TECHNISCHE
UNIVERSITÄT
DRESDEN

Anwendung von maschinellem Lernen zur Regression am Beispiel des Diphoton-Prozesses

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

vorgelegt von

Andreas Weitzel
geboren am 10.08.1999 in Fulda

Institut für Kern- und Teilchenphysik
Fakultät Physik
Bereich Mathematik und Naturwissenschaften
Technische Universität Dresden
2021

Eingereicht am xx. Monat 20xx

1. Gutachter: Prof. Dr. XX
2. Gutachter: Prof. Dr. YY

Was soll hier hin?

Zusammenfassung
Der Wirkungsquerschnitt für die Produktion von Photon-Paaren bei der Partonstreuung $q\overline{q} \to \gamma\gamma$ wird berechnet. Daraus wird mithilfe von Partondichtefunktionen der Wirkungsquerschnitt für den hadronischen Prozess $pp \to \gamma\gamma$ ermittelt. Die Anwendung von Methoden des Deep-Learning zur Näherung der differentiellen Wirkungsquerschnitte wird untersucht. Dabei bieten die exponentiellen Variationen der Wirkungsquerschnitte eine wesentliche Herausforderung. Weiterhin wird die Eignung und Anwendbarkeit von Transfer-Learning zur schnellen Adaption des Lernergebnisses an andere Partondichtefunktionen untersucht. Schließlich wird von Monte-Carlo-Methoden Gebrauch gemacht, um die differentiellen Wirkungsquerschnitte zu integrieren.

Abstract
English:

Inhaltsverzeichnis

1 Einleitung

Maschinelles Lernen (ML) ist ein Schlagwort und Konzept, das zwar schon lange im Umlauf ist, jedoch seit einiger Zeit extrem an Beliebtheit gewinnt. Auch in der Physik haben verschiedene Methoden bereits Einzug gehalten. Eine davon ist Deep-Learning, das einen Bereich des maschinellen Lernens bezeichnet, in dem tiefe neuronale Netze verwendet werden. In dieser Arbeit soll die Eignung neuronaler Netzen zur Regression von differentiellen Wirkungsquerschnitten untersucht werden. Dies wird am Beispiel des Diphoton-Prozess durchgeführt, dessen differentieller Wirkungsquerschnitt sowohl auf partonischer Ebene, als auch auf hadronischer Ebene in führender Ordnung analytisch hergeleitet wird.

Die Photon-Paar-Produktion in Proton Kollisionen trug durch den Zerfallskanal $H \to \gamma\gamma$ des Higgs-Bosons wesentlich zu dessen Nachweis [1, 11] bei. Auch aktuell ist der Prozess interessant, so wird er dazu verwendet sehr massereiche Resonanzen, vorhergesagt von Theorien jenseits des Standardmodells, zu suchen [2]. Zur Simulation von Photon-Paar-Produktion werden aufwändige numerische Methoden, wie der Event-Generator SHERPA [12] verwendet, die sehr rechenintensiv sein können. Machine-Learning-Algorithmen können im Vergleich effizienter sein, indem sie den Wirkungsquerschnitt, nach Vorarbeit eines rechnerisch anspruchsvollen Algorithmus, erlernen. Der Vorteil liegt hierbei darin, die aufwändigen numerischen Methoden zur Berechnung einer ausreichenden Anzahl an Phasenraumpunkten nur einmalig zu verwenden, um mit diesen den ML-Algorithmus zu trainieren und anschließend eine größere Zahl an Punkten zu generieren.

In *Kapitel 2* wird mit der theoretischen Behandlung des Diphoton-Prozesses im Rahmen der Quantenelektrodynamik begonnen, wobei Ausdrücke für den differentiellen Wirkungsquerschnitt für den partonischen und hadronischen Prozess analytisch hergeleitet werden. *Kapitel 3* beschäftigt sich zunächst mit den Konzepten hinter Maschinellem Lernen und speziell Deep-Learning mit tiefen neuronalen Netzen (DNN). Am Ende des Kapitels werden die Grundlagen einer Monte-Carlo-Integration (MC-Integration) besprochen. Die Anwendung der DNN folgt in *Kapitel 4*, wobei zunächst die differentiellen Wirkungsquerschnitte des Diphoton-Prozesses genähert werden. Anschließend wird das Lernen von Gewichten zur Umgewichtung von Ergebnissen für unterschiedliche Sets von Partondichtefunktionen (PDF) und die Eignung von Transfer-Learning (TL) untersucht.

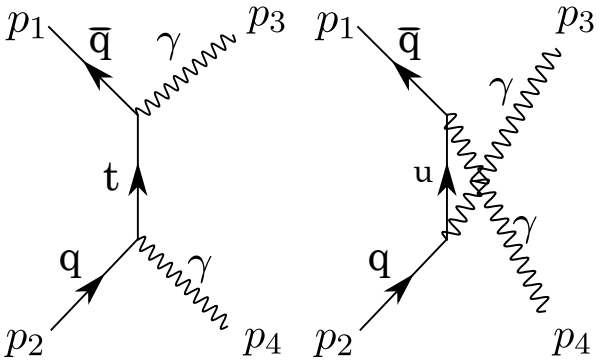
In dieser Arbeit verwendete Abkürzungen sind in *Tabelle A.11* zusammengefasst. Es werden durchweg natürliche Einheiten, sprich $\hbar = c = 1$ verwendet. Vektoren werden mit Fett gedruckten Kleinbuchstaben (Bsp. $\mathbf{x}$) und Matrizen oder Tensoren mit Fett gedruckten Großbuchstaben (Bsp. $\mathbf{M}$) notiert. Speziell Dreiervektoren werden mit einem Pfeil gekennzeichnet (Bsp. $\vec{p}$). Vierervektoren ergeben sich aus dem Kontext.

Unter `https://github.com/andiw99/Bachelor-Thesis` kann der gesamte Python-Code, der während dieser Arbeit entstanden ist, eingesehen werden. Hierbei sind alle Skripte zur Erzeugung der Diagramme im Ordner „Plotscripts“ durchnummeriert zu finden. Alle mit ML in Verbindung stehenden Funktionen und Klassen sind in ml.py definiert. Analoges gilt für MC.py. Es wird TensorFlow [25] 2.4.1 genutzt, wobei die Skripte auch mit TensorFlow 2.... getestet wurden.

2 Diphoton-Prozess

2.1 Matrixelement des partonischen Diphoton-Prozess

Zunächst wird der differentielle Wirkungsquerschnitt des partonischen Diphoton-Prozesses $q\bar{q} \to \gamma\gamma$ aus den Feynman-Regeln der Quantenelektrodynamik (QED) in führender Ordnung hergeleitet. Es werden hochrelativistische Quarks betrachtet, deren Ruhemasse vernachlässigt werden kann.



