Learning Feynman Diagrams using Graph Neural Networks

Harrison Mitchell
Cavendish Laboratory
University of Cambridge
hgwm2@cantab.ac.uk

Alexander Norcliffe
Computer Laboratory
University of Cambridge
alin2@cam.ac.uk

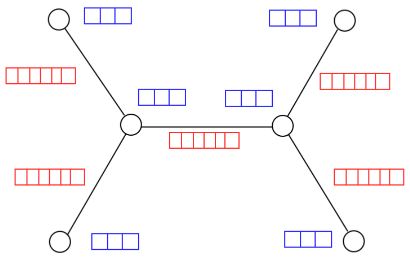
Pietro Liò
Computer Laboratory
University of Cambridge
pl219@cam.ac.uk

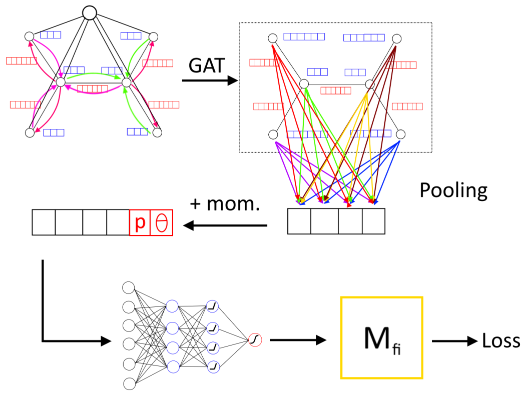
Abstract

In the wake of the growing popularity of machine learning in particle physics, this work finds a new application of geometric deep learning on Feynman diagrams to make accurate and fast matrix element predictions with the potential to be used in analysis of quantum field theory. This research uses the graph attention layer which makes matrix element predictions to 1 significant figure accuracy above 90% of the time. Peak performance was achieved in making predictions to 3 significant figure accuracy over 10% of the time with less than 200 epochs of training, serving as a proof of concept on which future works can build upon for better performance. Finally, a procedure is suggested, to use the network to make advancements in quantum field theory by constructing Feynman diagrams with effective particles that represent non-perturbative calculations.

1 Introduction and Related Work

Particle physics has recently seen a rise in the popularity of machine learning [1], [2], [3]. Applications include, analysing particle jets [4] [5], tracking particle paths [6] and detecting collider events [7]. This work explores a new application of geometric deep learning [8][9] on graph neural networks (GNNs) by predicting matrix elements from Feynman diagrams for simple particle interactions. Analytically, matrix elements are computed from Feynman diagrams, a pictorial representation of a particle interaction, using a set of rules called the Feynman rules, which translate the diagram into a mathematical expression. The matrix elements then find wide use in computing probabilities in particle physics. Graph neural networks are a type of network that operate on graph valued data, to make node, edge or graph level predictions. In this work, a graph level prediction is used for matrix elements from a graph representation of Feynman diagrams. The convolutional layer used is the graph attention layer (GAT) [10], which previously has shown application in text classification [11], material property predictions [12] and classical physics [13]. This work also aims to extend the testing of the performance of the GAT layer and the effectiveness of its attention head mechanism on Feynman diagrams. The advantage of using machine learning for physics is partly in faster computation time for complicated matrix elements, but the envisioned future of this study is that the neural network learns all Feynman rules so that it can be connected to a dynamic graph network (for example, the temporal graph network [14]). Such a network would progressively construct new nodes and edges, creating new Feynman diagrams which would be passed to the learned GNN whose output would be used to fit to matrix elements from experimental data. This may predict effective Feynman diagrams in non-perturbative regimes, such as low energy QCD, where truncating the infinite series of perturbations does not approximate the matrix elements. Many of the matrix elements that were used extensively in the learning process for the neural networks were adapted from the 1995 work by Borodulin [15].


(a) An example tree-level graph, that the network can take as an input. The work of this paper is on diagrams with this structure but with different edge features. Blue vectors are node features - interactions. Red vectors are edge features - particles.



(b) Top left image depicts the action of the GAT layer. Top right is a portrayal of the output of the GAT layers being pooled into a graph representation and momenta added. This is passed to an FCN to get the predictions.

Figure 1: Sample input (left) and network architecture (right).

