

Untersuchung von lorentz-äquivalenten Transformern als Matrixelement-Surrogate

Bachelor-Arbeit
zur Erlangung des Hochschulgrades
Bachelor of Science
im Bachelor-Studiengang Physik

vorgelegt von

LEOPOLD MAX MICHAEL
geboren am 12.04.2004 in Ebersbach

Institut für Kern- und Teilchenphysik
Fakultät Physik
Bereich Mathematik und Naturwissenschaften
Technische Universität Dresden
2025

Eingereicht am 10. Juni 2025

1. Gutachter: Prof. Dr. Michael Kobel
2. Gutachter: Prof. Dr. Arno Straessner

Zusammenfassung

Zusammenfassung

Die Verwendung von Monte-Carlo-Methoden für die ungewichtete Ereignisgeneration von Teilchenkollisionen ist ein notwendiger, aber sehr zeitintensiver Bestandteil der angewandten Teilchenphysik. Aushilfe verschafft ein surrogatbasierter Ungewichtungsalgorithmus, für den machine-learning zum Einsatz kommt. In dieser Arbeit wird die neue Transformerstruktur L-GATr an von *SHERPA* generierten Daten eines Beispielprozesses der Prozessgruppe $p p \rightarrow Z + 6 \text{ jets}$ untersucht und mit dem bisher besten Surrogatansatz verglichen. Dabei erreicht die neue Architektur nach der Optimierung ihrer Hyperparameter einen mittleren effektiven gain-Faktor von $f_{\text{eff}} = 16,8 \pm 3,1_{\text{stat}}$ beziehungsweise $f_{\text{eff}} = 25,4 \pm 0,6_{\text{stat}}$, abhängig von der Vorhersagemethode. Somit ist L-GATr vergleichbar beziehungsweise besser als das bisherige Surrogat eines MLP mit Dipolansatz.

Abstract

The application of Monte Carlo methods for unweighted event generation of proton collisions at the LHC is a necessary step, albeit a very resource-intensive one. Surrogate-based unweighting algorithms using machine learning are required to keep up with the ever-growing influx of data. In this work, a new machine learning architecture called L-GATr is examined using collision data of a $p p \rightarrow Z + 6 \text{ jets}$ process generated by *SHERPA*. The optimized network structure achieves a mean effective gain factor of $f_{\text{eff}} = 16,8 \pm 3,1_{\text{stat}}$ and $f_{\text{eff}} = 25,4 \pm 0,6_{\text{stat}}$, depending on its implementation. Hence, L-GATr is comparable and even exceeds the current best surrogate model.

Inhaltsverzeichnis

1	Einleitung	1
2	Physikalische Grundlagen	3
2.1	Das Standardmodell der Teilchenphysik	3
2.2	Der Wirkungsquerschnitt	4
2.3	Monte-Carlo-Integration	5
2.4	Ungewichtete Ereignisgeneration	6
2.5	Ungewichtete Ereignisgeneration mit Surrogaten	7
3	Maschinelles Lernen	9
3.1	Multi-Layer-Perzeptron (MLP)	9
3.2	QCD-Dipolentwicklung	11
3.3	Klassischer Transformer	12
3.4	Lorentz-äquivarianter Transformer (L-GATr)	14
3.5	Training mit OPTIMA	17
4	Lorentz-äquivariante Transformer als Matrixelement-Surrogate	21
4.1	MLP mit Dipolansatz als Referenz	21
4.2	L-GATr ohne kinematische Invarianten	25
4.3	L-GATr mit kinematischen Invarianten	29
4.4	L-GATr mit Dipolansatz	31
4.5	Hyperparameteroptimierung	33
5	Zusammenfassung	37
A	Anhang	39
A.1	HPO Ergebnisse	39
A.2	Ergebnisse mit 90%-Quantil	39
B	Literatur	41
C	Glossar	43

1 Einleitung

Seit seiner Inbetriebnahme im September 2008 hat sich der Large Hadron Collider (LHC) als das größte, komplizierteste und womöglich auch bekannteste Präzisionsmessgerät der modernen Physik etabliert. Mit der Entdeckung des Higgs-Bosons 2012, das bereits 1964 von Peter Higgs postuliert wurde [1], gelang dem LHC ein erster großer Erfolg [2]. Derzeit werden in 4 großen Experimenten (ATLAS, CMS, ALICE und LHCb) und 5 kleineren Experimenten (MoEDAL-MAPP, TOTEM, LHCf, SND@LHC und FASER) verschiedene Bereiche des Standardmodells der Teilchenphysik vermessen.

Um die Theorie des Standardmodells zu nutzen und konkrete physikalische Aussagen aufzustellen, nutzt man Monte-Carlo-Ereignisgeneratoren, in denen der gesamte Vorgang von der Teilchenkollision bis zur Messung mit einem Detektor simuliert wird. Im Vergleich mit tatsächlichen Messdaten lässt sich so herausfinden, an welchen Stellen das Standardmodell erweitert werden muss. Der mit den Simulationen verbundene Rechenaufwand ist enorm, sodass geschickte Datenverarbeitungsmethoden zwingend notwendig sind. Eine bewährte Herangehensweise ist die Durchführung eines Ungewichtungsalgorithmus, der die Anzahl der zu simulierenden Ereignisse für die weiterführenden Schritte reduziert. Stattet man diesen Algorithmus mit maschinellem Lernen aus, erhält man eine zusätzliche Einsparung an Rechenzeit.

In dieser Arbeit wird eine neuartige Netzwerkarchitektur untersucht, die für die Anwendung in der Hochenergiephysik besonders gut geeignet sein soll. Genauer gesagt soll diese neue Struktur als Surrogat für die Abschätzung des Matrixelementgewichts eingesetzt werden.

In Kapitel 2 werden zunächst die physikalischen Grundlagen, die Monte-Carlo-Integration und die Ereignisgeneration kurz erläutert. Kapitel 3 befasst sich dann mit allgemeinen Konzepten des maschinellen Lernens und der Beschreibung der in dieser Arbeit verwendeten Netzwerkarchitekturen. In Kapitel 4 werden diese dann im Kontext der Ereignisgeneration angewendet und miteinander verglichen. Zum Schluss findet noch eine Optimierung der verwendeten Netzwerke statt, durch die eine größtmögliche Effizienz gewährleistet werden soll.

2 Physikalische Grundlagen

2.1 Das Standardmodell der Teilchenphysik

In der Teilchenphysik wird das gesamte Universum durch quantisierte Felder beschrieben. Eine Anregung eines solchen Feldes wird als Teilchen interpretiert. Dabei unterscheidet man zwei Arten von Teilchen: Fermionen sind die Elementarteilchen, aus denen alle Materie besteht, und Bosonen vermitteln die verschiedenen Wechselwirkungen zwischen den Fermionen. Man unterteilt die Fermionen weiter in Leptonen, welche an der schwachen Wechselwirkung und teilweise auch an der elektromagnetischen Wechselwirkung beteiligt sind, und Quarks, die zusätzlich stark wechselwirken können. Die vierte fundamentale Kraft, die Gravitation, wird im Standardmodell allerdings nicht beschrieben.

Wie der Name LHC bereits vermuten lässt, kollidieren in diesem Teilchenbeschleuniger Protonen, welche jeweils aus Quarks und Gluonen bestehen. Daher ist die Quantenchromodynamik (QCD), welche die starke Wechselwirkung beschreibt, ein unabdingbarer Bestandteil aller Interaktionen im Detektor. Es ist somit sehr wichtig, diese Prozesse so genau wie möglich beschreiben zu können. Um neue Physik zu entdecken, werden besonders seltene Kollisionsereignisse gemessen und analysiert. Aufgrund ihrer Seltenheit müssen daher möglichst viele unwichtige Hintergrundereignisse ausgeschlossen werden, was deren genaues Verständnis voraussetzt.

Ein Feynman-Diagramm eines dieser Hintergrundereignisse ist in Abbildung 2.1 dargestellt. Man erkennt 5 auslaufende Gluonen und ein auslaufendes Up-Quark, welche im hadronischen Kalorimeter des Detektors als 6 Jets gemessen werden. Außerdem strahlt das einlaufende Up-Quark ein Z-Boson ab, das dann in ein Elektron-Positron-Paar zerfällt. Dieser Prozess wird daher der Prozessgruppe $p p \rightarrow Z + 6 \text{ jets}$ zugeordnet und spielt eine entscheidende Rolle für den Hintergrund in einer Proton-Proton-Kollision. Insbesondere die große Anzahl von QCD-Jets ist hier bemerkenswert, da dies die genaue Beschreibung des Prozesses im Rahmen des Standardmodells erschwert.

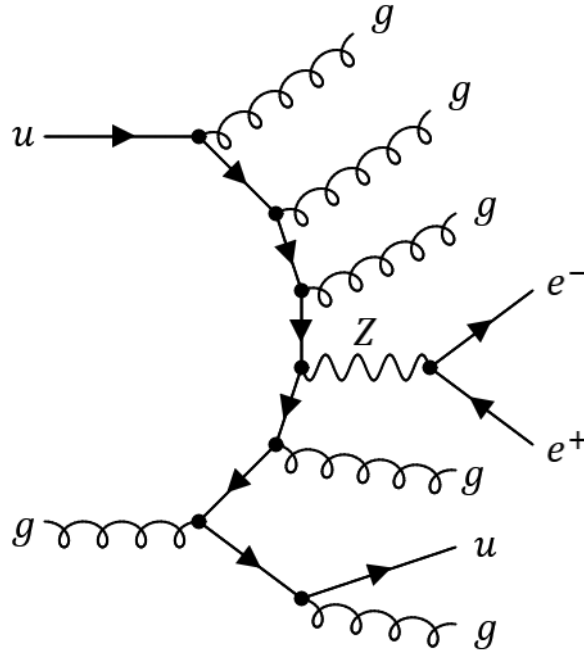


Abbildung 2.1: Ein Feynman-Diagramm führender Ordnung für den Prozess $g u \rightarrow e^+ e^- g g g g g u$ aus der Prozessgruppe $p p \rightarrow Z + 6 \text{ jets}$. Erstellt mit „Feynman diagram maker“ [3].

2.2 Der Wirkungsquerschnitt

Die zentrale Größe einer Teilchenkollision ist der Wirkungsquerschnitt σ . Er gibt ein Maß für die Wahrscheinlichkeit an, dass ein spezieller Endzustand aus dem gegebenen Anfangszustand hervorgeht. Experimentell bestimmt man σ über die Formel

$$N = \sigma \cdot L \quad (2.1)$$

aus der integrierten Luminosität L des Detektors und der Anzahl N der Kollisionen. Der Wirkungsquerschnitt lässt sich aber auch aus dem sogenannten Matrixelement M_{fi} durch eine Integration über den verfügbaren Phasenraum berechnen [4].

$$\sigma = \frac{\hbar^2 \mathcal{S}}{4\sqrt{(p_1 \cdot p_2)^2 - p_1^2 p_2^2}} \left[\int \prod_{i=1}^m \frac{c \cdot d^3 \vec{p}'_i}{2E'_i (2\pi)^3} \right] |M_{fi}|^2 (2\pi)^4 \delta^{(4)} \left(\sum_{i=1}^m p'_i - p_2 - p_1 \right) \quad (2.2)$$

Hier sind p_1 und p_2 die Viererimpulse der beiden Teilchen im Anfangszustand, p'_i und $\vec{p}'_i$ bezeichnen die Viererimpulse beziehungsweise die Dreierimpulse der m Teilchen im Endzustand und E'_i ihre zugehörigen Energien. $\mathcal{S}$ ist ein statistischer Faktor, der die Ununterscheidbarkeit identischer Teilchen im Endzustand berücksichtigt. $\delta^{(4)}$ ist als die Delta-Distribution zu interpretieren, die auf die einzelnen Komponenten der Viererimpulse wirkt. Sie sorgt für die Energie- und Impulserhaltung zwischen dem Anfangs- und Endzustand. Die größte Bedeutung

in dieser Formel hat allerdings das Matricelement M_{fi} . Es beschreibt die Interaktionen der Teilchen während der Kollision und ist somit abhängig von der vorausgesetzten Theorie. Beispielsweise könnte Supersymmetrie (SUSY) als „beyond-the-standard-model“-Theorie (BSM) für die Berechnung von M_{fi} genutzt werden, wodurch sich die Vorhersagen über die Wechselwirkungswahrscheinlichkeiten ändern. Im Abgleich mit tatsächlichen Messungen können so neue Theorien bestätigt oder widerlegt werden.

2.3 Monte-Carlo-Integration

In der Physik kommt es häufig vor, dass man Integrale berechnen muss, für die es keine analytische Lösung gibt. Insbesondere die Formel (2.2) im vorherigen Abschnitt ist äußerst nicht-trivial. Um diese Integrale numerisch annähern zu können, müssen Computer und geschickte Integrationsmethoden benutzt werden. Insbesondere ist die Methode der Monte-Carlo-Integration relevant für die Teilchenphysik, da hier häufig mehrdimensionale Integrale berechnet werden müssen, für die herkömmliche Integrationsverfahren keine ausreichende Genauigkeit liefern.

Sei $\Omega := [0, 1]^n$ der n -dimensionale Einheitshyperwürfel und sei $f : \Omega \rightarrow \mathbb{R}$ eine Funktion. Dann wird das zu berechnende Integral mit der Monte-Carlo-Integration angenähert als

$$I = \int_{\Omega} f(\vec{x}) d^n \vec{x} \approx \frac{1}{N} \sum_{i=1}^N f(\vec{x}_i), \quad (2.3)$$

wobei alle $\vec{x}_i$ zufällig uniform aus Ω gewählt werden und N möglichst groß sein soll. Definiert man die Gewichte $w_i := f(\vec{x}_i)$, dann ergibt sich das Integral gerade aus dem Erwartungswert dieser Gewichte [5] [6].

$$I \approx \frac{1}{N} \sum_{i=1}^N w_i = \langle w \rangle \quad (2.4)$$

Insbesondere ist die Berechnung des Wirkungsquerschnitts nach Gleichung (2.2) interessant. Um σ mit der Monte-Carlo-Integration berechnen zu können, definiert man das Matricelementgewicht

$$w_{\text{ME}} := |M_{fi}|^2 \quad (2.5)$$

und das Phasenraumgewicht

$$w_{\text{PS}} := \left[\prod_{i=1}^m \frac{c \cdot d^3 \vec{p}'_i}{2E'_i (2\pi)^3} \right] (2\pi)^4 \delta^{(4)} \left(\sum_{i=1}^m p'_i - p_2 - p_1 \right). \quad (2.6)$$

Daraus ergibt sich dann das Gesamtgewicht

$$w = w_{\text{ME}} \cdot w_{\text{PS}}, \quad (2.7)$$

welches für die Approximation des Integrals nach Gleichung (2.4) genutzt werden kann.

2.4 Ungewichtete Ereignisgeneration

Um Fortschritte in der Teilchenphysik erreichen zu können, muss man Theorievorhersagen mit tatsächlichen Messwerten vergleichen. Da dieser Vergleich eine nahezu astronomische Präzision erfordert, muss jeder Schritt von der Kollision der Teilchen bis zum eigentlichen Messprozess im Detektor simuliert werden. Dabei ist insbesondere die Detektorsimulation sehr rechenintensiv. Um die gewünschte Präzision zu erreichen, benötigt man eine große Anzahl von Ereignissen im Bereich von einigen Millionen oder mehr. Dabei treten häufig auch Ereignisse auf, deren Gewicht um mehrere Größenordnungen kleiner ist als das von anderen. Diese Ereignisse tragen dann nicht besonders stark zum Wirkungsquerschnitt bei, müssen aber trotzdem in allen Berechnungen berücksichtigt werden. Im Hinblick auf den hohen Rechenaufwand der Detektorsimulation ist es jedoch ein unnötiger Verbrauch wertvoller Rechenressourcen, alle diese Ereignisse zu simulieren, wenn ihr Beitrag im Endergebnis beinahe minimal ist. Man kann aber auch nicht einfach einige Ereignisse vernachlässigen, denn dann beschreibt das Endergebnis nicht mehr dieselbe Physik. Die Lösung ist ein Ungewichtungsalgorithmus, der die Anzahl der Ereignisse reduziert, ohne die Physik zu verändern [5].

Algorithmus 1 : Die herkömmliche ungewichtete Ereignisgeneration.

```

while true do
    Erzeuge einen Phasenraumpunkt  $u$ ;
    Berechne das exakte Gewicht  $w$ ;
    Erzeuge eine uniform verteilte Zufallszahl  $R \in [0, 1)$ ;
    if  $w > R \cdot w_{\max}$  then
        | return  $u$  und  $\tilde{w} = \max(1, w/w_{\max})$ 
    end
end

```

Hier wird mit Monte-Carlo-Methoden ein Phasenraumpunkt u aus dem erlaubten Phasenraum generiert. Dieses Ereignis besitzt zunächst das Gewicht w , welches ein Maß für seine Wahrscheinlichkeit ist.

