion also suggests simply counting the number of non-isomorphic

decompletions of a graph. We expect a large random graph to have no non-trivial automorphisms. Computations show that a completion with $\ell + 2$ vertices tends to have on average $\ell + 2$ non-isomorphic decompletions [8]. This suggests that an overwhelming majority of the decompletions for higher loop orders are non-isomorphic.

Number of Planar Decompositions

The appearance of new families of numbers in the Feynman period tends to happen when the graph is highly non-planar, with the same behaviour appearing in planar graphs, only at higher loop order. Special attention to planar graphs in period prediction is thus reasonable. In fact, as shown in [8], there appears to be a roughly linear relationship between the Feynman period and the ratio of planar decompletions. The ratio is given by:

$$\frac{\text{Number of Planar Decompletions}}{\ell(G) + 2}$$

Symmetry Factor of Planar Decompletions

With the above considerations together, we also include the symmetry factor of planar decompletions, given by

$$\frac{1}{\text{Number of planar decompletions}} \sum_{\text{planar decompletion } g} \frac{1}{|\text{Aut}(g)|}$$

5.5.3 Distance Features

The Feynman period of the graph is seen to get larger as the graph itself visually looks "larger" as well [8], so it is useful to consider various notions of wideness for the graph among our features.

Radius and Diameter of Graphs

The diameter of a graph is defined to be the distance between two vertices in the graph that have the have the maximum shortest path between them. That is

$$\text{diam}(G) = \max_{v_1, v_2 \in V(G)} (\text{length of the shortest } v_1, v_2 \text{ path})$$

To define the radius, first note that the eccentricity of a vertex v is defined to be the maximum distance between v and any another vertex of the graph. So the radius is the minimum of the eccentricity running over the vertices.

$$\text{radius}(G) = \min_{v \in V(G)} (\text{eccentricity of } v)$$

For more on the diameter, see [8]. It should be noted that for this work, the graphs generally fall within several values for the diameter. For example, Balduf showed that for the 755643 graphs of loop order 13, they all had diameters of $2, 3$ or 4 [8]. As a result of this, it is potentially useful to consider the mean distance instead, as we outline next.

Mean, First Moments of Distance Between Vertices

The mean distance between pairs of vertices is the average of the length of the shortest paths between all pairs of vertices in the graph.

$$\text{d}(G) = \text{mean}_{v_1, v_2 \in V(G)} (\text{length of the shortest } v_1, v_2 \text{ path}) \tag{5.11}$$

As shown in [8], there are 51 mean distance values for loop order 13, compared to the 3 values for the diameter. As well as the mean, we can consider the moments of the distribution of lengths of shortest paths. For more on first moments see section 7 of [8].

5.5.4 Matrix Features

Given the importance of the dual Kirchhoff polynomial, which comes as a determinant of the expanded Laplacian, and the importance of spectral graph theory in general, we also consider a number of matrix based features.

Eigenvalues of the Distance Matrix

Continuing on the importance of distance, we define the distance matrix to be matrix formed by the distances between all pairs of the vertices. We specifically consider the eigenvalues of the distance matrix for the models will use. This will be outlined in chapter 7.

Eigenvalues of the Laplacian Matrix

The Laplacian matrix of our Feynman graphs can also be considered. In this case, we take the Laplacian matrix without variables, in contrast to section 5.1.4. Namely, if $|V(G)| = n$, then the Laplacian is an $n \times n$ matrix such that that entry at index ij is $\deg(v_i)$ if $i = j$, -1 if the vertiecs i and j are adjacent in the graph, or zero otherwise.

The eigenvalues of the Laplacian are well studied in spectral graph theory [35]. In our case, they will be used for the models that we talk about in chapter 7.

Scaled Traces of Adjacency Matrix

The adjacency matrix of a graph with $|V(G)| = n$ is an $n \times n$ matrix A where $A_{ij} = 1$ if the vertices i and j are connected by an edge, and zero otherwise. Note that the diagonal consists of all zeros . The scaled traces are given by $\frac{\mathrm{Tr}(A^k)}{(\ell(G)+2)2^k}$.

5.5.5 Electrical Ideas

Given the origin of the Kirchhoff polynomial in electrical networks and the analogy between momentum space Feynman integrals and electrical flows, various electric features are also reasonable to consider.

Mean Electrical Distance

The Kirchhoff index of a graph G is defined to be the average of the resistances between all of the pairs of vertices v in the vertex set of our graph $V(G)$ [71].

$$R(G) = \frac{2}{|V(G)|(|V(G)|-1)} \sum_{v_1 < v_2 \in V(G)} r_{v_1,v_2}$$

where r_{v_1,v_2} is the resistance between the vertices v_1 and v_2, where every edge has resistance 1, and following Kirchhoff's laws. There is a nice way to compute this using the Moore-Penrose pseudoinverse of the Laplacian matrix, as described in section 2.1 of [9].

Minimum, Maximum, Standard Deviation of Electrical Resistance

We consider the minimum, maximum, and standard deviation, of electrical resistance between pairs of vertices in our graph, using the same notion of electrical resistance as above. We additionally considered the absolute mean, minimum, and maximum of the first 4 Ursell functions [118]. An Ursell function is a cumulant of a random variable, and they arise naturally by summing over connected Feynman graphs.

The mean, and mean of absolute value, and the entries of the transfer current matrix are also considered. The transfer current matrix of an electrical network expresses how the current through an edge changes if a unit current is imposed parallel to another edge.

Autocorrelation Function of Random Walks

Since quantum walks are a quantum analog of what we just discussed, we also include the first 9 entries of the autocorrelation function of random walks of length 10^4, which we average over 5000 random labellings of the vertices. For more on autocorrelation functions see [25].

5.6 Motivation from Quantum Field Theory

Feynman periods arise in QFT as finite, scale-independent residues of divergent Feynman integrals associated with primitive graphs. In perturbative QFT, physical quantities such as scattering amplitudes or correlation functions are computed as a formal power series whose coefficients are the Feynman integrals [123, 76].

Feynman periods are particularly important for the following reasons:

- They encode the combinatorics of UV divergences. Since primitive graphs correspond to the simplest divergent structures in perturbation theory, their periods provide a clean, universal description of the leading divergent behaviour in a theory.
- They are independent of physical parameters. Unlike full Feynman integrals, which depend on masses and momenta, periods are dimensionless and independent of kinematics. They depend only on the structure of the graph in the form of the dual Kirchhoff polynomial. This makes them ideal objects for mathematical study.

- They connect QFT with number theory and algebraic geometry. Many known Feynman periods evaluate to presumed transcendental numbers, such as multiple zeta values. This has motivated deep connections between QFT and the theory of motives, periods, and arithmetic geometry.

Because of their mathematical structure and physical relevance, Feynman periods serve as a bridge between QFT and several areas of pure mathematics. They are also a natural object of study for classification and computational methods, including those explored in this thesis.

Chapter 6

Introduction to Machine Learning

6.1 Motivation

This chapter presents an overview of neural networks, with a focus on the fundamental concepts, advanced architectures, and specialized graph-based models relevant to understanding my joint work with Paul-Hermann Balduf on using machine learning to predict Feynman periods [9]. Starting from perceptrons and multilayer perceptrons, we develop the formalism of feedforward neural networks, including activation functions. We then explore advanced network architectures such as convolutional neural networks (CNNs). Next, we introduce graph neural networks (GNNs) and their variants, like graph convolutional networks (GCNs) and GraphSAGE, delineating how graph structures are represented and processed in neural models. After this, we introduced our new work by connecting these machine learning concepts and using them for predictions in ϕ^4-theory.

6.2 Fundamental Concepts of Machine Learning

The aim of machine learning is train a computer to be able to perform a given task, which in this case is predicting a Feynman period for a given Feynman graph. In general, the goal is to take pre-existing data and use its characteristics to output a desired value, and the data itself will have *features* or *parameters* that characterize the data [54]. The exact features for our problem will be expanded in section 7.1.1, where for example we will consider features such as the number of 3-cuts, 6-cuts and the Hepp bound in our Feynman

graph, as described in section 5.5 [9]. Each particular data point can be considered a vector $\mathbf{x}$, such that each given entry x_i is a value for a feature of that point in our dataset [54].

In machine learning, a model is a program that captures patterns in data. It maps inputs (such as feature vectors) to outputs (such as predictions or classifications), typically by learning features from data through a training process. The model encapsulates the assumptions, structure, and learned knowledge which is then used to make inferences or decisions about new, unseen data [54]. As a simplifying assumption, all of our inputs and outputs are real-valued, and this will have no effect on the outlines of the models in this chapter.

There are two very classical types of machine learning problems that this subsection will outline: classification and regression. In *classification* problems, the task is to assign an input to one of several predefined categories. Given k possible categories and an input represented as a feature vector $\mathbf{x} \in \mathbb{R}^n$, the model learns a function of the form $f : \mathbb{R}^n \to 1, \ldots, k$ that maps inputs to discrete category labels [54]. In *regression* problems, the task instead is to predict a real value based on a set of data values for the features of the model. In this sense, if we were to have n features for each data entry, we would say that our model produces a function $f : \mathbb{R}^n \to \mathbb{R}$ [54]. For the scope of this thesis, our models are regression models, with the goal of predicting the Feynman period.

In order to evaluate the performance of a machine learning model, we split the data into a training set $(\mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}})$ and a test set $(\mathbf{X}^{\text{test}}, \mathbf{Y}^{\text{test}})$. The training set is used to learn the features of the model, while the test set is kept separate during training, and serves to evaluate the model's ability to make accurate predictions on unseen data. This process assesses the model's generalization, which refers to its capacity to perform well on new, previously unobserved inputs [54, 128]. High generalization indicates that the model has learned meaningful patterns rather than simply memorizing the training data. A key concern in this context is overfitting, which occurs when the model fits the training data too closely, including its noise or outliers. While such a model may achieve very low training error, it tends to perform poorly on the test set, revealing its inability to generalize [54]. Overfitting is particularly likely when the model is overly complex relative to the size or variability of the training data. To mitigate this, techniques such as dropout are employed, which we will discuss later in this section. Within the scope of this project, challenges arose due to disparities in dataset sizes across loop orders as described in [9], as discussed further in chapter 7.

6.2.1 Linear Models for Regression

A machine learning model is said to be linear for a regression problem if its output is a linear function of the input features. More precisely, given an input vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, a linear model computes a prediction of the form

$$\hat{y} = \mathbf{w}^T\mathbf{x} + b$$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of weights we try to optimize and $b \in \mathbb{R}$ is a scalar bias, also subject to optimization [54]. While the weight vector $\mathbf{w}$ controls the orientation of the hyperplane (or decision boundary), the bias term b allows the model to shift this hyperplane away from the origin. The expression above defines an affine function (a linear function plus a constant term). The set of all such functions forms the model class, also known as the hypothesis class, the collection of functions that the learning algorithm is able to explore when searching for an optimal model. In the case of regression, this yields an affine map whose graph is a hyperplane in $\mathbb{R}^{n+1}$. In classification, a linear model defines a decision boundary as the set $\mathbf{x} \in \mathbb{R}^n : \mathbf{w}^T\mathbf{x} + b = 0$, which is a hyperplane that separates $\mathbb{R}^n$ into two half-spaces corresponding to different class predictions [38]. Thus, linear models can only separate or fit data that is linearly structured (meaning data that is well-approximated or well-separated by a hyperplane). When this is not the case, additional feature transformations or nonlinear models must be used. To make the calculations easier, instead of explicitly including a bias term, we append every data entry with a 1, and optimize the weight vector to include the bias as the last term.