2 Method

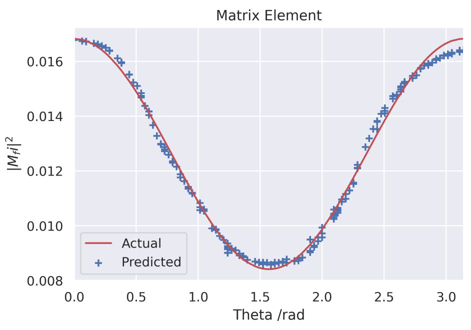
2.1 Dataset encoding

Feynman diagrams have a natural graph representation as seen in Figure 1a, with interaction vertices being represented by nodes, and particles being represented by edges. The edge features contain all the information to describe a particle, whilst the targets are the matrix elements, averaged over the helicity combinations. In our graph encodings, the momenta are not included in the edge features and are instead concatenated after the GNN has produced a graph representation. The concatenation process is outlined in Figure 1b. Since the Feynman rules are independent of momentum in both QED and QCD, the representation of each graph should be as well. This method provides scalability by removing the need for multiple diagrams for each possible helicity combination. This is a more efficient procedure when the number of Feynman diagrams is large. The particle encoding is given in Equation 1, where masses are on-shell and in mega electronvolts (MeV).

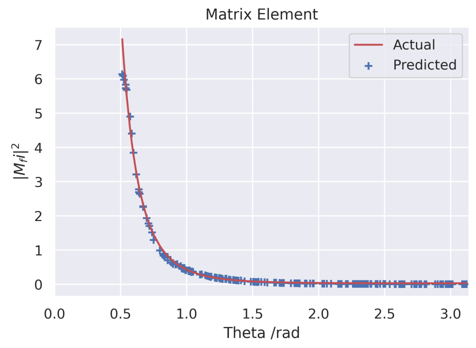
$$\text{edge vector} = [m, S, I_L^3, Y_L, I_R^3, Y_R, r, g, b, \bar{r}, \bar{g}, \bar{b}] \tag{1}$$

All data is given in the centre of mass frame with azimuthal angle $\phi = 0$ and particles aligned with the z-axis, so that only the polar angle, θ, and the momentum, p, are needed to categorise the interaction. Each data point includes a graph, representing the Feynman diagram, and a (p, θ) pair. A large dataset consisting of 80,000 such data points with ultra-relativistic momenta in the range $[1, 1000]$GeV, was then constructed and training was carried out on subsets containing only the appropriate Feynman diagrams for the experiment. Each subset consisted of 2000 data points, sampled evenly over the range of target matrix elements. The network is fed three random splits of the dataset: training, validation and test datasets The split ratio was 6:2:2 respectively. The training dataset is what the model sees and uses to conduct back-propagation and the order is shuffled each epoch. The validation dataset is passed through the model at the end of each epoch. The test dataset is passed through after training.

A global node was also added to improve the training speed and performance of the network. The three interaction strengths for QED, QCD and the weak force $(\alpha_{QED}, \alpha_S, \alpha_W)$ were stored in the global node vector, each given at zero energy. The logarithmic running of these constants can be learnt during propagation through subsequent node embeddings of the global node in the forward pass. This addition reduced the number of required layers and trainable parameters in the GNN since node information on opposite ends of the graph only needs two layers to pass to each other via the global node. It was seen that even when the number of attention heads and layers in the GNN were reduced, the network performed similarly to before adding the global node.



(a) Results for $e^-e^+ \to \mu^-\mu^+$ from a GNN that has seen both matrix elements



(b) Results for $e^-e^+ \to e^-e^+$ Bhabha scattering matrix elements.

Figure 2: Results for training on three diagrams

2.2 Network Architecture

The aim is that the GNN will create a representation that encodes information for the Feynman rules. The network consists of a GNN connected to a fully connected network (FCN). The GNN architecture is built using three key components: a convolutional layer, a linear layer and a pooling layer. The convolutional layer used is the graph attention layer [10]. The linear layer invokes simple fully connected layer with a leaky ReLu activation function, needed to condense the size of the graph, since the attention mechanism generates a larger graph depending on the number of attention heads. The pooling layer uses global sum pooling in order to capture information across all nodes as well as distinguishing different sized graphs by the magnitude of the values in the representation. The graph representation is then passed to the FCN consisting of two hidden layers and a leaky ReLU activation function. The final output layer has just one neuron whose value is passed through a sigmoid activation function so that the output is squeezed between zero and one. The architecture is summarised in the schematic in Figure 1b. Further implementation details are included in Appendix B.