Für die weitere Verarbeitung benötigt man allerdings ungewichtete Ereignisse, also solche mit $w = 1$. Im Algorithmus 1 werden Ereignisse daher zufällig akzeptiert oder verworfen mit einer Wahrscheinlichkeit, die ihrem relativen Gewicht $w/w_{\max}$ entspricht. Dadurch wird die Anzahl der unwichtigen Ereignisse stark reduziert. Man kann dann eine Ungewichtungseffizienz definieren als

$$\epsilon_{\text{full}} := \frac{N}{N^{\text{trials}}} \approx \frac{\langle w \rangle}{w_{\max}}, \quad (2.8)$$

wobei N^{trials} die Anzahl der gewichteten Ereignisse ist und N die Anzahl der ungewichteten Ereignisse, die nach dem Algorithmus übrig bleiben. Hierbei wird $w_{\max}$ vor der Berechnung

als das 99,9%-Quantil aller Gewichte w definiert, um den Einfluss besonders großer Ausreißer auf die Ungewichtungseffizienz zu reduzieren [5]. Es muss allerdings angemerkt werden, dass dadurch nach der Anwendung des Algorithmus nicht alle Ereignisse ein Gewicht von 1 haben. Man erreicht nur eine partielle Ungewichtung.

2.5 Ungewichtete Ereignisgeneration mit Surrogaten

Der nächste Engpass in Algorithmus 1 ist nun allerdings die Berechnung der Ereignisgewichte w . Diese kann besonders für Ereignisse mit einer großen Anzahl von QCD-Jets sehr zeitaufwändig werden. Beispielsweise für den Prozess aus Abbildung 2.1 beträgt die Berechnungszeit für w_{ME} circa 1,23 Sekunden pro Ereignis. Um dieses Problem zu beheben, kann man den Algorithmus 1 anpassen, indem man einen zweiten Ungewichtungsschritt einführt.

Algorithmus 2 : Zweistufige ungewichtete Ereignisgeneration mit einem Surrogat.

```

while true do
    Erzeuge einen Phasenraumpunkt  $u$ ;
    Berechne ein approximatives Gewicht  $s$ ;
    Erzeuge eine uniform verteilte Zufallszahl  $R_1 \in [0, 1)$ ;
    # Erster Ungewichtungsschritt
    if  $s > R_1 \cdot s_{\text{max}}$  then
        Berechne das exakte Gewicht  $w$ ;
        Bestimme das Verhältnis  $x = w/s$ ;
        Erzeuge eine uniform verteilte Zufallszahl  $R_2 \in [0, 1)$ ;
        # Zweiter Ungewichtungsschritt
        if  $x > R_2 \cdot x_{\text{max}}$  then
            | return  $u$  und  $\tilde{w} = \max(1, s/s_{\text{max}}) \cdot \max(1, x/x_{\text{max}})$ 
        end
    end
end
end

```

In Algorithmus 2 wird zunächst nur eine Annäherung s an das tatsächliche Gewicht w_{ME} berechnet. Diese Annäherung wird durch ein sogenanntes Surrogat bestimmt, beispielsweise durch ein MLP (siehe 3.1) oder ein anderes machine-learning-Verfahren, und setzt sich zusammen aus der Vorhersage a des Surrogats und dem Phasenraumgewicht w_{PS} als $s = a \cdot w_{\text{PS}}$. Hier kann man wieder die entsprechenden Ungewichtungseffizienzen für die einzelnen Schritte definieren [5].

$$\epsilon_{\text{full}} := \frac{N}{N^{\text{trials}}} \approx \frac{\langle w \rangle}{w_{\text{max}}} \quad (2.9)$$

$$\epsilon_{1\text{st, surr}} := \frac{N_{2\text{nd, surr}}^{\text{trials}}}{N_{1\text{st, surr}}^{\text{trials}}} \approx \frac{\langle s \rangle}{s_{\text{max}}} \quad (2.10)$$

$$\epsilon_{2\text{nd, surr}} := \frac{N}{N_{2\text{nd, surr}}^{\text{trials}}} \approx \frac{\langle x \rangle}{x_{\text{max}}} \quad (2.11)$$

Hier sind $N_{\text{step}}^{\text{trials}}$ die Anzahlen der Ereignisse, die in den jeweiligen Schritt des Algorithmus eingehen, und N ist die Anzahl der ungewichteten Ereignisse. ϵ_{full} ist die Ungewichtungseffizienz von Algorithmus 1 und $\epsilon_{1\text{st, surr}}$ und $\epsilon_{2\text{nd, surr}}$ sind die Effizienzen der beiden Schritte in Algorithmus 2. w_{max} und s_{max} sind analog zum Algorithmus 1 die 99,9%-Quantile aller w beziehungsweise s . Der Wert x_{max} wird dagegen als das x_i gewählt, für das $s_i \cdot x_i$ das 99,9%-Quantil von $\vec{s} \cdot \vec{x}$ ist [6].

Mit einem perfekten Surrogat gilt $x = 1$ für alle Ereignisse. In diesem Fall ist die gesamte Ungewichtungseffizienz von Algorithmus 2 identisch zum Algorithmus 1. Wenn allerdings die Berechnung von a schneller ist als die von w_{ME} , dann kann die Gesamtberechnungsdauer deutlich abnehmen. Das exakte Matricelementgewicht muss nämlich nur noch für die Ereignisse berechnet werden, die den ersten Ungewichtungsschritt passieren. Um den Vorteil von Algorithmus 2 quantitativ zu untersuchen, definiert man den effektiven gain-Faktor [7]

$$f_{\text{eff}} := \frac{T_{\text{Standard}}}{T_{\text{Surrogat}}}. \quad (2.12)$$

T_{Standard} und T_{Surrogat} sind dabei die Zeiten, die die Algorithmen 1 und 2 im Mittel benötigen, um gleich viele (partiell) ungewichtete Ereignisse zu produzieren. Diese sind gegeben durch

$$\begin{aligned} T_{\text{Standard}} &= N^{\text{trials}} \cdot (t_{\text{ME}} + t_{\text{PS}}) \\ T_{\text{Surrogat}} &= N_{1\text{st, surr}}^{\text{trials}} \cdot (t_{\text{surr}} + t_{\text{PS}}) + N_{2\text{nd, surr}}^{\text{trials}} \cdot t_{\text{ME}}. \end{aligned} \quad (2.13)$$

Hier sind t_{ME} , t_{PS} und t_{surr} jeweils die mittleren Zeiten, um das Matricelementgewicht, das Phasenraumgewicht beziehungsweise die Surrogatapproximation einmalig zu berechnen. Mit den Definitionen aus den Gleichungen (2.9) bis (2.11) ergibt sich durch Einsetzen:

$$f_{\text{eff}} = \left(\frac{t_{\text{surr}} + t_{\text{PS}}}{t_{\text{ME}} + t_{\text{PS}}} \cdot \frac{\epsilon_{\text{full}}}{\epsilon_{1\text{st, surr}} \cdot \epsilon_{2\text{nd, surr}}} + \frac{t_{\text{ME}}}{t_{\text{ME}} + t_{\text{PS}}} \cdot \frac{\epsilon_{\text{full}}}{\epsilon_{2\text{nd, surr}}} \right)^{-1}. \quad (2.14)$$

Mit diesem Faktor lässt sich beschreiben, ob sich die Anwendung eines Surrogats lohnt. Bei $f_{\text{eff}} > 1$ ist die Berechnungszeit mit dem Surrogat trotz der geringeren Genauigkeit schneller, und somit ist dieser Ansatz empfehlenswert.

3 Maschinelles Lernen

In der modernen Physik sind viele Gesetzmäßigkeiten so kompliziert geworden, dass eine Berechnung per Hand unmöglich beziehungsweise unpraktisch ist. Daher werden heutzutage sehr oft Computer zu Hilfe genommen, um physikalische Zusammenhänge zu berechnen oder zu modellieren. In der klassischen Programmierung schreibt man dabei ein Programm, das alle relevanten physikalischen Gesetze und Annahmen enthält. Diesem Programm übergibt man die Startbedingungen des Experiments, und der Computer berechnet daraus die Vorhersagen, die aus der Theorie hervorgehen. Allerdings ist es insbesondere in der Teilchenphysik so, dass man (noch) nicht alle Gesetzmäßigkeiten kennt oder dass eine direkte Berechnung zu zeitaufwendig ist. In diesem Fall kann man maschinelles Lernen verwenden, um trotzdem zufriedenstellende Ergebnisse zu erhalten. Hierbei übergibt man dem Rechner die Startbedingungen und die Zielwerte, die beispielsweise aus tatsächlichen Messungen hervorgehen. Der Computer kann aus diesen Trainingsdaten einen Algorithmus entwerfen, der die Startbedingungen auf die Zielwerte abbildet. Ein gut trainiertes Surrogat kann dann auch Vorhersagen aus Eingabedaten berechnen, die nicht während des Trainings verwendet wurden. Für die Umsetzung dieses Lernprozesses gibt es viele verschiedene Netzwerkstrukturen, von denen die für diese Arbeit relevanten in den folgenden Abschnitten vorgestellt werden.

3.1 Multi-Layer-Perzeptron (MLP)

Ein Multi-Layer-Perzeptron (kurz MLP) lässt sich durch die folgenden mathematischen Operationen definieren:

$$\text{Linear}(\vec{x}) := \vec{x}W + \vec{b} \quad (3.1)$$

$$\text{Layer}(\vec{x}) := f(\text{Linear}(\vec{x})) \quad (3.2)$$

Hier ist $\vec{x} \in \mathbb{R}^m$ ein Zeilenvektor mit m Einträgen, $\text{Linear}(\vec{x}) \in \mathbb{R}^n$, $\text{Layer}(\vec{x}) \in \mathbb{R}^n$ und $\vec{b} \in \mathbb{R}^n$ sind Zeilenvektoren mit n Einträgen und W ist eine $m \times n$ -Matrix. $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ ist eine nichtlineare Aktivierungsfunktion. In der Anwendung von MLPs im Rahmen dieser Bachelorarbeit wird für f immer die Funktion $\text{swish} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ verwendet. Die Ausgabe dieser Funktion hat die Komponenten

$$(\text{swish}(\vec{y}))_i = \frac{y_i}{1 + e^{-y_i}}. \quad (3.3)$$

Der Verlauf von swish im eindimensionalen Fall ist in Abbildung 3.3 dargestellt. Der vollständige MLP setzt sich dann zusammen als

$$\text{MLP}(\vec{x}) := \text{Linear} \circ \underbrace{\text{Layer} \circ \dots \circ \text{Layer}}_N \circ \text{Layer}(\vec{x}). \quad (3.4)$$

Das Netzwerk besteht somit aus einer Eingabeschicht, einer Ausgabeschicht und N versteckten Schichten. Die Struktur eines solchen Netzes ist in Abbildung 3.1 anschaulich dargestellt.

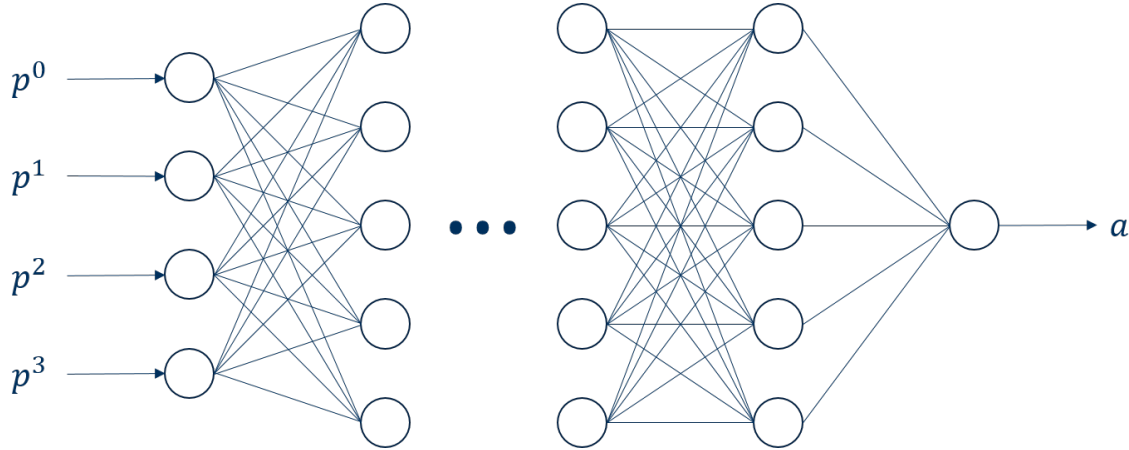


Abbildung 3.1: Schematischer Aufbau eines MLPs. Dieses Netz nimmt als Eingabewerte die Komponenten p^μ eines Viererimpulses und liefert die Approximation a an das Matrixelementgewicht w_{ME} als Ausgabe.

Der hier abgebildete MLP hat zur Vereinfachung nur 4 Neuronen in der Eingabeschicht, beliebig viele versteckte Schichten mit jeweils 5 Neuronen und eine Ausgabeschicht mit einem Neuron. Die Anzahl der Neuronen bestimmt die Dimension der Vektoren, die dem Netzwerk als Eingabe gegeben und als Ausgabe entnommen werden. Außerdem wird dadurch die Größe der Weight-Matrizen W und der Bias-Vektoren $\vec{b}$ (siehe Gleichung (3.1)) festgelegt. In der Anwendung der MLPs zur Bestimmung der Approximation s werden dem Netz als Eingabe alle Komponenten der Viererimpulse aller Teilchen des Anfangs- und Endzustands übergeben, sowie weitere kinematische Konstanten.

Um einen MLP zu trainieren, benötigt man neben den Eingabe- und Zieldaten noch eine sogenannte loss-Funktion. Diese dient während des Trainings als Maß für die Genauigkeit des Netzes. Die loss-Funktion ist abhängig von den Werten von W und $\vec{b}$, welche im Folgenden häufig kollektiv als Parameter des Netzes bezeichnet werden. Für das Erlernen der Matrixelementgewichte wird die folgende loss-Funktion verwendet:

$$l := \frac{1}{n} \sum_{i=1}^n \left(\text{arsinh} \left(\frac{a_i}{S_{\text{pred}}} \right) - \text{arsinh} \left(\frac{w_{\text{ME},i}}{S_{\text{pred}}} \right) \right)^2 \quad (3.5)$$

Hier ist n die Anzahl der Trainingsdaten und S_{pred} wird als das minimale w_{ME} aus dem Trainingsdatensatz gewählt. Das Training des Netzes geschieht dann durch die Anpassung der

Parameter gemäß den Formeln der Backpropagation.

$$P^{\text{neu}} = P^{\text{alt}} - h \cdot \frac{\partial l}{\partial P} \quad (3.6)$$

Hier steht P für einen einzelnen Parameter, also beispielsweise für einen Eintrag in einer Weight-Matrix. Der Wert h ist die Lernrate, die die Konvergenzgeschwindigkeit des Algorithmus beeinflusst. Die Berechnung in Gleichung (3.6) wird für jeden Parameter des Netzes von dem Optimierer *ADAM* durchgeführt [8]. Nachdem die Parameter angepasst wurden, wird die Backpropagation solange wiederholt, bis das Netz eine zufriedenstellende Genauigkeit erreicht hat. In der mathematischen Beschreibung entspricht Gleichung (3.6) dem iterativen Annähern eines Minimums der loss-Funktion. Idealerweise wünscht man sich, dass das globale Minimum gefunden wird. In der Praxis findet man aber meistens nur ein lokales Minimum. Für eine genauere und intuitivere Betrachtung dieser auch als „neuronales Netz“ bezeichneten Architektur sei auf [6] und auf die Playlist [9] verwiesen.

Die exakte Berechnung der loss-Funktion erfordert die Iteration über den gesamten Trainingsdatensatz. Ein vollständiges Durchlaufen des Trainingsdatensatzes wird als Epoche bezeichnet. Da unser Datensatz aus mehreren hunderttausend Ereignissen besteht, würde ein einzelner Trainingsschritt sehr lange dauern. Stattdessen wird der Trainingsdatensatz in viele kleinere Batches aufgeteilt, sodass mehrere Trainingsschritte pro Epoche vorgenommen werden können. Das Netzwerk kann dann mit Matrix-Matrix-Multiplikationen arbeiten, anstatt nur mit Vektor-Matrix-Multiplikationen.