(a) t-Kanal **(b)** u-Kanal

Abb. 2.1: Feynman-Diagramme des Diphoton-Prozesses $q\bar{q} \to \gamma\gamma$ führender Ordnung

In *Abb. 2.1* sind die Feynman Diagramme führender Ordnung gezeigt. Hieraus können die Matrixelemente aus *Gl. 2.1* abgeleitet werden. Es werden die Notation $\gamma^\mu p_\mu = \not{p}$, die Mandelstam-Variablen[1], sowie $\epsilon_\mu(p_i) \equiv \epsilon_{\mu,\lambda_i}$ verwendet, um die Matrixelemente zu vereinfachen (siehe *Gl. 2.2*). λ_i beschreibt die Polarisation des Photons.

$$\begin{aligned}
\mathcal{M}_t &= \overline{\nu}\left(p_1\right)\left(-iQ_q e\gamma^\mu\right)\epsilon^*_\mu\left(p_3\right)\left(\frac{\gamma^\alpha\left(p_{1,\alpha}-p_{3,\alpha}\right)}{\left(p_1-p_3\right)^2}\right)\left(-iQ_q e\gamma^\nu\right)\epsilon^*_\nu\left(p_4\right)u\left(p_2\right) \\
\mathcal{M}_u &= \overline{\nu}\left(p_1\right)\left(-iQ_q e\gamma^\rho\right)\epsilon^*_\rho\left(p_4\right)\left(\frac{\gamma^\beta\left(p_{1,\beta}-p_{4,\beta}\right)}{\left(p_1-p_4\right)^2}\right)\left(-iQ_q e\gamma^\sigma\right)\epsilon^*_\sigma\left(p_3\right)u\left(p_2\right)
\end{aligned} \tag{2.1}$$

$$\begin{aligned}
\mathcal{M}_t &= -\frac{Q_q^2 e^2}{t}\left[\overline{\nu}(p_1)\gamma^\mu\epsilon^*_{\mu,\lambda_3}\left(\not{p}_1-\not{p}_3\right)\gamma^\nu\epsilon^*_{\nu,\lambda_4}u(p_2)\right] \\
\mathcal{M}_u &= -\frac{Q_q^2 e^2}{u}\left[\overline{\nu}(p_1)\gamma^\rho\epsilon^*_{\rho,\lambda_4}\left(\not{p}_1-\not{p}_4\right)\gamma^\sigma\epsilon^*_{\sigma,\lambda_3}u(p_2)\right]
\end{aligned} \tag{2.2}$$

Die Vierervektoren sind wie in *Abb. 2.2* gewählt und in *Gl. 2.3* aufgeführt. Die Mandelstam-Variablen ergeben sich zu Gl. 2.4.

$$p_1 = \begin{pmatrix} p \\ 0 \\ 0 \\ p \end{pmatrix} \quad p_2 = \begin{pmatrix} p \\ 0 \\ 0 \\ -p \end{pmatrix} \quad p_3 = \begin{pmatrix} p \\ \sin(\theta)p \\ 0 \\ \cos(\theta)p \end{pmatrix} \quad p_4 = \begin{pmatrix} p \\ -\sin(\theta)p \\ 0 \\ -\cos(\theta)p \end{pmatrix} \tag{2.3}$$

[1] $t = (p_1 - p_3)^2$, $u = (p_1 - p_4)^2$

$$t = -4p^2 \cos^2\left(\frac{\theta}{2}\right) \quad \text{und} \quad u = -4p^2 \sin^2\left(\frac{\theta}{2}\right) \ . \tag{2.4}$$

Das totale Matrixelement wird durch Summation der Anteile des u- und t-Kanals berechnet:

$$\begin{aligned} \mathcal{M} = \mathcal{M}_u + \mathcal{M}_t &= \mathcal{F}\left[\overline{\nu}(p_1)\left(\frac{\Gamma_t}{a} + \frac{\Gamma_u}{b}\right) u(p_2)\right] \\ &= \mathcal{F}\left[\overline{\nu}(p_1)\Gamma u(p_2)\right] \ , \end{aligned} \tag{2.5}$$

wobei die Ersetzungen aus *Gl. 2.6* gewählt wurden.

$$\begin{aligned} \Gamma_t = \gamma^\mu \epsilon^*_{\mu,\lambda_3}(\not{p}_1 - \not{p}_3)\gamma^\nu \epsilon^*_{\nu,\lambda_4} \quad &\text{und} \quad \Gamma_u = \gamma^\rho \epsilon^*_{\rho,\lambda_4}(\not{p}_1 - \not{p}_4)\gamma^\sigma \epsilon^*_{\sigma,\lambda_3} \\ \text{sowie} \quad \mathcal{F} = \frac{Q_q^2 e^2}{4p^2} \quad &\text{und} \quad \Gamma = \frac{\Gamma_t}{\cos^2\left(\frac{\theta}{2}\right)} + \frac{\Gamma_u}{\sin^2\left(\frac{\theta}{2}\right)} \\ \cos^2\left(\frac{\theta}{2}\right) = a \quad &\text{und} \quad \sin^2\left(\frac{\theta}{2}\right) = b \end{aligned} \tag{2.6}$$

Bei der Berechnung des gemittelten Quadrats des Betrages des Matrixelementes, müssen die möglichen Anfangszustände der Quarks und Endzustände der Photonen berücksichtigt werden. Während die Endzustände eine Summe über mögliche Helizitäten s_3, s_4 und Polarisationen λ_3, λ_4 ergeben, können die Quarks drei verschiedene Farbzustände und jeweils zwei verschiedene Helizitäten annehmen, sodass die Anfangszustände einen Faktor 1/12 liefern:

$$\left\langle |\mathcal{M}|^2 \right\rangle = \frac{1}{12} \sum_{s_3,s_4} \sum_{\lambda_3,\lambda_4} |\mathcal{M}|^2 \ . \tag{2.7}$$

Um die Summe über die Helizitäten auszuführen, wird Casimirs Trick verwendet:

$$\sum_{s_3,s_4} |\mathcal{M}|^2 = \mathcal{F}^2 \sum_{s_3,s_4} \left[\overline{\nu}(p_1)\Gamma u(p_2)\right]\left[\overline{\nu}(p_1)\Gamma u(p_2)\right]^* = \mathcal{F}^2 \ \mathrm{Tr}\left[\Gamma \not{p}_2 \overline{\Gamma} \not{p}_1\right] \ , \tag{2.8}$$

wobei $\overline{\Gamma} = \gamma^0 \Gamma^\dagger \gamma^0 = \frac{\overline{\Gamma}_t}{a} + \frac{\overline{\Gamma}_u}{b}$ die Dirac-Adjungierte bezeichnet. Für die Dirac-adjungierten $\overline{\Gamma}_t, \overline{\Gamma}_u$ ergibt sich:

$$\overline{\Gamma}_t = \gamma^\nu \epsilon_{\nu,\lambda_4}(\not{p}_1 - \not{p}_3)\gamma^\mu \epsilon_{\mu,\lambda_3} \quad \text{und} \quad \overline{\Gamma}_u = \gamma^\sigma \epsilon_{\sigma,\lambda_3}(\not{p}_1 - \not{p}_4)\gamma^\rho \epsilon_{\rho,\lambda_4} \ . \tag{2.9}$$

