A factorisation aware matrix-element emulator

[arXiv:2107.06625] with Daniel Maître

Henry Truong

11th March 2022

Institute for Particle Physics Phenomenology
Institute for Data Science
Durham University

Table of contents

Introduction

Motivation

- High multiplicity matrix elements are computationally expensive to evaluate.

Motivation

- High multiplicity matrix elements are computationally expensive to evaluate.
- High energy collider experiments are becoming increasingly more precise, and with HL-LHC we need to improve the speed of event generation.

Motivation

- High multiplicity matrix elements are computationally expensive to evaluate.
- High energy collider experiments are becoming increasingly more precise, and with HL-LHC we need to improve the speed of event generation.
- Successfully emulating matrix elements will provide a fast and accurate alternative to more traditional matrix element providers.

Electron-positron scattering

- We investigate using a neural network model to emulate tree-level matrix elements for $e^+e^- \to q\bar{q} + ng$, up to five jets.

Electron-positron scattering

- We investigate using a neural network model to emulate tree-level matrix elements for $e^+e^- \to q\bar{q} + ng$, up to five jets.
- Matrix elements are plagued with singularities arising because of infrared divergences.

Electron-positron scattering

- We investigate using a neural network model to emulate tree-level matrix elements for $e^+e^- \to q\bar{q} + ng$, up to five jets.
- Matrix elements are plagued with singularities arising because of infrared divergences.
- Small changes in phase-space kinematics can induce large changes to the matrix element.

Why use neural networks?

- Neural networks are good function approximators. In principle, they can approximate any arbitrary function.

Why use neural networks?

- Neural networks are good function approximators. In principle, they can approximate any arbitrary function.
- Recent advancements in GPU technology and availability means that the training and inference times of neural networks are massively accelerated.

Why use neural networks?

- Neural networks are good function approximators. In principle, they can approximate any arbitrary function.
- Recent advancements in GPU technology and availability means that the training and inference times of neural networks are massively accelerated.
- Neural networks have been shown to scale well with large datasets.

Why use neural networks?

- Neural networks are good function approximators. In principle, they can approximate any arbitrary function.
- Recent advancements in GPU technology and availability means that the training and inference times of neural networks are massively accelerated.
- Neural networks have been shown to scale well with large datasets.
- Inference on a neural network is simple to implement manually so possible to interface to existing event generators.

Brief overview of neural networks

Building block of a neural network is the neuron

A neuron is modelled as

$$y = \phi(\mathbf{w}^T\mathbf{x} + b)\,. \tag{1}$$

ϕ is usually a non-linear mapping that allows the neuron to represent non-linear functions.

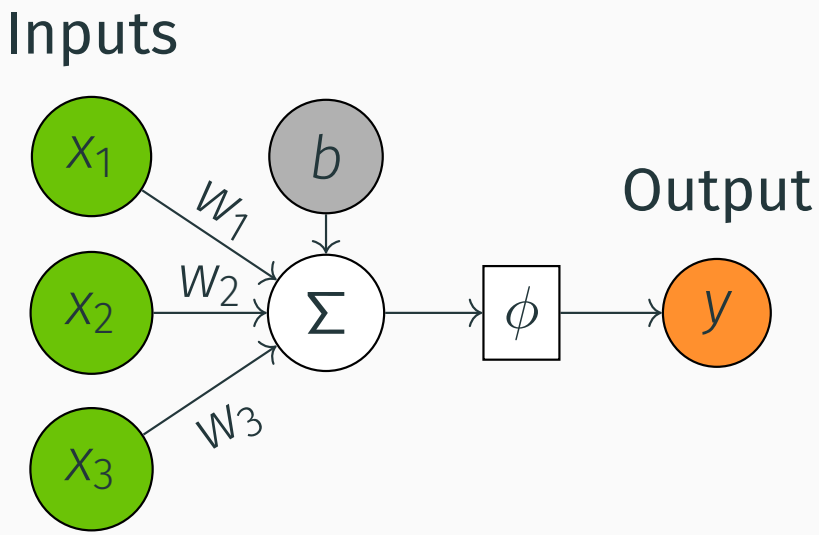


Figure 1: A schematic diagram of a neuron.

Connecting the neurons

By stacking neurons in layers, and then connecting these layers together, we build up the representation of the target function as

$$f(\mathbf{x};\theta) \approx \phi^{(n)}(\ldots\phi^{(2)}(\phi^{(1)}(\mathbf{x}))\ldots)\,, \tag{2}$$

where θ are the weights and biases of the network. Together we refer to them as the parameters of the network.

Connecting the neurons

By stacking neurons in layers, and then connecting these layers together, we build up the representation of the target function as

$$f(\mathbf{x};\theta) \approx \phi^{(n)}(\ldots\phi^{(2)}(\phi^{(1)}(\mathbf{x}))\ldots)\,, \tag{2}$$

where θ are the weights and biases of the network. Together we refer to them as the parameters of the network.

Fitting a target function essentially boils down to optimising the parameters θ.

Dense neural network

A neural network where every neuron is connected to every other neuron in neighbouring layers is referred to as a fully-connected or dense neural network.

This architecture is used often because they are easy to build.

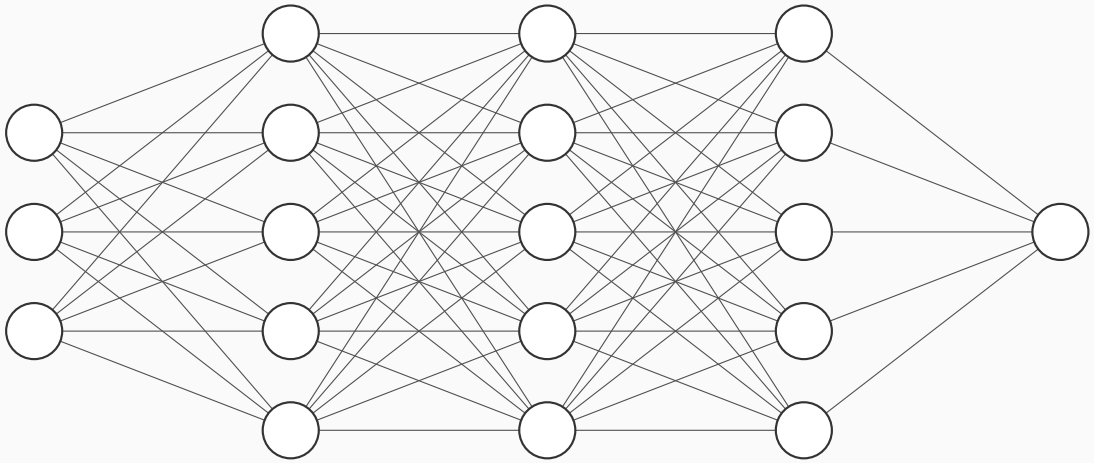


Figure 2: A dense neural network with three hidden layers.

Optimising θ

Before optimising θ we need an objective function to quantify how well the network is performing.

Optimising θ

Before optimising θ we need an objective function to quantify how well the network is performing.

For regression problems this is very commonly the mean squared error

$$L_{\text{MSE}} = \frac{1}{N} \sum_{n=1}^{N} (y - f(\mathbf{x}; \theta))^2 \,, \tag{3}$$

where y is the true (target) value and $f(\mathbf{x}; \theta)$ is the network prediction.

Optimising θ

Now that we can evaluate $L(\theta)$ to quantify the network performance, we want to minimise it.

Optimising θ

Now that we can evaluate $L(\theta)$ to quantify the network performance, we want to minimise it.

We minimise the loss function by optimising the values of θ according to an update rule

$$\theta = \theta - \alpha \nabla_\theta L(\theta)\,, \tag{4}$$

where α is the learning rate and ∇_θ denotes the gradient with respect to parameters θ.

Optimising θ

Now that we can evaluate $L(\theta)$ to quantify the network performance, we want to minimise it.

We minimise the loss function by optimising the values of θ according to an update rule

$$\theta = \theta - \alpha \nabla_\theta L(\theta)\,, \tag{4}$$

where α is the learning rate and ∇_θ denotes the gradient with respect to parameters θ.

This algorithm is gradient descent and is the algorithm of choice* for optimising θ.

IR divergences and Catani-Seymour dipole factorisation

Infrared divergences

Infrared divergences arise in matrix elements in certain regions of phase-space.