Zusätzlich zu den Parametern W und $\vec{b}$ gibt es noch andere Größen, die den MLP beschreiben. So sind zum Beispiel die Lernrate h oder die Anzahl N der versteckten Schichten Zahlen, die im Training und im Aufbau eine entscheidende Rolle spielen. Diese Größen werden allerdings nicht im Verlauf des Trainings verändert und werden somit kollektiv als Hyperparameter bezeichnet. Die Leistung eines Netzwerks hängt mitunter stark von den konkreten Werten der Hyperparameter ab.

3.2 QCD-Dipolentwicklung

Das Matrixelementgewicht für den Prozess 2.1 weist starke Schwankungen über mehrere Größenordnungen auf. Daher ist das direkte Erlernen von w aus den Trainingsdaten nicht möglich. Stattdessen wird der Ansatz aus [7] gewählt. Hier wird das Matrixelementgewicht in eine Summe aus QCD-Dipolfunktionen D_{ijk} entwickelt.

$$|M_{fi}|^2 = w_{\text{ME}} \approx \sum_{\{ijk\}} C_{ijk} D_{ijk} \quad (3.7)$$

Die Summe läuft hier jeweils über die drei Teilchen, die an der Dipolfunktion beteiligt sind.

Das Surrogat soll dann die Koeffizienten C_{ijk} aus den Trainingsdaten erlernen. Genauer gesagt erlernt das Netz die Koeffizienten c_{ijk} , welche über

$$C_{ijk} = S_{\text{coef}} \cdot \sinh c_{ijk} \quad (3.8)$$

in die Dipolkoeffizienten umgerechnet werden [7]. Die Koeffizientenskalierung S_{coef} berechnet sich hierbei aus der Vorhersageskalierung S_{pred} in Gleichung (3.5) und aus der Dipolskalierung S_{dipole} , die auf den Median aller Dipolfunktionswerte gesetzt wird. Dann gilt

$$S_{\text{coef}} = S_{\text{pred}} / S_{\text{dipole}}. \quad (3.9)$$

3.3 Klassischer Transformer

2017 stellte Google den Transformer als neue Architektur für maschinelles Lernen vor [10]. Seitdem erfreut sich diese Netzwerkstruktur großer Beliebtheit aufgrund ihrer hervorragenden Leistung, großer Flexibilität und guten Skalierbarkeit. In [10] wird der Transformer als Encoder-Decoder-Struktur eingeführt, welche beispielsweise für das Modellieren von Sprachen besonders effizient ist. Da wir für die Berechnung des Matrixelementgewichts jedoch kein autoregressives Netzwerk benötigen, können wir die in [10] dargestellte Struktur wesentlich vereinfachen. Wenn das Trainingsdatenset, wie in Abschnitt 3.1 beschrieben, in mehrere Batches unterteilt wird, kann der Transformer mit Matrizen als Eingabewerten arbeiten. Die allgemeine Struktur eines Transformers ist schematisch in Abbildung 3.2 dargestellt.

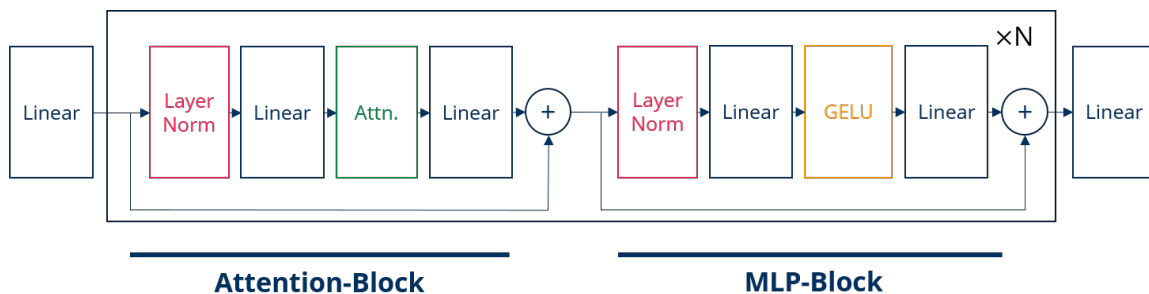


Abbildung 3.2: Schematischer Aufbau eines einfachen Transformers. Jedes der Rechtecke stellt eine der Operationen in den Gleichungen (3.10) bis (3.19) dar. Die eingekreisten Additionen sind die Rückkopplungen in Gleichung (3.16) bzw. (3.17). Diese Abbildung ist eine Nachempfindung aus [11].

Jedes der Rechtecke stellt hierbei eine der folgenden Operationen dar (vergleiche [12]):

$$\text{Linear}(x) = xW + b \quad (3.10)$$

$$\text{LayerNorm}(x) = \bar{x} = x \cdot \left(\frac{1}{n_E} \sum_{c=1}^{n_E} x_c^2 + \epsilon \right)^{-1/2} \quad (3.11)$$

$$(\text{Attention}(Q, K, V))_{ic} = \sum_{j=1}^{n_B} \text{softmax}_j \left(\sum_{q=1}^{n_Q} \frac{Q_{iq} K_{jq}}{\sqrt{n_E}} \right) V_{jc} \quad (3.12)$$

$$\text{Activation}(x) = \text{GELU}(x) \quad (3.13)$$

Hier sind $x \in \mathbb{R}^{n_B \times n_E}$ und $\bar{x} \in \mathbb{R}^{n_B \times n_E}$ Matrizen, wobei die Batch-Größe n_B , die Embedding-Größe n_E und die Query-Größe n_Q Hyperparameter des Transformers sind. ϵ ist eine kleine Zahl, durch die LayerNorm für kleine Eingabewerte stabilisiert wird. Die Matrizen Q, K und V ergeben sich durch Matrixmultiplikation als

$$Q = \bar{x}W_Q \quad (3.14)$$

und analog für K und V . Die Matrizen W_Q, W_K und W_V sind dann die Parameter des Attention-Blocks, die während des Trainings angepasst werden.

Die Funktion $\text{softmax} : \mathbb{R}^{n_E} \rightarrow \mathbb{R}^{n_E}$ wird zeilenweise auf die Einträge der Eingabematrix angewendet und liefert Vektoren mit den Einträgen

$$(\text{softmax}(\vec{v}))_i = \frac{e^{v_i}}{\sum_{k=1}^{n_E} e^{v_k}}. \quad (3.15)$$

Die Schreibweise softmax_j soll dann die Auswertung von softmax auf die j -te Zeile der Matrix ausdrücken. Die Funktion $\text{GELU} : \mathbb{R}^{n_B \times n_E} \rightarrow \mathbb{R}^{n_B \times n_E}$ bildet die Komponenten der Eingabematrix unabhängig voneinander ab. Auch die Funktion $\text{ReLU} : \mathbb{R}^{n_B \times n_E} \rightarrow \mathbb{R}^{n_B \times n_E}$, welche im späteren Verlauf relevant wird, ist so zu verstehen. Der Verlauf von $\text{GELU}(x)$ und $\text{ReLU}(x)$ im eindimensionalen Fall ist in Abbildung 3.3 dargestellt.

Der Transformer ist dann definiert als (vergleiche [12]):

$$\text{AttentionBlock}(x) = x + \text{Linear} \circ \text{Attention}(\text{Linear}(\bar{x}), \text{Linear}(\bar{x}), \text{Linear}(\bar{x})) \quad (3.16)$$

$$\text{MLPBlock}(x) = x + \text{Linear} \circ \text{Activation} \circ \text{Linear}(\bar{x}) \quad (3.17)$$

$$\text{Block}(x) = \text{MLPBlock} \circ \text{AttentionBlock}(x) \quad (3.18)$$

$$\text{Transformer}(x) = \text{Linear} \circ \underbrace{\text{Block} \circ \dots \circ \text{Block}}_N \circ \text{Linear}(x). \quad (3.19)$$

Die Anzahl N der Transformerblöcke ist ein weiterer Hyperparameter. Je nach Anwendung kann der Transformer auch erweitert werden, beispielsweise durch eine Maskierung im Attention-

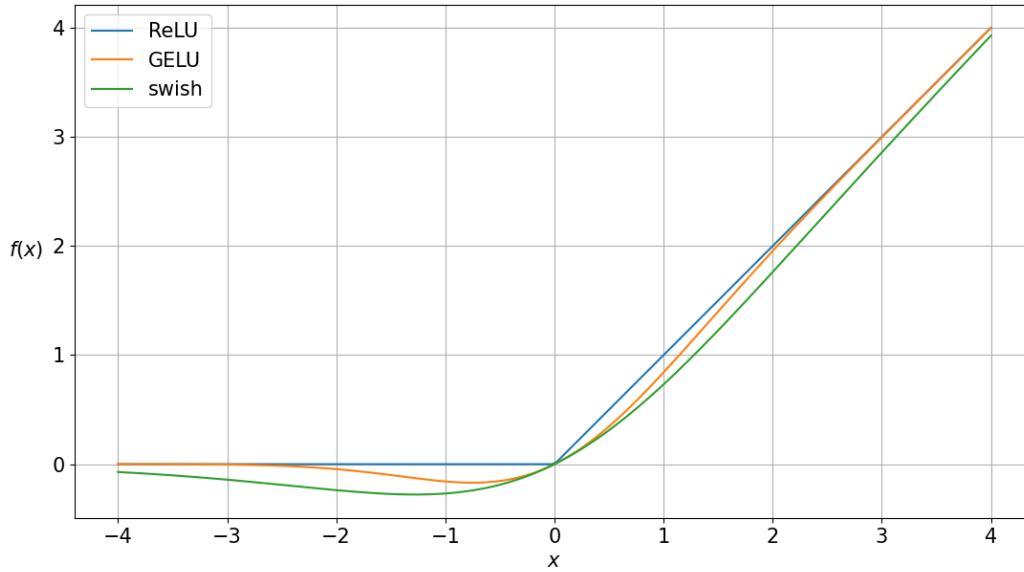


Abbildung 3.3: Verlauf der Aktivierungsfunktionen ReLU, GELU und swish im eindimensionalen Fall. Erstellt mit Matplotlib.

Mechanismus oder durch Multi-Headed-Attention [10]. Das Training eines Transformers erfolgt analog zum MLP über die Definition einer loss-Funktion und mit der Backpropagation in Gleichung (3.6). Für die genauere Betrachtung der einzelnen Komponenten des Transformers sei hier auf [9] verwiesen.

3.4 Lorentz-äquivarianter Transformer (L-GATr)

Der Transformer und auch der MLP funktionieren sehr gut in vielfältigen Anwendungsbereichen. Für die Modellierung physikalischer Gesetzmäßigkeiten, insbesondere in der Hochenergiephysik, fehlt ihnen allerdings eine möglicherweise entscheidende Logik. Die Eingabedaten für einen MLP und einen herkömmlichen Transformer sind reine Zahlen. Das bedeutet, dass jeder Wert unabhängig von allen anderen Eingabewerten ist. Dem Netzwerk werden jedoch als Eingabedaten unter anderem die Komponenten der Viererimpulse übergeben, welche immer die Gleichung

$$p_\mu p^\mu = E^2 - \vec{p}^2 = m^2 \quad (3.20)$$

(in Heaviside-Lorentz-Einheiten) erfüllen müssen. Weder der MLP noch der klassische Transformer kennen diese Beziehung und müssen sie daher aus den Trainingsdaten erlernen. L-GATr (kurz für Lorentz-Equivariant Geometric Algebra Transformer) ist eine Modifikation der herkömmlichen Transformerarchitektur, in der die Lorentz-Invarianz (3.20) bereits a priori eingebaut ist. Das wird erreicht, indem L-GATr nicht mit unabhängigen Zahlen arbeitet, wie der

herkömmliche Transformer, sondern stattdessen mit Multivektoren der Form [12]

$$u = u^S 1 + u_\mu^V \gamma^\mu + u_{\mu\nu}^B \sigma^{\mu\nu} + u_\mu^A \gamma^\mu \gamma^5 + u^P \gamma^5 \in \text{Cl}_{1,3}. \quad (3.21)$$

Multivektoren sind Objekte aus der Mathematik, besser gesagt aus der geometrischen Algebra, und werden hier nur kurz eingeführt. Die Multivektoren, die für die Teilchenphysik relevant sind, stammen aus der Clifford-Algebra $\text{Cl}_{1,3}$ mit Signatur $(1, 3)$ (siehe hierzu Gleichung (3.23)). Diese Multivektoren haben eine Skalkomponente u^S , 4 Vektorkomponenten u_μ^V , 6 Bivektorkomponenten $u_{\mu\nu}^B$, 4 Axialvektorkomponenten u_μ^A und eine Pseudoskalarkomponente u^P . Für zwei Multivektoren $u, w \in \text{Cl}_{1,3}$ definiert man das geometrische Produkt als

$$\text{GP}(u, w) := uw \in \text{Cl}_{1,3} \quad (3.22)$$

Hierbei befolgen die Vektoren γ^μ die Relationen in Gleichung (3.23) (vergleiche mit der Minkowski-Metrik $(+1, -1, -1, -1)$).

$$\gamma^\mu \gamma^\nu = \begin{cases} +1, & \text{falls } \mu = \nu = 0 \\ -1, & \text{falls } \mu = \nu \in \{1, 2, 3\} \\ \sigma^{\mu\nu} = -\gamma^\nu \gamma^\mu & \text{falls } \mu \neq \nu \end{cases} \quad (3.23)$$

Der Pseudoskalar γ^5 ist definiert als

$$\gamma^5 := \gamma^0 \gamma^1 \gamma^2 \gamma^3 \quad (3.24)$$

(vergleiche auch mit der Definition des Paritätsoperators der Dirac-Spinoren). Alle anderen Eigenschaften der geometrischen Algebra lassen sich aus diesen beiden Gleichungen und der Assoziativität des geometrischen Produkts herleiten. Als weitere Operation wird die blade-Projektion $\langle \cdot \rangle_k$ mit $k \in \{0, 1, 2, 3, 4\}$ definiert, die die k -te Komponente der Multivektordarstellung (3.21) isoliert. Damit ist beispielsweise

$$\langle u \rangle_1 = u_\mu^V \gamma^\mu.$$

Außerdem benötigt man das innere Produkt zweier Multivektoren,

$$u \cdot w, \quad (3.25)$$

das dem herkömmlichen Vektorskalarprodukt (der Minkowski-Metrik) entspricht, falls u und w nur Vektorkomponenten besitzen.

Die Grundidee dieser geometrischen Algebra $\text{Cl}_{1,3}$ ist, dass die gesamte Struktur der Raumzeit, inklusive der Lorentztransformation Λ , in einer einheitlichen Darstellung beschrieben wird. Die

Eingaben in das L-GATr Netzwerk sind dann Matrizen $x \in \text{Cl}_{1,3}^{n_t \times n_c}$ aus Multivektoren. Hier sind n_c die Anzahl der Multivektorkanäle des Transformers und die Anzahl n_t der Eingabetoken ergibt sich aus dem Produkt der Teilchenanzahl pro Ereignis und der Batch-Größe. Wenn nun L-GATr lorentz-invariant gebaut werden soll, muss

$$\text{L-GATr}(\Lambda(x)) = \Lambda(\text{L-GATr}(x)) \quad (3.26)$$

gelten. Dafür müssen die Transformer-Operationen in den Gleichungen (3.10) bis (3.19) modifiziert werden [12]:

$$\text{Linear}(x) = \sum_{k=0}^4 (v_k \langle x \rangle_k + w_k \gamma^5 \langle x \rangle_k) \quad (3.27)$$

$$\text{LayerNorm}(x) = \bar{x} = x \cdot \left(\frac{1}{n_c} \sum_{c=1}^{n_c} \sum_{k=0}^4 |\langle x_c \rangle_k \cdot \langle x_c \rangle_k| + \epsilon \right)^{-1/2} \quad (3.28)$$

$$(\text{Attention}(Q, K, V))_{ic} = \sum_{j=1}^{n_t} \text{softmax}_j \left(\sum_{c'=1}^{n_c} \frac{Q_{ic'} \cdot K_{jc'}}{\sqrt{16n_c}} \right) V_{jc} \quad (3.29)$$

$$\text{Activation}(x) = x \cdot \text{GELU}(\langle x \rangle_0) \quad (3.30)$$

$$\text{AttentionBlock}(x) = x + \text{Linear} \circ \text{Attention}(\text{Linear}(\bar{x}), \text{Linear}(\bar{x}), \text{Linear}(\bar{x})) \quad (3.31)$$

$$\text{MLPBlock}(x) = x + \text{Linear} \circ \text{Activation} \circ \text{Linear} \circ \text{GP}(\text{Linear}(\bar{x}), \text{Linear}(\bar{x})) \quad (3.32)$$

$$\text{Block}(x) = \text{MLPBlock} \circ \text{AttentionBlock}(x) \quad (3.33)$$

$$\text{L-GATr}(x) = \text{Linear} \circ \underbrace{\text{Block} \circ \text{Block} \circ \dots \circ \text{Block}}_N \circ \text{Linear}(x) \quad (3.34)$$

Die Struktur von L-GATr ist dann analog zum klassischen Transformer.