Gl. 2.8 wird damit zu:

$$\mathrm{Tr}\left[\Gamma \not{p}_2 \overline{\Gamma} \not{p}_1\right] = \mathrm{Tr}\left[\frac{1}{a^2}\Gamma_t \not{p}_2 \overline{\Gamma}_t \not{p}_1 + \frac{1}{ab}\Gamma_t \not{p}_2 \overline{\Gamma}_u \not{p}_1 + \frac{1}{ba}\Gamma_u \not{p}_2 \overline{\Gamma}_t \not{p}_1 + \frac{1}{b^2}\Gamma_u \not{p}_2 \overline{\Gamma}_u \not{p}_1\right] \ . \tag{2.10}$$

Bei Einsetzen von *Gl. 2.10* in *Gl. 2.7* ergeben sich Terme in folgendem Schema:

$$T_{ij} = \frac{1}{12} \sum_{\lambda_3,\lambda_4} \mathcal{F}^2 \ \mathrm{Tr}\left[\frac{1}{ij}\Gamma(i) \not{p}_2 \overline{\Gamma}(j) \not{p}_1\right] \quad \text{mit} \quad i,j \in \{a,b\} \ . \tag{2.11}$$

Hierbei wird $\Gamma(a) = \Gamma_t$ und $\Gamma(b) = \Gamma_u$ identifiziert. Zunächst wird in *Gl. 2.12* der Fall $i = j$ evaluiert,

wobei diverse Spur-Methoden und Identitäten der Dirac-Matrizen verwendet werden.

$$\begin{aligned}
T_{aa} &= \frac{\mathcal{F}^2}{12a^2} \sum_{\lambda_3,\lambda_4} \mathrm{Tr}\left[\gamma^\mu \epsilon^*_{\mu,\lambda_3}(\not{p}_1 - \not{p}_3)\gamma^\nu \epsilon^*_{\nu,\lambda_4}\not{p}_2\gamma^{\nu'}\epsilon_{\nu',\lambda_4}(\not{p}_1 - \not{p}_3)\gamma^{\mu'}\epsilon_{\mu',\lambda_3}\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{12a^2} \sum_{\lambda_3,\lambda_4} \epsilon^{*\mu}_{\lambda_3}\epsilon^{\mu'}_{\lambda_3}\epsilon^{*\nu}_{\lambda_4}\epsilon^{\nu'}_{\lambda_4}\ \mathrm{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_3)\gamma_\nu\not{p}_2\gamma_{\nu'}(\not{p}_1 - \not{p}_3)\gamma_{\mu'}\not{p}_1\right] \\
&\overset{(2.15)}{=} \frac{\mathcal{F}^2}{12a^2}\ g^{\mu\mu'}g^{\nu\nu'}\ \mathrm{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_3)\gamma_\nu\not{p}_2\gamma_{\nu'}(\not{p}_1 - \not{p}_3)\gamma_{\mu'}\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{12a^2}\ \mathrm{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_3)\gamma_\nu\not{p}_2\gamma^\nu(\not{p}_1 - \not{p}_3)\gamma^\mu\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{3a^2}\ \mathrm{Tr}\left[(\not{p}_1 - \not{p}_3)\not{p}_2(\not{p}_1 - \not{p}_3)\not{p}_1\right] \\
&= \frac{8\mathcal{F}^2}{3a^2}(p_3\cdot p_2)(p_3\cdot p_1)
\end{aligned} \tag{2.12}$$

Wobei die Vollständigkeitsrelation für reale Photonen (siehe *Gl. 2.14*) verwendet wurde. Es folgt analog:

$$T_{bb} = \frac{8\mathcal{F}^2}{3b^2}(p_4\cdot p_2)(p_4\cdot p_1)\ . \tag{2.13}$$

$$\sum_{\lambda=1}^{2}\epsilon^\mu_\lambda\epsilon^{*\nu}_\lambda = -g^{\mu\nu} \tag{2.14}$$

Für $i \neq j$ ergibt sich:

$$\begin{aligned}
T_{ab} &= \frac{\mathcal{F}^2}{12ab}\ \mathrm{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_4)\gamma_\nu\not{p}_2\gamma^\mu(\not{p}_1 - \not{p}_3)\gamma^\nu\not{p}_1\right] \\
&= \frac{4\mathcal{F}^2}{3ab}\left[(p_1\cdot p_2)\left[-2\,(p_1\cdot p_4) + (p_3\cdot p_4)\right] - (p_1\cdot p_3)(p_2\cdot p_4) + (p_2\cdot p_3)(p_1\cdot p_4)\right]\ .
\end{aligned} \tag{2.15}$$

und analog:

$$T_{ba} = \frac{4\mathcal{F}^2}{3ab}\left[(p_1\cdot p_2)\left[-2\,(p_1\cdot p_3) + (p_3\cdot p_4)\right] - (p_1\cdot p_4)(p_2\cdot p_3) + (p_1\cdot p_3)(p_2\cdot p_4)\right] \tag{2.16}$$

Beim Einsetzen der expliziten Vierervektoren aus *Gl. 2.3*, fällt auf, dass $T_{ab} + T_{ba} = 0$. Die Summe über die Helizitäten und Polarisationen wurde nun ausgeführt, sodass *Gl. 2.7* umgeschrieben werden kann zu:

$$\begin{aligned}
\left\langle|\mathcal{M}|^2\right\rangle &= \frac{8}{3}\mathcal{F}^2\left(\frac{1}{a^2}(p_3\cdot p_2)(p_3\cdot p_1) + \frac{1}{b^2}(p_4\cdot p_2)(p_4\cdot p_1)\right) \\
&= \frac{2}{3}Q_q^4e^4\left[\frac{1-\cos^2(\theta)}{\cos^4\left(\frac{\theta}{2}\right)} + \frac{1-\cos^2(\theta)}{\sin^4\left(\frac{\theta}{2}\right)}\right] \\
&= \frac{4}{3}Q_q^4e^4\frac{1+\cos^2(\theta)}{\sin^2(\theta)} = \frac{4}{3}Q_q^4e^4\cosh(2\eta)\ .
\end{aligned} \tag{2.17}$$

Hierbei ist $\eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right)$ die Pseudo-Rapidität.