6.2.2 Linear Regression

Linear regression is one of the most classic linear model, and it even pre-dates modern machine learning, but can be recast in this context and works well. In linear regression, the model finds the desired weights that minimize the difference between the predicted value ($\hat{y}$) and the desired output (y), which is the Feynman period in our case. The difference is calculated using a loss function. A loss function that often works well for regression problems is *mean squared error loss function* given by:

$$MSE_{test} = \frac{1}{m} \mid\mid \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \mid\mid_2^2$$

Where we are using the Euclidean distance for the norm, symbolized by the subscript 2 [54], and m is the number of testing examples. Linear regression is a linear model, which we compare to other linear models in section 7.3.1. To train a linear regression model, we aim

to find the weight vector $\mathbf{w}$ that minimizes the MSE between the predicted outputs $\hat{\mathbf{Y}}^{\text{train}}$ and the training labels $\mathbf{Y}^{\text{train}}$. In this context, training consists of solving an equation to minimize the mean squared error, by solving for where its gradient is zero, as outlined in [54]. Note that $\mathbf{X}^{train}$ refers to the data used for training and $\mathbf{X}^{test}$ is the data used for testing.

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0 \tag{6.1}$$

$$\nabla_{\mathbf{w}} \frac{1}{m} ||\hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train}||_2^2 = 0 \tag{6.2}$$

$$\frac{1}{m} \nabla_{\mathbf{w}} ||\hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train}||_2^2 = 0 \tag{6.3}$$

We then use the fact that $\hat{\mathbf{Y}}^{train} = \mathbf{X}^{train}\mathbf{w}$, as we have included the bias term in the weights.

$$\frac{1}{m} \nabla_{\mathbf{w}} ||\mathbf{X}^{train}\mathbf{w} - \mathbf{Y}^{train}||_2^2 = 0 \tag{6.4}$$

$$\frac{1}{m} \nabla_{\mathbf{w}} (\mathbf{X}^{train}\mathbf{w} - \mathbf{Y}^{train})^T (\mathbf{X}^{train}\mathbf{w} - \mathbf{Y}^{train}) = 0 \tag{6.5}$$

$$\nabla_{\mathbf{w}} (\mathbf{w}^T \mathbf{X}^{(train)T} \mathbf{X}^{train}\mathbf{w} - 2\mathbf{w}^T \mathbf{X}^{(train)T} \mathbf{Y}^{train} + \mathbf{Y}^{(train)T} \mathbf{Y}^{train} = 0 \tag{6.6}$$

$$2\mathbf{X}^{(train)T} \mathbf{X}^{train}\mathbf{w} - 2\mathbf{X}^{(train)T} \mathbf{Y}^{train} = 0 \tag{6.7}$$

Thus we can solve this system to show that the optimal weights are given by

$$\mathbf{w} = (\mathbf{X}^{(train)T} \mathbf{X}^{train})^{-1} \mathbf{X}^{(train)T} \mathbf{Y}^{train} \tag{6.8}$$

Note that equation (6.6) simplifies $\mathbf{Y}^{(train)^T} \mathbf{X}^{train}\mathbf{w} = \mathbf{w}^T \mathbf{X}^{(train)^T} \mathbf{Y}^{train}$.

6.2.3 Perceptrons

A perceptron is the simplest model of an artificial neuron, first introduced in 1957 [102]. It performs a linear combination of its inputs followed by an activation decision [38]. Formally, given an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$, a perceptron computes a weighted sum $z = \mathbf{w} \cdot \mathbf{x} + b$, where $\mathbf{w} = (w_1, \ldots, w_m)$ is a weight vector and b is a bias term. The

perceptron's output $f(\mathbf{x})$ is then given by an activation function $h(z)$, traditionally the Heaviside step function $H(z)$ for binary classification [38, 65]. In equation form:

$$f(\mathbf{x}) = H(\mathbf{w} \cdot \mathbf{x} + b) \in \{\, 0, 1 \,\}$$

Here $H(z) = 1$ if $z > 0$ and $H(z) = 0$ otherwise [65]. This defines a linear decision boundary in $\mathbb{R}^m$. The perceptron is thus a linear classifier: it can separate data that are linearly separable by some hyperplane. If the bias b is included, it shifts the hyperplane away from the origin. Although the Heaviside function is nonlinear in the sense that it discretizes the output, the overall model remains a linear classifier because the decision boundary is determined by a linear function of the input.

In order to tell how "easy" or "hard" a given problem or task may be, the notion of a margin is used. Given a dataset and a hyperplane, the margin is defined to be the distance between the hyperplane and the nearest point. The margin of a dataset $\mathbf{D}$ with weights $\mathbf{w}$ and bias b defining the hyperplane, is denoted $margin(\mathbf{D}, \mathbf{w}, b) = min_{(x,y)\in\mathbf{D}}\ y(\mathbf{w} \cdot \mathbf{x} + b)$ if $\mathbf{w}$ separates $\mathbf{D}$ and negative infinity otherwise. The margin on the entire dataset is the largest attainable margin of the dataset [38]. Training a perceptron involves adjusting $\mathbf{w}$ and b to correctly classify labelled examples. There is a perceptron algorithm that specifies how to initialize your weights, and how to train a model. This can be considered an easy test as to how linearly separable a dataset can be. There is a proof of perceptron convergence, that specifies how many updates of training a model will need based on the margin of a dataset to find a linear separator for the dataset if it is linearly separable. For a proof of the perceptron convergence algorithm, see the perceptron chapter of [38]. Perceptrons however cannot solve problems that aren't linearly separable, which motivates multilayer perceptrons.

6.3 Complex Models

6.3.1 Quadratic Regression

Quadratic regression is a natural generalization of linear regression in which the model incorporates second-degree (squared) terms of the input features to capture nonlinear relationships between variables. While linear regression fits a hyperplane to the data, quadratic regression fits a quadratic surface, allowing for curvature. Concretely, for an input vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the model predicts an output $\hat{y}$ of the form

$$\hat{\mathbf{y}} = \mathbf{w}^T\mathbf{x} + \mathbf{x}^T Q\mathbf{x} + \mathbf{b} \tag{6.9}$$

where $\mathbf{w} \in \mathbb{R}^n$ is a weight vector for the linear term, $Q \in \mathbb{R}^{n \times n}$ is a symmetric matrix encoding pairwise interactions and squared terms (i.e., $x_i x_j$), and $b \in \mathbb{R}$ is a bias term [83].

This model can be trained by minimizing a loss function such as mean squared error, just as in linear regression. Since the model includes all quadratic combinations of the features, it can represent more complex surfaces that fit curved data better than linear models. However, this comes at the cost of a larger parameter space: quadratic regression has $O(n^2)$ features compared to $O(n)$ in linear regression, which may lead to overfitting if not regularized properly or trained on sufficient data.

6.3.2 Multilayer Perceptrons

Multilayer perceptrons (MLPs) or feedforward neural networks are neural network composed of multiple layers of neurons (perceptrons) with nonlinear activation functions [54], as seen in figure 6.1. In an MLP, neurons are arranged in an input layer, one or more hidden layers (meaning they go in between the input and output), and an output layer, and each neuron in a layer connects to every neuron in the next layer [54, 38]. To explain more about MLPs, it is first important to understand the significance of activation functions.

6.3.3 Activation Functions

Activation functions introduce nonlinearity into neural networks, allowing them to model complex, non-linear patterns in data. Without such non-linearities, a neural network composed of only linear transformations would itself represent a linear function, regardless of depth, defeating the purpose of having multiple layers. Nonlinear activations therefore give neural networks their expressive power. We briefly review two widely used activation functions and then elaborate on their behaviour, particularly focusing on the vanishing gradient problem and dying ReLUs.

- Sigmoid (Logistic) Function: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$ [54]. This smooth, S-shaped function maps real-valued inputs to the interval $(0, 1)$, as shown in figure 6.2 It was historically popular in early neural networks, especially for binary classification, since it allows for probabilistic interpretation of outputs. However, sigmoid functions suffer from a major drawback during training: the vanishing gradient problem.

  The derivative of the sigmoid is $\sigma' = \sigma(x)(1 - \sigma(x))$. This derivative becomes very small when x is either large and positive (in which case $\sigma(x) \approx 1$) or large and negative (then $\sigma(x) \approx 0$). This leads to gradients close to zero during backpropagation

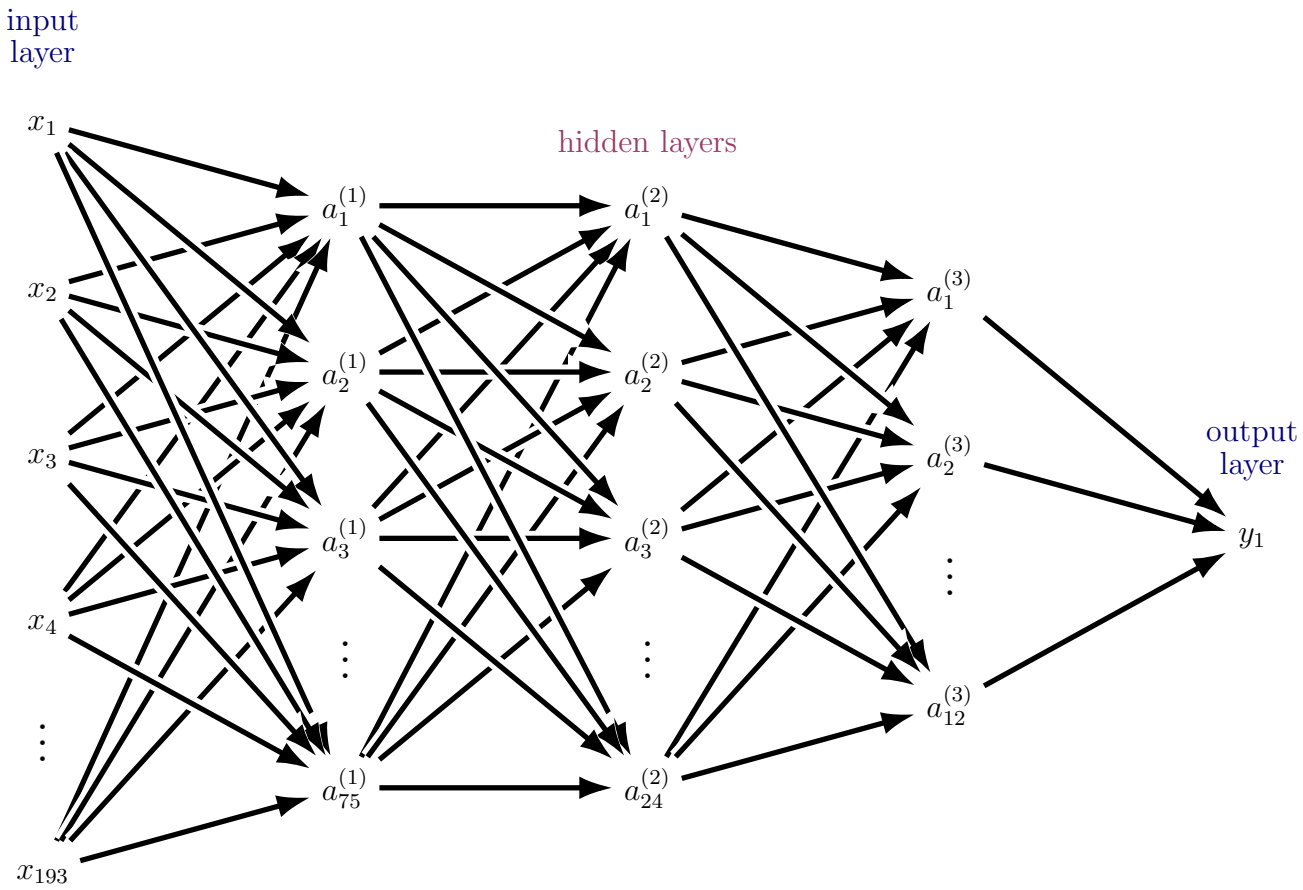


Figure 6.1: figure
MLP Architecture

(see section 6.3.6), particularly in deep networks where these small derivatives are compounded across many layers. As a result, weights in early layers receive negligible updates, effectively freezing them, and learning becomes slow or even impossible [54, 12].