Infrared divergences

Infrared divergences arise in matrix elements in certain regions of phase-space.

For our process $e^+e^- \to q\bar{q} + ng$ they occur

- when gluons are *collinear* to the quark or anti-quark, $s_{qg} \to 0$.
- when gluons are *soft*, $E_g \to 0$

Infrared divergences

Infrared divergences arise in matrix elements in certain regions of phase-space.

For our process $e^+e^- \to q\bar{q} + ng$ they occur

- when gluons are *collinear* to the quark or anti-quark, $s_{qg} \to 0$.
- when gluons are *soft*, $E_g \to 0$

Singularities make it difficult to model the phase-space effectively since a single neural network struggles to simultaneously fit the well-behaved regions and the singular regions.

Dipole factorisation formula

Dipole factorisation was used by Catani and Seymour [1] originally to construct subtraction terms in NLO QCD calculations.

[1] S. Catani and M.H. Seymour. "A General algorithm for calculating jet cross-sections in NLO QCD". In: Nucl. Phys. B 485 (1997), pp. 291–419.

Dipole factorisation formula

Dipole factorisation was used by Catani and Seymour [1] originally to construct subtraction terms in NLO QCD calculations.

The authors introduced universal dipoles that smoothly interpolates between the soft and collinear limits.

[1] S. Catani and M.H. Seymour. "A General algorithm for calculating jet cross-sections in NLO QCD". In: Nucl. Phys. B 485 (1997), pp. 291–419.

Dipole factorisation formula

Dipole factorisation was used by Catani and Seymour [1] originally to construct subtraction terms in NLO QCD calculations.

The authors introduced universal dipoles that smoothly interpolates between the soft and collinear limits.

We use these dipoles to factorise out the IR singular structure from matrix elements

$$|\mathcal{M}_{n+1}|^2 \to |\mathcal{M}_n|^2 \otimes \mathbf{V}_{ij,k}\,, \tag{5}$$

where all divergences are isolated in the process independent factor $\mathbf{V}_{ij,k}$.

[1] S. Catani and M.H. Seymour. "A General algorithm for calculating jet cross-sections in NLO QCD". In: Nucl. Phys. B 485 (1997), pp. 291–419.

Dipole factorisation formula

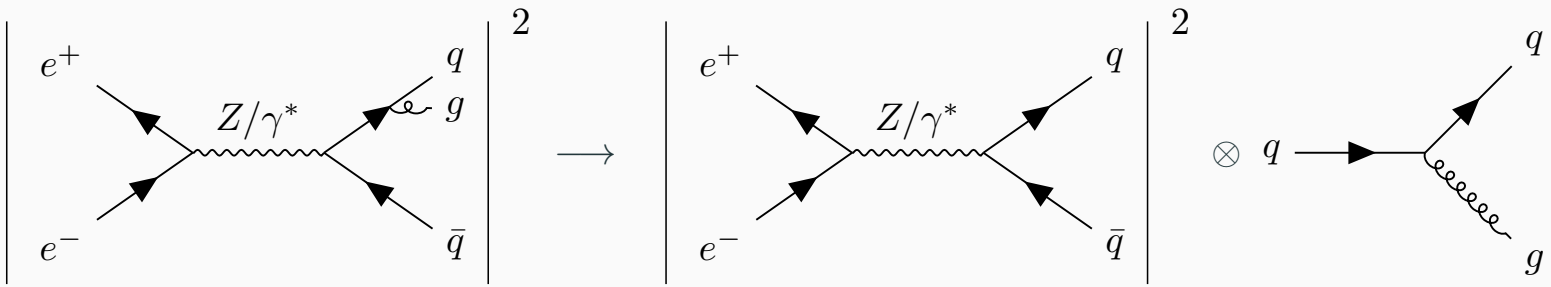


Figure 3: Schematic of dipole factorisation.

Dipole factorisation formula

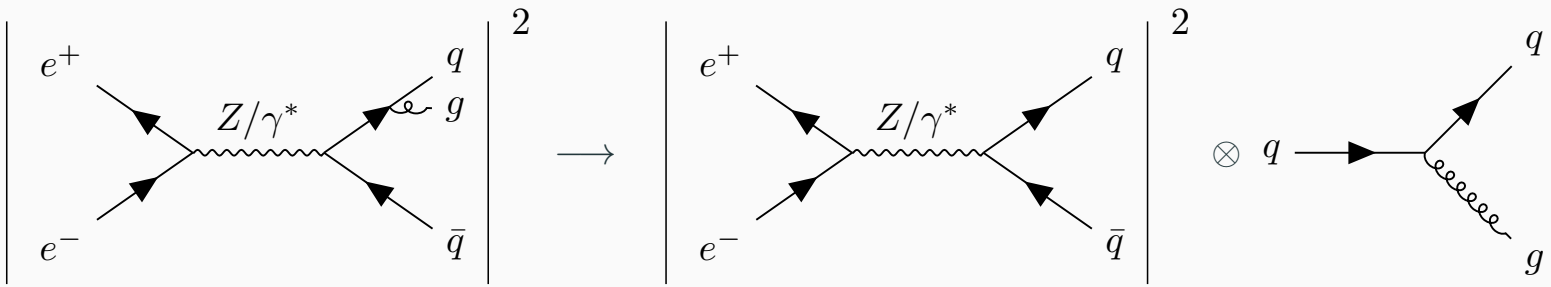


Figure 3: Schematic of dipole factorisation.

For singly unresolved limits, this factorisation isolates all the divergent behaviour in $\mathbf{V}_{ij,k}$ leaving the reduced matrix element $\langle|\mathcal{M}_n|^2\rangle$ free of divergences.

Spin-averaged Catani-Seymour dipoles

The dipoles are given as

$$\langle V_{q_i g_j,k} \rangle = 8\pi\alpha_s C_F \left[\frac{2}{1 - z_i(1 - y_{ij,k})} - (1 + z_i) \right], \tag{6}$$

$$\langle V_{g_i g_j,k} \rangle = 16\pi\alpha_s C_A \left[\frac{1}{1 - z_i(1 - y_{ij,k})} + \frac{1}{1 - z_j(1 - y_{ij,k})} - 2 + z_i z_j \right], \tag{7}$$

where

$$z_i = \frac{p_i p_k}{(p_i + p_j) p_k} \quad \text{and} \quad z_j = 1 - z_i$$

$$y_{ij,k} = \frac{p_i p_j}{p_i p_j + p_j p_k + p_k p_i}.$$

Dipole neural network emulator

Ansatz

We use the dipole factorisation formula to build an ansatz of the colour and helicity summed $n+1$-body matrix element

$$\langle |\mathcal{M}_{n+1}|^2 \rangle = \sum_{\{ijk\}} C_{ijk} D_{ij,k} \,, \quad \text{where} \quad D_{ij,k} = \frac{\langle V_{ij,k} \rangle}{s_{ij}} \,. \tag{8}$$

Ansatz

We use the dipole factorisation formula to build an ansatz of the colour and helicity summed $n+1$-body matrix element

$$\langle |\mathcal{M}_{n+1}|^2 \rangle = \sum_{\{ijk\}} C_{ijk} D_{ij,k} \,, \quad \text{where} \quad D_{ij,k} = \frac{\langle V_{ij,k} \rangle}{s_{ij}} \,. \tag{8}$$

C_{ijk} are the coefficients we fit using the neural network. They can be interpreted as the reduced matrix element in n-body phase-space.