3 Results and Discussion

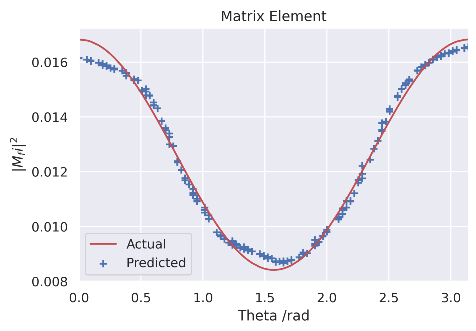
The network was trained on high energy tree-level QED diagrams involving electrons and muons on which the model accurately predicts matrix elements for several processes. There were 2 main metrics used to evaluate performance: L^1 loss and the accuracy. Accuracy is defined as the proportion of results in the dataset that are the same as the target after rounding up to a number of decimal places (d.p.), a, as shown in Equation 2 .

$$\mathcal{A}(\{y_i\},\{\hat{y}_i\},a) = \frac{1}{N}\sum_{i}^{N}\left(1-\Theta\left[L^1\left(\lceil y_i\rceil_a,\lceil \hat{y}_i\rceil_a\right)\right]\right) \tag{2}$$

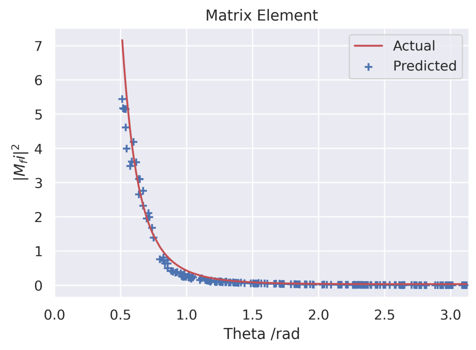
Where $\lceil x\rceil_a$ means round up to the a^{th} decimal place and $\Theta(x)$ is the step function. The predicted matrix elements are plotted against the polar angle in the range $[0, 2\pi]$. At large momenta, the matrix element is independent of momentum, and so the variation is in the angle. These metrics are tabulated in each experiment's data tables. A further extension of the model is tested on QCD, involving quark diagrams. The results are included in Appendix C.

3.1 Three diagram dataset

The initial model was tested on two graphs consisting of three Feynman diagrams: the electron-positron to muon-antimuon s-channel diagram; and the two Bhabha electron-positron to electron-positron s and t-channel diagrams, batched into one graph with ultra-relativistic matrix element. Since both graphs are combined into a single dataset the GNN is forced to distinguish between the two



(a) Results from the fully trained network on all tree-level QED with electrons and muons, for the $e^-e^+ \to \mu^-\mu^+$ matrix element.



(b) Results from the fully trained network, for the $e^-e^+ \to e^-e^+$ matrix element.

Figure 3: Training on full dataset

different inputs, testing its ability to construct a sufficiently detailed graph representation to feed to the FCN to communicate which matrix element is to be fitted.

Figures 2a and 2b exhibit the model's performance in fitting over the range of angles. The predictions are good with a small discrepancy for the muon matrix element at large angles. More detail on the loss and accuracy values are given in Table 1, where it can be seen that the accuracy to one decimal place is high but just shy of 100% because the model predicts some matrix elements very close to the target, but some much farther away so that the L^1 loss is consistently low but allowing for a much tighter fit for some of the data points reaching two and three decimal place accuracy over 81% and 11% of the time, respectively.

Table 1: Loss and accuracy values for the three combined Feynman diagrams for $e^-e^+ \to \mu^-\mu^+$ and $e^-e^+ \to e^-e^+$

Test metric	Data
Test accuracy (1d.p.)	99.00%
Test accuracy (2d.p.)	81.50%
Test accuracy (3d.p.)	11.50%
Test L1 Loss	0.0049

3.2 Tree-level QED dataset for electrons and muons

The dataset was expanded to encompass six graphs with four high energy matrix elements at tree-level QED, for each of the permutations of electrons and muons in the initial and final states and their allowed s and t-channels. There is a drop in performance but the overall target shape is maintained. Figure 3a shows the predicted and theoretical matrix elements for the same s-channel $e^-e^+ \to \mu^-\mu^+$ interaction. This time, this is one of many graphs that the network is able to predict, but despite the increase in number of points to fit, we still see strong performance. For this interaction the discrepancy at the extreme angles 0 and 2π is more noticeable, but there is additional error at the minimum $\theta = \pi$. Since the model performs best in the regions that are close to linear, this may be indicative of a requirement for additional non-linearities.

