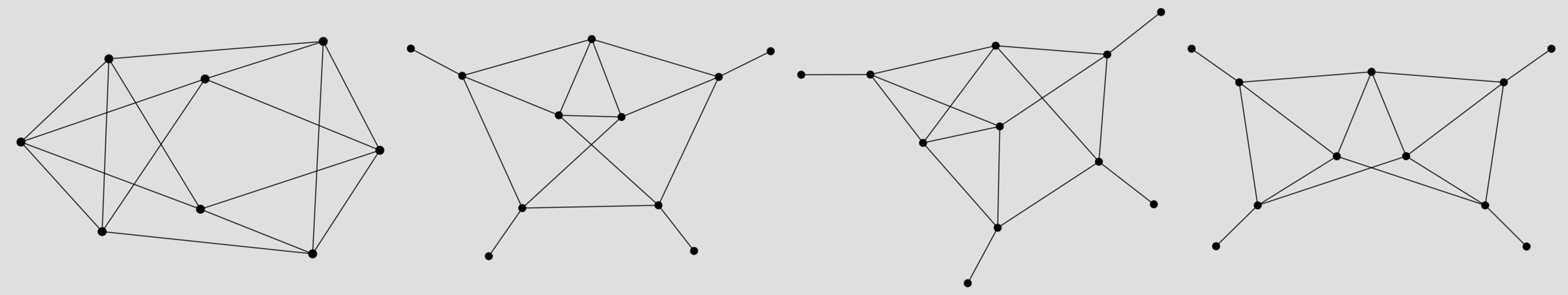
Predicting Feynman Periods with Machine Learning

Kimia Shaban, Paul-Hermann Balduf

University of Waterloo, Department of Combinatorics & Optimization

Physical Background

Quantum field theory (QFT) describes the fundamental constituents of matter and all fundamental forces except gravity. The probability of a scattering process can be computed with the help of *Feymnman integrals*. These integrals are indexed by *Feynman graphs*.



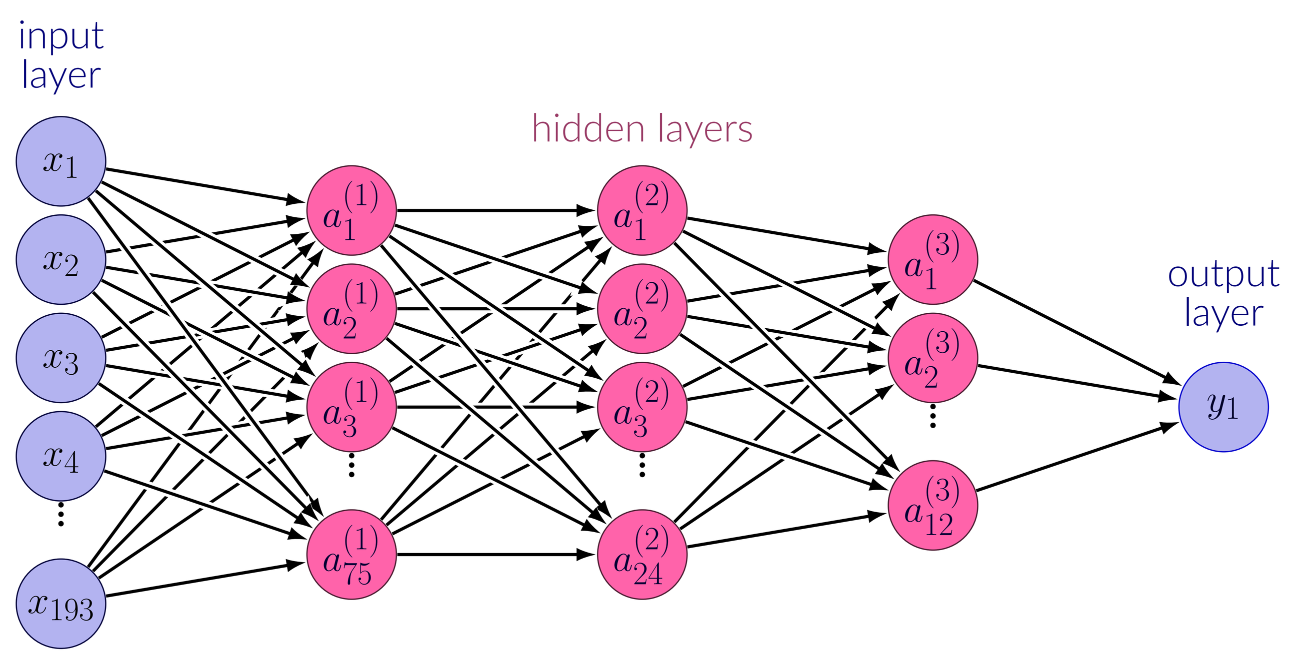
We consider Feynman graphs that represent quantum corrections to a $2 \to 2$ scattering process of ϕ^4-theory. This theory is a simplified model theory consisting of only one type of particle and where vertices in Feynman graphs can only be 4-valent. A 4-regular graph is a *completion*, a graph with 4 1-valent vertices is a *decompletion*. The *Feynman period* of a primitive graph is the dependence of its scattering amplitude on the energy scale.