Ansatz

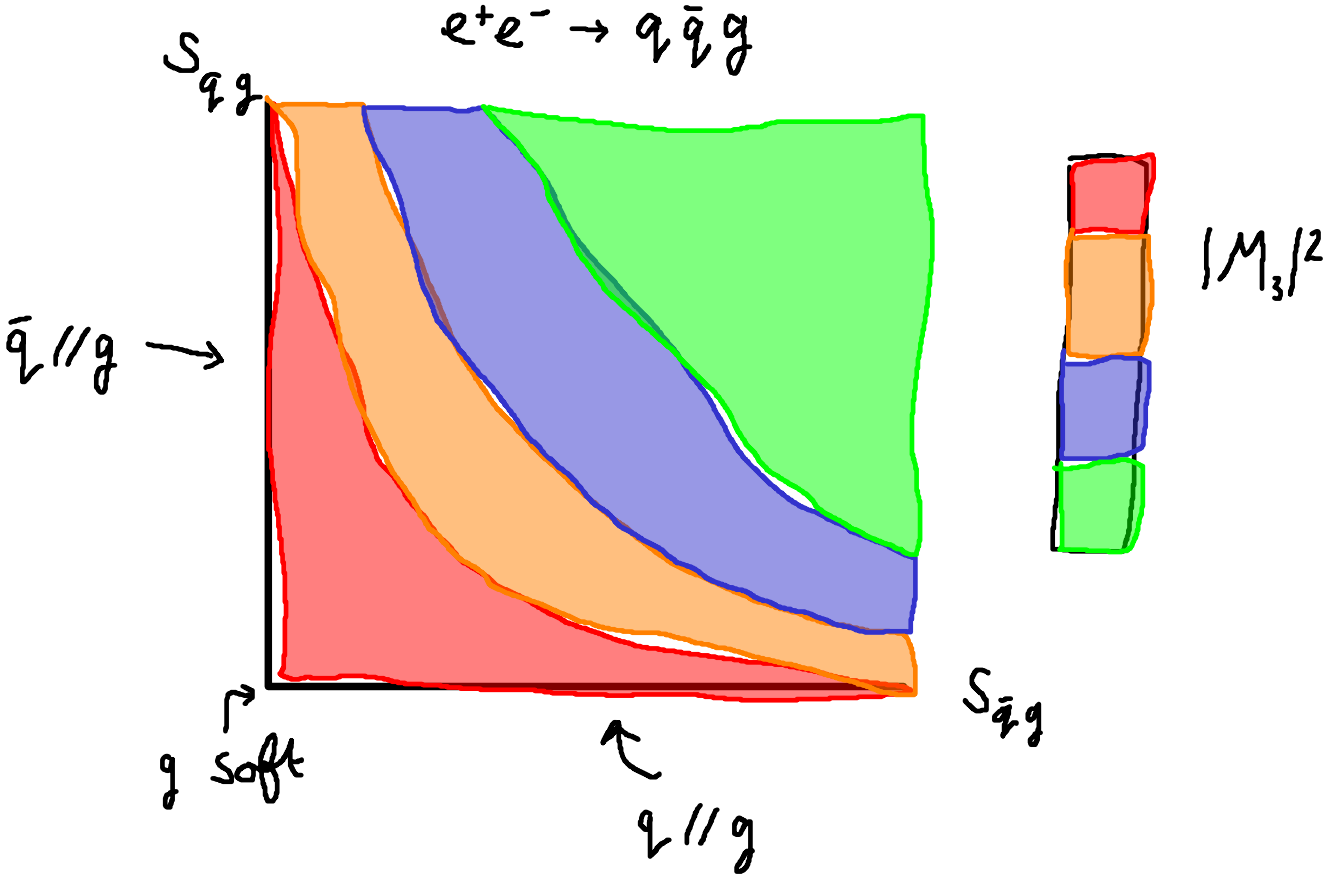
We use the dipole factorisation formula to build an ansatz of the colour and helicity summed $n+1$-body matrix element

$$\langle |\mathcal{M}_{n+1}|^2 \rangle = \sum_{\{ijk\}} C_{ijk} D_{ij,k} \,, \quad \text{where} \quad D_{ij,k} = \frac{\langle V_{ij,k} \rangle}{s_{ij}} \,. \tag{8}$$

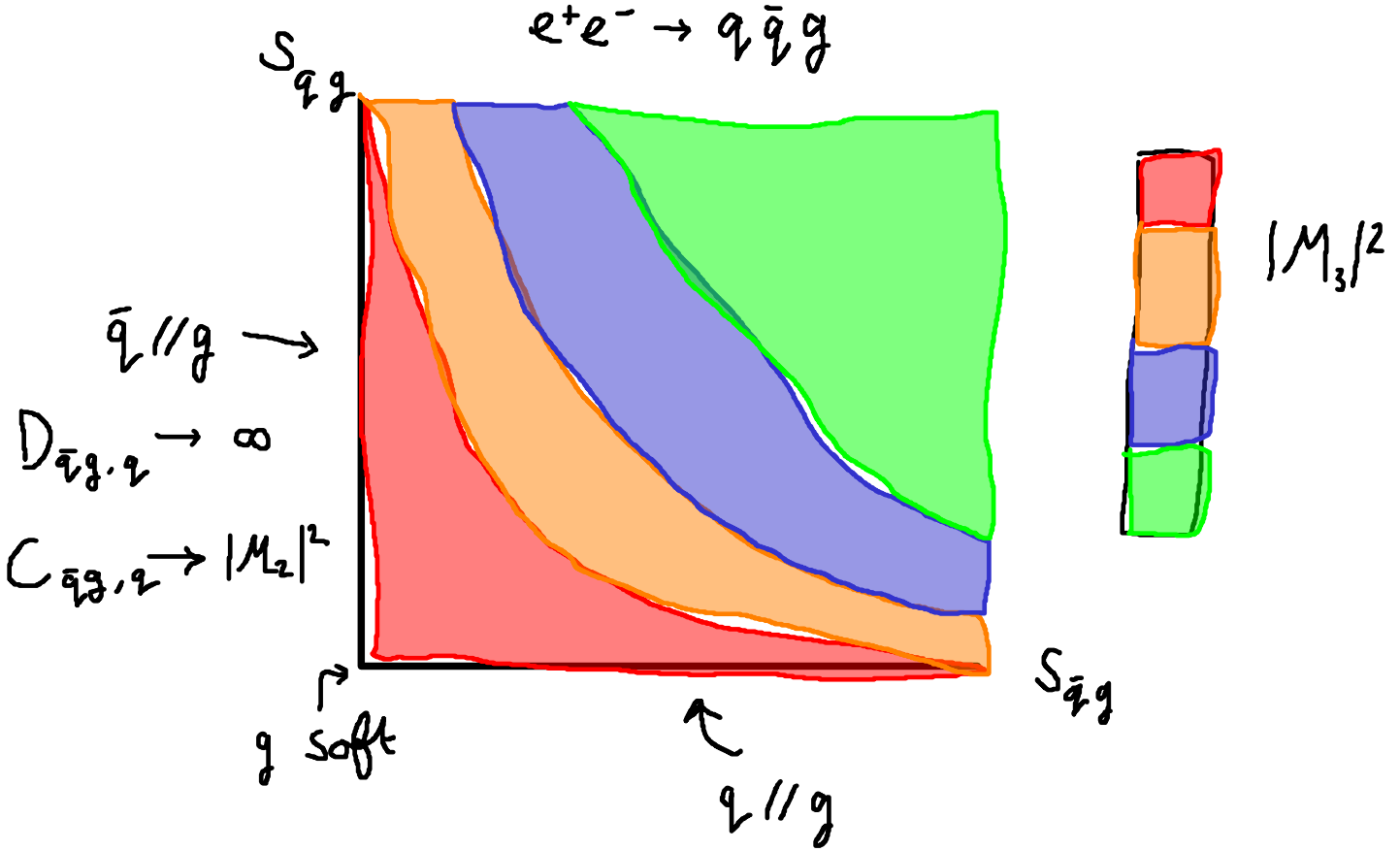
C_{ijk} are the coefficients we fit using the neural network. They can be interpreted as the reduced matrix element in n-body phase-space.

The sum over $\{ijk\}$ denotes the sum over relevant permutations of external final state particles.