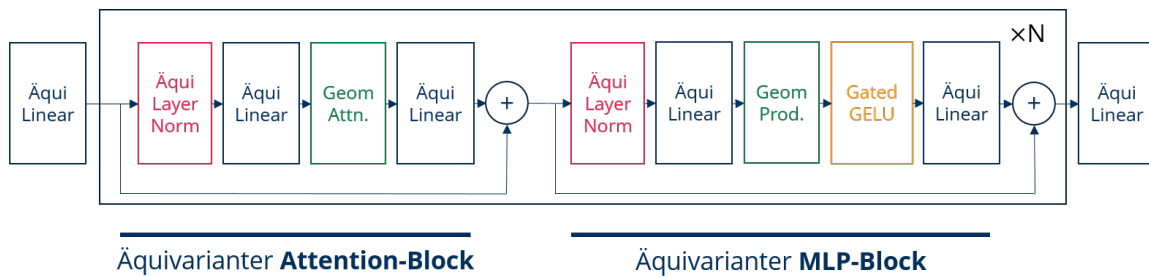


Abbildung 3.4: Schematischer Aufbau von L-GATr. Analog zu Abbildung 3.2 entspricht jedes Rechteck einer Operation von (3.27) bis (3.30). Bemerkenswert ist, dass der strukturelle Aufbau nahezu identisch zum klassischen Transformer ist. Diese Abbildung ist eine Nachbildung aus [11].

Die erlernbaren Parameter von L-GATr sind die Vektoren $v, w \in \mathbb{R}^5$ der linearen Operationen sowie die Matrizen W_Q, W_K und W_V des Attention-Mechanismus. Im Vergleich zum Abschnitt 3.3 fällt auf, dass viele Operationen strukturell sehr ähnlich aussehen. Das ist einer der großen

Vorteile von L-GATr. Insbesondere der Attention-Mechanismus in Gleichung (3.29) kann für große n_c rechenaufwendig werden, da hier zwei Matrizen der Größe $n_t \cdot n_c$ multipliziert werden. In L-GATr wird diese Operation in zwei Schritte unterteilt. Man berechnet eine Liste von Vorzeichen und die klassischen euklidischen Skalarprodukte. Die Vorzeichen müssen nur einmal berechnet werden und können danach wiederverwendet werden. Für das euklidische Skalarprodukt können Optimierungen wie FlashAttention [13] genutzt werden, welche ursprünglich für den klassischen Transformer entwickelt wurden.

Im MLPBlock wird das geometrische Produkt als zusätzlicher Rechenschritt eingeführt. Es dient hier als zusätzliche nichtlineare Operation, welche nun durch die Verwendung von Multivektoren zur Verfügung steht.

Durch diese neuen Definitionen ist L-GATr eine lorentz-invariante Netzwerkstruktur. Dies kann allerdings auch hinderlich sein, wenn L-GATr eine Situation modellieren soll, in der die Lorentz-Invarianz teilweise verletzt ist. Das ist beispielsweise bei einer Teilchenkollision der Fall, da hier die Strahlachse des Detektors den Raum auszeichnet. Dieses Problem kann allerdings behoben werden, indem man dem Netzwerk im Training und in der Anwendung zusätzliche Multivektoren als Eingaben übergibt, die die Lorentz-Invarianz brechen. In [12] werden hierzu die Multivektoren σ^{12} und γ^0 vorgeschlagen, welche die xy -Ebene und die Richtung der Zeit beschreiben.

3.5 Training mit OPTIMA

Die Datensätze für alle Surrogate wurden mit *SHERPA* [14] generiert. Insgesamt handelt es sich hierbei um circa 1,7 Millionen Ereignisse, welche zu jeweils etwa 50 % in einen Trainingsdatensatz und einen Testdatensatz aufgeteilt wurden. Der Trainingsdatensatz wird wiederum zu 80 % beziehungsweise 20 % in Lerndaten und Validierungsdaten aufgeteilt. Die genaue Aufteilung ist in Tabelle 3.1 angegeben. Das Training der verschiedenen Surrogate wird mit OPTIMA [15] durchgeführt. Hier können die verschiedenen Netzwerke erstellt und nach Belieben modifiziert werden. Eine Konfigurationsdatei beschreibt die Struktur des Surrogats sowie alle relevanten Hyperparameter und Metriken. Eine Metrik dient neben der loss-Funktion dazu, die Leistung des Surrogats während des Trainings auszuwerten.

Datensatz	Ereignisanzahl
Trainingsdaten	791873
- davon Lerndaten	633498
- davon Validierungsdaten	158375
Testdaten	873770
Gesamt	1665643

Tabelle 3.1: Aufteilung aller mit *SHERPA* generierten Ereignisse auf die drei Datensätze, die für das Training beziehungsweise die Auswertung verwendet werden.

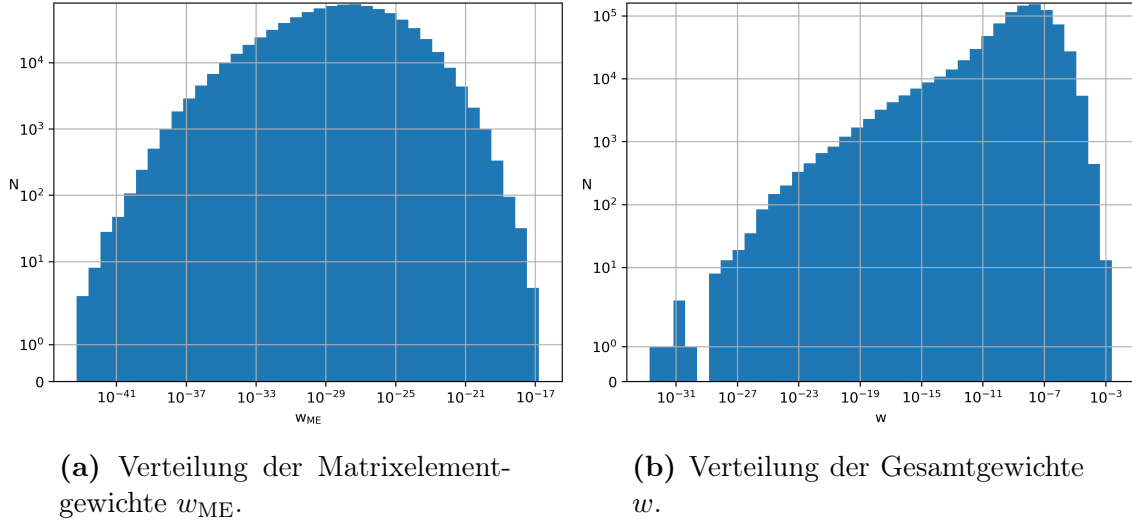


Abbildung 3.5: Doppeltlogarithmische Histogramme der Ereignisse für den Prozess aus Abbildung 2.1. Dargestellt sind die Anzahl der Ereignisse in den verschiedenen Größenordnungen von w_{ME} beziehungsweise w . Diese mit *SHERPA* generierten Daten bilden den Testdatensatz für alle hier untersuchten Modelle in Kapitel 4.

Das Training selbst geschieht folgendermaßen: Das Netzwerk wird auf den Lerndaten ausgewertet und über die Backpropagation in Gleichung (3.6) werden die internen Parameter angepasst. Danach wird das Netzwerk auf den Validierungsdaten ausgewertet. Wenn es dabei einen geringeren Metrikwert erreicht als je zuvor, dann hat sich das Netzwerk durch das Training verbessert. Danach wird die Reihenfolge der Daten in jedem Datensatz zufällig geändert und die nächste Epoche beginnt. Verbessert sich das Surrogat innerhalb einer gewissen Anzahl von Epochen nicht, dann tritt early-stopping in Kraft, wodurch das Training beendet wird. Early-stopping soll hauptsächlich die Effekte von Overtraining und Undertraining reduzieren. Diese sind schematisch in Abbildung 3.6 dargestellt.

Dort erkennt man, dass die violette Regressionskurve nicht das exakte Verhalten erfassen kann. Das konstante Polynom ist in gewisser Weise „zu einfach“, um den linearen Anstieg abbilden zu können. Dies entspricht der Situation des Undertraining, bei dem das Netzwerk nicht genügend lange trainiert hat. Andererseits kann das kubische Polynom in diesem Fall die 4 Datenpunkte perfekt abbilden. Für kleinere und größere x - Werte weicht es dafür trotzdem sehr stark vom exakten Verhalten ab. In diesem Fall hat das Netzwerk deutlich zu lange trainiert und letztlich nur die Trainingsdaten mit ihrem statistischen Rauschen auswendig gelernt. Early-stopping soll nun dafür sorgen, dass das Training weder zu früh noch zu spät abgebrochen wird.

Nachdem ein Surrogat fertig trainiert ist, wird es auf dem Testdatensatz ausgewertet, um die finale Leistung des Netzes zu überprüfen. Daher wurden in den kommenden Abschnitten alle Diagramme mit dem Testdatensatz erstellt.

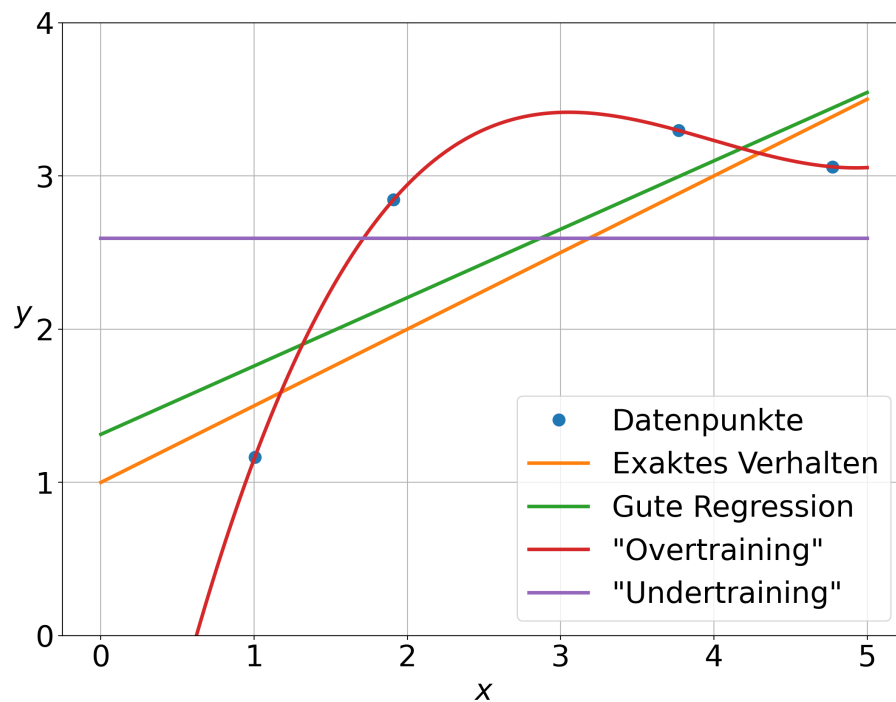


Abbildung 3.6: Konzeptionelle Verdeutlichung von Overtraining und Undertraining. Das exakte Verhalten (orange) ist die lineare Funktion $f(x) = 0,5x + 1$. Die Datenpunkte (blau) liegen gaußverteilt um das exakte Verhalten herum. Die Regressionskurven (violett, grün und rot) sind ein konstantes, ein lineares und ein kubisches Polynom mit frei wählbaren Koeffizienten. Erstellt mit Matplotlib.

4 Lorentz-äquivariante Transformer als Matrixelement-Surrogate

4.1 MLP mit Dipolansatz als Referenz

Um bewerten zu können, ob L-GATr in der praktischen Anwendung lohnenswert ist, benötigt man als Referenz ein möglichst gutes Surrogat, welches auf bisherigen Methoden basiert. Als Orientierung dient hierzu das Bachelorprojekt „Optimierung von neuronalen Netzen für Matrixelement-Surrogate in Prozessgruppen“ von Mathis Schenker [6]. Sein Modell ist in der Programmiersprache Python implementiert, genauer gesagt mit dem machine-learning-Framework Keras und TensorFlow. Die Implementierung von L-GATr wird als Open-Source-Code [16] auf GitHub zur Verfügung gestellt. Diese verwendet jedoch das Framework PyTorch, welches nicht mit Keras kompatibel ist. Um beide Modelle besser vergleichen zu können, musste das Modell aus [6] originalgetreu in PyTorch nachgestellt werden. Hierfür wurden die Hyperparameter in Tabelle 4.1 verwendet. Diese weichen teilweise von den gefundenen Werten in [6] ab, da auch der betrachtete Prozess ein anderer ist. Insbesondere werden fünf versteckte Schichten verwendet, da das in Abbildung 5.2 in [6] im Mittel beste Netzwerk mit nur einer versteckten Schicht für diesen komplizierteren Prozess nicht sinnvoll erscheint.

Hyperparameter	Wert
Anzahl versteckte Schichten	5
Neuronen pro Schicht	128
Lernrate	0,001
Batch-Größe	1024
Aktivierungsfunktion	swish
$ADAM \beta_1$	0,9
$ADAM \beta_2$	0,999
$ADAM \varepsilon$	10^{-8}
Weight Initialisierung	He uniform
Early-stopping-Geduld	40 Epochen

Tabelle 4.1: Die relevanten Hyperparameter für das Training des MLP als Referenz.

Weitere wichtige Größen sind die Werte von S_{pred} und S_{coef} , sowie die Zeiten t_{ME} , t_{PS} und t_{surr} in Tabelle 4.2. Da diese weitestgehend aus dem Trainingsdatensatz hervorgehen, nehmen sie

auch in den folgenden Abschnitten dieselben Werte an. Außerdem ist die Zeit t_{surr} relevant, welche analog zu [6] mit $t_{\text{surr}} = 0,3$ ms abgeschätzt wird.

Konstante Parameter	Wert (gerundet)
S_{pred}	$3,24269 \cdot 10^{-44}$
S_{coef}	$8,41526 \cdot 10^{-40}$
t_{ME}	1233,95 ms
t_{PS}	8,113 ms

Tabelle 4.2: Konstante Werte, die für alle Trainings verwendet wurden.

Die Eingaben für das Netzwerk bestehen dann aus den Komponenten der Viererimpulse p_i aller Anfangs- und Endzustandsteilchen, den Phasenraumabbildungen y_{ijk} der Dipole und den invarianten Massen s_{ij} [7]. Die Indizes i, j und k zählen jeweils über die Teilchen des Anfangs- und Endzustands. Jede dieser drei Gruppen wird vorher auf den Mittelwert $\langle \cdot \rangle = 0$ und auf die Standardabweichung $\sigma = 1$ normiert mittels

$$p_i^{\text{norm}} = \frac{p_i - \langle p \rangle}{\sigma_p} \quad (4.1)$$

und analog für y_{ijk} und s_{ij} . Die normierten Gruppen werden aneinandergereiht und an das Netzwerk übergeben. Die Ausgaben sind dann die Koeffizienten c_{ijk} aus Gleichung (3.8). Anders als in [6] wird ausschließlich die loss-Funktion in Gleichung (3.5) verwendet. Ein Wechsel zum inversen gain-Faktor als loss findet nicht statt. In [6] mussten außerdem einige Mittelwerte innerhalb eines Batch für die Ermittlung des gain-Faktors berechnet werden. Das macht ihn anfällig gegenüber Ausreißern, also einzelnen Ereignissen, deren Gewicht deutlich über dem der restlichen Ereignisse liegt. Da jedoch hier der gain-Faktor nur als Metrik über eine gesamte Epoche verwendet wird, ist auch die Verwendung einer kleineren Batch-Größe von 1024 plausibel anstelle des besten Werts von 2048 in [6].