2.2 Differentieller Wirkungsquerschnitt des partonischen Prozesses

Mithilfe von Fermis goldener Regel kann aus dem Betragsquadrat des Übergangsmatrixelementes der Wirkungsquerschnitt berechnet werden. Im Schwerpunktsystem mit vernachlässigbaren Ruhemassen ergibt sich der Zusammenhang in *Gl. 2.18*, wobei $\mathrm{d}\Omega = \sin(\theta)\mathrm{d}\theta\mathrm{d}\varphi$ das Raumwinkelelement und $s = (p_1 + p_2)^2$ der Betrag der Schwerpunktsenergie bezeichnet.

$$\sigma = \frac{1}{64\pi^2 s}\int \left\langle |\mathcal{M}|^2 \right\rangle \mathrm{d}\Omega = \frac{1}{32\pi s}\int \left\langle |\mathcal{M}|^2 \right\rangle \sin(\theta)\mathrm{d}\theta \tag{2.18}$$

Für den differentiellen Wirkungsquerschnitt $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ ergibt sich *Gl. 2.19*, wobei ein Symmetriefaktor $\frac{1}{2}$ hinzukommt, da die beiden Photonen im Endzustand identisch sind.

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\theta} = \frac{1}{2}\cdot\frac{Q_q^4 e^4}{24\pi s}\frac{1+\cos^2(\theta)}{\sin(\theta)} \tag{2.19}$$

Mithilfe der Kettenregel lässt sich der differentielle Wirkungsquerschnitt in Abhängigkeit von η bestimmen:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\eta} = \frac{\mathrm{d}\theta}{\mathrm{d}\eta}\frac{\mathrm{d}\sigma}{\mathrm{d}\theta} = \frac{Q_q^4 e^4}{48\pi s}(1+\tanh^2(\eta)) \ . \tag{2.20}$$

2.3 Hadronischer Diphoton Prozess

Aufgrund des *Confinement* kommen Quarks nicht als freie Teilchen vor, sodass lediglich der hadronische Prozess $pp \to \gamma\gamma$ beobachtet werden kann. Die Protonen besitzen hierbei zwei up-Quarks und ein down-Quark als Valenzquarks, sowie verschiedene Quark-Antiquark-Paare als Seequarks. Prallen zwei Protonen mit hohen Energien aufeinander, wird die Substruktur des Protons aufgelöst und die Konstituenten des Hadrons können miteinander interagieren. Bei diesen Interaktionen können die Quarks dann als quasi-freie Teilchen betrachtet werden.

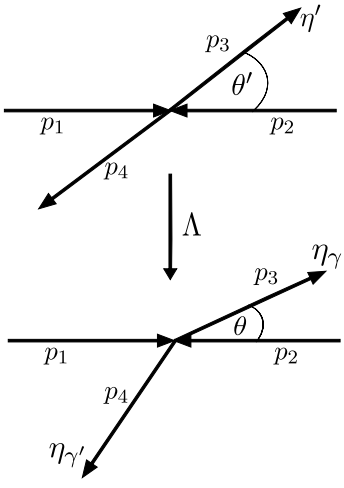


Abb. 2.2: Kinematik der Stoßprozesse im Labor- und Schwerpunktssystem

Das Schwerpunktsystem der Quarks unterscheidet sich im Allgemeinen vom Schwerpunktsystem der Protonen (Laborsystem). Der Impulsbruchteil eines Quarks innerhalb eines Hadrons ist nicht fest definiert, sodass ihm zunächst ein unbestimmter Bruchteil x des Gesamtimpulses zugeordnet wird. Das Proton wird nun in einem System betrachtet, indem es eine sehr hohe Energie $E \gg m_p$ besitzt, sodass seine Ruhemasse vernachlässigt werden und sein Vierervektor als $\mathbf{p}_p = (E, 0, 0, E)$ geschrieben werden kann. Im vorliegenden System können die Transversalimpulse der Partonen vernachlässigt werden, sodass ihr Vierervektor sich zu Gl. 2.21 ergibt.

$$\mathbf{p}_q = (xE, 0, 0, xE) = x\mathbf{p}_p \ . \tag{2.21}$$

Findet bei einer Interaktion ein Impulsübertrag $\mathbf{q}$ statt, so geht der Vierervektor des Partons $x\mathbf{p}_p \rightarrow (x\mathbf{p}_p + \mathbf{q})$ über. Durch Einsetzen der invarianten Masse beider Zustände (siehe *Gl. 2.22*) ineinander, kann nach dem Impusbruchteil x aufgelöst werden (siehe *Gl. 2.23*).

$$\left(x\mathbf{p}_p\right)^2 = m_q^2 \quad \text{und} \quad \left(x\mathbf{p}_p + q\right)^2 = \left(x\mathbf{p}_p\right)^2 + 2x\mathbf{p}_p \cdot \mathbf{q} + \mathbf{q}^2 = m_q^2 \tag{2.22}$$

$$2x\mathbf{p}_p \cdot \mathbf{q} + \mathbf{q}^2 = 0 \quad \Rightarrow \quad x = \frac{-\mathbf{q}^2}{2\mathbf{p}_p \cdot \mathbf{q}} \tag{2.23}$$

Das x in *Gl. 2.23* ist hierbei die Bjorken-Skalenvariable. Sie repräsentiert bei hohen Proton-Impulsen den Impulsbruchteil, den ein Parton im Proton trägt.

Da es nicht möglich ist die Impulsbruchteile vor der Reaktion zu kennen, wird der Prozess im Schwerpunktsystem der kollidierenden Protonen beschrieben. Hier folgt der Impulsbruchteil x einer Wahrscheinlichkeitsverteilung, der *Partondichtefunktion* (PDF) $f_i(x, Q^2)$, des Protons. Diese PDFs beschreiben die Wahrscheinlichkeitsdichte, bei einer Energieskala $Q^2 = -\mathbf{q}^2$, das entsprechende Parton i mit dem Impulsbruchteil x zu finden. Sie können nicht aus ersten Prinzipien abgeleitet und müssen experimentell bestimmt werden.

Die Partondichtefunktionen können genutzt werden, um einen Ausdruck für den totalen Wirkungsquerschnitt $pp \rightarrow \gamma\gamma$ zu finden. Ist der totale Wirkungsquerschnitt eines partonischen Prozesses zwischen den Partonen i und j, bei den festgelegten Impulsbruchteilen x_1 und x_2 und der Energieskala Q^2 (genannt $\tilde{\sigma}_{i,j}(x_1, x_2, Q^2)$) bekannt, dann kann mithilfe der PDF der Wirkungsquerschnitt $\sigma_{i,j}$ für die Reaktion der Partonen i und j bei dem Zusammenstoß von zwei Protonen berechnet werden (siehe *Gl. 2.24*).

$$\sigma_{i,j} = \int f_i(x_1, Q^2) f_j(x_2, Q^2) \tilde{\sigma}_{i,j}(x_1, x_2, Q^2) \mathrm{d}x_1 \mathrm{d}x_2 \tag{2.24}$$