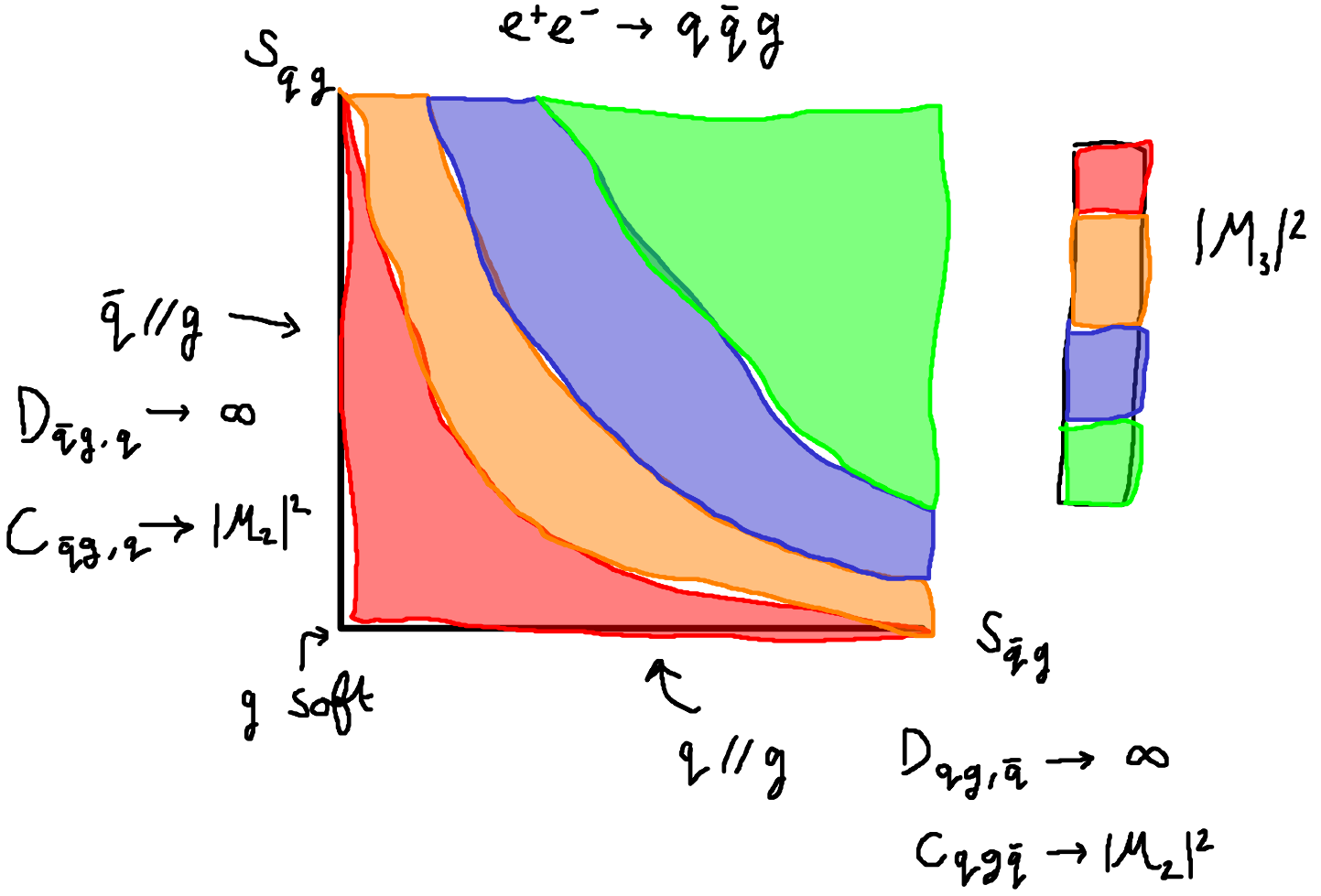
Intuition on ansatz



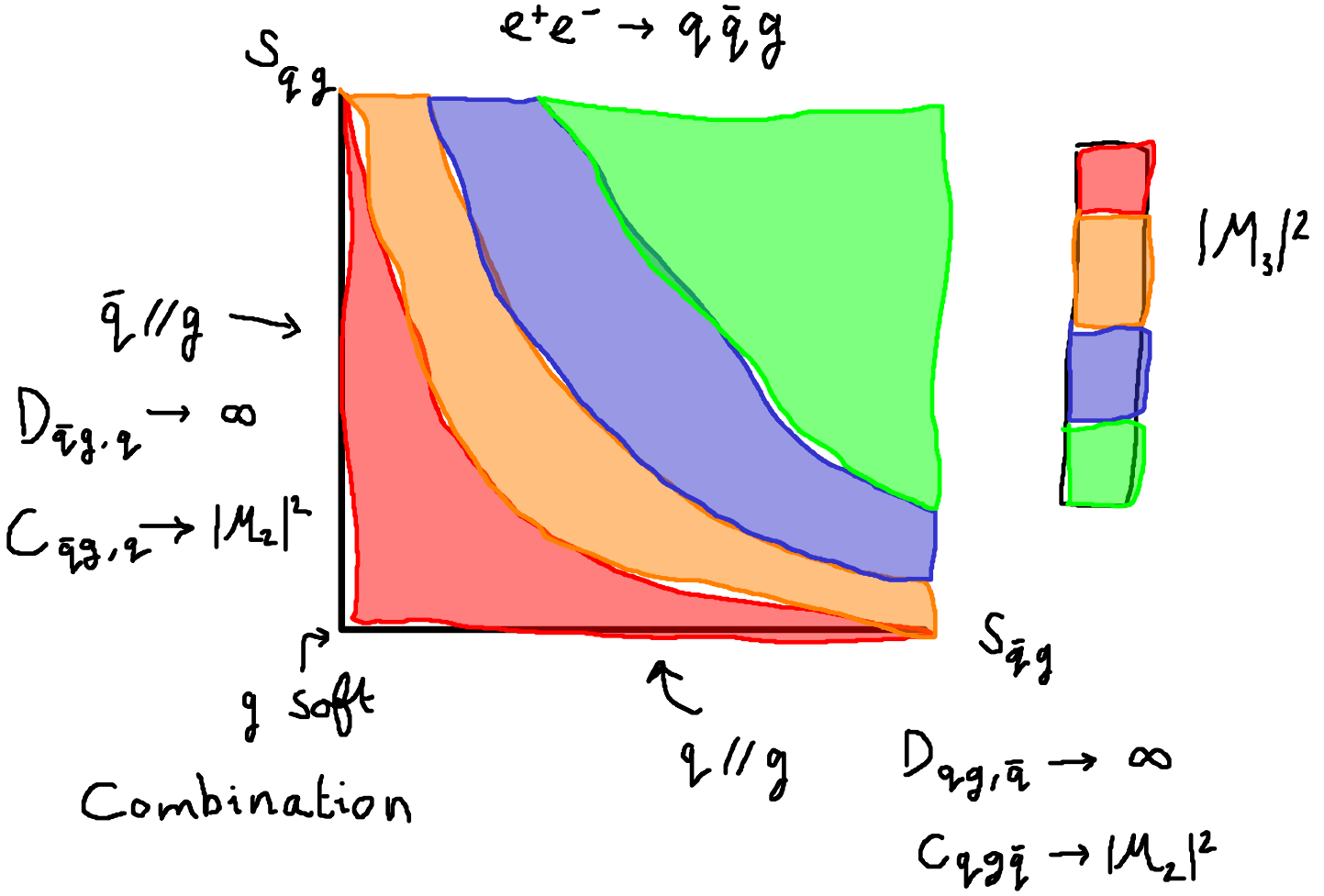
Intuition on ansatz



Intuition on ansatz



Intuition on ansatz



Inputs of the neural network

Direct inputs to network

- Phase-space points: $p = [E, p_x, p_y, p_z]$

Inputs of the neural network

Direct inputs to network

- Phase-space points: $p = [E, p_x, p_y, p_z]$
- Recoil factors: $y_{ij,k} = \dfrac{p_i p_j}{p_i p_j + p_j p_k + p_i p_k}$

Inputs of the neural network

Direct inputs to network

- Phase-space points: $p = [E, p_x, p_y, p_z]$
- Recoil factors: $y_{ij,k} = \dfrac{p_i p_j}{p_i p_j + p_j p_k + p_i p_k}$

These inputs are scaled to have a mean of zero and unit variance.

Inputs of the neural network

Direct inputs to network

- Phase-space points: $p = [E, p_x, p_y, p_z]$
- Recoil factors: $y_{ij,k} = \dfrac{p_i p_j}{p_i p_j + p_j p_k + p_i p_k}$

These inputs are scaled to have a mean of zero and unit variance.

Phase-space points sampled with RAMBO and clustered with `FastJet`.

Global phase-space cuts are applied according to $y_{ij} \geq y_{\text{cut}}$, where we explore three values of $y_{\text{cut}} = [0.01, 0.001, 0.0001]$.

Inputs of the neural network

Indirect inputs to network

- Spin-averaged dipoles $D_{ij,k} = \dfrac{\langle V_{ij,k} \rangle}{s_{ij}}$

Inputs of the neural network

Indirect inputs to network

- Spin-averaged dipoles $D_{ij,k} = \dfrac{\langle V_{ij,k} \rangle}{s_{ij}}$
- Averaging over spins means that we have lost information about the spin-correlation in $g \to gg$ splitting.