Die Vorhersagen a des Surrogats müssen dann noch mit den Phasenraumgewichten w_{PS} multipliziert werden, um das volle Ereignisgewicht zu erhalten. Für die Berechnung der loss-Funktion definiert man die Terme y und $\hat{y}$.

$$y := \text{arsinh} \left(\frac{w}{S_{\text{pred}}} \right) \quad (4.2)$$

$$\hat{y} := \text{arsinh} \left(\frac{a}{S_{\text{pred}}} \right). \quad (4.3)$$

Da y nur Werte aus dem Trainingsdatensatz enthält, können diese bereits vor dem Start des Trainings berechnet werden. Um die numerischen Werte der loss-Funktion mit [6] vergleichen zu können, werden y und $\hat{y}$ noch normiert mit dem Mittelwert $\langle y \rangle$ und der Standardabweichung

σ_y der y -Werte.

$$y^{\text{norm}} = \frac{y - \langle y \rangle}{\sigma_y} \quad (4.4)$$

$$\hat{y}^{\text{norm}} = \frac{\hat{y} - \langle y \rangle}{\sigma_y} \quad (4.5)$$

Die loss-Funktion reduziert sich damit auf einen einfachen MSE-Loss („mean squared error“) über die n_B Daten eines Batch.

$$l = \frac{1}{n_B} \sum_{i=1}^{n_B} (\hat{y}^{\text{norm}} - y^{\text{norm}})^2 \quad (4.6)$$

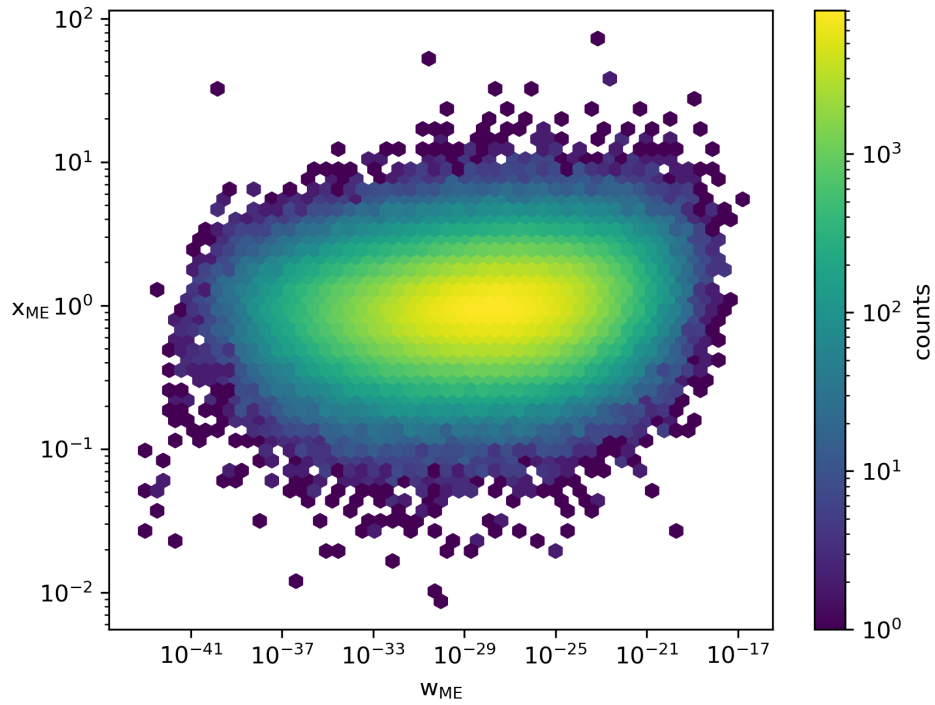
Idealerweise entsteht dabei die Verteilung in Abbildung 3.5b. Für die tatsächliche Verteilung ist insbesondere die grafische Darstellung der Verhältnisse $x = w/s$ in Abhängigkeit von w interessant. Wie bereits in Abschnitt 2.5 angemerkt, liefert ein perfektes Surrogat immer $x = 1$, also $s = w$. Ein gutes Surrogat sollte daher eine Verteilung $x(w)$ produzieren, die um 1 zentriert ist. Wie man in Abbildung 4.1b erkennt, ist dies auch der Fall für den MLP mit Dipolansatz. Die dargestellte Verteilung hat einen Mittelwert um 1 und eine Verbreiterung von etwa einer Größenordnung. Es gibt einzelne Ausreißer, die etwa um 2 Größenordnungen vom Mittelwert abweichen, doch für den Großteil der Ereignisse (dargestellt in gelb) wird die richtige Größenordnung vorhergesagt. Da die Verteilung der Lerndaten sehr ähnlich ist zum Histogramm in Abbildung 3.5b, war dieses Verhalten auch zu erwarten. Im Bereich $w \in [10^{-11}, 10^{-5}]$ liegen schließlich die meisten Trainingsdaten, sodass das Netzwerk deren Verhalten auch am häufigsten trainiert. Selbstverständlich erreicht die Verteilung von x nur diese Form, da auch $x_{\text{ME}} := w_{\text{ME}}/a$ in Abbildung 4.1a dieselben qualitativen Eigenschaften aufweist. Die Symmetrie bezüglich $x = 1$ beider Verteilungen lässt sich dadurch erklären, dass durch die Verwendung des arsinh in der loss-Funktion zunächst nur das Erlernen der richtigen Größenordnung von s beziehungsweise a relevant ist.

Mit den oben genannten Hyperparametern erreicht der MLP einen effektiven gain-Faktor von

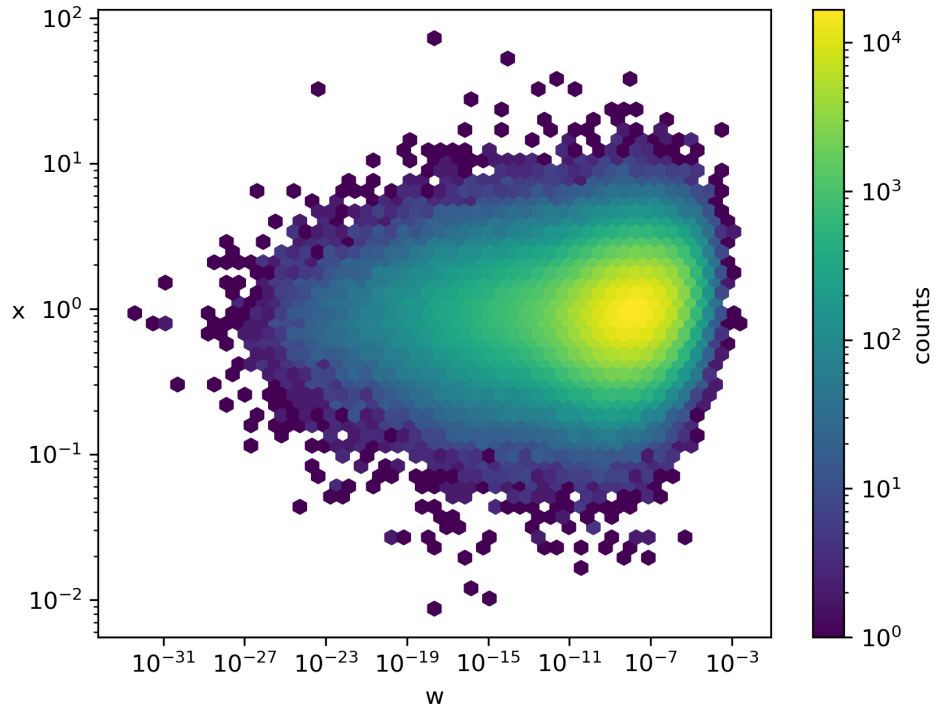
$$f_{\text{eff}} = 12,78997. \quad (4.7)$$

Dies entspricht einer 12,8-fachen Zeiteinsparung mit dem Algorithmus 2 gegenüber der klassischen Ungewichtung mit dem Algorithmus 1.

Im Folgenden wird nun untersucht, ob der lorentz-invariante Transformer auch diesen gain-Faktor erreichen oder sogar übertreffen kann.



(a) Verteilung von $x_{\text{ME}} = w_{\text{ME}}/a$ in Abhängigkeit von w_{ME} .



(b) Verteilung von $x = w/s$ in Abhängigkeit von w .

Abbildung 4.1: Doppeltlogarithmische hexagonale Histogramme der Verteilungen von x_{ME} und x für die Referenzanalyse des MLP mit Dipolansatz mit den Hyperparametern in Tabelle 4.1.

4.2 L-GATr ohne kinematische Invarianten

In der Implementierung von L-GATr in [16] können zusätzlich zu den Multivektoren auch klassische Skalare an das Netzwerk übergeben werden. Diese besitzen keine geometrische Interpretation und werden analog zum klassischen Transformer verarbeitet. Es bietet sich daher an, dem L-GATr-Netzwerk ebenfalls dieselben Eingaben zu übergeben wie dem MLP in der Referenzanalyse, wobei jedoch die Viererimpulse in Multivektoren umgewandelt werden. Da jedoch aus physikalischer Sicht die Viererimpulse bereits sämtliche Informationen über den Streuprozess enthalten, ist es zunächst interessant, herauszufinden, ob L-GATr auch ohne die zusätzlichen Skalare das Matrixelementgewicht gut vorhersagen kann.

Die allgemeine Vorgehensweise ist hierbei identisch zu Abschnitt 4.1. Da die Viererimpulse jetzt eine geometrische Interpretation haben, ist eine Normierung auf den Mittelwert $\langle p \rangle = 0$ nach Gleichung (4.1) nicht physikalisch. Stattdessen findet lediglich eine Umskalierung mit ihrer Standardabweichung σ_p statt.

$$p_i^{\text{norm}} = \frac{p_i}{\sigma_p} \quad (4.8)$$

Diese normierten Viererimpulse werden an ein L-GATr-Netzwerk mit folgenden Hyperparametern übergeben:

Hyperparameter	Wert
Anzahl Blöcke	8
Interne Skalarkanäle	8
Interne Multivektorkanäle	8
Anzahl Attentionheads	8
MLP Aktivierung	GELU
Symmetriebrechung	σ^{12} und γ^0
Lernrate	0,001
Batch-Größe	1024
ADAM β_1	0,9
ADAM β_2	0,999
ADAM ε	10^{-8}
Weight Initialisierung	He uniform
Early-stopping-Geduld	40 Epochen

Tabelle 4.3: Die relevanten Hyperparameter für das Training des L-GATr ohne zusätzliche Skalare für die direkte Vorhersage des Matrixelementgewichts.

Die direkte Voraussage von w_{ME} ist nicht möglich, da die Matrixelementgewichte über viele Größenordnungen schwanken, wie in Abbildung 3.5a zu erkennen ist. Stattdessen soll das Netzwerk den Term $\hat{y} = \text{arsinh}(a/S_{\text{pred}})$ aus der loss-Funktion (3.5) vorhersagen. Da man hier jedoch die Vorhersage a für das Matrixelementgewicht aus einer einfachen Umrechnung erhält, wird dieses Verfahren trotzdem als „direkte Vorhersage des Matrixelementgewichts“ bezeichnet. Für die Berechnung der loss-Funktion werden die Ausgaben des Netzwerks wieder analog zum MLP mit den Gleichungen (4.4) und (4.5) normiert.

An dieser Stelle muss man jedoch auch auf die Numerik von Computern achten. Im Training und in der Anwendung gibt es zwei grundlegende Datentypen, nämlich 32-Bit- und 64-Bit-Gleitkommazahlen. Alle Rechenschritte mit 32-Bit sind deutlich schneller, aber auch ungenauer als mit 64-Bit. Der Wert von S_{pred} ist etwa $3,24269 \cdot 10^{-44}$, was deutlich kleiner ist als die betraglich kleinste normalisiert darstellbare 32-Bit-Gleitkommazahl $2^{-126} \approx 1,17549 \cdot 10^{-38}$. Eine Division durch S_{pred} sorgt somit dafür, dass der Quotient numerisch unendlich werden würde, wodurch eine sinnvolle Berechnung der loss-Funktion nicht mehr möglich wäre. Aus diesem Grund müssen alle Bearbeitungsschritte, die S_{pred} und später auch S_{coef} verwenden, mit 64-Bit-Genauigkeit durchgeführt werden. Dies war für den MLP kein Problem, da 64-Bit der Standardzahlentyp in Python ist. Für L-GATr muss man jetzt aber darauf achten, da die Implementierung von FlashAttention und die verwendete Maskierung im Attention-Mechanismus für 32-Bit optimiert sind. Die Reihenfolge der Datenverarbeitung ist daher wie folgt:

1. Vorverarbeitung der Eingabedaten in 64-Bit
2. Umwandlung der Daten zu 32-Bit
3. Eingabe in L-GATr-Netzwerk
4. Umwandlung der Ausgaben zu 64-Bit
5. Nachverarbeitung in 64-Bit

Außerdem muss die Zeit t_{surr} angepasst werden, da L-GATr durch seine komplexere Struktur auch eine etwas längere Rechenzeit benötigt. Für die tatsächliche Anwendung der Netzwerke in einer produktionsreifen Software werden die trainierten Modelle kompiliert, um ihre Ausführungsgeschwindigkeit bestmöglich zu optimieren. Der Wert $t_{\text{surr}} = 0,3 \text{ ms}$ ist eine Abschätzung für die Laufzeit eines kompilierten MLP. Um die Laufzeit eines kompilierten L-GATr abzuschätzen, wurden der unkompilierte MLP und der unkompilierte L-GATr auf dem gesamten Testdatensatz mit einer Batch-Größe von 1024 ausgewertet.

Modell	Laufzeit
MLP	23,27456 s
L-GATr	55,10312 s

Tabelle 4.4: Laufzeiten der unkompilierten Netzwerke für die Auswertung aller Testdaten. Der MLP nutzt die Hyperparameter aus Tabelle 4.1 und L-GATr die Hyperparameter aus Tabelle 4.3.

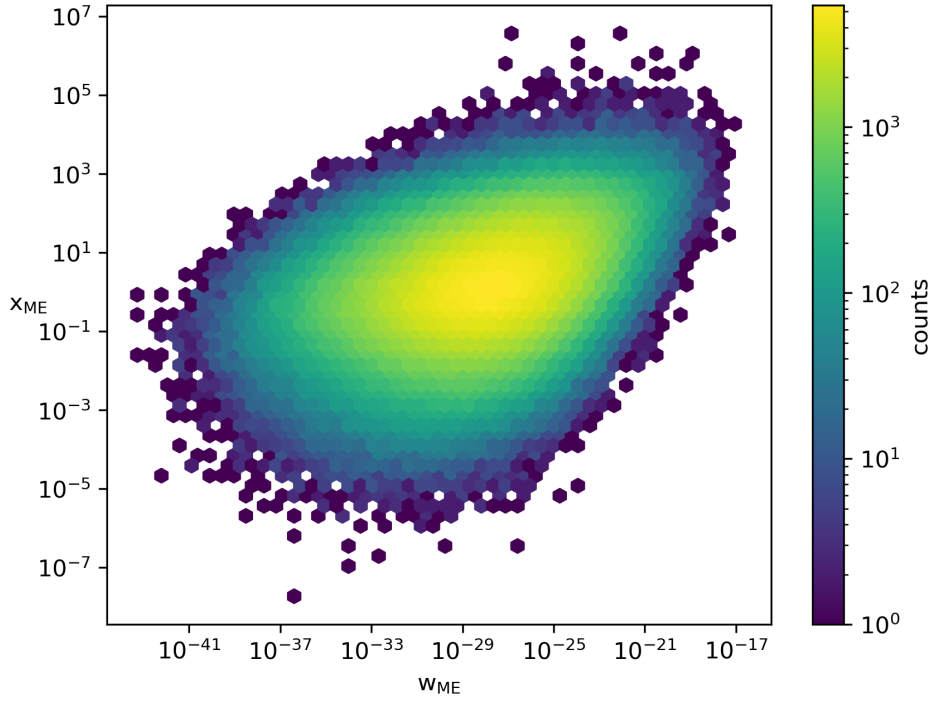
Aus diesen gemessenen Zeiten entsteht über eine einfache Verhältnisgleichung eine Abschätzung für die Laufzeit eines kompilierten L-GATr.

$$t_{\text{surr}} = 0,3 \text{ ms} \cdot \frac{t_{\text{L-GATr}}}{t_{\text{MLP}}} \approx 0,71 \text{ ms} \quad (4.9)$$

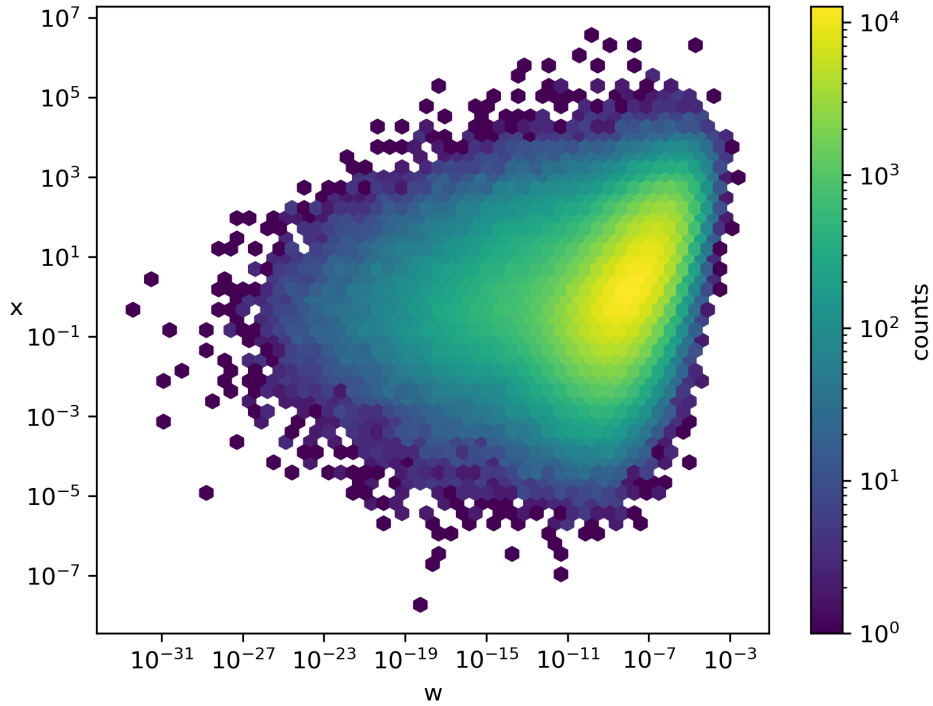
Mit diesen Angaben wird ein Training mit dem L-GATr-Netzwerk durchgeführt. Der erreichte gain-Faktor beträgt hier allerdings nur

$$f_{\text{eff}} = 5,08430 \cdot 10^{-5}. \quad (4.10)$$

Auch die Verteilungen von x_{ME} und x zeigen, dass dieser Ansatz des L-GATr-Netzwerks nicht ausreichend ist. Man erkennt zwar, dass die Verteilungen einigermaßen um 1 zentriert sind, doch die starke Verbreiterung um etwa 5 Größenordnungen zeigt, dass dieses Netzwerk das Verhalten der Matricelementgewichte nicht gut beschreiben kann. In Abbildung 4.2a sieht man, dass L-GATr kleine w_{ME} eher überschätzt und große w_{ME} tendenziell unterschätzt. Das erkennt man an der diagonalen Verzerrung der Verteilung.