$$\mathcal{P}(G) = \left(\prod_{e \in E_G} \int_0^\infty \mathrm{d}\, a_e \right) \delta \left(1 - \sum_{e \in E_G} a_e \right) \frac{1}{\psi_G^2}. \tag{1}$$

The Symanzik polynomial ψ_G is the sum over all spanning trees, and consists of the edge variables $a_{e_1} a_{e_2} \cdots a_{e_L}$ for the edges $\{e_1, \ldots, e_L\}$ *not* in the spanning tree. For a L-loop graph, it is of degree L.

Unlike more general Feynman integrals, the period has the advantage that it is a single finite number, not a function of momenta of external particles, and therefore easy to handle numerically. Many numerical [1] and analytical [2, 3] results are known. Periods are also of interest in number theory [4], they form a class of numbers that exceeds $\mathbb{Q}$ and does not exhaust $\mathbb{R}$.

At 16 loops, there are around 1 billion non-isomorphic completions. It is impossible to numerically compute *all* their periods. If we know an approximate value of the period beforehand, we can select the most important ones.



Implementation of Neural Network

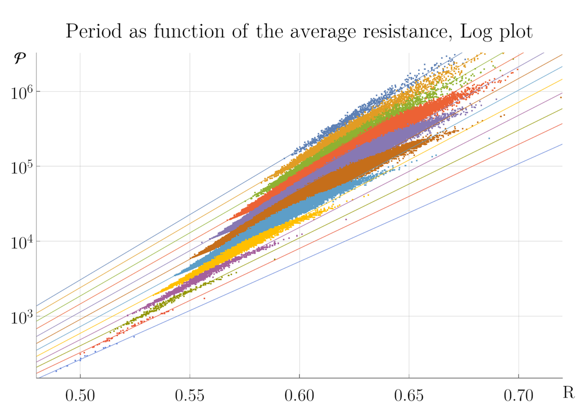
- Given a large dataset with approximately 2 million Feynman periods computed, and 193 features for each Feynman graph.
- Used a multi-layer feedforward neural network to predict the Feynman period.
- Applied a sigmoid activation function between each layer of the neural network.
- Key packages: Python 3.10.4, PyTorch 2.0.0, torch-geometric 2.3.0.
- Applied Adam optimizer from PyTorch with the default settings.
- Used mean squared error (MSE) loss function to compute the difference between our input **x**, and the Feynman periods **y** to compute:

$$\frac{1}{n} \sum_{i=1}^{n} ||f(\mathbf{x}_i, \mathbf{W}) - y_i||^2 \tag{2}$$

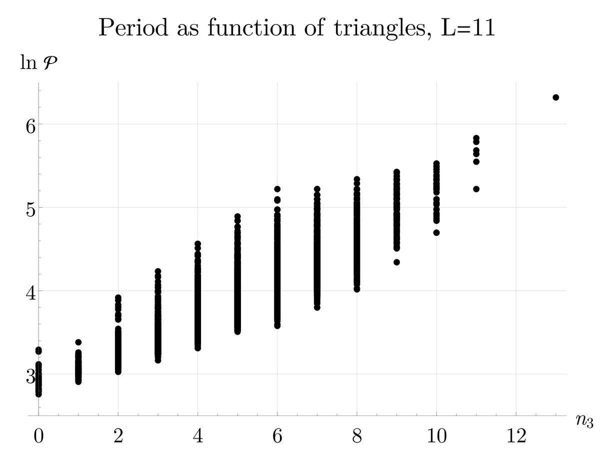
for optimized weights **W**, and a model **f**.

Features of Feynman Graphs

Apart from the graph itself (via the incidence matrix), 193 features of each graph were used to construct the datasets.



- Dimension of cycle space: *loop order* $L = |V| - 2$.
- Size of automorphism group (*symmetry factor*).
- Number of non-isomorphic decompletions, how many are planar, their symmetry factor.
- Number of ways the graph can be cut by removing r edges, for various r.
- Number of ways the graph can be cut by removing r edges such that one obtains exactly 2 connected components.
- Number of cycles of a fixed length l, for various l.
- Number of ways the graph can be turned into c disjoint cycles by splitting vertices (*circuit partition polynomial*).
- Mean and moments of the distribution of distances between any two vertices.
- Mean and moments of the distribution of resistances between any two vertices, if the graph were an electrical network where every edge has unit resistance.
- Traces and Eigenvalues of various graph matrices.
- A simplification of equation (1) where the Symanzik polynomial is replaced with the single largest monomial (*Hepp bound*).