Figure 3b shows the predicted and theoretical matrix element for the s and t-channel $e^-e^+ \to e^-e^+$ interactions, where we see much stronger agreement. Increasing the number of dense neurons showed signs of improvement, but significantly increased the training time, but since this target consists of two largely linear regimes, the models performs well considering the training time. Table 2 shows the metrics for the learned GNN. The most noticeable difference is the much higher loss. This behaviour is expected for a larger dataset as there are more points for the network to fit to, but despite increasing the width of the FCN, the GNN was unable to fit as accurately as before. This may have been, in part, due to a need for an increased number of neurons in the FCN to be able to decode and fit multiple different functions depending on the graph representation it sees. Alternatively it may be that the GNN's graph representations were not large enough to discriminate between the graphs. The graph representation was kept to a constant dimension in the hope that the strong fit to the smaller three diagram dataset implied that the Feynman rules were sufficiently encoded. Increasing this may help to encode further detail of the Feynman rules with different starting and final states. Finally, the minima search may be further improved by smoothing the landscape with stochastic weight averaging by allowing the network to find lower minima and therefore reduce the loss.

However, the ability to identify different graphs and reconstruct different functions suggests that the GNN has the capacity to encode the Feynman rules into its graph representation. With some improvement in implementation it is possible that the GNN can make more accurate predictions and for a wider range of particle states.

Table 2: Loss and accuracy values for all tree-level QED Feynman diagrams involving all permutations of e^-e^+ and $\mu^-\mu^+$ pairs

Test metric	Data
Test accuracy (1d.p.)	96.75%
Test accuracy (2d.p.)	44.00%
Test accuracy (3d.p.)	0.75%
Test L1 Loss	0.015

4 Conclusion

Graph neural networks have been shown to be able to make basic matrix element predictions by fitting to analytical data. This provides improved computation time for matrix element calculations and scales well when introducing new matrix elements. This work suggests that it is possible for the GNN's graph representation to encode the elements of the Feynman rules, and for the FCN to decode it into a matrix element. Following on from the research in the field of geometric deep learning, graph attention networks have been shown to be effective in regression problems for Feynman diagrams. This supports the use of geometric deep learning in particle physics beyond experimental analysis. The aspiration is that this work will be extended to have learnt a full dataset with all fundamental forces that, in itself, would be useful in making experimental predictions. To further build on this, if the dataset is extended to loop diagrams, the network will learn its own procedure to deal with renormalization. This could then be connected to a network for dynamic graphs, which will learn the edges and nodes to build, which upon passing to the GNN, correctly fit to experimental data. The constructed diagrams are then ideally interpretable as Feynman diagrams, corresponding to physical processes, and if not, then at least it may provide effective Feynman diagrams, whose matrix elements represent an infinite series in non-perturbative QFT.

Acknowledgments and Disclosure of Funding

We thank Bentley Carr for his time and insight during the research of this project. Additionally we would like to take the chance to thank the reviewers for raising challenging points that drove us to find new perspectives and better the future direction of this work.

Finally, we also thank the Cavendish Laboratory and Computer Laboratory at the University of Cambridge for facilitating the original research.

References

[1]M. Feickert and B. Nachman, "A living review of machine learning for particle physics", 10.48550/ARXIV.2102.02770 (2021).

[2]D. Guest, K. Cranmer, and D. Whiteson, "Deep learning and its application to lhc physics", Annual Review of Nuclear and Particle Science **68**, 161–181 (2018).

[3]J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics", Machine Learning: Science and Technology **2**, 021001 (2021).

[4]J. Barnard, E. N. Dawe, M. J. Dolan, and N. Rajcic, "Parton shower uncertainties in jet substructure analyses with deep neural networks", Physical Review D **95**, 10.1103/physrevd.95.014018 (2017).

[5]L. de Oliveira, M. Kagan, L. Mackey, B. Nachman, and A. Schwartzman, "Jet-images — deep learning edition", Journal of High Energy Physics **2016**, 10.1007/jhep07(2016)069 (2016).