Der totale Wirkungsquerschnitt ergibt sich dann als Summe aller möglichen $\sigma_{i,j}$, wobei im Diphoton-Prozess lediglich $i = q = \overline{j}$ beitragen:

$$\sigma = \sum_q \left(\sigma_{q,\overline{q}} + \sigma_{\overline{q},q}\right) \ . \tag{2.25}$$

In *Abschnitt 2.2* wurde der differentielle Wirkungsquerschnitt für den partonischen Prozess σ_p im Schwerpunktsystem der Konstituenten berechnet. $\tilde{\sigma}_{q,\overline{q}}(x_1, x_2, Q^2)$ ergibt sich damit nach *Gl. 2.26*.

$$\tilde{\sigma}_{q,\overline{q}}(x_1, x_2, Q^2) = \int \frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta}\left(x_1, x_2, Q^2\right) \mathrm{d}\eta \tag{2.26}$$

Gl. 2.26 gilt im Schwerpunktsystem der Partonen und muss im Folgenden in das Laborsystem transformiert werden. Weiterhin muss die Mandelstam-Variable s, die das Quadrat der Schwerpunktsenergie der Partonen darstellt, in Abhängigkeit von x_1, x_2 ausgedrückt werden. Für die Partonen q und $\overline{q}$ mit den Impulsbruchteilen x_1 und x_2 lassen sich im Schwerpunktsystem der beiden Hadronen ihre Vierervektoren schreiben als *Gl. 2.27*, wobei E die Strahlenergie bezeichnet.

$$\mathbf{p}_q = (x_1 E, 0, 0, x_1 E) \quad \text{und} \quad \mathbf{p}_{\overline{q}} = (x_2 E, 0, 0, -x_2 E) \tag{2.27}$$

Mit *Gl. 2.27* ergibt sich die Schwerpunktenergie zu:

$$\sqrt{s} = 2\sqrt{x_1 x_2}E \ . \tag{2.28}$$

Im folgenden werden Variablen im Laborsystem ungestrichen und Variablen im Schwerpunktsystem der Partonen gestrichen benannt. Die Pseudo-Rapidität, die sich für masselose Teilchen additiv bei Inertialsystemwechsel verhält, transformiert sich, wenn sich das Schwerpunktsystem der Partonen mit der Geschwindigkeit β zum Laborsystem bewegt, nach *Gl. 2.29*.

$$\eta' = \eta + \frac{1}{2}\ln\left(\frac{1-\beta}{1+\beta}\right) \quad \Rightarrow \quad \frac{\mathrm{d}\eta'}{\mathrm{d}\eta} = 1 \tag{2.29}$$

Damit ergibt sich für den Wirkungsquerschnitt im Laborsystem:

$$\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta} = \frac{\mathrm{d}\eta'}{\mathrm{d}\eta}\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta'} = \frac{Q_q^4 e^4}{48\pi s}(1+\tanh^2(\eta')). \tag{2.30}$$

Die Geschwindigkeit β ergibt sich mit den Dreierimpulsen $\vec{p}$ zu *Gl. 2.31*.

$$\beta = \frac{|\vec{p}_q + \vec{p}_{\overline{q}}|}{m_q + m_{\overline{q}}} = \frac{(x_1 - x_2)E}{(x_1 + x_2)E} = \frac{x_1 - x_2}{x_1 + x_2} \tag{2.31}$$

Durch Einsetzen der gefunden Ausdrücke für s, η', und β in *Gl. 2.30*, ergibt sich mit $Q^2 = 2x_1x_2E^2$ insgesamt für die differentiellen Wirkungsquerschnitt im Laborsystem:

$$\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta}\left(x_1, x_2, Q^2, q\right) = \frac{Q_q^4 e^4}{96\pi Q^2}\left(1 + \tanh^2\left(\eta + \frac{1}{2}\ln\left(\frac{x_2}{x_1}\right)\right)\right) \ . \tag{2.32}$$

Schließlich können mithilfe von *Gl. 2.24*, *Gl. 2.26* und *Gl. 2.25* die Ausdrücke *Gl. 2.33* für den totalen und *Gl. 2.34* für den dreifach differentiellen Wirkungsquerschnitt gefunden werden.

$$\sigma = \sum_q \int \left[f_q(x_1, Q^2) f_{\overline{q}}(x_2, Q^2) + f_{\overline{q}}(x_1, Q^2) f_q(x_2, Q^2)\right] \frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta} \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}\eta \tag{2.33}$$

$$\frac{\mathrm{d}^3\sigma}{\mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}\eta} = \sum_q \left[f_q(x_1, Q^2) f_{\overline{q}}(x_2, Q^2) + f_{\overline{q}}(x_1, Q^2) f_q(x_2, Q^2)\right] \frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta} \tag{2.34}$$

2.4 Umgewichtung zwischen PDF-Sets

Das quantitative Ergebnis von *Gl. 2.34* hängt von der verwendeten Partondichtefunktionen ab. Je nach Messung und Anpassung ergeben sich kleine Unterschiede zwischen den verschiedenen Sets. Die *Umgewichtung* entspricht einem Umrechnungsfaktor zwischen differentiellen Wirkungsquerschnitten, die mit verschiedenen Partondichtefunktionen berechnet wurden. Aus *Gl. 2.34* ergeben sich die Gewichte zwischen den PDF-Sets $f^{(1)}$ und $f^{(2)}$ zu *Gl. 2.35*.

$$w\left(x_1, x_2, Q^2\right) = \frac{\sum_q Q_q^4 \left[f_q^{(1)}(x_1, Q^2) f_{\overline{q}}^{(1)}(x_2, Q^2) + f_{\overline{q}}^{(1)}(x_1, Q^2) f_q^{(1)}(x_2, Q^2)\right]}{\sum_q Q_q^4 \left[f_q^{(2)}(x_1, Q^2) f_{\overline{q}}^{(2)}(x_2, Q^2) + f_{\overline{q}}^{(2)}(x_1, Q^2) f_q^{(2)}(x_2, Q^2)\right]} \tag{2.35}$$

3 Maschinelles Lernen und tiefe neuronale Netzwerke

3.1 Einführung in Maschinelles Lernen

Das Konzept „Maschinelles Lernen“ befasst sich damit, aus Informationen, beispielsweise Messwerte, ein statistisches Modell zu entwickeln, das die Muster hinter den Lerndaten erkennt und übertragen kann. Es werden die Teilgebiete Klassifizierung und Regression unterschieden.