Four Different Models

Basic Neural Network

- 4 linear layers of size (193, 75, 24, 12).
- Did not use incidence matrix or other information about the structure of graph.

Convolutional Neural Network

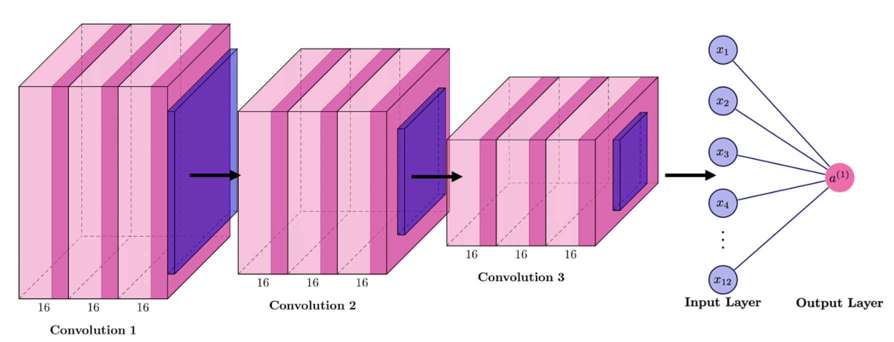
- Includes all features of the *basic* model, with the incidence matrices trained on a convolutional neural network.
- At each convolutional layer, we use hidden layer $H^{l+1} = \sigma(DADH^l W^l)$ [5]. In this case $D = (\sum_j A_{i,j})^{-1/2}$, where A is the incidence matrix. H^l and W^l represent the hidden layer, and weights used for the l^{th} convolutional layer.

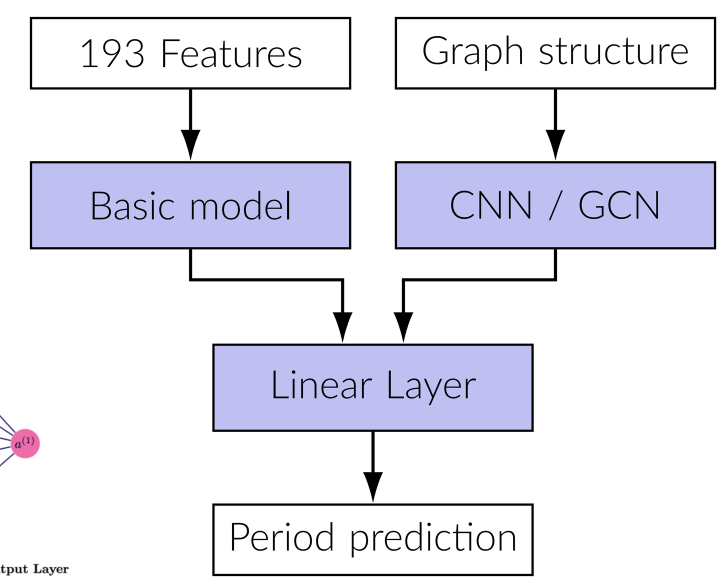
Stack Neural Network

- 4 linear layers of size (193, 75, 24, 12).
- Uses flattened incidence matrix.
- Useful to see what significance the distribution of single edges of the incidence matrix may have.

Graph Convolutional Network

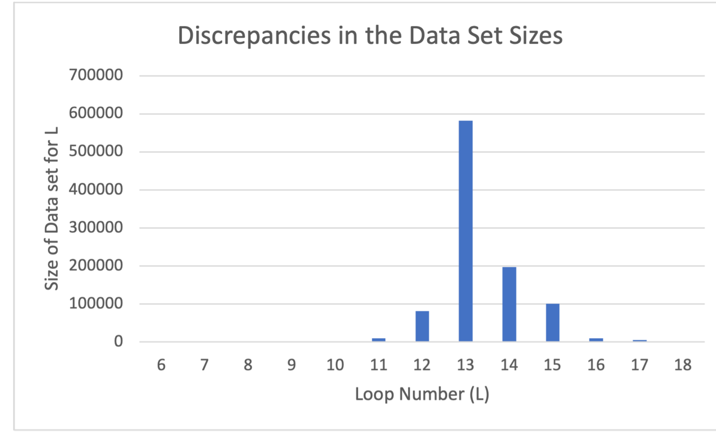
- 4 linear layers of size (193, 75, 24, 12).
- Data set includes edge-index matrices.
- Similar to CNN, a linear layer was used to account for both predictions, and a final prediction was made using a linear layer.