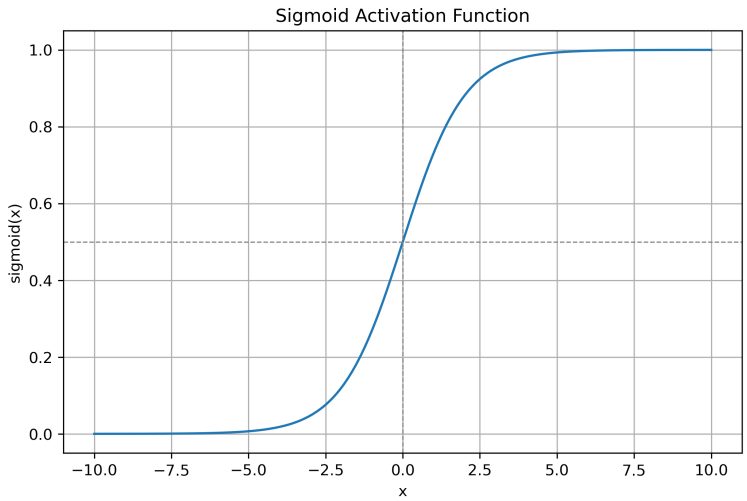


Figure 6.2: Sigmoid Function

- Rectified Linear Unit (ReLU): $\text{ReLU}(x) = \max\{0, x\}$ [54]. ReLU is a piecewise linear function that outputs zero for negative inputs and identity for positive ones, as shown in figure 6.3. Its derivative is 0 for any $x \in \mathbb{R}$ less than 0, or otherwise 1, meaning that extremely large or negative values do not cause the vanishing gradient problem during backpropogation. As a result, ReLU became widely adopted because it avoids the vanishing gradient issue [52].

  However, ReLU introduces a new issue: the dying ReLU problem. If a neuron's input becomes negative, it outputs zero. If this happens consistently during training (e.g., due to poor initialization or large gradient steps), then the neuron will always output zero and its gradient will also be zero. Consequently, the weights connected to this neuron stop updating entirely, effectively "killing" the neuron. As we discussed earlier, this is not optimal and networks with many such neurons may suffer a significant reduction in capacity [80].

  To mitigate this, variants like Leaky ReLU and Parametric ReLU have been proposed. With Leaky ReLU, value is x if $x > 0$ and αx if $x \leq 0$, where $\alpha \in (0, 1)$ (typically

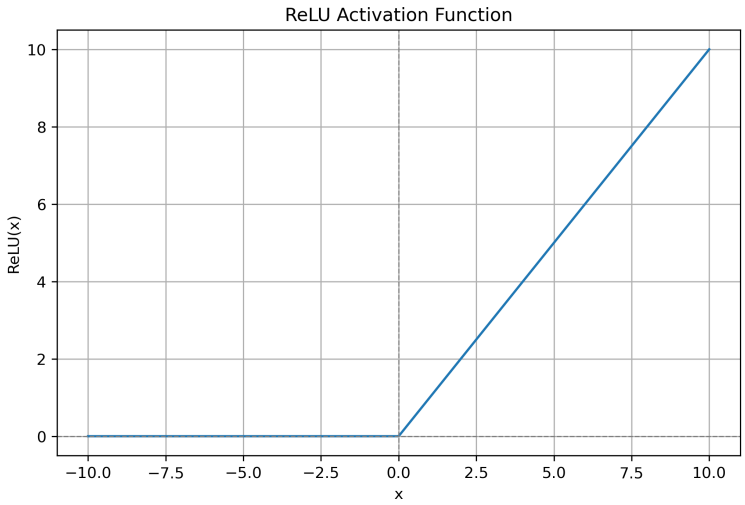


Figure 6.3: ReLU Function

$\alpha = 0.01$) [81]. These variants allow a small, non-zero gradient for negative inputs, preventing neurons from dying and encouraging better learning dynamics.

In this project, Leaky ReLU was frequently used as it empirically helped prevent the issue of dead neurons while maintaining computational efficiency and improved convergence properties [9].

6.3.4 Training Neural Networks

Training a neural network involves optimizing its features, typically weights $\mathbf{W}$ and biases b, to minimize a loss function $\mathcal{L}(\hat{y}, y)$ that measures the error between the network's prediction $\hat{y}$ and the corresponding value from our dataset y [54]. This is generally done through variants of gradient descent, an iterative optimization algorithm that updates the parameters in the direction that most rapidly decreases the loss.

Gradient Descent

Given a loss function $L(W)$ defined over the features of the network (like MSE section 6.2.2), the classical gradient descent algorithm updates parameters using the rule

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L(\mathbf{W}) \tag{6.10}$$

where $\eta > 0$ is the learning rate, and is a hyperparameter that can be further optimized, and $\nabla_{\mathbf{W}} L(\mathbf{W})$ is the gradient of the loss with respect to the weights. Intuitively, $\nabla_W L(\mathbf{W})$ points in the direction of steepest ascent, so moving in the opposite direction reduces the loss.

For a dataset $\{ (\mathbf{x}, y_i) \}_{i=1}^m$, the loss function is typically defined as the average of a per-sample loss $\mathcal{L}$

$$L(\mathbf{W}) = \frac{1}{m} \mathcal{L}(f_{\mathbf{W}}(\mathbf{X}_i), y_i) \tag{6.11}$$

In regression problems, a common choice is the mean squared error loss $\mathcal{L}(\hat{y}, y) = ||\hat{y} - y||_2^2$, as described earlier in section 6.2.2 [54].

6.3.5 Stochastic Gradient Descent (SGD)

Computing the full gradient over all m training samples can be computationally expensive, especially for large datasets. To improve efficiency, it is possible to use stochastic gradient descent (SGD), which updates the parameters based on a randomly selected subset of data, known as a mini-batch [24, 54]:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L_B(\mathbf{W}) \tag{6.12}$$

where $L_B(W)$ is the loss computed over the mini-batch $B \subseteq \{ 1, \ldots, m \}$. Though this introduces noise into the update direction, SGD often converges faster and can escape shallow local minima.

6.3.6 Backpropagation

The gradient $\nabla_W L(W)$ in multilayer neural networks is efficiently computed via the back-propagation algorithm, which applies the multivariable chain rule recursively layer by layer [105, 54]. In brief, backpropagation works by first computing the error at the output layer, then using the chain rule to express the gradient of the loss with respect to the weights in each hidden layer. The name backpropgation is due to the fact that we propagate gradients backward throughout the network, and we start this process at the end of the network [38]. For a full overview of back propagation, see chapter 6.5 of [38].

6.3.7 Optimizers

Basic stochastic gradient descent (SGD) updates model parameters by taking small steps in the direction of the negative gradient of the loss function. While simple and widely used, SGD is sensitive to the choice of learning rate and can converge slowly, especially in regions where the loss surface has steep slopes or noisy gradients. These challenges have motivated the development of adaptive optimizers, which automatically adjust the learning rate during training based on the behaviour of past gradients.

One of the most widely used adaptive optimizers is Adam (Adaptive Moment Estimation) [79]. Adam extends SGD by maintaining exponentially decaying averages of past gradients (which acts like momentum) and past squared gradients (to adjust learning rates per parameter). The "momentum" component helps smooth updates by dampening oscillations, while the "adaptive" component allows each parameter to have its own learning rate, depending on how frequently and strongly it has changed in the past. This allows Adam to perform well even when gradients vary significantly in magnitude or direction. Adam has become a standard optimizer in deep learning because of its fast convergence, minimal need for manual tuning, and robustness across a wide range of tasks.

6.3.8 Dropout Layers

Dropout is a regularization technique used in neural networks to prevent overfitting by reducing co-adaptation among neurons [110]. During training, a dropout layer randomly sets a fraction of its input units to zero at each update step. This forces the network to learn redundant, robust representations, as it cannot rely on the presence of any single feature. Dropout applies a random binary mask, with a fixed probability. It has been shown to improve generalization, particularly in fully connected layers of deep networks. In our work, dropout was applied between hidden layers of the MLPs to reduce overfitting and promote more distributed feature representations.

6.4 Advanced Neural Networks

We now delve into specialized neural network architectures that extend the basic MLPs to handle data with specific structures: images, sequences, and in general, data with spatial or temporal locality.

6.4.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a specialized class of neural networks designed for data with spatial or grid-like structure, such as images. Unlike fully connected layers where each neuron is connected to every input unit, CNNs use small, overlapping regions of the input. This means each neuron looks at only a part of the input image (known as local connectivity), allowing the network to detect simple patterns like edges or textures. Additionally, the same set of weights is used across different parts of the image, which helps the network recognize the same pattern regardless of its position (a concept known as weight sharing). This framework allows CNNs to be computationally efficient and well-suited for extracting local patterns such as edges or textures [54].

Convolution is the name for the mathematical operation used, and is a linear combination of products. One example of a convolution operation is outlined in [54], where we are tracking the location of a spaceship with a laser sensor. The laser sensor has output $x(t)$, that provides the location of the spaceship at time t. If we assume that the laser sensor has some noise, then we it makes the most sense to average several measurements, while giving a higher precedence to more recent measurements. In this sense, we use a weight function $w(a)$, where a is the age of the measurement. We can put this all together to get a smoothed estimate of the position of the spaceship given by:

$$s(t) = \int x(a)w(t-a)da$$

The operation above is called convolution, and is typically denoted with an asterisk $*$ [54].

$$s(t) = (x * w)(t)$$

In this case, w is a valid probability density function, that must be zero for all negative values. The input to the CNN in this case is the function x. The function w is the kernel, and the output is denoted a feature map. To consider a discrete convolution, we have the following equation [54]:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

In machine learning, we typically work with discrete data represented as multidimensional arrays (tensors). The input I is often a 2D array (such as a grayscale image), and

the kernel K is another 2D array of learnable parameters. We define the 2D discrete convolution between I and K denote $(I * K)(i, j)$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

6.4.2 Kernels

To illustrate how a kernel works, it is best to start with an example. Suppose that the input to our model was a $2D$ 3×3 array (or in standard math notation an element of $\mathbb{R}^{3\times3}$). Then an example of a 2×2 kernel window is given by the following:

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix} \tag{6.13}$$

where equation (6.13) follows [128]. The kernel slides along each part of the matrix. For example, the 19 in the output is obtained from $0*0+1*1+3*2+4*3 = 19$. Similarly, we have $4 * 0 + 5 * 1 + 7 * 2 + 8 * 3 = 43$.

A kernel governs how local features are extracted from input data. Formally, a kernel is a multidimensional array of features that is convolved with the input tensor to produce an output known as a feature map. The kernel acts as a localized detector, scanning the input to identify specific patterns such as edges, corners, textures, or more abstract features in deeper layers of the network.

A key property of the kernel is that it is significantly smaller than the input. For instance, in image processing tasks, kernels commonly have spatial dimensions of 3×3, 5×5 or 7×7, while the input may be a full image with hundreds of pixels in each dimension. The kernel is applied uniformly across the input via a sliding window mechanism, which means the same set of weights is used at each location, as shown in our example above. This architectural design embodies the characteristics that are central to CNNs: local connectivity (only a small patch of the input affects each output unit) and feature sharing (the same weights are reused across spatial positions).

During training, the entries of the kernel tensor are optimized using gradient descent and backpropagation, enabling the network to learn task-relevant features directly from the data. In a given convolutional layer, multiple kernels are typically used in parallel, each learning to extract a different type of feature from the same input. The result is

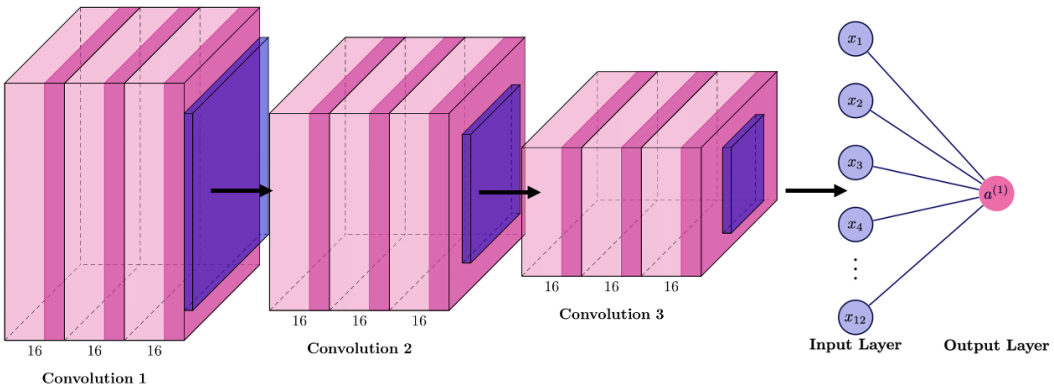


Figure 6.4: Example CNN Architecture

a collection of feature maps that together represent the presence and spatial location of various learned patterns.