Klassifizierung ordnet Objekten ihre jeweilige Gruppe, auch genannt *Label* zu. Dies geschieht auf Grundlage der Eigenschaften eines Objektes, den sogenannten *Features*. In dieser Arbeit wird **Regression** behandelt, wobei hier anstatt einer diskreten Zuordnung eine reelle Zahl ausgegeben wird. Wird eine Funktion $f : \mathbb{R}^n \to \mathbb{R}, \mathbf{x} \mapsto f(\mathbf{x})$ betrachtet, werden die Einträge des Vektors $\mathbf{x}$ als Features und der Funktionswert $f(\mathbf{x}) \in \mathbb{R}$ als Label bezeichnet. Am Beispiel des hadronischen Diphoton Prozess *Gl. 2.34*, kann man x_1, x_2, η als Features und den zugehörigen differentiellen Wirkungsquerschnitt als Label identifizieren. Im **überwachten** Lernen sind alle Trainingsdaten mit Labels versehen, sodass die Vorhersage des Modells mit dem wahren Ergebnis abgeglichen werden und das Netz seine Parameter entsprechend anpassen kann, um eine minimale Abweichung zu erreichen.

Die konkrete Art des Machine-Learning, die in dieser Arbeit untersucht wird, ist das Deep-Learning, dessen Prinzip auf künstlichen neuronalen Netzwerken beruht. Diese neuronalen Netze sollen im folgenden verwendet werden, um einen Regressionsalgorithmus zu entwickeln, der gegebenenfalls hochdimensionale Funktionen erlernen und damit die Effizienz der numerischen Berechnung von differentiellen Wirkungsquerschnitten steigern kann.

3.2 Neuronale Netze

Eine Veranschaulichung des Konzeptes eines neuronalen Netzes ist in Abb. 3.1 gezeigt. Den Grundbaustein eines DNN, in dem die elementaren Berechnungen durchgeführt werden, stellt das Neuron dar, dessen Name durch das biologische Nervensystem inspiriert ist. Diese Neuronen, die auch *Units* genannt werden, können unterschiedlich stark aktiviert sein. Die Units sind in Schichten, genannt *Layern*, organisiert, zwischen denen die Ausgabewerte der Neuronen hin- und hertransferiert werden. Die Units des Layers l nehmen als Funktionsargumente die Aktivierung von Neuronen der Schicht $l-1$ und geben ihrerseits wieder einen reellen Wert aus. Während im ersten Layer, genannt Input-Layer, die Aktivierung der Neuronen durch den Wert der eingehenden Features gegeben ist, beherbergt die

letzte Schicht, der sogenannte Output-Layer nur noch eine Node, dessen Aktivierung die Vorhersage des Netzes darstellt.

Es wird sich im folgenden auf vollständig verbundene Feedforward-Netze beschränkt. Während sich vollständig verbunden darauf bezieht, dass ein Neuron mit allen Neuronen der vorhergehenden Schicht verbunden ist, versteht man unter Feedforward-Netzen, dass die Ausgabe von Units der Schicht $l-1$ nur Neuronen in Layer l beeinflusst.

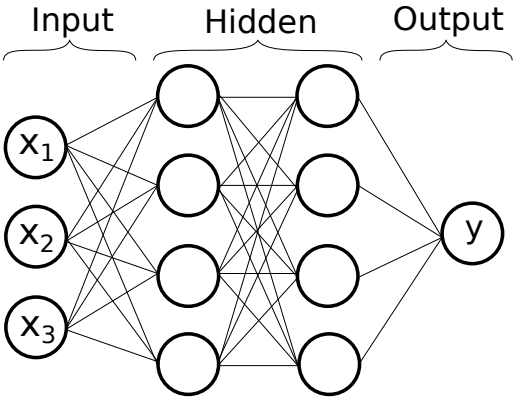


Abb. 3.1: Konzept eines mehrschichtigen Perzeptron [24], Kreise stellen Neuronen dar.

Jedes Neuron stellt zunächst eine lineare Funktion von den Ausgaben des vorhergegangen Layers $l-1$ dar, die im Vektor $\mathbf{y}_{l-1}$ zusammengefasst sind. Die Ausgabe des n-ten Neurons der Schicht l, bezeichnet mit y_l^n, wird als Skalarprodukt zwischen den Gewichten der Node $\mathbf{w}_l^n$ dargestellt, wobei zusätzlich das Bias b_l^n addiert wird. Auf diese Lineare Funktion wird anschließend eine nichtlineare Aktivierungsfunktion σ angewendet, die es dem Netz ermöglicht nichtlineare Zusammenhänge zu erlernen.

$$y_l^n = \sigma\left(\mathbf{w}_l^n \cdot \mathbf{y}_{l-1} + b_l^n\right) \tag{3.1}$$

Für den ersten Layer gilt $\mathbf{y}_0 = \mathbf{x}$, die Features entsprechen also y_0. In einem vollständig verbundenen Netz ergibt sich pro Node eine lineare Gleichung der Form *Gl. 3.1*. Insgesamt können die Rechenoperation, die in einem Layer stattfindet also als Matrixmultiplikation formuliert werden. Die Vektoren $\mathbf{w}_l^n$ werden hierbei zu den Zeilen der Matrix $\mathbf{W}_l$, die b_l^n in Vektoren zusammengefasst.

$$\mathbf{y}_l = \sigma\left(\mathbf{W}_l \cdot \mathbf{y}_{l-1} + \mathbf{b}_l\right) \tag{3.2}$$

Im Neuron des Output-Layers findet schließlich die Ausgabe des Funktionswertes y statt. Das Ziel ist es nun, die Abweichung des Ausgabewertes y des Netzes vom wahren Wert $\tilde{y}$ zu minimieren. Mathematisch wird die Abweichung als eine Metrik definiert und das Erlernen der freien Parameter eines künstlichen neuronalen Netzes wird damit zum Optimierungsproblem. Im Kontext von ML wird die zu minimierende Metrik, die abhängig von allen Gewichten $\mathbf{W}$ und Biases $\mathbf{b}$ ist, als Kostenfunktion oder Verlustfunktion (Loss-Funktion) bezeichnet, wobei hier eine mögliche Wahl die mittlere quadratische Abweichung ist.

$$C\left(\mathbf{W}, \mathbf{b}\right) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \tilde{y}^{(i)}\right)^2 \tag{3.3}$$

Als Kostenfunktion kann prinzipiell jede Metrik verwendet werden, die zielführend erscheint und es muss je nach Problemstellung abgewogen werden, welche Wahl die besten Ergebnisse liefert. Da die analytische Berechnung von Extremstellen der Verlustfunktion von neuronalen Netzen nicht möglich ist, wird auf *Gradient-Descent* [10] zurückgegriffen. Hierbei wird, beim Output-Layer beginnend, der Gradient der Kostenfunktion berechnet und per Kettenregel zum vorherigen Layer fortgepflanzt. Die-

ser Vorgang, für alle Units einen Gradienten zu berechnen, nennt sich *Backpropagation* und führt in der Anwendung auf die Methode des automatischen Ableitens zurück. Die Gradienten werden immer gemittelt für eine Anzahl an Trainingspunkten, genannt Batch, berechnet und auf die Gewichte angewendet, sodass man sich einem lokalen, oder auch globalen, Minimum nähern kann (Stochastic Gradient Descent, SGD [23]).