[6]J. Duarte and J.-R. Vlimant, "Graph neural networks for particle tracking and reconstruction", in *Artificial intelligence for high energy physics* (WORLD SCIENTIFIC, Feb. 2022), pp. 387–436.

[7]Y. Alanazi, N. Sato, P. Ambrozewicz, A. Hiller-Blin, W. Melnitchouk, M. Battaglieri, T. Liu, and Y. Li, "A survey of machine learning-based physics event generation", in Proceedings of the thirtieth international joint conference on artificial intelligence (Aug. 2021).

[8]F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model", IEEE Transactions on Neural Networks **20**, 61–80 (2009).

[9]M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data", IEEE Signal Processing Magazine **34**, 18–42 (2017).

[10]P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks", 10.48550/ARXIV.1710.10903 (2017).

[11]L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification", Proceedings of the AAAI Conference on Artificial Intelligence **33(01)**, 7370–7377 (2019).

[12]S.-Y. Louis, Y. Zhao, A. Nasiri, X. Wang, Y. Song, F. Liu, and J. Hu, "Graph convolutional neural networks with global attention for improved materials property prediction", Phys. Chem. Chem. Phys. **22**, 18141–18148 (2020).

[13]R. Wang, D. Maddix, C. Faloutsos, Y. Wang, and R. Yu, "Bridging physics-based and data-driven modeling for learning dynamical systems", in Proceedings of the 3rd conference on learning for dynamics and control, Vol. 144, edited by A. Jadbabaie, J. Lygeros, G. J. Pappas, P. Parrilo, B. Recht, C. J. Tomlin, and M. N. Zeilinger, Proceedings of Machine Learning Research (2021), pp. 385–398.

[14]E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs", 10.48550/ARXIV.2006.10637 (2020).

[15]V. I. Borodulin, R. N. Rogalyov, and S. R. Slabospitsky, "Core 2.1 (compendium of relations, version 2.1)", 10.48550/ARXIV.HEP-PH/9507456 (1995).

[16]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: an imperative style, high-performance deep learning library", in *Advances in neural information processing systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), pp. 8024–8035.

[17]M. Fey and J. E. Lenssen, *Fast Graph Representation Learning with PyTorch Geometric*, https://github.com/pyg-team/pytorch_geometric, May 2019.

[18]W. Falcon and The PyTorch Lightning team, *PyTorch Lightning*, https://github.com/PyTorchLightning/pytorch-lightning, version 1.4, Mar. 2019.

[19]DeepFindr, *A graph neural network project on hiv data*, `https://github.com/deepfindr/gnn-project`, 2021.

[20]Philippe, *Uvadlc notebooks*, `https://github.com/phlippe/uvadlc_notebooks/blob/master/docs/tutorial_notebooks/tutorial7/GNN_overview.ipynb`, 2022.

[21]J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs”, `10.48550/ARXIV.1312.6203` (2013).

[22]R. Mertig, M. Böhm, and A. Denner, “Feyn calc - computer-algebraic calculation of feynman amplitudes”, Computer Physics Communications **64**, 345–359 (1991).

[23]V. Shtabovenko, R. Mertig, and F. Orellana, “New developments in FeynCalc 9.0”, Computer Physics Communications **207**, 432–444 (2016).

[24]V. Shtabovenko, R. Mertig, and F. Orellana, “FeynCalc 9.3: new features and improvements”, Computer Physics Communications **256**, 107478 (2020).

[25]W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs”, `10.48550/ARXIV.1706.02216` (2017).

[26]W. L. Hamilton, “Graph representation learning”, Synthesis Lectures on Artificial Intelligence and Machine Learning **14**, 1–159.

[27]J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry”, `10.48550/ARXIV.1704.01212` (2017).

[28]H. Dai, B. Dai, and L. Song, “Discriminative embeddings of latent variable models for structured data”, `10.48550/ARXIV.1603.05629` (2016).

[29]A. Butter and T. Plehn, “Generative networks for lhc events”, `10.48550/ARXIV.2008.08558` (2020).

[30]P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, “Averaging weights leads to wider optima and better generalization”, `10.48550/ARXIV.1803.05407` (2018).

[31]Y. Bengio, “Practical recommendations for gradient-based training of deep architectures”, `10.48550/ARXIV.1206.5533` (2012).