6.4.3 Pooling

Often, convolutional layers are followed by pooling to reduce spatial resolution and increase robustness. Pooling takes a neighbourhood (like 2×2) and outputs a summary statistic, typically max pooling (maximum value) or average pooling. Pooling reduces the data size and computation and helps achieve some translational invariance (small shifts in input cause minimal changes after pooling) [54]. A 2×2 max-pooling, for example, down-samples the image by a factor of 2 in each dimension, taking the maximum in non-overlapping 2×2 windows. An illustration of a CNN network is given in figure 6.4. In the illustration, the convolutional layers are light pink, followed by dark pink pooling layers, purple dropout layers, followed by a 12 layer fully connected layer to make it a single numerical output for the Feynman period.

For this work, a CNN model was used where the input was not a natural image but a graph adjacency matrix interpreted as an image. Specifically, we padded each graph's adjacency matrix to 20×20 and treated it as a grayscale image. The CNN had 4 convolutional layers with 2×2 pooling, using leaky ReLU activations. This is a direct application of CNNs to graph data by imposing a grid structure, however this has limitations because

the adjacency matrix is sensitive to node labelling (permutations of nodes alter the matrix) [9].

6.4.4 Graph Neural Networks

Graph neural networks (GNNs) are neural architectures tailored to graph-structured data, where input consists of nodes and edges with arbitrary connections.

A graph $G = (V, E)$ with node set V and edge set E can be represented in neural networks by:

- Adjacency matrix A of size $n \times n$ (for $n = |V|$) where $A_{uv} = 1$ if $(u, v) \in E$ (or some weight if weighted graph), and $A_{uv} = 0$ otherwise [120].
- Node feature matrix X of size $n \times d$, where the uth row $\mathbf{x}_u$ is a feature vector of node u (could encode attributes of the node).

Early approaches to machine learning on graphs fed such representations (possibly flattened) into an MLP or CNN (like the stack model or CNN1 section 7.3.2. However, flattening the adjacency matrix loses permutation invariance (relabelling nodes permutes the matrix arbitrarily). GNNs aim to perform learning on graphs in a way that is invariant or equivariant to graph isomorphism (permutation of node labels) [120].

6.4.5 Graph Convolutional Networks (GCNs)

The Graph Convolutional Network, first introduced by Kipf & Welling in 2017 is foundational architecture for learning on graph-structured data [70]. They extend classical neural networks to graphs by incorporating information from a node's local neighbourhood at each layer. The GCN layer can be written in matrix form as:

$$H^{(t+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(t)}\Theta^{(t)})$$

Where, from [70]:

- $H^{(t)}$ is the $n \times d_t$ matrix of node embeddings at layer t (with $H^{(0} = X$ the input features).
- $\Theta^{(t)}$ is the trainable weight matrix for layer t (of size $d_t \times d_{t+1}$).

- $\tilde{A} = A + I$ is the adjacency matrix with added self-loops (each node connected to itself).
- $\tilde{D}$ is the degree matrix of $\tilde{A}$ (diagonal with $\tilde{D}uu = \sum v\tilde{A}_{uv}$).
- $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ is the symmetrically normalized adjacency matrix.
- σ is an activation function like ReLU applied element wise.

In the work, GCNS are implemented using PyTorch Geometric's GCNConv with 4 graph convolution layers and leaky ReLU [9]

6.4.6 GraphSAGE

GraphSAGE (Graph Sample and Aggregate) is a graph neural network architecture designed for inductive learning on large graphs, that is, it learns a function that can generate embeddings for previously unseen nodes without retraining the entire model [58]. This is in contrast to transductive methods (which are models that do not need to generalize to unseen examples), which require access to the full graph at inference time and cannot generalize to new nodes or graphs. This inductive capability is especially valuable in real-world applications like dynamic social or citation networks, where the graph is constantly evolving and new nodes frequently appear. For our context, this would be promising as it would suggest that we could extend to higher loop order graphs, in a way that the models could learn from smaller loop order graphs that would be easier to train.

GraphSAGE extends the neighbourhood aggregation framework by allowing for flexible aggregation functions (e.g., mean, pooling) and by sampling a fixed-size set of neighbours for each node, which keeps computations tractable on high-degree or large-scale graphs. Unlike GCNs that average neighbour messages and apply a shared weight matrix, GraphSAGE concatenates a node's own representation with its aggregated neighbours before applying the transformation, allowing the model to distinguish between self-information and contextual information. GraphSaGE also often uses a fixed number of sampled neighbours (to control computational cost in graphs with very high degree nodes). For example, sample up to 10 neighbours at random for each node instead of taking full neighbourhood, which can reduce variance and complexity.

In our work, the GraphSAGE model was used with 4 layers, similar to the GNN setup [9].

6.5 Conclusion of ML

The use of neural network in our work showcases both the promise and challenges of applying deep learning in mathematical physics contexts, which we will expand on in chapter 7. The fundamental and advanced models were all tested. Simpler models (such as linear regression) performed surprisingly well due to the highly structured nature of the problem, and modelling without the use of neural networks also proved successful [9]. More advanced architectures did not automatically discover significantly better representations given the data available and the complexity of extracting needed invariants from raw graphs. This underscores the importance of combining deep learning with domain knowledge, here the known combinatorial features of graphs, to achieve the best results.

Recently, machine learning has been used on questions that are somewhat related to, but not identical with, our own endeavour to predict Feynman integrals in quantum field theory. In [94], machine learning was used in a Feynman path integral framework to describe microbiological effects. [11] considers machine learning for graph-structured quantum states, as arise in constructing quantum computers. In [57], quantum field theory is suggested as a theoretical framework to understand neural networks. There is significant literature on the simulation or prediction of various types of scattering amplitudes in quantum field theory with machine learning methods as well, see e.g. [6, 85, 7, 64, 34]. Although the scattering amplitude in quantum field theory is expressible as a sum of all Feynman integrals, these works have in practice focused on a very different aspect of the prediction: For amplitudes, the relevant input would be the kinematics of the scattering process, and the output the amplitude, whereas in our case, the input is a single Feynman graph and the output its period, which is independent of kinematics. From a machine learning perspective, predicting periods avoids the complications of external momenta and regulator schemes, offering a cleaner and more structured target that depends only on the internal topology of the graph. This makes it an ideal object of study for understanding how graph structure encodes physically meaningful information. Moreover, Feynman periods have arithmetic structure, appearing in the study of motives and multiple zeta values, giving further theoretical motivation for their study independent of physical scattering processes. All of these reasons are motivators to using machine learning techniques for predicting Feynman periods.

Chapter 7

Period Estimation using Machine Learning

7.1 Machine learning for graphs

When considering machine learning for Feynman period estimation, a natural question is the use of graph neural networks given the natural graph structure of the problem. Graph neural networks have become a more popular topic in machine learning in the most recent years. For our present project, we aim to use a graph as an input to a model, and obtain a real number – the period (definition 5.3.1) – as an output. For the rest of this introduction, we will provide an overview of how this problem is similar to previously explored topics, but also slightly different, and some areas of consideration. Some problems discussed in the literature include, for example, *classification* (section 6.2) of graphs into finitely many sets (e.g. [77]), or using data *on* a graph (such as values assigned to vertices) as an input, or using a graph to describe the *structure* of a neural network itself, or using models that have graphs as *output* (e.g. predicting missing edges in graphs that represent friendship relations in social networks). From a machine learning perspective, our task presents a slightly different structure from many standard graph learning problems.

In graph classification tasks, the model aims to assign a discrete label to each graph from a finite set of classes, often using summary statistics or learned embeddings to reduce graph structure to a classification-ready form [77]. In node classification or graph signal processing tasks, the input includes not only the graph's structure but also additional features on the nodes or edges, and the goal is to label or predict properties of individual nodes. There are also generative graph models, where the graph itself is the output—such as in molecular

graph generation or link prediction problems. These use graph representations as either structured inputs, outputs, or both, but the learning objective is always framed within a discrete, categorical, or structured prediction context.

In contrast, our task is learning a function from unlabelled graphs, and additional features such as the Hepp bound, Martin invariant, and so forth (section 5.5), to real numbers. While we use the incidence matrix to represent the graph in our models, this representation is not inherently isomorphism-invariant: permuting the order of vertices or edges can drastically change the matrix, even though the underlying graph remains the same. This contrasts with tasks like node classification, where vertex identities are fixed across instances, and thus permutation invariance is less critical. Our use of the incidence matrix allows us to train models like MLPs and CNNs, but also forces the model to learn structural invariants from fixed but arbitrary representations, a fundamentally different challenge than those encountered in typical graph classification or node prediction pipelines. In our case, a single $L = 13$ loop graph has up to $15! > 10^{12}$ different orderings of vertices, which makes it practically impossible to use all of them. Conversely, one might want to fix one canonical order of vertices. This can be done uniquely (and we do this in all our algorithms to avoid duplicates, using `nauty` [87]), but it imposes an artificial structure on the graph, which obscures the fact that the graph really encodes *adjacencies*, and not a particular ordering of vertices.

In our particular application, we use numerous graph-related quantities as input to our machine learning models, see section 7.1.1. As stated earlier, the Hepp bound can predict the Feynman period with an average error of less than 1% [8]. It is also possible to consider how we can directly provide the machine learning model with the graph structure. In the machine learning literature, the term *embedding* is used for a mapping of a graph to some vector space of features, a survey of numerous embedding techniques can be found in [121, 55, 68, 32]. The precise definition of what counts as an embedding depends on the particular setting, commonly one demands that the vectors produced by an embedding function for two distinct input graphs are close if the graphs are *similar*, in a sense that depends on the context.

7.1.1 Input features and data sets

For the various machine learning models, we used all the features that have been found helpful in [8] and [9] for the linear models, and a few additional ones. The following is an overview of all 22 features, where references indicate where this feature has been used earlier to predict periods. See section 5.5 for an overview of these features.

1. Loop number (section 5.1.2).
2. Hepp bound (section 5.5.1, [8, Sec. 6.5], [95, 74]).
3. Whether the graph is planar.
4. Size of the automorphism group (section 5.5.2, [8, Sec. 6.1]).
5. Number of non-isomorphic decompletions (section 5.5.2, [8, Sec. 6.1]).
6. Number of planar decompletions (section 5.5.2, [8, Sec. 6.1]).
7. Symmetry factor of only the planar decompletions (section 5.5.2)
8. Number of n-vertex cuts for $n = 3, 4, 5$ (section 5.5.1).
9. Number of n-edge cuts for $n = 6, 8, 10, 12, 14$ (section 5.5.1, [8, Sec. 6.4]).
10. Number of connected n-edge cuts for $n = 6, 8, 10, 12, 14$.
11. Radius and diameter of the graph (i.e. minimum and maximum of the distance between vertices) (section 5.5.3, [8, Sec. 6.2]).
12. Mean, and first moments, of the distance between vertices (section 5.5.3, [8, Sec. 6.2]).
13. Eigenvalues of the distance matrix (section 5.5.4).
14. 22 binary parameters corresponding to whether or not an matrix eigenvalue is present for the loop order under consideration (see below).
15. Eigenvalues of the Laplacian matrix (section 5.5.4).
16. Scaled traces $\frac{\mathrm{Tr}(A^k)}{(L+2)2^k}$, for $k \leq 20$, where A is the adjacency matrix (section 5.5.4).
17. Number of n-cycles for $n = 3, \ldots, 10$.
18. Mean electrical resistance (section 5.5.5).
19. Minimum, maximum, and standard deviation, of electrical resistances between pairs of vertices. Absolute mean, minimum, and maximum, of the first 4 Ursell functions. Mean, and mean of absolute value, of the entries of the transfer current matrix (section 5.5.5).