(a) Verteilung von $x_{\text{ME}} = w_{\text{ME}}/a$ in Abhängigkeit von w_{ME} .



(b) Verteilung von $x = w/s$ in Abhängigkeit von w .

Abbildung 4.2: Doppeltlogarithmische hexagonale Histogramme der Verteilungen von x_{ME} und x für den L-GATr ohne kinematische Invarianten für die direkte Vorhersage des Matrixelementgewichts mit den Hyperparametern in Tabelle 4.3.

4.3 L-GATr mit kinematischen Invarianten

Im vorherigen Ansatz hat man gesehen, dass die Informationen der Viererimpulse nicht ausreichend sind, damit L-GATr brauchbare Vorhersagen treffen kann. Wie bereits angemerkt, kann man aber auch zusätzliche Skalare als Eingaben an das Netzwerk übergeben. In diesem Versuch werden jetzt also neben den Viererimpulsen auch die Phasenraumabbildungen y_{ijk} der Dipole und die invarianten Massen s_{ij} als Eingaben verwendet. Somit erhält L-GATr dieselben Informationen wie der MLP in der Referenzanalyse in Abschnitt 4.1. Die Ausgabe des Netzwerks ist aber weiterhin der Term $\hat{y} = \operatorname{arsinh}(a/S_{\text{pred}})$ aus dem vorherigen Abschnitt. L-GATr soll also wieder das Matrixelementgewicht direkt vorhersagen. Die restlichen Hyperparameter sind ebenfalls identisch zum vorherigen Abschnitt. Die Umwandlungsschritte der Gleitkommazahlen aus Abschnitt 4.2 müssen natürlich auch beachtet werden.

Hyperparameter	Wert
Anzahl Blöcke	8
Interne Skalarkanäle	8
Interne Multivektorkanäle	8
Anzahl Attentionheads	8
MLP Aktivierung	GELU
Symmetriebrechung	σ^{12} und γ^0
Lernrate	0,001
Batch-Größe	1024
<i>ADAM</i> β_1	0,9
<i>ADAM</i> β_2	0,999
<i>ADAM</i> ε	10^{-8}
Weight Initialisierung	He uniform
Early-stopping-Geduld	40 Epochen

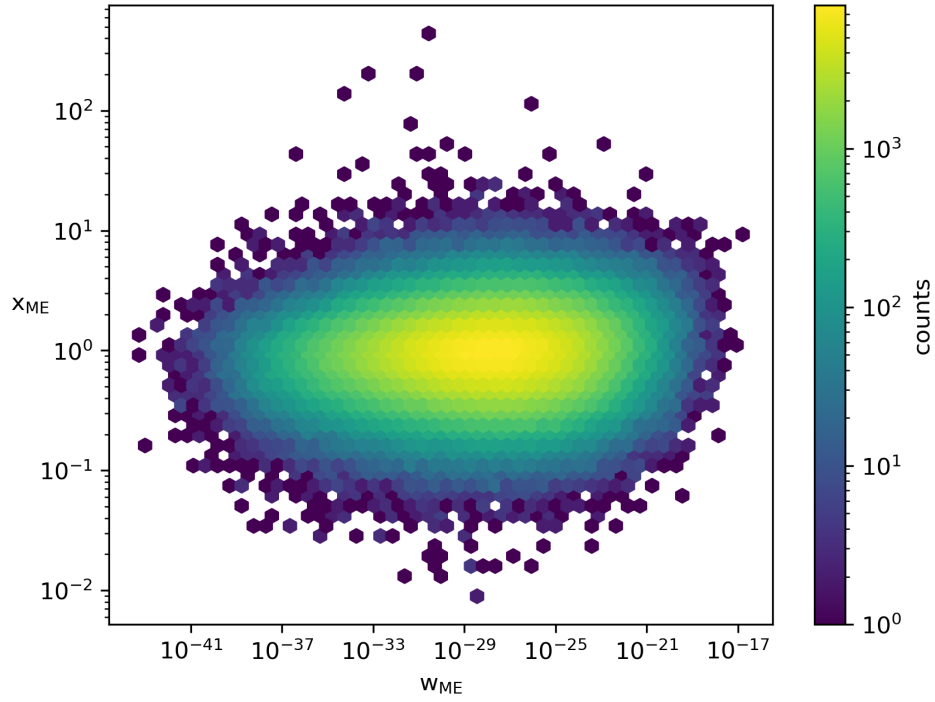
Tabelle 4.5: Die relevanten Hyperparameter für das Training des L-GATr mit der Eingabe der zusätzlichen Skalare für die direkte Vorhersage des Matrixelementgewichts.

Nun erreicht L-GATr einen effektiven gain-Faktor von

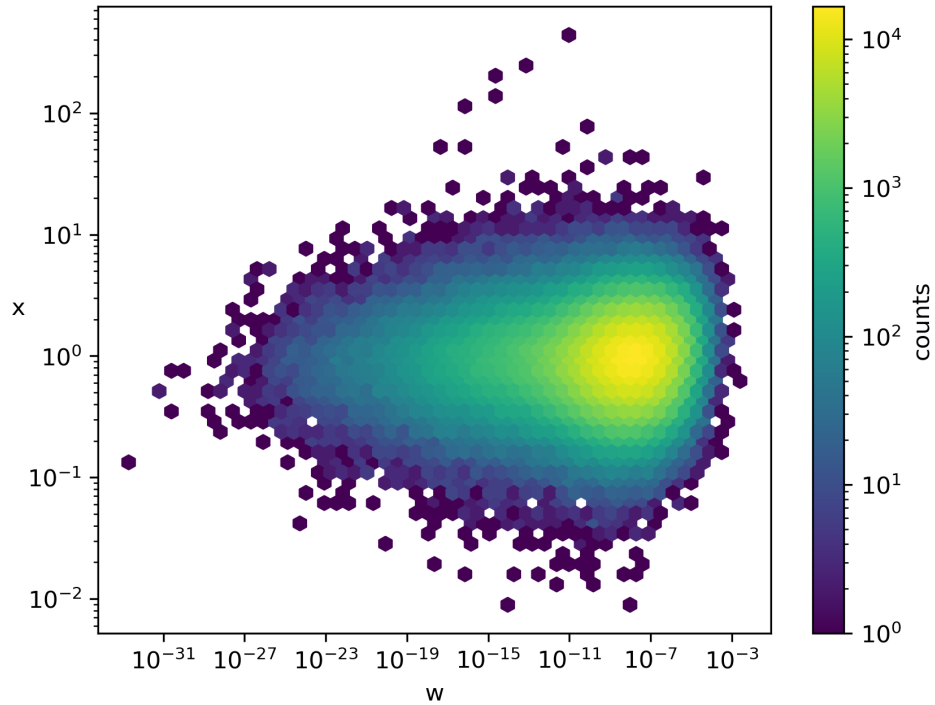
$$f_{\text{eff}} = 8,61657. \quad (4.11)$$

Dies entspricht noch nicht ganz dem Zielwert von etwa $f_{\text{eff}} = 12,8$, den der MLP mit Dipolansatz erreichen konnte. Es zeigt aber, dass L-GATr bereits ohne die Dipolentwicklung vergleichbare gain-Faktoren erzielen kann. Vergleicht man die Diagramme in Abbildung 4.3 mit den Abbildungen der vorherigen Abschnitte, dann fällt auf, dass sie den Verteilungen des MLP in Abbildung 4.1 sehr ähnlich sehen und sich deutlich gegenüber dem L-GATr ohne kinematische Invarianten in Abbildung 4.2 verbessert haben. Zwar gibt es wieder einzelne Ausreißer, bei denen sich das Surrogat um etwa 2 Größenordnungen verschätzt hat, doch die grobe Form der Verteilung ähnelt der des MLP jetzt besser. Insbesondere wurden die Zentrierung

um $x = 1$ beziehungsweise $x_{\text{ME}} = 1$ und die Verbreiterung von etwa einer Größenordnung wiederhergestellt.



(a) Verteilung von $x_{\text{ME}} = w_{\text{ME}}/a$ in Abhängigkeit von w_{ME} .



(b) Verteilung von $x = w/s$ in Abhängigkeit von w .

Abbildung 4.3: Doppeltlogarithmische hexagonale Histogramme der Verteilungen von x_{ME} und x für den L-GATr mit kinematischen Invarianten für die direkte Vorhersage des Matrixelementgewichts mit den Hyperparametern in Tabelle 4.5.

4.4 L-GATr mit Dipolansatz

Da L-GATr bereits ohne Dipolentwicklung einen beeindruckenden gain-Faktor vorweisen konnte, wird nun noch untersucht, ob die Vorhersage der Dipolkoeffizienten analog zum MLP in Abschnitt 4.1 eine weitere Verbesserung erzielt. Die Eingaben sind wie im Abschnitt 4.3 die Viererimpulse als Multivektoren, die Phasenraumabbildungen y_{ijk} und die invarianten Massen s_{ij} . Die Ausgaben sind jetzt allerdings die Koeffizienten c_{ijk} aus Gleichung (3.8). Die Hyperparameter und die Umwandlungsschritte der Gleitkommazahlen werden wieder aus den vorherigen Abschnitten übernommen.

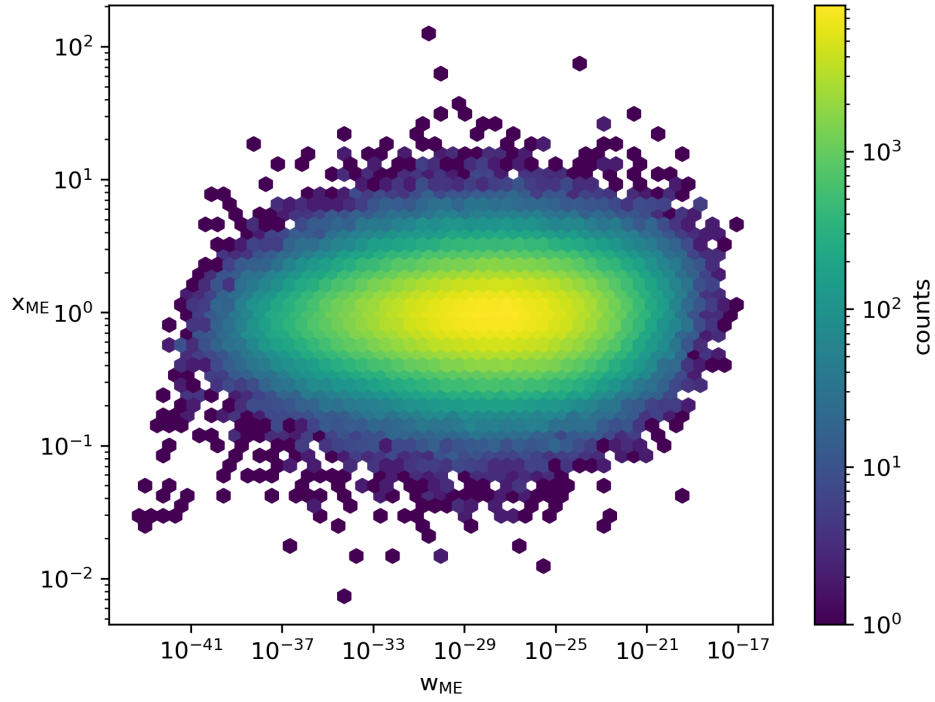
Hyperparameter	Wert
Anzahl Blöcke	8
Interne Skalarkanäle	8
Interne Multivektorkanäle	8
Anzahl Attentionheads	8
MLP Aktivierung	GELU
Symmetriebrechung	σ^{12} und γ^0
Lernrate	0,001
Batch-Größe	1024
<i>ADAM</i> β_1	0,9
<i>ADAM</i> β_2	0,999
<i>ADAM</i> ε	10^{-8}
Weight Initialisierung	He uniform
Early-stopping-Geduld	40 Epochen

Tabelle 4.6: Die relevanten Hyperparameter für das Training des L-GATr mit der Eingabe der zusätzlichen Skalare für die Vorhersage der Dipolkoeffizienten.

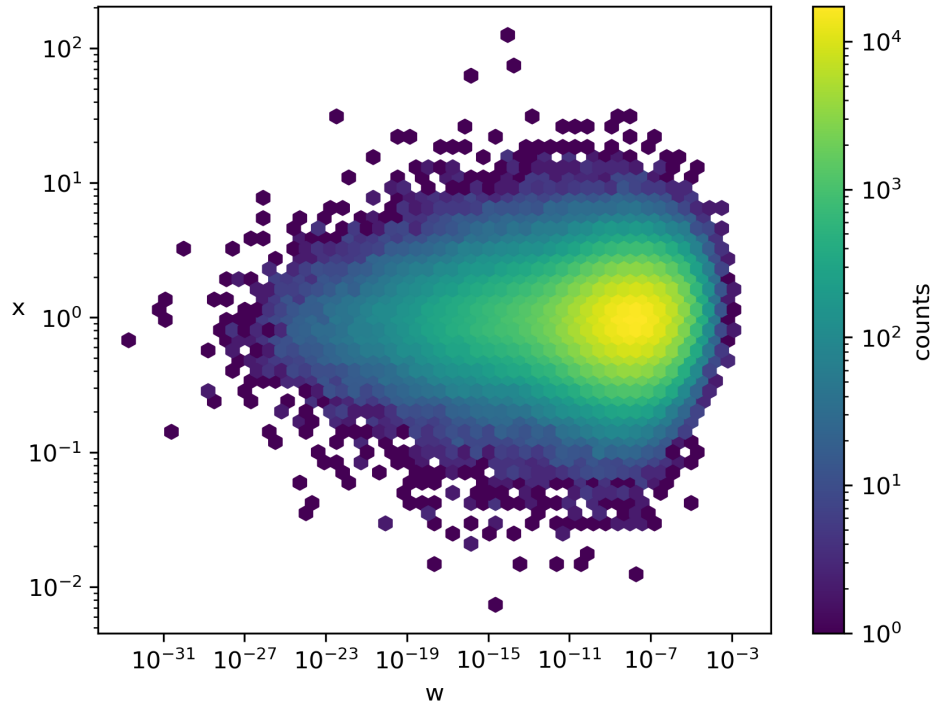
Mit diesem Ansatz erreicht man einen effektiven gain-Faktor von

$$f_{\text{eff}} = 17,49540. \quad (4.12)$$

Auch die Verteilungen von x_{ME} und x weisen qualitativ dieselben Eigenschaften auf, wie in den Abschnitten 4.1 und 4.3. Man erkennt zwar visuell gegenüber den Diagrammen in Abbildung 4.1 kaum eine Verbesserung, doch der höhere gain-Faktor spricht für sich. Mit dieser Erweiterung des L-GATr auf die Vorhersage der Dipolkoeffizienten gelingt dem Modell aus dem vorläufigen Training also ein gain-Faktor, der höher ist als der des MLP aus der Referenzanalyse.



(a) Verteilung von $x_{\text{ME}} = w_{\text{ME}}/a$ in Abhängigkeit von w_{ME} .