Inputs of the neural network

Indirect inputs to network

- Spin-averaged dipoles $D_{ij,k} = \dfrac{\langle V_{ij,k} \rangle}{s_{ij}}$
- Averaging over spins means that we have lost information about the spin-correlation in $g \to gg$ splitting.
  - Introduce a pair of terms $S_{ij}\,\mathbf{sin}(2\phi_{ij}) + C_{ij}\,\mathbf{cos}(2\phi_{ij})$ in the ansatz for each pair of gluons in the final state.
  - ϕ_{ij} is the azimuthal angle of the decay particles in the plane perpendicular to the parent particle momentum.

Outputs of neural network

The output of the neural network is the colour and helicity summed matrix element

$$\langle |\mathcal{M}_{n+1}|^2 \rangle = \sum_{\{ijk\}} C_{ijk} D_{ij,k} \,.$$

Outputs of neural network

The output of the neural network is the colour and helicity summed matrix element

$$\langle |\mathcal{M}_{n+1}|^2 \rangle = \sum_{\{ijk\}} C_{ijk} D_{ij,k} \,.$$

During training, the predictions are compared against the target value of the matrix element which we scale according to

$$y = \text{arcsinh}\left(\frac{\langle |\mathcal{M}_{n+1}|^2 \rangle}{S_{\text{pred}}} \right). \tag{9}$$

Outputs of neural network

The output of the neural network is the colour and helicity summed matrix element

$$\langle |\mathcal{M}_{n+1}|^2 \rangle = \sum_{\{ijk\}} C_{ijk} D_{ij,k} \,.$$

During training, the predictions are compared against the target value of the matrix element which we scale according to

$$y = \text{arcsinh}\left(\frac{\langle |\mathcal{M}_{n+1}|^2 \rangle}{S_{\text{pred}}} \right). \tag{9}$$

Targets are also standardised to zero mean and unit variance.

Neural network architecture

The basis of our neural network model is a dense neural network.

Neural network architecture

The basis of our neural network model is a dense neural network.

The number of nodes in the input and output layer vary depending on the final state multiplicity.

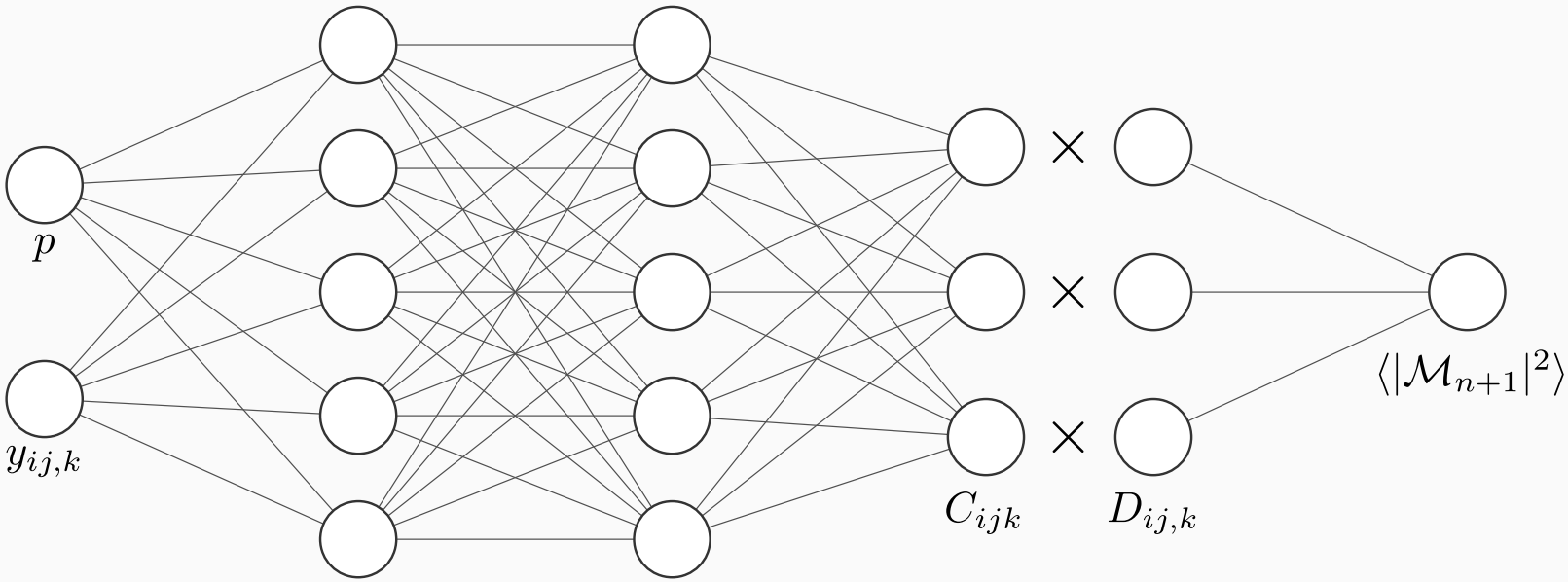


Figure 4: Schematic diagram of our neural network architecture.

Custom loss function

We use the mean squared error loss function along with a regularisation term

$$L = L_{\text{MSE}} + L_{\text{pen}}$$

$$L = \frac{1}{N} \sum_{n=1}^{N} \left[(y_n - f(\mathbf{x}_n; \theta))^2 + J \sum_i \frac{D_i^{-2}}{\sum_j D_j^{-2}} |C_i D_i| \right].$$

Custom loss function

We use the mean squared error loss function along with a regularisation term

$$L = L_{\text{MSE}} + L_{\text{pen}}$$

$$L = \frac{1}{N}\sum_{n=1}^{N}\left[(y_n - f(\mathbf{x}_n;\theta))^2 + J\sum_i \frac{D_i^{-2}}{\sum_j D_j^{-2}}|C_i D_i|\right].$$

We promote the network to learn about the universal factorisation property in matrix elements, hence becoming *factorisation-aware*.

Ensembling of models

Due to the stochastic nature of training neural networks, having one neural network is far from ideal.

Ensembling of models

Due to the stochastic nature of training neural networks, having one neural network is far from ideal.

Training a group of them and aggregating the results leads to much more robust predictions as we average out the stochasticity of the parameter optimisation.

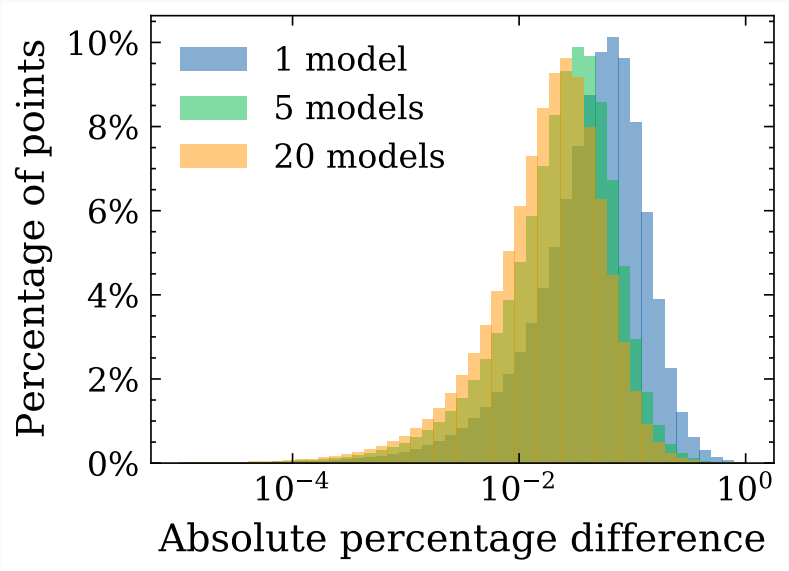


Figure 5: Ensembling more models increases predictive performance.

Results

Results

We present results for our matrix element emulator:

1. Compare results obtained with our method to those in a previous work by Aylett-Bullock and Badger [2].

[2] Simon Badger and Joseph Aylett-Bullock. "Using neural networks for efficient evaluation of high multiplicity scattering amplitudes". In: JHEP 06 (2020), p. 114.

Results

We present results for our matrix element emulator:

1. Compare results obtained with our method to those in a previous work by Aylett-Bullock and Badger [2].
2. Scaling performance by expanding the network size and number of training samples.