Basic Model with Weighted Data

The data sets of the different loop numbers are very different in size. If a model is trained on all data, it effectively uses almost only 13 loops. To compensate the bias, the loss functions scales each data set by $1 - N_L \cdot T^{-1}$, where N_L represents the total number of Feynman graphs of loop order L used to train the model, and T represents the total number of Feynman graphs of all loop orders used. The loss function is calculated as:



$$\left(1 - \frac{N_L}{T}\right) \frac{1}{T} \sum_{i=1}^{T} (f(\mathbf{x}_i, \mathbf{W}) - y_i)^2 \tag{3}$$

By employing this weighted training approach, greater emphasis is placed on those data entries that were underrepresented due to their smaller dataset size.

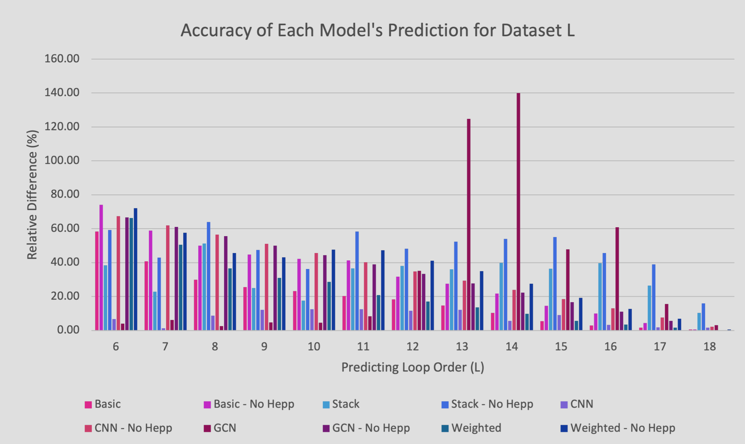
Results

	Average Relative Prediction Error												
Loop order	6	7	8	9	10	11	12	13	14	15	16	17	18
Basic	58.27	40.78	29.89	25.45	23.18	20.35	18.20	14.62	10.29	5.35	2.89	1.61	0.53
Stack	38.45	22.89	51.32	24.93	17.62	36.59	37.94	36.05	39.77	36.41	39.59	26.51	10.32
CNN	6.68	1.30	8.62	12.20	12.50	12.49	11.65	12.09	5.66	9.08	3.34	1.80	1.71
GCN	3.96	6.18	2.44	4.66	4.53	8.30	35.17	124.8	140.0	47.80	60.80	15.55	3.02
Weighted	66.28	50.46	36.62	31.01	28.55	20.86	17.06	13.65	9.71	5.64	3.37	1.66	0.11

Table 1. Relative Error of Prediction With Hepp Bound

	Average Relative Prediction Error												
Loop order	6	7	8	9	10	11	12	13	14	15	16	17	18
Basic	74.06	58.83	49.92	44.80	42.27	41.24	31.73	27.47	21.71	14.53	9.86	4.33	0.57
Stack	59.25	42.86	63.92	47.35	36.18	58.29	48.13	52.34	53.88	55.12	45.71	38.97	15.98
CNN	67.36	61.93	56.49	51.05	45.61	40.17	34.73	29.29	23.85	18.41	12.97	7.53	2.10
GCN	66.65	61.10	55.54	49.98	44.42	38.87	33.30	27.74	22.20	16.64	11.08	5.52	0.04
Weighted	72.12	57.61	45.64	43.09	47.54	47.22	41.19	34.98	27.57	19.17	12.59	6.93	0.49

Table 2. Relative Error of Prediction Without Hepp Bound



- Compared several Machine Learning models to predict Feynman Periods, to determine which would have the smallest loss.
- Reached conclusion that the basic and weighted model perform the best.
- Further investigations ongoing to determine best optimal weight of the weighted model, and different training methods for GCN.

References

[1] P.-H. Balduf, *Statistics of Feynman amplitudes in ϕ^4-theory*, (2023) `http://arxiv.org/abs/2305.13506` (visited on 06/24/2023), preprint.

[2] O. Schnetz, "Quantum periods: A census of ϕ^4-transcendentals", Commun.Num.Theor.Phys. **4**, 1–48 (2010), `http://arxiv.org/abs/0801.2856` (visited on 10/29/2021).

[3] O. Schnetz, "Numbers and Functions in Quantum Field Theory", Physical Review D **97**, 085018 (2018), `http://arxiv.org/abs/1606.08598` (visited on 09/04/2019).

[4] M. Kontsevich and D. Zagier, "Periods", in *Mathematics Unlimited - 2001 and Beyond* (Springer, 2001), pp. 771–808, `http://preprints.ihes.fr/M01/M01-22.ps.gz`.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks", arXiv preprint arXiv:1609.02907 (2016).

Special thank you to Prof. Karen Yeats.

November 2023

kimia.shaban@uwaterloo.ca