(b) Verteilung von $x = w/s$ in Abhängigkeit von w .

Abbildung 4.4: Doppeltlogarithmische hexagonale Histogramme der Verteilungen von x_{ME} und x für den L-GATr mit kinematischen Invarianten für die Vorhersage der Dipolkoeffizienten mit den Hyperparametern in Tabelle 4.6.

4.5 Hyperparameteroptimierung

Abschließend sollen nun die Modelle aus den vorläufigen Trainings optimiert werden. Die Hyperparameter für die Modelle in den Abschnitten 4.2 bis 4.4 wurden im Wesentlichen „nach Gefühl“ gewählt. Möglicherweise ist diese Kombination der Hyperparameter jedoch nicht optimal. Insbesondere können beispielsweise die Effekte des Undertraining und Overtraining die Leistung des Netzwerks verschlechtern. Es wird daher untersucht, welchen Einfluss die Batch-Größe und die Anzahl der Transformerblöcke auf den erreichbaren gain-Faktor haben.

Die Hyperparameteroptimierung (HPO) liefert zwar gute Kombinationen für die Hyperparameter eines Netzwerks, doch sie beeinflusst nicht die Funktionsfähigkeit eines Modells. Da also die direkte Vorhersage von w_{ME} ohne kinematische Invarianten in Abschnitt 4.2 nur einen sehr kleinen gain-Faktor erreichen konnte, wird dieses Modell nicht für die HPO berücksichtigt. Stattdessen werden nur die Modelle aus den Abschnitten 4.3 und 4.4 optimiert.

Die HPO wird wieder mit OPTIMA durchgeführt. Hier können jetzt mehrere Netzwerke mit unterschiedlichen Hyperparametern parallel trainiert werden. Dafür muss in der Konfigurationsdatei des Netzwerks ein Suchraum definiert werden. Für beide Optimierungen werden dabei die Kombinationsmöglichkeiten aus Tabelle 4.7 gewählt.

Hyperparameter	mögliche Werte
Batch-Größe	{256, 512, 1024, 2048}
Anzahl Blöcke	{2, 3, 4, 5, 6, 7, 8}

Tabelle 4.7: Mögliche Werte für die Verwendung in der HPO. Die Werte beider Hyperparameter können unabhängig voneinander gewählt werden.

Eine geringere Anzahl von Transformerblöcken ist hierbei vorteilhaft, da dies direkt die Auswertungszeit des Netzwerks reduziert. Eine Verschlechterung durch Overtraining wird ebenfalls dadurch minimiert. Trotzdem wird die Abschätzung für $t_{\text{surr}} = 0,71$ ms beibehalten.

Eine geringere Batch-Größe sorgt für eine ungenauere Annäherung der loss-Funktion. Das klingt zwar zunächst unvorteilhaft, kann aber trotzdem die Leistung des Netzwerks verbessern. Durch die geringere Batch-Größe erhöht sich die Schrittweite in der Backpropagation. Das ermöglicht es dem Netzwerk unter Umständen, aus lokalen Minima der loss-Funktion zu entkommen und ein tieferliegendes Minimum zu erreichen.

Alle weiteren Hyperparameter des Netzwerks werden bei der Optimierung konstant gewählt. Einige der Hyperparameter sind hierbei anders, als in den vorläufigen Trainings. Insbesondere wurde die Anzahl der internen Multivektorkanäle und der Attentionheads auf 4 statt 8 gesetzt und der MLP-Block verwendet nun die Aktivierungsfunktion ReLU statt GELU. Diese Änderungen wurden lediglich vorgenommen, um die Trainingsdauer zu reduzieren.

Für beide Netzwerk-Strukturen werden jetzt jeweils 24 Trainings gestartet. OPTIMA wählt dabei die Hyperparameter der ersten Netzwerke zufällig und konzentriert sich danach auf die Kombinationen, die die beste Leistung erzielen konnten.

Hyperparameter	Wert
Interne Skalarkanäle	8
Interne Multivektorkanäle	4
Anzahl Attentionheads	4
MLP Aktivierung	ReLU
Symmetriebrechung	σ^{12} und γ^0
Lernrate	0,001
<i>ADAM</i> β_1	0,6
<i>ADAM</i> β_2	0,999
<i>ADAM</i> ε	10^{-8}
Weight Initialisierung	He uniform
Early-stopping-Geduld	40 Epochen

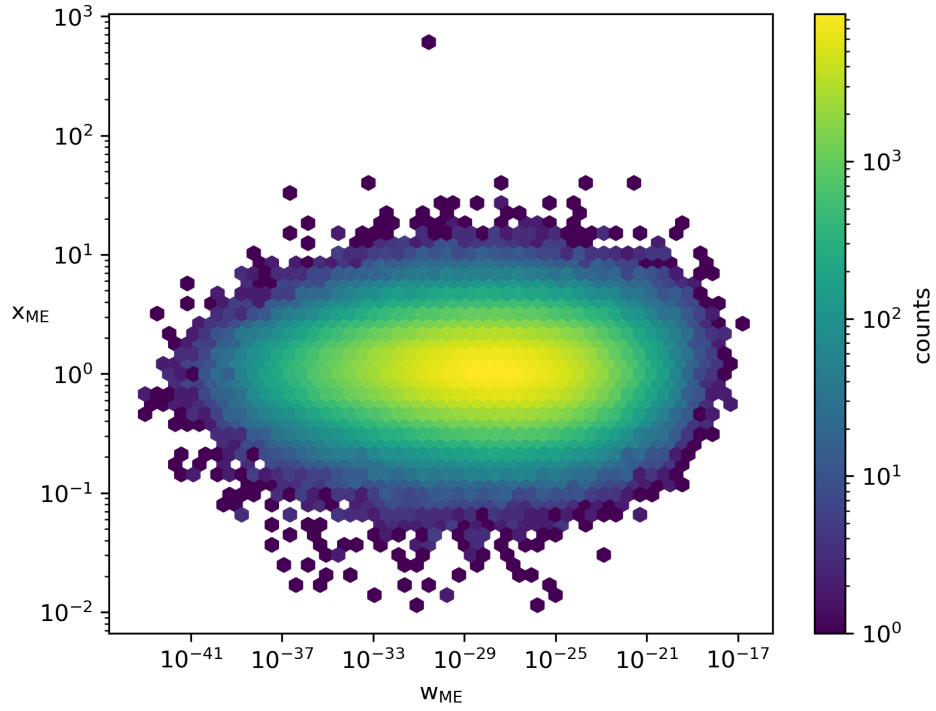
Tabelle 4.8: Die konstanten Hyperparameter für die HPO der Netzwerke aus den Abschnitten 4.3 und 4.4.

Die beiden besten Netzwerke aus der HPO haben die folgenden Hyperparameter in Tabelle 4.9.

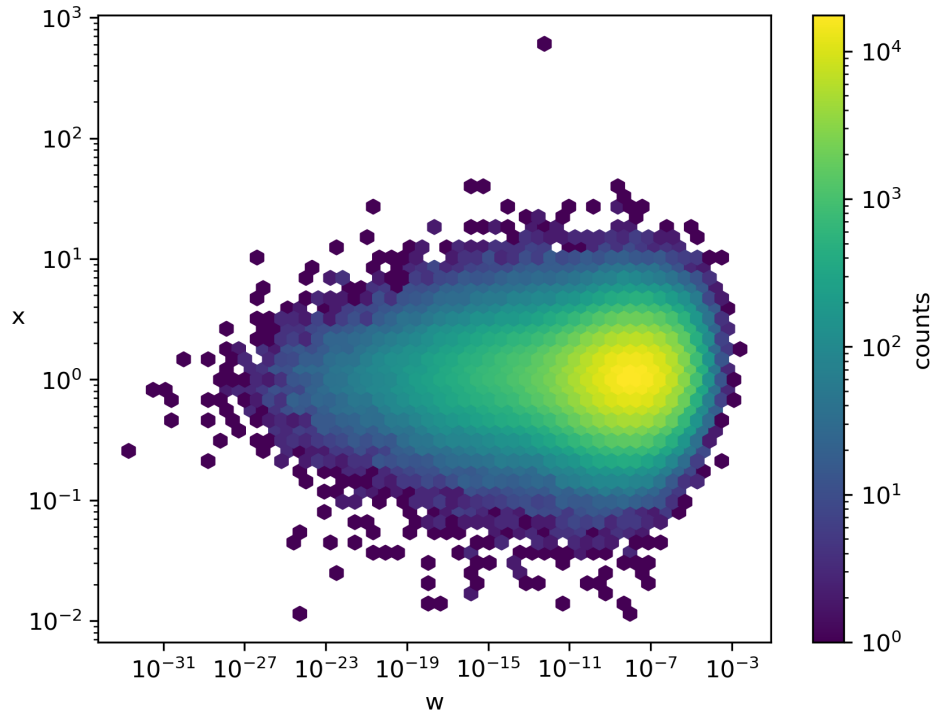
Netzwerkmodell	Batch-Größe	Anzahl Blöcke	gain-Faktor
Mit Skalaren, w_{ME} direkt	256	4	20,97155
Mit Skalaren, Dipolansatz	256	5	25,97959

Tabelle 4.9: Die besten Kombinationen der Hyperparameter aus dem Suchraum in Tabelle 4.7 für die Modelle aus den Abschnitten 4.3 und 4.4.

Man erkennt, dass die optimierten Netzwerke eine deutliche Verbesserung gegenüber den vorläufigen Trainings darstellen. Nun erreicht auch L-GATr mit der direkten Vorhersage von w_{ME} einen höheren gain-Faktor als der MLP in Abschnitt 4.1. Die Diagramme der Verteilungen von x und x_{ME} sind wieder sehr ähnlich zu denen der Abschnitte 4.3 und 4.4. Eine Verbesserung kann wieder nicht visuell festgestellt werden.

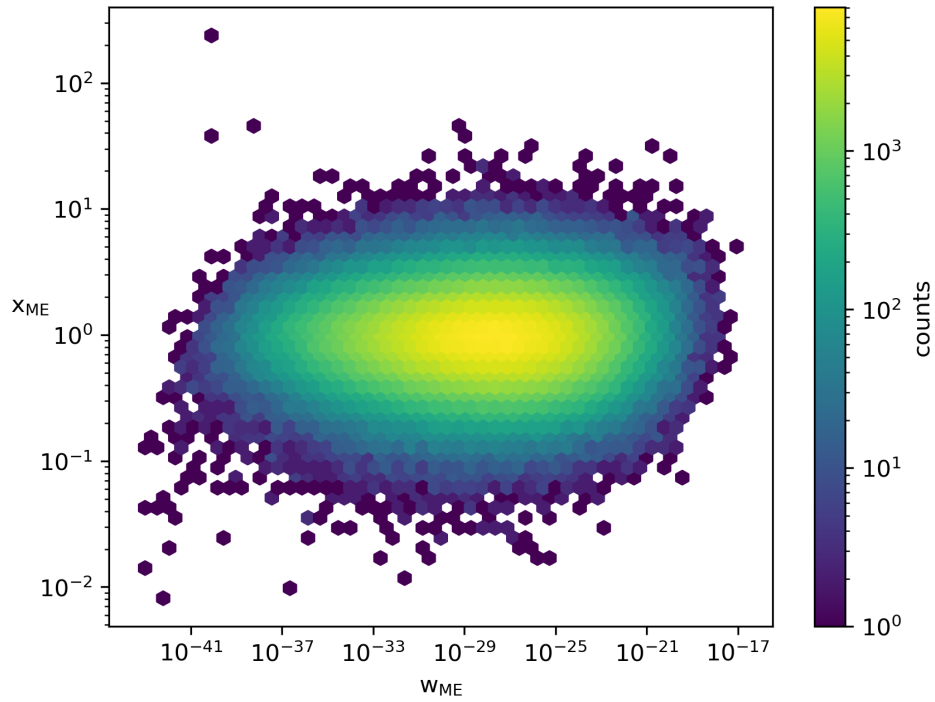


(a) Verteilung von $x_{ME} = w_{ME}/a$ in Abhängigkeit von w_{ME} .

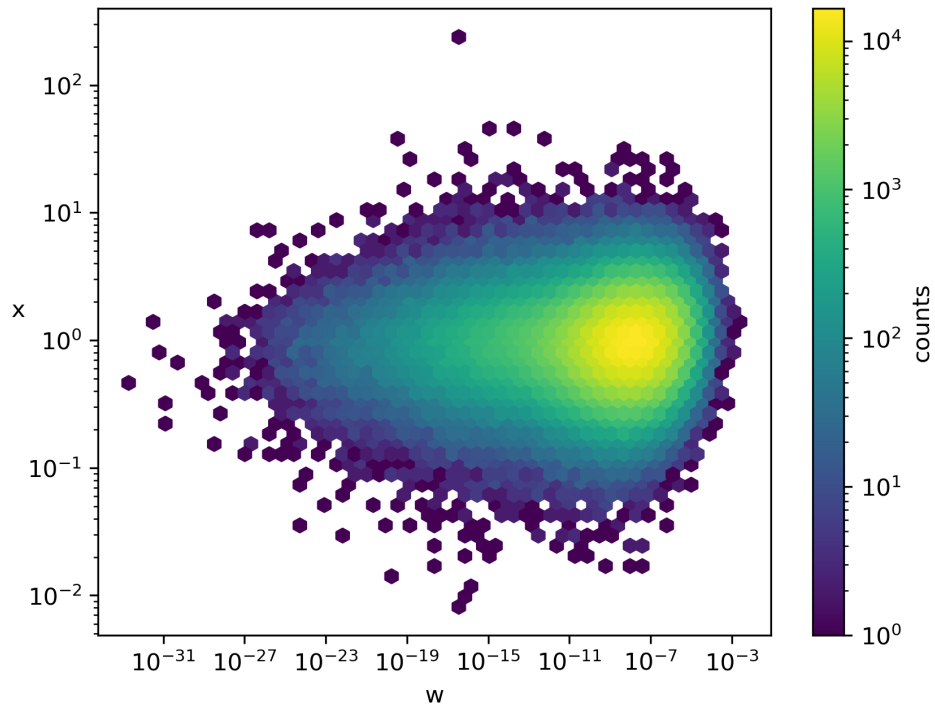


(b) Verteilung von $x = w/s$ in Abhängigkeit von w .

Abbildung 4.5: Doppeltlogarithmische hexagonale Histogramme der Verteilungen von x_{ME} und x für den L-GATr mit kinematischen Invarianten für die direkte Vorhersage des Matricelementgewichts mit den Hyperparametern in Tabelle 4.8 und Tabelle 4.9.



(a) Verteilung von $x_{\text{ME}} = w_{\text{ME}}/a$ in Abhängigkeit von w_{ME} .



(b) Verteilung von $x = w/s$ in Abhängigkeit von w .

Abbildung 4.6: Doppeltlogarithmische hexagonale Histogramme der Verteilungen von x_{ME} und x für den L-GATr mit kinematischen Invarianten für die Vorhersage der Dipolkoeffizienten mit den Hyperparametern in Tabelle 4.8 und Tabelle 4.9.

5 Zusammenfassung

Die Teilchenphysik stellt hohe Anforderungen nicht nur an die Genauigkeit der Messungen, sondern auch an die große Anzahl simulierter Teilchenkollisionen für den Vergleich zwischen Theorie und Realität. Aufgrund begrenzter Rechenressourcen sind hier clevere Algorithmen und der Einsatz maschineller Lernverfahren notwendig. Je schneller und genauer die Vorhersagen solcher als Surrogate bezeichneten Netzwerke sind, desto nützlicher sind sie für den Einsatz in Ereignisgeneratoren wie beispielsweise *SHERPA*.

In dieser Arbeit wurde eine neuartige Struktur mit dem Namen L-GATr auf ihre Funktionalität untersucht. Diese beinhaltet bereits die Gesetze der speziellen Relativitätstheorie und verspricht somit bessere Vorhersagen in Situationen, in denen die Teilchen eine Geschwindigkeit nahe der Lichtgeschwindigkeit c besitzen. Da das in Teilchenkollisionen am LHC immer der Fall ist, kann diese Netzwerkstruktur hier besonders hilfreich sein.

Das Netzwerk wird auf die Vorhersage des betragsquadrierten Matrixelements $w_{\text{ME}} = |M_{fi}|^2$ des Prozesses $g u \rightarrow e^+ e^- g g g g g u$ aus der Prozessgruppe $p p \rightarrow Z + 6 \text{ jets}$ trainiert. Da die direkte Vorhersage von w_{ME} aus den Viererimpulsen der ein- und auslaufenden Teilchen nur einen effektiven gain-Faktor von etwa $5,1 \cdot 10^{-5}$ liefert, wird der Einfluss von kinematischen Invarianten als weitere Eingabewerte und die Vorhersage von Dipolkoeffizienten untersucht. Im Vergleich mit einem MLP, der die derzeit beste Leistung erbringt, stellt man hier jetzt fest, dass L-GATr wie erwartet vergleichbare beziehungsweise bessere Vorhersagen trifft. Durch die anschließende Hyperparameteroptimierung wird dann die Leistung von L-GATr weiter gesteigert. Die besten gefundenen gain-Faktoren sind zum Vergleich in Tabelle 5.1 angegeben.