20. First 9 entries of the autocorrelation function of random walks of length 10^4, averaged over 5000 random relabellings of the vertices (section 5.5.5).

21. Scaled logarithms of the Martin invariants $\frac{\ln(M^{[k]})}{2^{k+1}}$ (definition 5.5.4, [8, Sec. 6.6], [96]), with upper limit $k \leq 5$ depending on loop order.

22. Coefficients of the Martin polynomial (definition 5.5.3, [96]).

The size of the matrices, and therefore the number of eigenvalues, depends on the loop order of the graph. In order to have a consistent number of 22 features for each matrix, across all loop orders, we have padded the missing entries with zeros. Even if the loop order already implies how many of the features are non-trivial, we supplied the models with an explicit vector of ones and zeros to indicate whether the corresponding entries contain information or are merely padded zeros, which is feature 14 in the list.

7.2 Dataset

The number of Feynman graphs grows factorially with loop order [37, 20], and as shown in the counts given in [8]. Our dataset contains all periods up to $L = 13$ loops, but only non-complete samples for higher loop order, as illustrated by the red dots in figure 7.1. The red dots correspond to the number of numerically known 3-vertex irreducible periods, and the blue dots are the number of known periods after using all period symmetries. This is no problem as long as one only trains models for one particular loop order. However, our goal is to predict periods across different loop orders. If we merely merge all available data to train our models, $L = 13$ and $L = 14$ loops will be dominant by far. Consequently, for the models that are trained on all loop orders simultaneously, we use non-complete random samples of the data sets for $L = 13, 14$ to reduce their dominance.

We split the data so that only 80% of the data was used for training, and the reported performance of the models refers to testing the model on the remaining 20% of data. This does imply that the testing data is heavily skewed towards the 13 and 14 loop order data.

If we train the models on all available data, they are effectively mostly being trained on $\ell \in \{ 12, 13, 14, 15 \}$ loops, since most of the data has this loop order. They then show poor performance for smaller or larger loop orders. To remedy this imbalance, we only use a random subset of the graphs for training. Concretely, at 12 loops 50% of the graphs are used for training, at 13 loops 33%, and at 14 and 15 loops 10%, and all remaining

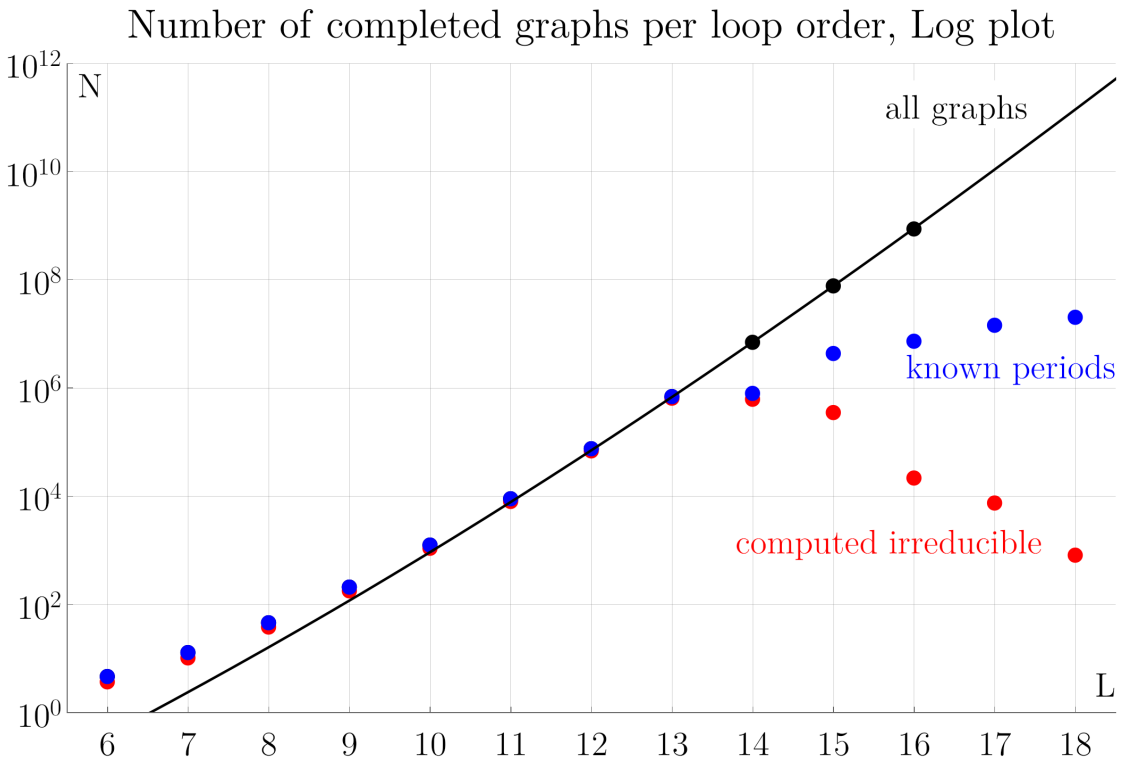


Figure 7.1: Number of Completed Feynman graphs per Loop Order. The black dots are the number of completions, which represents the total number of periods at that loop order, the black line is the asymptotics. The red points indicate the number of numerically known 3-vertex irreducible periods. Blue dots are the number of known periods after using all period symmetries

graphs are used for validation. These percentages stem from trying to make a more fair contribution from all loop orders. While this cutoff is not equal as shown in figure 7.1, these restrictions improved the accuracy of the test set. Given that the regression models showed consistent performance, we use all available graphs for them (including for loop orders 13 and 14).

7.3 Statistical Notations

We use the error measures δ, Δ as follow for measuring prediction accuracy:

Definition 7.3.1 (Relative Standard Devitation)**.** The relative standard difference between a Feynman period $\mathcal{P}$ and the estimated Feynman period $\bar{\mathcal{P}}$ is given by:

$$\delta\left(\frac{\mathcal{P}}{\bar{\mathcal{P}}}\right) = \frac{\sigma(\frac{\mathcal{P}}{\bar{\mathcal{P}}})}{\langle\frac{\mathcal{P}}{\bar{\mathcal{P}}}\rangle}$$

Definition 7.3.2 (Average Relative Difference)**.** The average relative difference between a Feynman period $\mathcal{P}$ and the estimated Feynman period $\bar{\mathcal{P}}$ is given by:

$$\Delta\left(\frac{\mathcal{P}}{\bar{\mathcal{P}}}\right) = \left\langle\sqrt{(\frac{\mathcal{P}}{\bar{\mathcal{P}}} - 1)^2}\right\rangle = \langle\frac{\mathcal{P} - \bar{\mathcal{P}}}{\bar{\mathcal{P}}}\rangle$$

To improve numerical stability, we consider rescaling the Feynman period $\mathcal{P}$ and Hepp bound $\mathcal{H}$ in accordance to the loop order ℓ.

$$\mathcal{P}'(G) = \frac{\mathcal{P}(G)}{(\frac{3}{2})^\ell \ell^{5/2}} \tag{7.1}$$

$$\mathcal{H}'(G) = \frac{\mathcal{H}(G)}{4^\ell} \tag{7.2}$$

This is referred to later as scaling the Feynman period and Hepp bound based on their asymptotic growth.

7.3.1 Standard Regression Techniques

For the machine learning techniques in this section, we used the python library `scikit-learn` [98] and its auxiliary functions such as `PolynomialFeatures` and `MinMaxScaler`.

Linear Regression

We constructed three distinct linear regression models as described in section 6.2.2 for this work. For all three of the models we created, we normalized all features $x \in \vec{x}$ (section 7.1.1) to lie within the interval $(0, 1)$. The first model **LR1** aims to predict the Feynman period directly. The second model **LR2** instead scales the period and Hepp bound to their leading asymptotic growth with loop order according to section 7.3. Consequently, for that model the Hepp bounds and periods across all loop orders are of similar magnitude. Finally, the third model **LR3** aims to predict the natural logarithm $y = \ln(\mathcal{P})$ of the period, and uses the logarithm of the Hepp bound as an input, both scaled to their asymptotic growth (section 7.3).

The *coefficient of determination* for a N-element data set is defined as

$$R^2 := 1 - \frac{\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2}{\sum_{i=1}^{N}(y_i - \langle y \rangle)^2}, \tag{7.3}$$

where $\langle y \rangle$ denotes arithmetic mean. R^2 is a measure for how much better the multi-linear function describes the data, compared to merely taking averages.

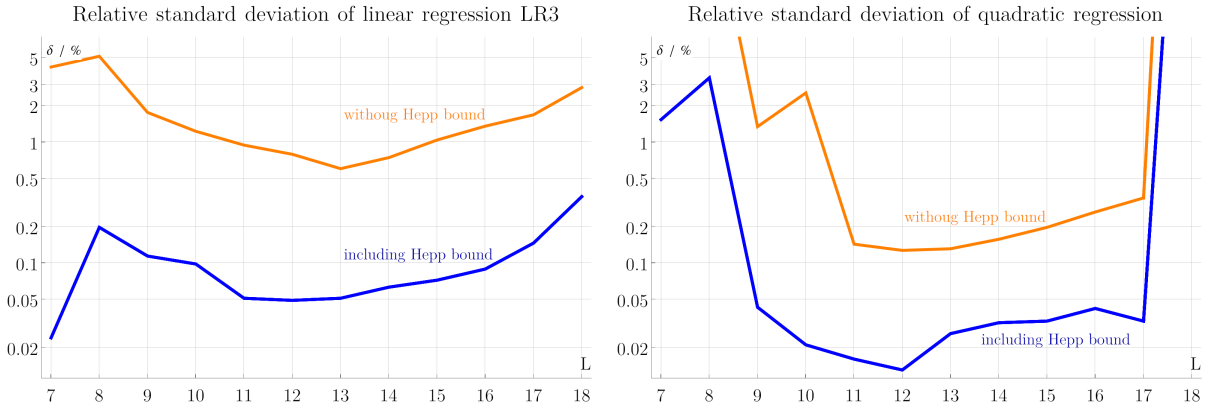
Model	R^2 Training	R^2 Testing	Δ (%)
LR1	0.9981745	0.9982034	3.37
LR2	0.9996490	0.9996573	0.04
LR3	0.9999994	0.9999994	0.04

Table 7.1: Coefficient of determination (equation (7.3)) for the three models using linear regression; average relative difference Δ (definition 7.3.2). The models LR2 and LR3, which include scaling to the asymptotics (section 7.3), are by far superior to the unscaled model LR1.

The values shown in table 7.1 suggest that scaling to the leading asymptotics by the period scaling is very helpful in order to construct a model that works across all loop orders. The accuracy of the model LR3 for individual loop orders is shown in table 7.2

From these preliminary results, we trained the remainder of our models in section 4 with the dataset whose Hepp bound and Feynman period were scaled with the leading asymptotics of the period scaling, the natural logarithm and they were further normalized to lie within $(0, 1)$. Normalization is a common preprocessing step in machine learning that ensures all input features contribute comparably to the model. A standard linear regression model was trained, and then evaluated on the test data set, which we split by loop order to evaluate Δ and δ, which are defined in section 7.3.

Recall that the Hepp bound (section 5.5.1) is a strong predictor of the Feynman period, and thus results will often be provided with and without using the Hepp bound in the training dataset.



(a) (b)

Figure 7.2: **(a)** Relative standard deviation δ (definition 7.3.1) for the linear regression model LR3. Including the Hepp bound improves accuracy by roughly one order of magnitude. Overall, δ is lower for larger the data sets than for smaller ones.
(b) Relative standard deviation δ for quadratic regression. If the data set is large enough, then the accuracy is better than the linear model (a), but the quadratic model suffers more strongly from having only small, or non-complete, data sets.