3.3 Training und Hyperparameter

Man nennt Parameter, die Programmierende vorher festlegen müssen und die nicht vom Algorithmus erlernt werden, Hyperparameter. Es werden im Weiteren die folgende Hyperparameter besprochen, wobei nicht zwischen solchen Hyperparametern, die die Architektur des Netzes bestimmen und Trainingsparametern, die das Lernverhalten bestimmen, differenziert wird:

- Anzahl der Layer und Units
- Kostenfunktion
- Aktivierungsfunktion der Neuronen
- Initialisierung der Gewichte
- Optimizer(Lernart)
- Learning-Rate(Lernrate)
- Batch-Größe
- Anzahl der Trainingsepochen
- Normalisierung

Die Architektur eines neuronalen Netzes wird durch die Anzahl an **Layer und Units** festgelegt. Tiefere neuronale Netze mit größeren Anzahlen an Neuronen sind in der Lage kompliziertere Sachverhalte genauer zu lernen, allerdings steigt die Anzahl an zu trainierenden Parametern und auszuführenden Rechnungen. Bei zu komplexen Modellen für simple Sachverhalte mit wenigen Trainingspunkten kommt es häufig zur Überanpassung, bei der sich das Modell zu sehr auf die vorliegenden Daten spezialisiert und seine Generalisierungsfähigkeit verliert.

Die **Loss-Funktion** bestimmt das Lernverhalten des Netzes maßgeblich, denn für sie werden letztendlich die Gradienten berechnet. Die in dieser Arbeit verwendeten Kostenfunktionen sind in *Tabelle A.1* definiert.

Die **Aktivierungsfunktion** bricht die Linearität des Netzes und sorgt dafür, dass dieses nichtlineare Funktionen erlernen kann. Die Form und Ableitung der Aktivierungsfunktion bestimmt den Gradienten während der Backpropagation. Speziell für Aktivierungsfunktionen mit verschwindenden Ableitungen, besonders die namensgebende Funktion ReLU [13], kann das *Dying-ReLU-Problem* auftreten. Hierbei werden durch große Gradienten die Gewichte eines Neurons so verändert, dass es, fast unabhängig von seinen Funktionswerten, nur noch Null ausgibt[1]. Da die Ableitung in diesem Definitionsbereich ebenfalls Null ist, kann sich das Neuron nicht regenerieren und trägt im Folgenden nicht mehr zum Lernprozess des Netzes bei. Die untersuchten Aktivierungsfunktionen sind in *Abb. A.1* gezeigt.

An welchem Punkt des hochdimensionalen Phasenraums der Kostenfunktion der Lernprozess beginnt, wird von der **Initialisierung** festgelegt. Die Initialisierung der Gewichte ist eng verknüpft mit der

[1] Null speziell für ReLU

verwendeten Aktivierungsfunktion. Zum Beispiel hat sich die HeNormal-Initialisierung [14] für die ReLU-Aktivierungsfunktion etabliert.

Die hier besprochenen Lernalgorithmen basieren auf Gradientenabstieg. Die konkrete Implementation des Gradient-descent, die sich gegebenenfalls an die vorliegende Situation anpasst, wird **Optimizer** genannt. Die **Learning-Rate** ist hierbei der Faktor, mit dem der Gradient skaliert wird, bevor er auf die Gewichte angewendet wird. Diese muss so gewählt werden, dass das Lernen weder in einem zu hohen lokalen Maximum zum Erliegen kommt, noch so groß ist, dass es zu Sprüngen über das Minimum kommt. Einige auf Gradientenabstieg basierende Optimizer sind in *Tabelle A.2* erläutert.

Die **Batch-Größe** beschreibt, wie viele Objekte in einem Durchgang vom neuronalen Netz behandelt werden. Große Batch-Größen dämpfen Ausreißer und beschleunigen die Trainingszeit, wobei ein Training mit kleineren Batches detailreicher und genauer sein kann.

Die Anzahl **Trainingsepochen** beschreibt, wie oft während des Lernvorgangs über die Trainingsdaten iteriert wird. Die Präzision eines neuronalen Netzes konvergiert idealerweise, daher kann eine Abbruchbedingung als minimale Verbesserung zwischen Epochen festgelegt werden.

Hat man Features, deren numerische Reichweite stark auseinandergeht, kann es sich lohnen die Eingabewert zu **normalisieren**. Das bedeutet, alle Features auf ein festgelegtes Intervall, zum Beispiel $I = [1, 0]$ anzupassen. So wird verhindert, dass einem Feature mit großem numerischen Wert zu viel Bedeutung zugeordnet wird.

Das Finden der besten Hyperparameter ist ein weiteres Optimierungsproblem, das abgesehen von der Suche der besten Gewichte gelöst werden muss. Für die Methoden des Grid- oder Random-Search wird ein Gitter an Hyperparametern festgelegt. Anschließend werden im ersten Fall alle und im zweiten lediglich zufällige Gitterpunkte trainiert und evaluiert. Fortgeschrittenere Methoden der Hyperparameteroptimierung wie Bayesian-Search [17] oder Hyperband [21], sollen in kürzerer Zeit bessere Parameter finden. Die Hyperparameteroptimierung kann bei vielen Parametern oder langen Lerndauern sehr aufwändig sein.

Implementierung mit Keras und Tensorflow Die Implementierung des ML-Algorithmus ist in dieser Arbeit mit der Open-Source Python-Bibliothek TensorFlow und Keras [5] umgesetzt. Keras fungiert hierbei als eine high-level API für TensorFlow. Das Erstellen eines Netzes wird mit den Modulen vereinfacht und sowohl Loss-Funktion, Optimizer als auch Initialisierungen sind bereits implementiert. Es können sowohl vorgefertigte Layer angepasst werden, als auch Layer und Trainingsroutine selbst schreiben, wobei die vorgefertigten Layer über einige Methoden verfügen, die den Umgang mit dem Netz komfortabler machen und das Speichern und Laden vereinfachen.

3.4 Transfer-Learning

Um die hohen Zeit- und Rechenkosten des Trainings zu verringern, können bereits trainierte Modelle an ein neues Problem anpasst werden. Außerdem kann mit dem sogenannten **Transfer-Learning**, die Menge an Daten, die benötigt wird, um ein brauchbares Modell zu erhalten, signifikant verringert werden.

Die Grundidee des Transfer-Learning besteht darin, dass der Algorithmus sein bereits erlerntes statistisches Modell auf eine andere Situation überträgt und gegebenenfalls nur noch die numerischen Ausgaben anpassen muss. Es ist beobachtet worden [22], dass Transfer-Learning die folgenden Vorteile bringt:

- Höherer Start, höhere Asymptote und höhere Steigung der Lernkurve
- signifikant weniger Messwerte benötigt, um brauchbare Ergebnisse zu erreichen

Konkret wird im Laufe dieser Arbeit das Transfer-Learning verwendet, um den differentiellen Wirkungsquerschnitt, berechnet mit einem PDF-Set, auf selbige, berechnet mit einem anderen PDF-Set, zu übertragen. Dabei wird von beiden angesprochenen Aspekten profitiert, da einerseits die Trainingszeit reduziert werden kann, als auch die Zeit zur Datengeneration verkürzt wird.