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See end of Section 3 and Appendix C.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix A.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our code will be made publicly available after the review period.
   (b) Did you specify all the training details (e.g., data splits, hyper-parameters, how they were chosen)? [Yes] See Appendix B.2.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] As part of this ongoing work we were investigating whether Graph Neural Networks *could* be applied to Feynman Diagrams, which we saw they can, using a single seed.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix B.2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] The python libraries are PyTorch and PyTorch Geometric.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We will provide the code publicly via Github after the review process.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

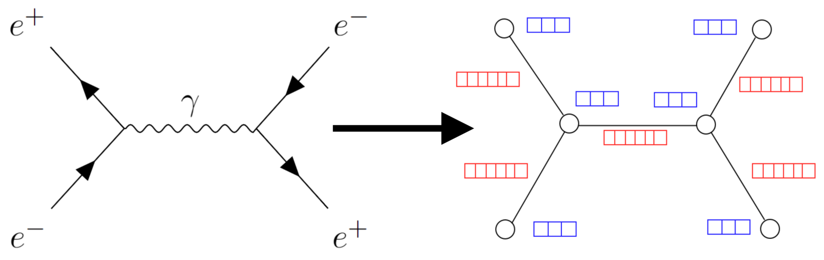


Figure 4: Encoding a Feynman diagram as a graph. The blue vectors are the node features and the red are the edge features.

A Societal impacts

This work has had two fields of development: one in the use of machine learning to further physics and the other to use physics to research machine learning. The development of physical theory has limited short term societal impact, since particle physics is many years from finding implementation in technology and engineering. This paper contributes towards the effectiveness of the graph attention layer but only in its application to Feynman diagrams and has its limitations. It does not suggest it could be easily substituted for unethical datasets and does not promote its use in immoral situations. The datasets used were self-constructed and did not use any sensitive information. Creating a trained network is also fast and not energy intensive with Google Colab's free GPUs being sufficient with 12.68GB RAM used to train the whole dataset, so there is little environmental impact.

B Implementation

B.1 Dataset encoding

Figure 4 gives an example Feynman diagram and a graphical depiction of its encoding to a graph.

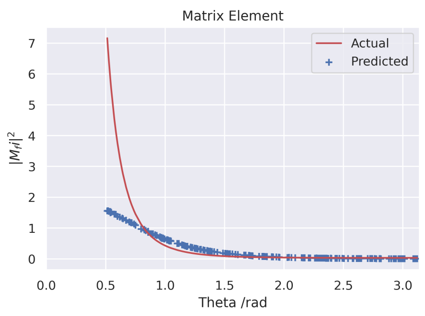
The nodes were also encoded as to make the graph orientable by capturing information about the time-like direction of the interaction. This was achieved by specifying which nodes are initial state particles, final state particles and virtual particles, using a one-hot encoding.

$$\text{node vector} = [\text{initial}, \text{virtual}, \text{final}] \tag{3}$$

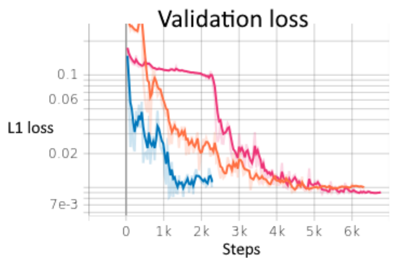
B.2 Training details

The network was constructed in PyTorch [16], with the aid of PyTorch Geometric [17] and PyTorch Lightning [18]. We used the help of several GitHub repositories including the work by DeepFindr [19] and the tutorial from Pytorch Geometric [20]. All of the code was written in Google Colab using their free GPUs, with about 13GB of RAM total.

The loss was generated with a variety of loss functions including mean square error (MSE), absolute error (L^1), and LogCosh with the loss of a batch being calculated by the sum of the individual losses. Ultimately all losses performed similarly, but L^1 loss was primarily used in this study as slight improvement on accuracy was found. A stochastic gradient descent optimizer with momentum 0.4 was used. Damping was also tested but did not improve the performance. The learning rate was started high at 1 but a scheduler that reduced the learning rate when the loss plateaued was implemented to hone in on the minima. The mini-batch size used depended on the dataset but was kept at powers of 2, between 16 and 64, to maximize storage use. This was a good compromise of speed and performance. The graph representation vector had dimension that depended on the complexity of the dataset, but reached a size of at most 8. This was sufficient for the network to encode the relevant information to pass to the FCN. The activation functions used throughout are ReLU and Leaky ReLU, these were fast to train and did not give stagnation problems that sigmoid and tanh do.