The plot in figure 7.2a shows how the linear regression model LR3 performs if it is trained with data of only one particular loop order. Performance tends to be overall better for the large data sets around $L = 13$ loops than for the smaller ones. The Hepp bound as an input feature has a large influence, improving the accuracy by roughly a factor of 10. This is expected from the findings of [8], that the Hepp bound alone is sufficient for a relatively accurate prediction. Nevertheless, even without the Hepp bound, the linear regression model reaches an accuracy of $\delta \leq 1\%$ if the data set is large enough.

	including Hepp bound		without Hepp bound	
L	$\Delta(\%)$	$\delta(\%)$	$\Delta(\%)$	$\delta(\%)$
7	0.102	0.024	7.399	4.183
8	0.141	0.197	3.861	5.136
9	0.094	0.114	1.514	1.759
10	0.079	0.098	0.927	1.229
11	0.040	0.051	0.690	0.944
12	0.033	0.049	0.530	0.793
13	0.039	0.051	0.424	0.603
14	0.048	0.063	0.552	0.744
15	0.055	0.072	0.787	1.041
16	0.070	0.089	1.056	1.356
17	0.114	0.146	1.305	1.683
18	0.276	0.354	2.201	2.830

Table 7.2: Obtained accuracies Δ and δ for the linear regression model LR3, separated by loop order L. The last two columns refer to a model where the Hepp bound has been omitted as an input feature.

Quadratic Regression

We extended standard regression techniques, to train models using quadratic regression as explained in section 7.3.1 [83]. In particular, this model for N input parameters involves not just N quadratic terms of the individual parameters, but $\frac{N(N-1)}{2}$ potentially mixed quadratic terms and N linear ones, resulting in $\frac{N^2+N+2}{2}$ fit parameters. If we were to use all 194 input features of section 7.1.1, then the model would have 18914 free fit parameters, making the model prone to overfitting. Consequently, we excluded list items 13, 14, 15 and 16 of section 7.1.1, and use only 83 features to train the model. The choices of features to discard is based on the coefficients of the features that were analyzed in the linear regression model. Using quadratic regression, we again obtain R^2 values (equation (7.3)) of better than 0.999. A model that included data from all loop orders reached $\Delta = 0.021\%$, and $\delta = 0.079\%$. For models trained only on one loop order, the performance is shown in figure 7.2b. Similar to the linear regression model (section 7.3.1), we observe that the accuracy of the quadratic model decreases by approximately one order of magnitude if the Hepp bound is not used as an input feature, suggesting that it is an important feature for the model to have.

L	$\Delta(\%)$	δ (%)
7	1.630	1.536
8	1.262	3.403
9	0.034	0.043
10	0.014	0.021
11	0.010	0.016
12	0.010	0.013
13	0.020	0.026
14	0.024	0.032
15	0.025	0.033
16	0.031	0.042
17	0.026	0.033
18	14195.258	259815.351

Table 7.3: Δ and δ by loop order L for quadratic regression.

It is clear that for the loop order 18 case, the model does not test well. This suggests that the limited size of the dataset that corresponds to loop order 18 is not enough for

the model to learn to predict the Feynman period for that loop order case. This is also a strong suggestion that the model will not generalize well to unseen loop orders (i.e., loop orders that were not used for training).

L	$\Delta(\%)$	δ (%)
6	361.829	523.333
7	57.822	76.363
8	36.218	122.359
9	0.932	1.343
10	0.339	2.555
11	0.097	0.143
12	0.079	0.127
13	0.097	0.131
14	0.121	0.157
15	0.152	0.197
16	0.203	0.264
17	0.268	0.345
18	150144.557	4851306.020

Table 7.4: Δ and δ by loop order L for quadratic regression without using the Hepp bound feature.

7.3.2 Neural Networks

In the present section, we discuss several neural network models we used to predict the period. They utilize different architectures and different input features, including five models that incorporate the structure of the Feynman graphs themselves into the prediction additionally to the features listed in section 7.1.1. This allows us to study the difference between models that only use features, models that only used the graph structure, and a mixture of both.

The neural networks used for this research were implemented using Python 3.10.4, and PyTorch 2.0.0 [97], along with mean squared error (MSE) loss function as explained in section 6.2.2, the dataset using the `RobustScaler` from `scikit learn`[98], which scales the data using the quantile range.

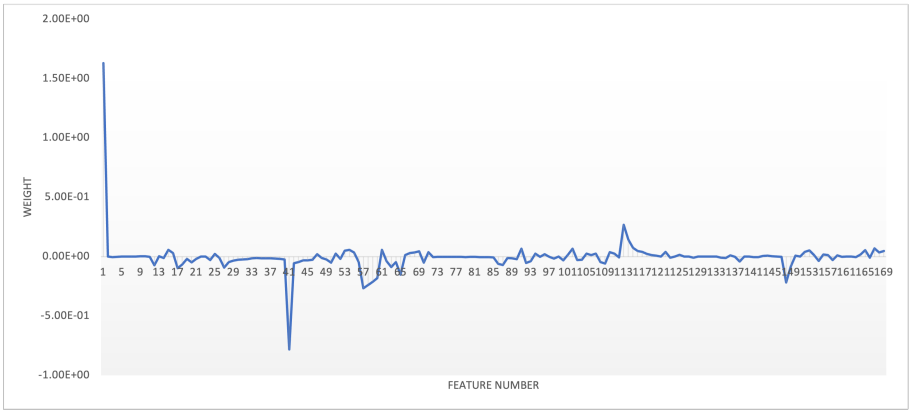


Figure 7.3: Weights of the Perceptron

Perceptron

As described in section 6.2.3, we had a perceptron to try to predict the Feynman period, which is also a good measure of how linearly separable our data is. We confirmed that the single layer perceptron resulted in a relative standard deviation δ that is similar to figure 7.2a for large data sets, but inferior for small data sets, presumably due to the iterative training being more susceptible to the influence of outliers in small data sets. When we plot the weights of the perceptron as shown in figure 7.3 it is evident that the most significant features are features 1, 41, 61 and 113 which correspond to the Hepp bound, one of the diagonals of the distance matrix and one of the random walk correlations. While the Hepp bound was always consistent, the exact diagonal of the distance matrix and random walk were subject to change based on each run of training.

Basic Model

For the *basic* model, we used a standard architecture for a neural network. Concretely, the basic model a feed-forward neural network with four fully connected linear layers, as sketched in figure 7.4, and described in section 6.3.2. It uses a ReLU activation functions between the layers followed by a batch normalization layer between all of the layers. The network includes a dropout layer with probability 0.3 between input and first hidden layer, and another dropout layer of probability 0.2 between first and second hidden layer in order to reduce susceptibility to overfitting [54]. We use the Adam optimizer with a mean squared error loss function (section 6.2.2). Everything was implemented in `PyTorch` [97].

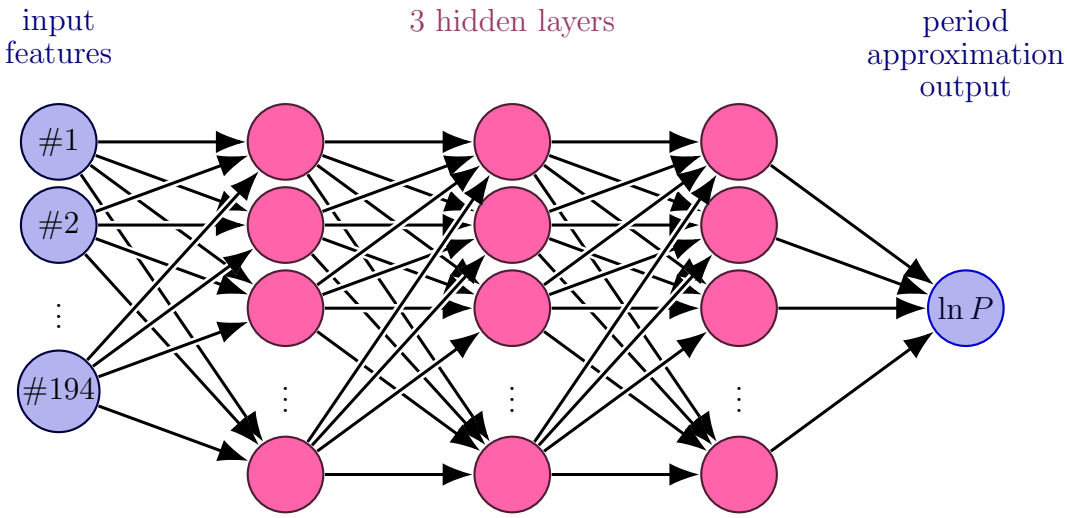


Figure 7.4: Sketch of the architecture of the basic model used in section 7.3.2. Note that it says 194 input features, given that some features (such as the matrices) require more than one spot in the array

The basic model only uses 83 input features listed in section 7.1.1, and did not use list items 13, 14, 15 and 16. Furthermore, it does not use the graph itself as an input.

The performance of the basic model is shown in figure 7.5a.

Unlike with the regression models in section 7.3.1, the performance of the basic model is only improved marginally if the Hepp bound is included. What is remarkable is that on the one hand, the regression models of section 7.3.1 are by far more accurate than the basic model, and on the other hand, they are much less susceptible to outliers and random fluctuations. In different runs of the neural network models, the performance fluctuated notably, and one occasionally observes a breakdown in accuracy as for $L = 10$ in the basic model in figure 7.5a. Note that the numbers can fluctuate based on each run.

Stack Model

The stack model uses both the features of section 7.1.1 (with or without Hepp bound), and the adjacency matrix itself, as input features. Concretely, the adjacency matrix was padded with zeros to a 20×20 matrix, flattened into a 1-dimensional vector [56], and concatenated with the initial 169 graph features to produce a vector of 569 input features

L	$\Delta(\%)$	δ (%)
7	10.181	11.528
8	11.582	11.031
9	6.586	7.877
10	13.084	100.822
11	6.939	5.298
12	7.235	7.445
13	7.056	7.664
14	5.797	6.798
15	6.664	8.110
16	7.842	9.561
17	10.595	12.977
18	8.986	11.571

Table 7.5: Δ and δ by loop order L for the basic model.

per graph. The model was trained using the same optimizer, loss function and activation function as the basic model (section 7.3.2). This model is useful to see what significance the distribution of single edges of the adjacency matrix may have on the prediction of the Feynman period by the neural network. As shown in figure 7.5a, the stack model overall shows similar performance as the basic model, and is superior only for $L \geq 10$ loops.

Models based on the graph structure

The fact that the stack model did not perform better than the basic model (section 7.3.2) suggests that the adjacency matrix, when flattened to a 1-dimensional vector, is not useful as an input feature. In this final section, we briefly comment on three different types of artificial neural networks that operate more closely to the graph structure itself.

Firstly, a convolutional neural network (CNN) (section 6.4.1) processes a 2-dimensional input array by convoluting it with a 2-dimensional kernel function [54], where the parameters of the kernel are subject to training as described in section 6.4.1.

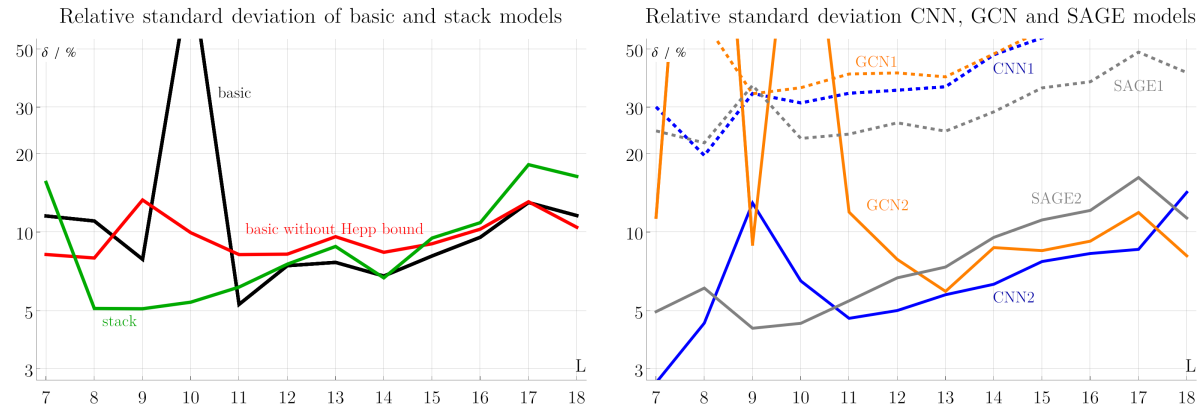
CNNs are commonly used for image classification and computer vision, for a review see e.g. [2], or section 6.4.1. In our case, we used the adjacency matrix, padded with zeros to have a size of 20×20, as an input *image*. We chose a CNN architecture with

L	$\Delta(\%)$	δ (%)
7	11.805	8.216
8	8.310	7.966
9	10.623	13.272
10	7.063	9.950
11	7.409	8.208
12	7.099	8.240
13	7.777	9.609
14	6.640	8.375
15	8.029	9.029
16	9.055	10.269
17	11.529	13.066
18	9.392	10.439

Table 7.6: Δ and δ by loop order L for the basic model without using the Hepp bound feature.