[2] Simon Badger and Joseph Aylett-Bullock. "Using neural networks for efficient evaluation of high multiplicity scattering amplitudes". In: JHEP 06 (2020), p. 114.

Results

We present results for our matrix element emulator:

1. Compare results obtained with our method to those in a previous work by Aylett-Bullock and Badger [2].
2. Scaling performance by expanding the network size and number of training samples.
3. Generalisation to unseen regions of phase-space by predicting on random phase-space trajectories.

[2] Simon Badger and Joseph Aylett-Bullock. "Using neural networks for efficient evaluation of high multiplicity scattering amplitudes". In: JHEP 06 (2020), p. 114.

Results

We present results for our matrix element emulator:

1. Compare results obtained with our method to those in a previous work by Aylett-Bullock and Badger [2].
2. Scaling performance by expanding the network size and number of training samples.
3. Generalisation to unseen regions of phase-space by predicting on random phase-space trajectories.
4. Compare MC statistical error to NN accuracy.

[2] Simon Badger and Joseph Aylett-Bullock. "Using neural networks for efficient evaluation of high multiplicity scattering amplitudes". In: JHEP 06 (2020), p. 114.

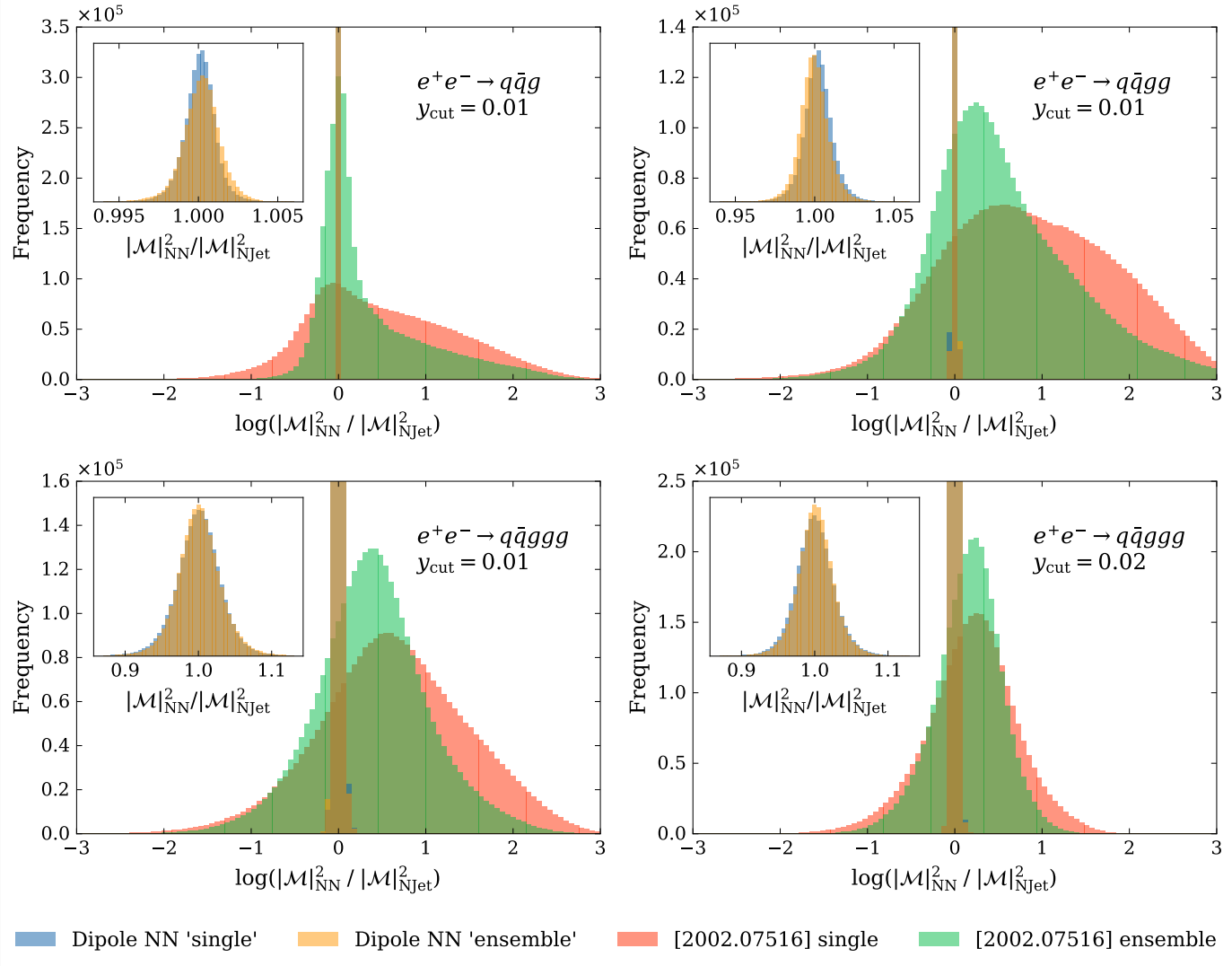
Results

We present results for our matrix element emulator:

1. Compare results obtained with our method to those in a previous work by Aylett-Bullock and Badger [2].
2. Scaling performance by expanding the network size and number of training samples.
3. Generalisation to unseen regions of phase-space by predicting on random phase-space trajectories.
4. Compare MC statistical error to NN accuracy.
5. Performance on a GPU machine.

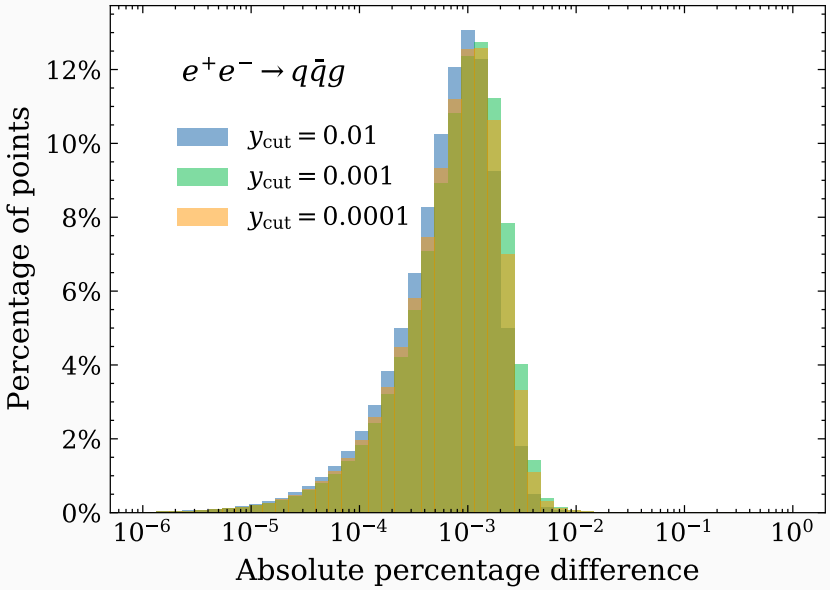
[2] Simon Badger and Joseph Aylett-Bullock. "Using neural networks for efficient evaluation of high multiplicity scattering amplitudes". In: JHEP 06 (2020), p. 114.