Netzwerkmodell	gain-Faktor
MLP mit Dipolansatz	12,78997
L-GATr, w_{ME} direkt	20,97155
L-GATr, Dipolansatz	25,97959

Tabelle 5.1: Vergleich der besten gain-Faktoren der verschiedenen Netzwerkstrukturen.

Da während der HPO mehrere Netzwerke mit denselben Hyperparametern, aber anders initialisierten Weights trainiert wurden, kann man auch einen mittleren gain-Faktor mit statistischer Unsicherheit angeben.

Netzwerkmodell	gain-Faktor
Mit Skalaren, w_{ME} direkt	$16,8 \pm 3, 1_{\text{stat}}$
Mit Skalaren, Dipolansatz	$25,4 \pm 0, 6_{\text{stat}}$

Tabelle 5.2: Mittlere gain-Faktoren aus der HPO für die beiden Netzwerkstrukturen mit den besten Hyperparametern in Tabelle 4.9.

Die statistischen Unsicherheiten wurden über die Kleinststatistik

$$\Delta_{\text{stat}} f_{\text{eff}} = \frac{\max_i f_{\text{eff},i} - \min_i f_{\text{eff},i}}{2} \quad (5.1)$$

angenähert. Diese Werte sollten allerdings mit großer Vorsicht betrachtet werden, da jeweils nur 3 beziehungsweise 2 Netzwerke zu ihrer Berechnung beigetragen haben (siehe Tabellen A.1 und A.2). Trotzdem erscheint es so, dass L-GATr mit diesen Hyperparametern im Mittel eine bessere Leistung erzielt als der MLP in Abschnitt 4.1.

Mit dieser Arbeit ist die Untersuchung von L-GATr jedoch längst nicht abgeschlossen. Als einfache Erweiterung könnte man andere Hyperparameter optimieren. Beispielsweise sollten die Anzahl der internen Skalar- und Multivektorkanäle und die Anzahl der Attentionheads einen möglicherweise entscheidenden Einfluss auf die Leistung des Netzwerks haben.

Eine weitere Idee ist eine Modifikation des Dipolansatzes. In Abschnitt 4.4 soll L-GATr die Koeffizienten C_{ijk} vorhersagen, die dann mit den Werten der Dipolfunktionen D_{ijk} multipliziert werden, um schließlich w_{ME} abzuschätzen. Eine Alternative wäre jedoch, die D_{ijk} als zusätzliche Skalare neben den kinematischen Invarianten als Eingabewerte an das Netzwerk zu übergeben und w_{ME} direkt vorherzusagen. Das erlaubt L-GATr eine größere Freiheit in der Verwendung der Eingabewerte und kann möglicherweise zu einer weiteren Verbesserung der Vorhersagen führen.

Natürlich sollte L-GATr auch für andere relevante Prozesse einer Protonenkollision getestet werden. Ein möglicher Prozess ist $g u \rightarrow e^+ e^- g g g u d \bar{d}$, da dessen Berechnung ebenfalls recht anspruchsvoll für bisherige Modelle war. Eine Anwendung für $t \bar{t}$ - Prozesse ist ebenfalls denkbar.

A Anhang

A.1 HPO Ergebnisse

Die folgenden beiden Tabellen enthalten nur die Netzwerke, deren Training ordnungsgemäß durch early-stopping beendet wurde. Die HPO in OPTIMA bricht nämlich manche Trainings auch vorzeitig ab, wenn die Metrikwerte deutlich schlechter sind als die von Netzwerken mit anderen Hyperparametern.

Anzahl Blöcke	Batch-Größe	Epochen	gain-Faktor
2	256	88	13,37609
4	256	71	14,68282
4	256	177	14,87279
4	256	161	20,97155
5	256	75	11,83004
8	2048	248	12,55548

Tabelle A.1: Alle Hyperparameterkombinationen der HPO für L-GATr mit Skalaren für die direkte Vorhersage von w_{ME} , für die das Training erst durch early-stopping beendet wurde.

Anzahl Blöcke	Batch-Größe	Epochen	gain-Faktor
2	256	96	14,20422
4	256	120	18,79703
5	256	178	24,75193
5	256	138	25,97959
6	256	69	15,29419
6	256	126	23,86049
8	2048	263	13,66096

Tabelle A.2: Alle Hyperparameterkombinationen der HPO für L-GATr mit Skalaren und Dipolansatz, für die das Training erst durch early-stopping beendet wurde.

A.2 Ergebnisse mit 90%-Quantil

Während des Suchens und Behebens der Fehler, die beim Programmieren zwangsläufig auftreten, wurden kurzzeitig die Definitionen von $w_{\max}$, $s_{\max}$ und $x_{\max}$ geändert. Statt des 99,9%-Quantils wurde dann nur das 90%-Quantil verwendet. Diese Definition ist für die Anwendung

nur wenig brauchbar, da die Algorithmen 1 und 2 jetzt mehr Ereignisse mit einem Gewicht ungleich 1 produzieren. Die Ergebnisse aus dieser Analyse sind nicht relevant für das Gesamtergebnis der Arbeit und werden hier nur zur vollständigen Dokumentation angegeben.

Netzwerkmodell	gain-Faktor
MLP mit Dipolansatz	37,44659
L-GATr ohne Dipolansatz	25,19888

Tabelle A.3: Vergleich der gain-Faktoren der verschiedenen Netzwerkstrukturen mit dem 90%-Quantil zum Zweck der Fehlersuche.

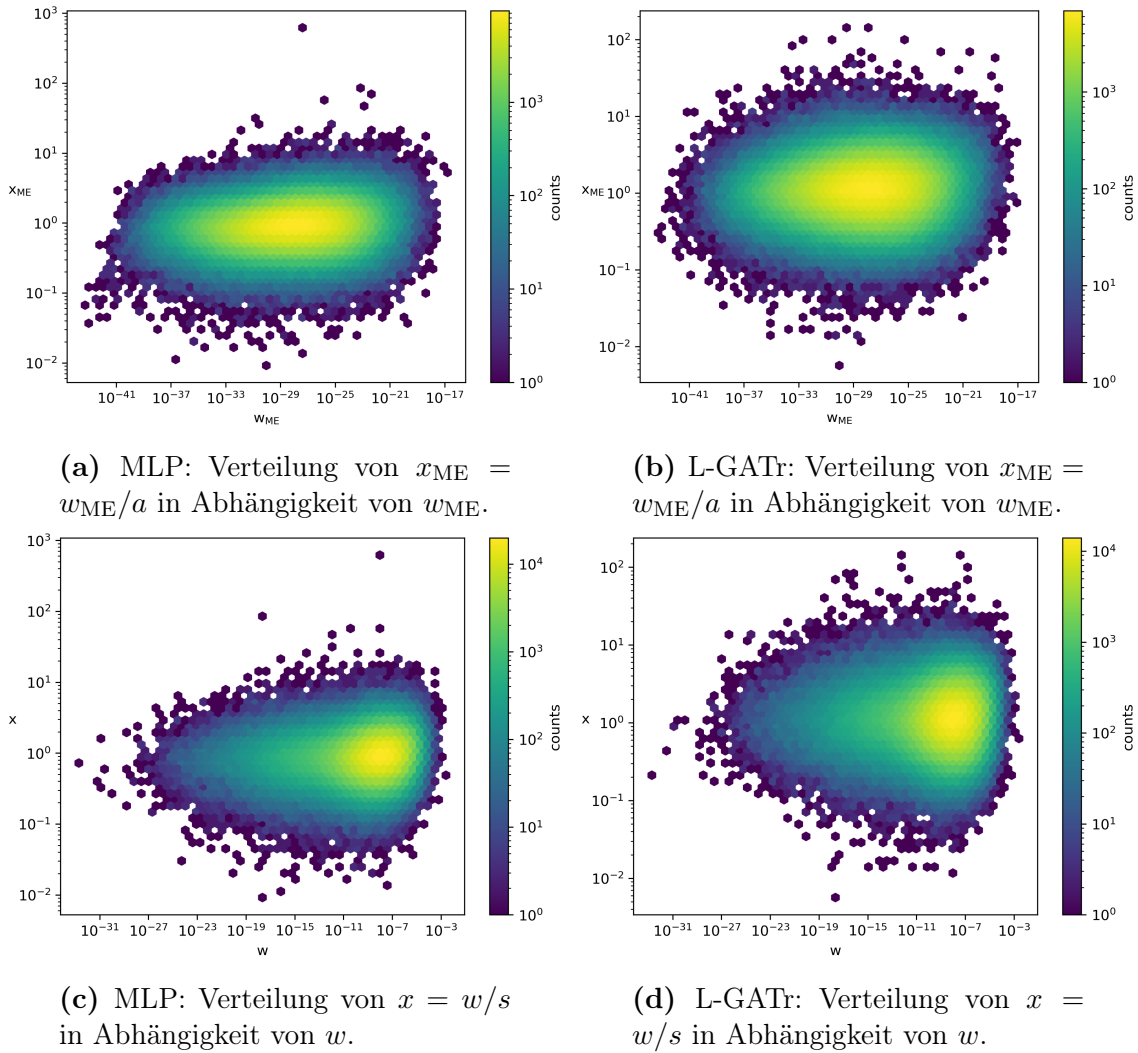


Abbildung A.1: Doppeltlogarithmische hexagonale Histogramme der Verteilungen von x_{ME} und x des MLP und L-GATr mit den Hyperparametern in Tabelle 4.1 beziehungsweise Tabelle 4.5 und mit dem 90%-Quantil zur Fehlerbehebung.

B Literatur

- [1] P.W. Higgs. „Broken symmetries, massless particles and gauge fields“. In: *Physics Letters* 12.2 (1964), S. 132–133. ISSN: 0031-9163. DOI: [https://doi.org/10.1016/0031-9163\(64\)91136-9](https://doi.org/10.1016/0031-9163(64)91136-9). URL: <https://www.sciencedirect.com/science/article/pii/0031916364911369>.
- [2] The ATLAS Collaboration. „Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC“. In: *Physics Letters B* 716.1 (Sep. 2012), S. 1–29. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214v2. URL: <http://dx.doi.org/10.1016/j.physletb.2012.08.020>.
- [3] Aidan Randle-Conde. *Feynman diagram maker*. URL: <https://www.aidansean.com/feynman/> (besucht am 23.04.2025).
- [4] Robert Mann. *An Introduction to Particle Physics and the Standard Model*. CRC Press Inc, 2009. ISBN: 978-1-4200-8298-2.
- [5] Katharina Danziger u. a. „Accelerating Monte Carlo event generation – rejection sampling using neural network event-weight estimates“. In: *SciPost Physics* 12.5 (Mai 2022). ISSN: 2542-4653. DOI: 10.21468/scipostphys.12.5.164. arXiv: 2109.11964v2 [hep-ph]. URL: <http://dx.doi.org/10.21468/SciPostPhys.12.5.164>.
- [6] Mathis Erik Schenker. *Optimierung von neuronalen Netzen für Matrixelement-Surrogate in Prozessgruppen*. Mai 2025. URL: <https://repository.cern/records/qe3xg-khb35>.
- [7] Timo Janßen u. a. „Unweighting multijet event generation using factorisation-aware neural networks“. In: *SciPost Physics* 15.3 (Sep. 2023). ISSN: 2542-4653. DOI: 10.21468/scipostphys.15.3.107. arXiv: 2301.13562v2 [hep-ph]. URL: <http://dx.doi.org/10.21468/SciPostPhys.15.3.107>.
- [8] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: <https://arxiv.org/abs/1412.6980>.
- [9] Grant Sanderson. *Neural Networks*. URL: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.
- [10] Ashish Vaswani u. a. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: <https://arxiv.org/abs/1706.03762>.

-
- [11] Johann Brehmer u. a. *L-GATr: Lorentz-Equivariant Geometric Algebra Transformer for High-Energy Physics*. URL: <https://indico.cern.ch/event/1498937/contributions/6311945/attachments/3003574/5293759/lgatr-cometa.pdf>.
 - [12] Johann Brehmer u. a. *A Lorentz-Equivariant Transformer for All of the LHC*. 2024. arXiv: 2411.00446 [hep-ph]. URL: <https://arxiv.org/abs/2411.00446>.
 - [13] Tri Dao u. a. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG]. URL: <https://arxiv.org/abs/2205.14135>.
 - [14] Enrico Bothmann u. a. *Event generation with Sherpa 3*. 2024. arXiv: 2410.22148 [hep-ph]. URL: <https://arxiv.org/abs/2410.22148>.
 - [15] Erik Bachmann. *OPTIMA: an Optimization Platform for Tuning Input Variables and Model Parameters*. URL: <https://gitlab.cern.ch/atlas-germany-dresden-vbs-group/optima>.
 - [16] Jonas Spinner, Víctor Bresó und Antoine Petitjean. *Lorentz-Equivariant Geometric Algebra Transformer*. URL: <https://github.com/heidelberg-hepml/lorentz-gatr/tree/main>.

C Glossar

Backpropagation

Ein Algorithmus zur effizienten Berechnung der partiellen Ableitungen der loss-Funktion von den Weight-Matrizen und Bias-Vektoren.

batch

Eine kleine meist zufällig gewählte Teilmenge des gesamten Trainingsdatensatzes, mit deren Hilfe die loss-Funktion angenähert wird, um so das Training deutlich zu beschleunigen.

early-stopping

Der vorzeitige Abbruch des Trainings, wenn sich das Surrogat innerhalb einer vorgegebenen Anzahl von Epochen nicht verbessert hat.

Epoche

Der einmalige Durchlauf aller Trainingsdaten durch das Surrogat, inklusive der anschließenden Validierung.

gain-Faktor

Der Gewinnfaktor f_{eff} der surrogat-basierten Ungewichtung. Wenn $f_{\text{eff}} > 1$, dann führt der Einsatz eines Surrogats zu einer schnelleren Berechnung.

HPO

Abkürzung für Hyperparameteroptimierung.

L-GATr

Eine Modifikation der klassischen Transformer-Architektur, sodass die Eingaben lorentz-invariant verarbeitet werden.

loss-Funktion

Eine Verlustfunktion, die während des Trainings als Maßstab für die Genauigkeit eines Surrogats dient.

machine-learning

Maschinelles Lernen. Eine Softwarestruktur, beispielsweise ein MLP, wird anhand von Beispieldaten trainiert, um eine gewisse Berechnungsvorschrift zu erlernen.

MLP

Eine einfache machine-learning-Architektur bestehen aus Schichten. Siehe Abschnitt 3.1.

overtraining

Das Surrogat trainiert zu lange auf den gleichen Eingabedaten und lernt diese effektiv „auswendig“. Die Leistung auf den Validierungs- und Testdaten sinkt damit wieder. Siehe auch undertraining.

QCD

Abkürzung für Quantenchromodynamik, die Theorie, die das Verhalten der starken Wechselwirkung von Quarks und Gluonen beschreibt.

Surrogat

Ein Programm oder eine Funktion, die die exakte Berechnung des Matrixelementgewichts durch eine weniger genaue aber deutlich schnellere Approximation ersetzt. Häufig verwendet man hier machine-learning.

undertraining

Das Training eines Surrogats wird beendet, bevor es das Verhalten der Eingabedaten genügend gut erlernen konnte. Die Leistung auf den Validierungs- und Testdaten erreicht somit nicht ihren besten Wert. Siehe auch overtraining.

Danksagung

An dieser Stelle möchte ich mich bei den Personen bedanken, die mich während der Arbeit an diesem Thema unterstützt haben.

Zunächst richtet sich mein Dank an Dr. Frank Siegert, Prof. Dr. Michael Kobel und Dr. Tim Herrmann, die mein Bachelorprojekt betreut und überhaupt erst ermöglicht haben.

Insbesondere Tim Herrmann und auch Erik Bachmann haben mir des Öfteren bei verschiedensten Problemen zur Seite gestanden.

Weiterer Dank gilt natürlich meiner Familie, und ganz besonders meinem Vater, der trotz mangelndem Fachwissen die Bürde des Korrekturlesens auf sich genommen hat.

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit im Rahmen der Betreuung am Institut für Kern- und Teilchenphysik ohne unzulässige Hilfe Dritter verfasst und alle Quellen als solche gekennzeichnet habe.

Leopold Max Michael
Dresden, Juni 2025