L	$\Delta(\%)$	δ (%)
7	16.052	15.492
8	4.205	5.511
9	4.643	5.098
10	4.988	5.402
11	4.374	6.168
12	5.995	7.545
13	6.642	8.818
14	4.923	6.689
15	7.935	9.496
16	8.880	10.872
17	14.961	18.107
18	12.959	16.326

Table 7.7: Δ and δ by loop order L for the stack model.



(a) (b)

Figure 7.5: **(a)** Relative standard deviation δ (definition 7.3.1) for the basic model (section 7.3.2) and the stack model (section 7.3.2). These models overall perform similarly, except from random fluctuation such as $L = 10$ for the basic model.
(b) Relative standard deviation δ for CNN (section 6.4.1), GCN (section 6.4.5) and SAGE (section 6.4.6) models. For each of the three, the 1-version (dashed) is the pure model, whereas the 2-version additionally use the input features of the basic model. We see that adding the more advanced models gives only small increases in accuracy compared to the basic model, and all these models are by far inferior to regression models (compare figure 7.2)

4 convolutional layers, and then pooled twice using a 2×2 pooling layer. Pooling can reduce computational effort and help eliminate noise from the data. Two CNN models were implemented. CNN1 is the actual CNN itself, as described above. For CNN2, we combined CNN1 with the basic model (section 7.3.2) and use a weighted average of the two as prediction.

The performance of the two CNN models is shown in figure 7.5b. CNN2 reaches δ between 5% and 10%, similar to the basic model in figure 7.5a, whereas CNN1 alone performs notably worse. From this we conclude that the CNN itself is essentially unable to predict the period. A probable reason for this is that the adjacency matrix is not invariant under vertex relabelings, and the CNN, which essentially treats the adjacency matrix as an *image*, was unable to incorporate this fact and therefore failed to infer the properties of the underlying graph from the given matrix.

In a graph convolutional network (GCN), the convolution is computed for adjacent vertices, following the edges of the graph, instead of for adjacent pixels of an input array.

L	CNN1 Δ(%)	CNN1 δ (%)	CNN2 Δ(%)	CNN2 δ (%)
7	86.157	30.073	21.739	2.661
8	37.668	19.642	15.247	4.494
9	37.981	33.854	12.952	12.924
10	26.615	31.168	10.563	6.509
11	27.962	33.934	9.153	4.686
12	27.529	34.851	7.940	5.019
13	27.879	35.931	7.384	5.767
14	38.696	47.590	6.043	6.324
15	46.611	55.235	7.394	7.723
16	50.717	61.173	7.772	8.288
17	65.681	76.400	7.683	8.596
18	59.486	71.259	7.721	14.196

Table 7.8: Δ and δ by loop order L for the stack model.

Thereby, the GCN truly captures the adjacency relations of the graph itself, and not just of some placeholder quantity like the adjacency matrix. For an explanation, review section 6.4.5. For our model GCN1, we used 4 GCNConv layers with leaky ReLU activation function from the `PyTorch Geometric` library as convolutional layers, and additionally 1 pooling layer and 3 dropout layers. The vertices where labelled with integers. The model GCN2 combines this graph convolutional network with the basic model (section 7.3.2)

Recall that the Graph Sample and Aggeregrate (GraphSAGE) model is an architecture that samples neighbours from the nodes of the graph and aggregates features from the node's neighbour to create an embedding as described in section 6.4.6. As for the GCN, the node features were taken to be integer vertex labels. The SAGE1 model used four GraphSAGE convolutional layers, 1 pooling layer and 3 dropout layers. The SAGE2 model combines SAGE1 with a basic model. The concept of combining both models was to see if the network could figure out anything useful about the graph structure, and use that in addition to the features we discussed in section 5.5, in order to improve its Feynman period predictions. As shown in figure 7.5b, the GCN and SAGE models perform similarly to the CNN.

L	GCN 1 $\Delta(\%)$	GCN 1 $\delta(\%)$	GCN 2 $\Delta(\%)$	GCN 2 $\delta(\%)$
7	305.555	60.308	15.684	11.389
8	242.612	63.766	929.124	2761.242
9	74.401	33.778	7.418	8.996
10	35.381	35.626	37.960	416.805
11	32.017	40.190	8.990	11.927
12	31.826	40.574	8.863	7.867
13	30.254	39.195	7.612	5.940
14	37.145	47.966	7.863	8.731
15	45.245	58.930	6.252	8.500
16	48.201	61.984	7.337	9.237
17	63.091	78.585	9.722	11.886
18	54.540	67.299	8.130	8.119

Table 7.9: Δ and δ for GC1 and GC2

L	SAGE1 $\Delta(\%)$	SAGE1 $\delta(\%)$	SAGE2 $\Delta(\%)$	SAGE2 $\delta(\%)$
7	115.178	24.368	7.753	4.969
8	96.156	21.938	16.646	6.144
9	59.639	36.368	10.756	4.294
10	33.945	22.800	9.018	4.485
11	30.033	23.683	7.140	5.464
12	30.510	26.159	8.304	6.690
13	27.840	24.310	8.543	7.358
14	35.137	28.812	8.763	9.550
15	43.210	35.571	9.773	11.133
16	46.936	37.545	10.827	12.105
17	63.754	48.724	15.244	16.167
18	57.125	40.700	10.369	11.325

Table 7.10: Δ and δ for SAGE1 and SAGE2

Chapter 8

Conclusion

This thesis has explored the intersection of combinatorics and physics, through causal set theory and the prediction of Feynman periods using machine learning approaches.

In the first part, we investigated structural aspects of causal set theory, by looking at the order dimension two case of posets and proving results on the structures of downsets of a fixed size of a poset. We introduced the covtree framework for modelling covariant growth, and defined a witness reconstruction problem, highlighting its special structure in the order dimension two case. Our results contribute to a clearer understanding of how local information (such as the set of downsets of fixed cardinality) can determine global causal structure.

The second part of the thesis turned to the combinatorics of Feynman diagrams, particularly the task of predicting Feynman periods associated to primitive graphs in ϕ^4 theory. By leveraging properties such as Kirchhoff polynomials, Hepp bounds, and graph invariants, we constructed datasets that informed machine learning models, both classical and neural network based. We further validated that certain graph-theoretic features serve as strong predictors of Feynman periods, such as the Hepp bound, and edge-cuts.

Taken together, these studies reflect a broader theme: that discrete structures, particularly posets and graphs, offer a powerful lens through which to understand the foundations of physics. While causal set theory seeks to discretize spacetime itself, and Feynman integrals encode quantum amplitudes via graph polynomials, both domains benefit from a combinatorial perspective. In both cases, we showed how problems traditionally viewed as computational or physical can be reformulated as questions about order, structure, and reconstruction.

There remain many open directions. In causal set theory, the extension of witness reconstruction techniques in the order dimension two case. In the Feynman periods context, more work on the generalization of the models we used here, and on the use of machine learning for Feynman periods for higher loop orders would further enhance this research. Thank you for reading my thesis!

References

[1] D. Ahmedt-Aristizabal, M.A. Armin, S. Denman, C. Fookes, and L. Petersson. A survey on graph-based deep learning for computational histopathology. *Computerized Medical Imaging and Graphics*, 95, 2022.

[2] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaria, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, 2021.

[3] Ibrahim Assem, Andrzej Skowronski, and Daniel Simson. *Elements of the representation theory of associative algebras: Volume 1: Techniques of representation theory*, volume 65. Cambridge University Press, 2006.

[4] Mariette Awad, Rahul Khanna, Mariette Awad, and Rahul Khanna. Support vector regression. *Efficient learning machines: Theories, concepts, and applications for engineers and system designers*, pages 67–80, 2015.

[5] Ali Azimi and Mohammad Farrokhi Derakhshandeh Ghouchan. Explicit formulas for matrices associated to ladder, circular ladder, and Mobius ladder graphs. (arXiv:2306.10497), June 2023. arXiv:2306.10497.

[6] Simon Badger and Joseph Bullock. Using neural networks for efficient evaluation of high multiplicity scattering amplitudes. *Journal of High Energy Physics*, 2020(6):114, June 2020.

[7] Simon Badger, Anja Butter, Michel Luchmann, Sebastian Pitz, and Tilman Plehn. Loop amplitudes from precision networks. *SciPost Physics Core*, 6(2):034, April 2023.

[8] Paul-Hermann Balduf. Statistics of Feynman amplitudes in ϕ^4-theory. *Journal of High Energy Physics*, 2023.11:160, November 2023.

[9] Paul-Hermann Balduf and Kimia Shaban. Predicting Feynman periods in ϕ^4-theory. *Journal of High Energy Physics*, 2024:38, 2024.

[10] Paul-Hermann Balduf and Johannes Thürigen. Primitive asymptotics in ϕ^4 vector theory. *arXiv preprint arXiv:2412.08617*, 2024.

[11] K. Beer, M. Khosla, J. Köhler, T.J. Osborne, and T. Zhao. Quantum machine learning of graph-structured data. *Physical Review A*, 108(1), 2023.

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[13] Dionigi M. T. Benincasa and Fay Dowker. Scalar curvature of a causal set. *Phys. Rev. Lett.*, 104:181301, 2010.

[14] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant Subgraph Aggregation Networks. *arXiv:2110.02910 [cs, stat]*, March 2022.

[15] Garrett Birkhoff. *Lattice theory*, volume 25. American Mathematical Soc., 1940.

[16] Spencer Bloch, Hélène Esnault, and Dirk Kreimer. On Motives Associated to Graph Polynomials. *Communications in Mathematical Physics*, 267(1):181–225, October 2006.

[17] Béla Bollobás. A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs. *European Journal of Combinatorics*, 1(4):311–316, December 1980.

[18] Luca Bombelli, Joe Henson, and Rafael D Sorkin. Discreteness without symmetry breaking: A theorem. *Modern Physics Letters A*, 24(32):2579–2587, 2009.

[19] Luca Bombelli, Joohan Lee, David Meyer, and Rafael D. Sorkin. Space-time as a causal set. *Phys. Rev. Lett.*, 59:521–524, 1987.

[20] Michael Borinsky. Renormalized asymptotic enumeration of Feynman diagrams. *Annals of Physics*, 385:95–135, October 2017.

[21] Michael Borinsky. Tropical Monte Carlo quadrature for Feynman integrals. *Annales de l'Institut Henri Poincaré D*, 10(4):635–685, January 2023.

[22] Michael Borinsky, Henrik J. Munch, and Felix Tellander. Tropical Feynman integration in the Minkowski regime. (arXiv:2302.08955), February 2023.

[23] Michael Borinsky and Oliver Schnetz. Recursive computation of Feynman periods. *Journal of High Energy Physics*, 2022(8):1–83, 2022.

[24] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[25] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[26] D. J. Broadhurst and D. Kreimer. Knots and Numbers in ϕ^4 Theory to 7 Loops and Beyond. *International Journal of Modern Physics C*, 06(04):519–524, August 1995.

[27] Francis Brown. Periods and Feynman amplitudes. *arXiv preprint arXiv:1512.09265*, 2015.