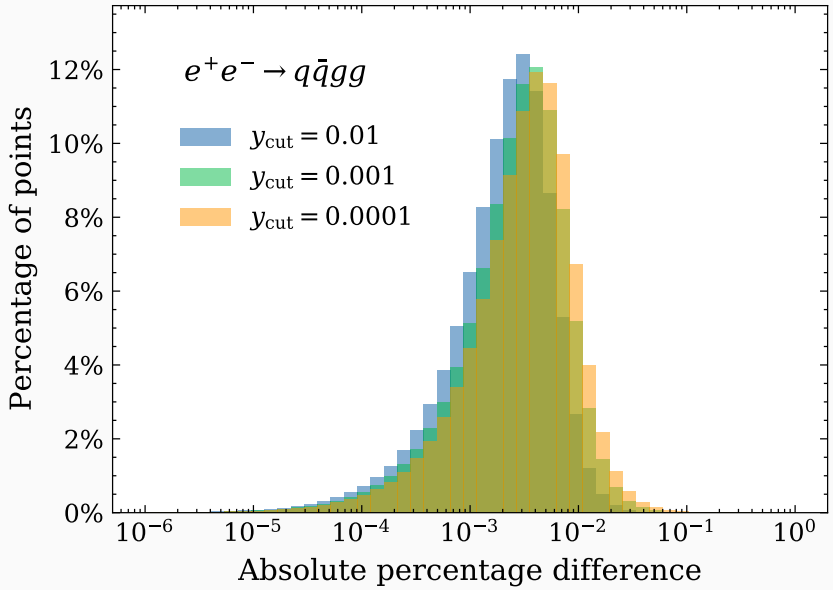
Comparison with n3jet[2]

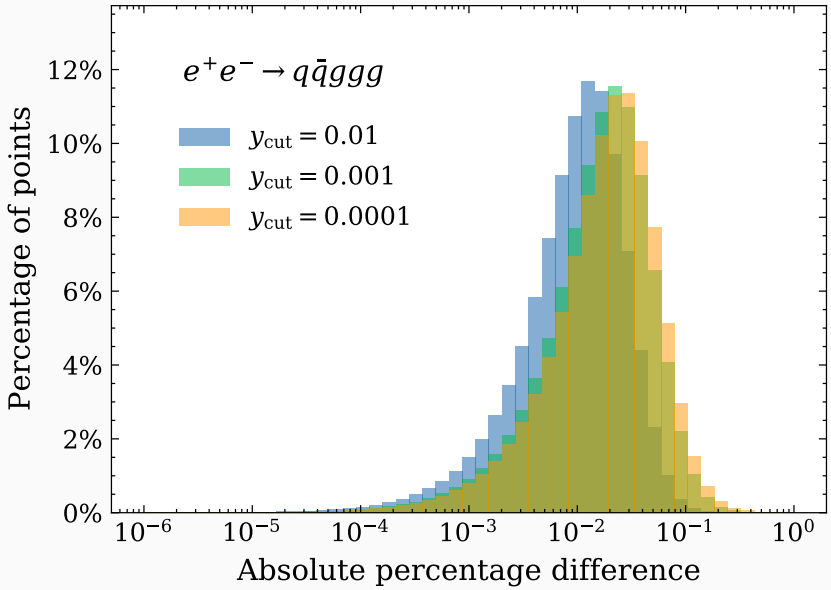


[2]Simon Badger and Joseph Aylett-Bullock. "Using neural networks for efficient evaluation of high multiplicity scattering amplitudes". In: JHEP 06 (2020), p. 114.