(a) Training on $e^-e^+ \to e^-e^+$ Bhabha scattering diagrams after only one epoch of training



(b) Graph showing the improvement of the network loss at the end of each epoch on the validation dataset. Blue: one graph. Pink: $e^-e^+ \to \mu^-\mu^+$ and $e^-e^+ \to e^-e^+$ graphs. Orange: whole dataset.

Figure 5: Graphs showing the speed to train the network

The number of dense neurons in the FCN was progressively increased if the model under-fitted, but a value between 64 and 128 was enough to be able to fit to the functions for this paper. The depth of the GNN was kept to only 2 or 3 layers, due to the global node.

B.3 Runtime

The GNN training was completed on the order of minutes. Figure 5a shows how the FCN begins to fit to the targets after just one epoch. Figure 5b shows how the training time increases as the complexity of the dataset grows. The blue line represents training on just one graph, which is why it converges on a minimum so quickly. The orange line represents the most complex dataset, and takes longer than the pink line to reach the same level of loss, being overtaken at around 5000 steps. The slow start is due to the stochastic nature of the optimization algorithm. Once it finds a minimum, the pink line converges more quickly.

Training time is largely due to the number of parameters to fit. As the dataset gets more complex, more parameters are required and so the parameter space to search is larger. With the large learning rates used, it is possible that steep local minima are also missed.

The sudden jumps in each line is due to the learning rate scheduler. As the model starts to oscillate around the minima, the speed is dropped so that the model does not overshoot.

B.4 Program code

`https://github.com/Clearbloo/Feynman_GNN.git`

C QCD

QCD was also trialed to test the models generalization to a different set of Feynman rules. The test process used was $u\bar{u} \to t\bar{t}$ via a gluon s-channel. Since top quarks are massive, and not in chirality eigenstates, this also doubled as a test for the network to distinguish momentum dependence. There was a significant drop in performance, as shown in Figure 6 and Table 3. However, the shape of the original QED predictions was retained and the model was able to identify the different curves based on momenta. This suggests the GNN can be scaled up to include more matrix elements and different sections of the standard model interaction Hamiltonian, with an increasingly large graph representation and improvements on implementation. This is expected as different Feynman rules need additional encodings on the graph representation.

Table 3: Loss and accuracy values for $u\bar{u} \to t\bar{t}$ via gluon exchange

Test metric	Data
Test accuracy (1dp)	94.0%
Test accuracy (2dp)	42.0%
Test accuracy (3dp)	3.75%
Test L1 Loss	0.0265

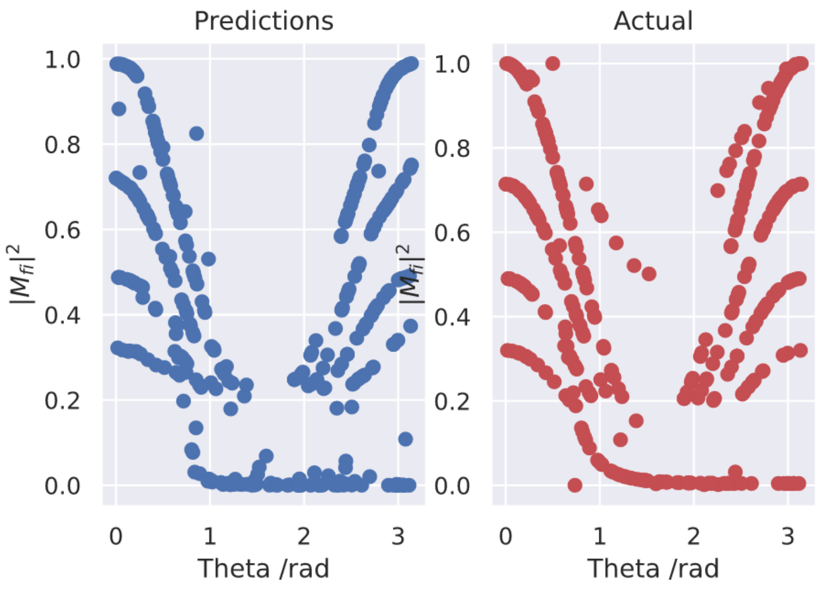


Figure 6: QCD results