[28] Francis Brown and Oliver Schnetz. A k_3 in ϕ^4. *arXiv preprint arXiv:1006.4064*, 2010.

[29] Francis C. S. Brown. On the periods of some Feynman integrals. 2010.

[30] James Robert Brown. How do Feynman diagrams work? *Perspectives on Science*, 26(4):423–442, 2018.

[31] E. Brézin, J. C. Le Guillou, and J. Zinn-Justin. Perturbation theory at large order. I. The φ^{2N} interaction. *Physical Review D*, 15(6):1544–1557, March 1977.

[32] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, September 2018.

[33] Diego Caudillo, Joscha Diehl, Kurusch Ebrahimi-Fard, and Emanuele Verri. Signatures of graphs for bicommutative Hopf algebras, February 2023.

[34] Ibrahim Chahrour and James Wells. Comparing machine learning and interpolation methods for loop-level calculations. *SciPost Physics*, 12(6):187, June 2022.

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

[36] Iain Crump. Graph invariants with connections to the Feynman period in ϕ^4 theory. *arXiv preprint arXiv:1704.06350*, 2017.

[37] Predrag Cvitanović, B. Lautrup, and Robert B. Pearson. Number and weights of Feynman diagrams. *Physical Review D*, 18(6):1939, 1978.

[38] Hal Daumé. *A course in machine learning*. Alanna Maldonado, 2023.

[39] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, 1997.

[40] Bernard Derrida, JP Bouchaud, and Antoine Georges. Introduction to random walks. In *Disorder and Mixing: Convection, Diffusion and Reaction in Random Materials and Processes*, pages 1–18. Springer, 1988.

[41] Fay Dowker. Causal sets and the deep structure of spacetime. In Abhay Ashtekar, editor, *100 years of relativity*, pages 445–464. 2005.

[42] Fay Dowker, Nazireen Imambaccus, Amelia Owens, Rafael Sorkin, and Stav Zalel. A manifestly covariant framework for causal set dynamics. *Classical and Quantum Gravity*, 37(8):085003, 2020.

[43] Fay Dowker and Stav Zalel. Evolution of universes in causal set cosmology. *Comptes Rendus. Physique*, 18(3-4):246–253, 2017.

[44] Peter G Doyle and J Laurie Snell. *Random Walks and Electric Networks*, volume 22 of *Carus Mathematical Monographs*. Mathematical Association of America, Washington DC, 2006.

[45] Ben Dushnik and Edwin W Miller. Partially ordered sets. *American journal of mathematics*, 63(3):600–610, 1941.

[46] Konrad Engel. *Sperner Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1997.

[47] Michael J Evans and Jeffrey S Rosenthal. Probability and Statistics: The Science of Uncertainty. 2023.

[48] Dowker Fay and Zalel Stav. Evolution of universes in causal set cosmology. *Comptes Rendus. Physique*, 18(3-4):246–253, 2017.

[49] Stefan Felsner, William T Trotter, and Veit Wiechert. The dimension of posets with planar cover graphs. *Graphs and Combinatorics*, 31(4):927–939, 2015.

[50] Loïc Foissy. Plane posets, special posets, and permutations. *Advances in Mathematics*, 240:24–60, 2013.

[51] Jane Gao. Tutte evolution of random graph orders and their dimensions, Jan 2025.

[52] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[53] Marcel Golz, Erik Panzer, and Oliver Schnetz. Graphical functions in parametric space. *Letters in Mathematical Physics*, 107:1177–1192, 2017.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[55] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, July 2018.

[56] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, July 2018.

[57] J. Halverson, A. Maiti, and K. Stoner. Neural networks and quantum field theory. *Machine Learning: Science and Technology*, 2(3), 2021.

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

[59] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 1025–1035, Red Hook, NY, USA, December 2017. Curran Associates Inc.

[60] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

[61] Klaus Hepp. Proof of the Bogoliubov-Parasiuk theorem on renormalization. *Communications in Mathematical Physics*, 2(1):301–326, December 1966.

[62] Simone Hu, Oliver Schnetz, Jim Shaw, and Karen Yeats. Further investigations into the graph theory of ϕ^4-periods and the c^2 invariant. *Annales de l'Institut Henri Poincaré D*, 9(3):473–524, 2022.

[63] James E. Humphreys. *Reflection Groups and Coxeter Groups*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1990.

[64] Philip Ilten, Tony Menzo, Ahmed Youssef, and Jure Zupan. Modeling hadronization using machine learning. *SciPost Physics*, 14(3):027, March 2023.

[65] Paul Kainen, V Krková, and Andrew Vogt. Best approximation by Heaviside perceptron networks. *Neural Networks*, 13(7):695–697, 2000.

[66] David Kaiser. Feynman diagrams. In *Compendium of Quantum Physics*, pages 235–239. Springer, 2009.

[67] Adrien Kassel and Wei Wu. Transfer current and pattern fields in spanning trees. *Probability Theory and Related Fields*, 163(1):89–121, October 2015.

[68] Megha Khosla, Vinay Setty, and Avishek Anand. A Comparative Study for Unsupervised Network Representation Learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[69] U-Rae Kim, Sungwoong Cho, and Jungil Lee. The art of Schwinger and Feynman parametrizations. *Journal of the Korean Physical Society*, pages 1–17, 2023.

[70] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR 2017*, February 2017.

[71] Gustav Kirchhoff. Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

[72] Steve Kirkland, Michael Neumann, and Bryan Shader. Distances in Weighted Trees and Group Inverse of Laplacian Matrices. *Siam Journal on Matrix Analysis and Applications - SIAM J MATRIX ANAL APPLICAT*, 18, October 1997.

[73] D. J. Klein and M. Randić. Resistance distance. *Journal of Mathematical Chemistry*, 12(1):81–95, December 1993.

[74] Mikhail V. Kompaniets and Erik Panzer. Minimally subtracted six loop renormalization of $o(n)$ -symmetric ϕ^4 theory and critical exponents. *Physical Review D*, 96(3):036016, August 2017.

[75] Maxim Kontsevich and Don Zagier. Periods. In *Mathematics Unlimited - 2001 and Beyond*, pages 771–808. Springer, 2001.

[76] Dirk Kreimer. Combinatorics of (perturbative) quantum field theory. *Physics Reports*, 363(4-6):387–424, 2002.

[77] A. Kroll, S. Ranjan, M.K.M. Engqvist, and M.J. Lercher. A general model to predict small molecule substrates of enzymes based on machine and deep learning. *Nature Communications*, 14(1), 2023.

[78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019.

[79] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[80] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying ReLu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[81] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[82] Claudia Malvenuto and Christophe Reutenauer. A self-dual hopf algebra on double partially ordered sets. *arXiv preprint arXiv:0905.3508*, 2009.

[83] Dastan Maulud and Adnan Mohsin Abdulazeez. A Review on Linear Regression Comprehensive in Machine Learning. *Journal of Applied Science and Technology Trends*, 1:140–147, December 2020.

[84] D. Maître and H. Truong. One-loop matrix element emulation with factorisation awareness. *Journal of High Energy Physics*, 2023(5):159, May 2023.

[85] Daniel Maître and Henry Truong. A factorisation-aware Matrix element emulator. *Journal of High Energy Physics*, 2021(11):66, November 2021.

[86] A. J. McKane, D. J. Wallace, and O. F. de Alcantara Bonfim. Non-perturbative renormalisation using dimensional regularisation: Applications to the ε expansion. *Journal of Physics A: Mathematical and General*, 17(9):1861–1876, June 1984.

[87] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, January 2014.

[88] Brendan D. McKay, Nicholas C. Wormald, and Beata Wysocka. Short Cycles in Random Regular Graphs. *The Electronic Journal of Combinatorics*, 11(1):R66, September 2004.

[89] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[90] David Alan Meyer. *The dimension of causal sets*. PhD thesis, Massachusetts Institute of Technology, 1988.

[91] Sebastian Mizera. Scattering with Neural Operators. *Physical Review D*, 108(10):L101701, November 2023.

[92] Sunil Mukhi. String theory: a perspective over the last 25 years. *Classical and Quantum Gravity*, 28(15):153001, 2011.

[93] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991.

[94] Huber Nieto-Chaupis. The Feynman path integral and machine learning algorithms to characterize and anticipate bacteria chemotaxis in a host healthy body. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0969–0973, 2019.

[95] Erik Panzer. Hepp's bound for Feynman graphs and matroids. *Annales de l'Institut Henri Poincaré D*, 10(1):31–119, December 2022.

[96] Erik Panzer and Karen Yeats. Feynman symmetries of the Martin and c_2 invariants of regular graphs, April 2023.

[97] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan

Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[98] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[99] Michael E Peskin. *An Introduction to quantum field theory*. CRC press, 2018.

[100] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[101] D. P. Rideout and R. D. Sorkin. Classical sequential growth dynamics for causal sets. *Phys. Rev. D*, 61:024002, Dec 1999.

[102] Frank Rosenblatt. The perceptron: A perceiving and recognizing automaton. Report, Project PARA, Cornell Aeronautical Laboratory, January 1957.

[103] Carlo Rovelli. *Quantum Gravity*. Cambridge University Press, 2004.

[104] Carlo Rovelli. Loop quantum gravity. *Living Reviews in Relativity*, 11:1–69, 2008.

[105] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. In *Backpropagation*, pages 1–34. Psychology Press, 2013.

[106] Oliver Schnetz. Quantum periods: A census of ϕ^4-transcendentals. *Commun.Num.Theor.Phys.*, 4:1–48, 2010.

[107] Oliver Schnetz. Numbers and Functions in Quantum Field Theory. *Physical Review D*, 97(8):085018, April 2018.

[108] Walter Schnyder. Planar graphs and poset dimension. *Order*, 5:323–343, 1989.

[109] Chris Sherlock and Gareth Roberts. Optimal scaling of the random walk Metropolis on elliptically symmetric unimodal targets. *Bernoulli*, 15(3):774–798, August 2009.

[110] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[111] Richard P Stanley. Enumerative combinatorics volume 1 second edition. *Cambridge studies in advanced mathematics*, 2011.

[112] Felsner Stefan and Massow Mareike. Linear extension diameter of downset lattices of 2-dimensional posets. *Electronic Notes in Discrete Mathematics*, 34:313–317, 2009.

[113] Sumati Surya. The causal set approach to quantum gravity. *Living Reviews in Relativity*, 22:1–75, 2019.

[114] Yves Tillé. *Sampling and Estimation from Finite Populations*. John Wiley & Sons, Ltd, 2020.

[115] Csaba Toth, Darrick Lee, Celia Hacker, and Harald Oberhauser. Capturing Graphs with Hypo-Elliptic Diffusions, May 2022.

[116] William T Trotter. *Combinatorics and partially ordered sets*. Johns Hopkins University Press, 1992.

[117] William T Trotter Jr and John I Moore Jr. The dimension of planar posets. *Journal of Combinatorial Theory, Series B*, 22(1):54–67, 1977.

[118] H. D. Ursell. The evaluation of Gibbs' phase-integral for imperfect gases. *Mathematical Proceedings of the Cambridge Philosophical Society*, 23(6):685–697, April 1927.

[119] Harry Wiener. Structural Determination of Paraffin Boiling Points. *Journal of the American Chemical Society*, 69(1):17–20, January 1947.

[120] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021.

[121] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):40–51, 2007.

[122] Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

[123] Karen Yeats. *A combinatorial perspective on quantum field theory.* Springer, 2017.

[124] Karen Yeats and Stav Zalel. Hopf algebras from poset growth models. *The Electronic Journal of Combinatorics*, 31, 2024.

[125] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNNExplainer: Generating Explanations for Graph Neural Networks. In *33rd Conference on Neural Information Processing Systems*, Vancouver, 2019.

[126] Stav Zalel. The structure of covtree: searching for manifestly covariant causal set dynamics. *Classical and Quantum Gravity*, 38(1):015001, 2020.

[127] Anthony Zee. *Quantum field theory in a nutshell*, volume 7. Princeton university press, 2010.

[128] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. *Dive into deep learning.* Cambridge University Press, 2023.