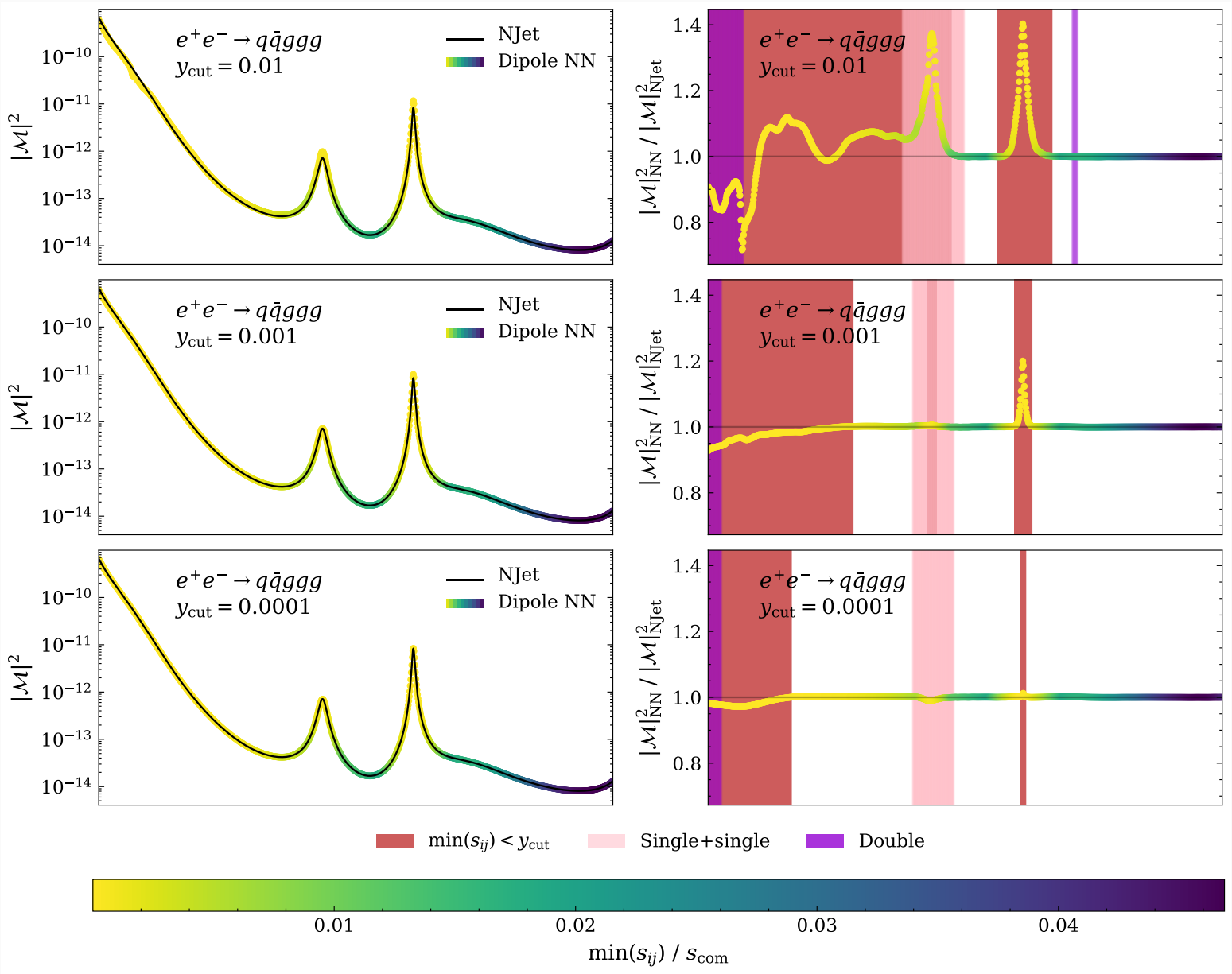
Error distributions for full size model



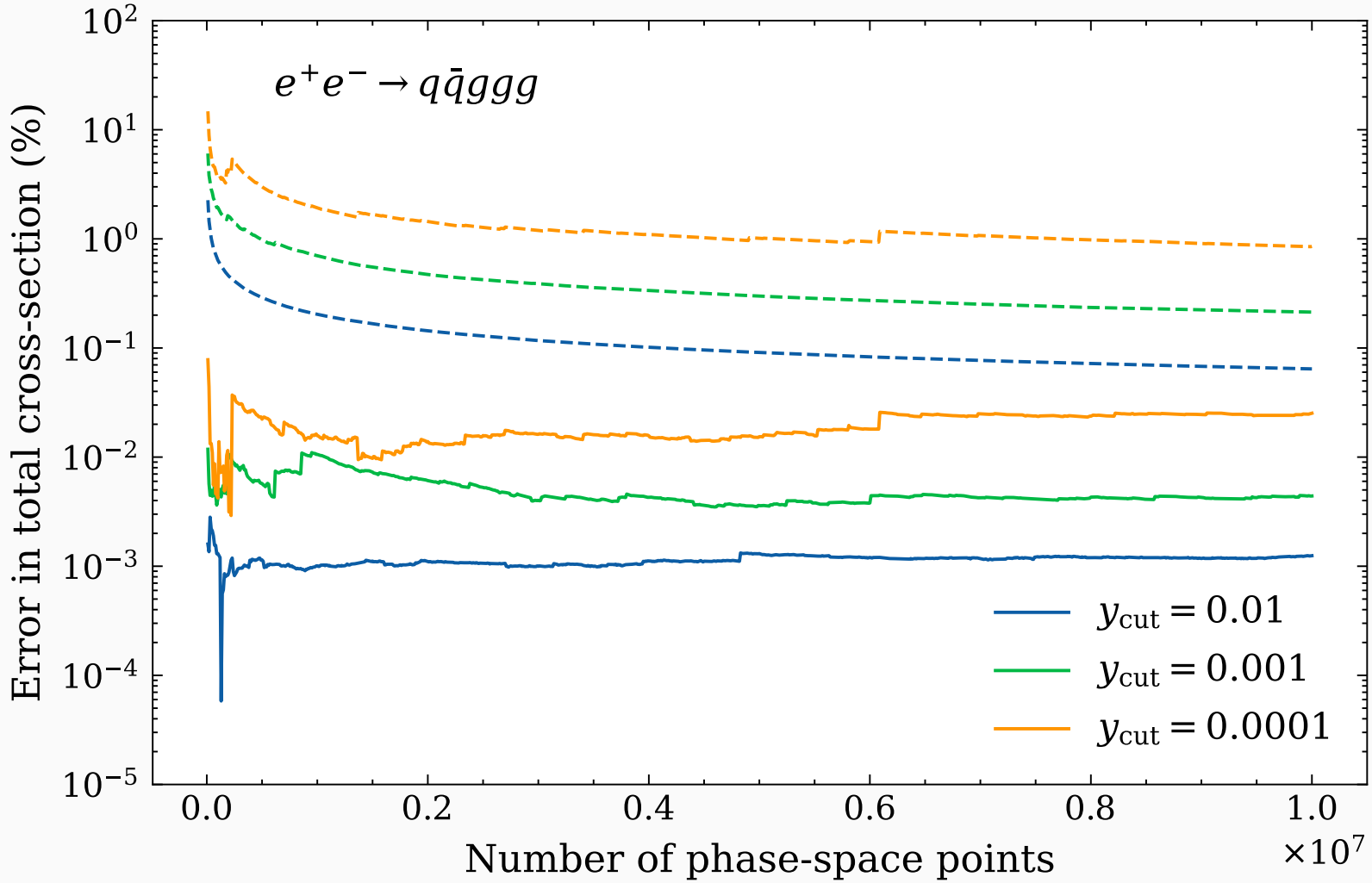




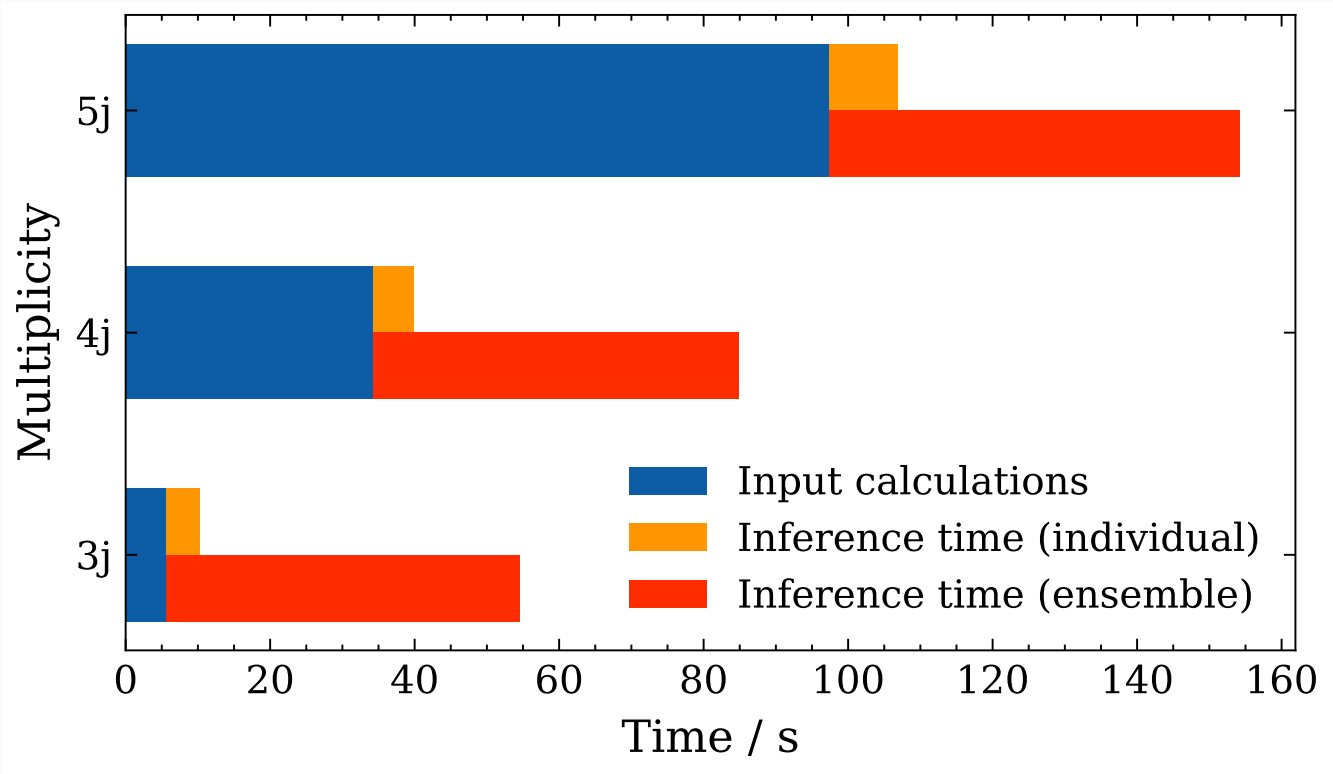
Random phase-space trajectory for 5 jets



Comparison of errors



Performance on GPU



Time taken to infer on 10 million phase-space points. Times measured on an Nvidia V100 32GB GPU and Intel Xeon Silver 4216 CPU @ 2.10GHz.

Conclusion and outlook

Conclusions

- We showed that it is possible to build a neural network emulator to accurately model matrix elements by incorporating our physics knowledge on infrared divergences.

Conclusions

- We showed that it is possible to build a neural network emulator to accurately model matrix elements by incorporating our physics knowledge on infrared divergences.
- We demonstrate that with careful treatment of divergences, our emulator shows drastically improved per-point performance compared to previous work.

Conclusions

- We showed that it is possible to build a neural network emulator to accurately model matrix elements by incorporating our physics knowledge on infrared divergences.
- We demonstrate that with careful treatment of divergences, our emulator shows drastically improved per-point performance compared to previous work.
- By building in basis functions into the emulator, we are able to safely extrapolate to unseen regions of phase-space.

Outlook

- Extension to one-loop matrix elements by changing the ingredients in ansatz.
- *pp* collisions
- Neural network error is lower than the statistical Monte Carlo error so can use the NN to augment datasets.

Thanks for listening.

Building the emulator

Our emulator is constructed with a densely-connected neural network built using the `Keras` API with the `TensorFlow` back-end with GPU support.

The pipeline looks like the following

- Generate datasets to train and test our neural network
- Create neural network architecture based on dipole factorisation
- Analyse performance by testing against two independent testing datasets.

Phase-space sampling

- Phase-space points are initially sampled uniformly using the RAMBO algorithm with a $\sqrt{s_{\text{com}}}$ = 1000 GeV.
- Global phase-space cuts are applied according to $y_{\text{cut}} \leq y_{ij}$ where y_{ij} are the Mandelstam invariants normalised by s_{com}.
- Phase-space points are then clustered using `FastJet` with the e^+e^- k_t algorithm.
- Jets are clustered exclusively with $d_{\text{cut}} = \max(2 \times y_{\text{cut}},\ 0.01 \times s_{\text{com}})$.
- We explore three different values of the global phase-space cut parameters $y_{\text{cut}} = [0.01, 0.001, 0.0001]$ to demonstrate the ability the factorisation-aware neural network to effectively interpolate in more and more singular regions of phase-space.

Specific details on network architecture

- Eight hidden layers consisting of $[64, 128, 256, 512, 768, 386, 128, 64]$ nodes
- `tanh` activation function on hidden layers
- Glorot uniform initialisation
- Adam optimiser with initial learning rate 0.001
- Training mini-batch size 4096
- `EarlyStopping` with patience of 40 epochs
- `ReduceLROnPlateau` with patience of 20 epochs with reduction factor 0.7
- These choices of hyperparameters were chosen empirically because they worked well from our testing. If we had infinite time then we would use better techniques to optimise them.

Random phase-space trajectories

- Pick two points in $4n$-dimensional hypercube
- Connect them
- Map these points using the RAMBO mapping
- There is no guarantee that the trajectory is well-behaved. Different nature to the test set which is generated in the same way as the training set.
- We show that even when the neural networks predicts on points that have never been seen before, it manages to extrapolate.