Transfer-Learning folgt üblicherweise dem Ablauf:

- Zunächst wird ein sogenanntes Source-Model an einer Source-Datenmenge bis zur Konvergenz trainiert.
- Eine kleinere Datenmenge an Zielwerten wird erstellt.
- Die oberste oder einige der oberen Schichten (sprich der Output-Layer und wenige darunterliegende Layer) werden entfernt.
- Die Gewichte der restlichen Layer werden zunächst fixiert, um diese nicht durch große Gradienten zu zerstören.
- Die entfernten Schichten werden mit neuen, trainierbaren Neuronen ersetzt.
- Schließlich wird das neue Modell an der kleinern Datenmenge trainiert.
- Als optionaler Letzter Schitt wird das sogenannte *Fine-Tuning*(FT) eingesetzt. Bei diesem werden die fixierten Gewichte wieder gelöst.

3.5 Monte-Carlo-Integration

Monte-Carlo-Integration unterscheidet sich von anderen numerischen Integrationsmethoden vor allem dadurch, dass die Konvergenz der Integration keine Abhängigkeit von der Dimensionalität des Integrals aufweist. Monte-Carlo-Methoden konvergieren hierbei immer mit $\propto \frac{1}{\sqrt{N}}$, wobei N die Anzahl der ausgewerteten Phasenraumpunkte ist. Es wird hierbei Gebrauch vom Gesetz der Großen Zahlen gemacht und die Integrale mittels Wahrscheinlichkeitstheorie gelöst.

Betrachte eine Funktion $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}, \mathbf{x} \mapsto f(\mathbf{x})$ und definiere ihren Erwartungswert $\langle f(\mathbf{X}) \rangle$ auf Ω, wobei $\mathbf{X}$ uniform auf Ω gezogen wird.

$$\langle f(\mathbf{X}) \rangle = \langle f \rangle = \frac{1}{\|\Omega\|} \int_\Omega f(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{3.4}$$

Es wird nun das Gesetz der Großen Zahlen angewendet und somit einen Schätzer für den Erwartungswert von f (*Gl. 3.5*) gefunden.

$$\langle \tilde{f} \rangle = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) \quad \text{mit} \quad \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) = \langle f \rangle \tag{3.5}$$

Da der Erwartungswert nicht exakt berechnet werden kann, weil es nicht möglich ist f an unendlich vielen Punkten zu evaluieren, wird genutzt, dass der Schätzer gegen den Erwartungswert konvergiert und $\langle \tilde{f} \rangle \approx \langle f \rangle$ genähert. Die Geschwindigkeit der Konvergenz der Näherung kann erhöht werden, indem in *Gl. 3.4* eine produktive Eins in Form einer Wahrscheinlichkeitsdichte $\rho : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}_{\geq 0}, x \mapsto \rho(x)$ mit $\int_\Omega \rho(x)\mathrm{d}x = 1$ eingeführt wird (*Gl. 3.6*)

$$I = \int_\Omega f(\mathbf{x})\mathrm{d}x = \int_\Omega \frac{f(\mathbf{x})}{\rho(\mathbf{x})}\rho(\mathbf{x})\mathrm{d}\mathbf{x} = \left\langle \left(\frac{f}{\rho}\right) \right\rangle_\rho \tag{3.6}$$

Dabei stellt $\left\langle \left(\frac{f}{\rho}\right) \right\rangle_\rho$ den Erwartungswert von $\frac{f}{\rho}$ unter der Bedingung dar, dass die $\mathbf{x}_i$ nach der Wahrscheinlichkeitsverteilung $\rho(\mathbf{x})$ gezogen werden. Der Schätzer ergibt sich dann zu *Gl. 3.7*.

$$I \approx \left\langle \left(\frac{\tilde{f}}{\rho}\right) \right\rangle_\rho = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)} \tag{3.7}$$

Die Konvergenz der MC-Integration ist am Schnellsten, wenn sich die Varianz von *Gl. 3.7* minimiert. Die Varianz ist gegeben durch *Gl. 3.8*.

$$\mathrm{Var}\left(\frac{f}{\rho}\right) = \left\langle \left(\frac{f}{\rho} - \left\langle \frac{f}{\rho} \right\rangle\right)^2 \right\rangle = \left\langle \left(\frac{f}{\rho}\right)^2 \right\rangle - \left\langle \frac{f}{\rho} \right\rangle^2 \approx \frac{1}{N} \sum_{i=1}^{N} \left(\frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)}\right)^2 - I^2 \tag{3.8}$$

Die Varianz minimiert sich also, wenn jeder Summand aus *Gl. 3.8* gleich groß ist. Der Vorgang die Wahrscheinlichkeitsdichte ρ an die Form unserer zu integrierenden Funktion f anzupassen, nennt man **Importance Sampling** (IS). Hierbei zieht man absichtlich die $\mathbf{x}_i$ mit höheren Wahrscheinlichkeiten aus den Regionen, in denen auch f den größten Beitrag liefert. Die Unsicherheit auf die Integration ergibt sich aus der Standardabweichung des Mittelwerts, sprich:

$$\sigma_{\left\langle \frac{f}{\rho} \right\rangle} = \frac{1}{\sqrt{N-1}} \cdot \sqrt{\mathrm{Var}\left(\frac{f}{\rho}\right)} \,. \tag{3.9}$$

Es werden im Folgenden simple Monte-Carlo-Methoden und das Importance Sampling verwendet, um aus den differentiellen Wirkungsquerschnitten die totalen Wirkungsquerschnitte zu erhalten.

4 Anwendung von Maschinellem Lernen auf den Diphoton Prozess

4.1 Diphoton-Prozess auf Parton-Ebene

Zunächst werden neuronale Netze zur Regression der eindimensionalen differentiellen Wirkungsquerschnitte aus *Abschnitt 2.2* benutzt. Dabei werden geeignete Wertebereiche gewählt, sprich $\theta \in [\epsilon, \pi - \epsilon]$ bzw. $\eta \in [-3, 3]$, wobei das Verhalten des DNN für verschiedene ϵ evaluiert wird. Kleine ϵ sind hierbei interessant, weil der Wirkungsquerschnitt für $\epsilon = 0$ am Rand des Intervalls divergiert und somit das Verhalten von neuronalen Netzen an Polstellen untersucht werden kann. Die quantitativen Werte im nächsten Abschnitt sind für einen $d\bar{d} \to \gamma\gamma$-Prozess mit einer Schwerpunktsenergie von $\sqrt{s} = 200$ GeV berechnet. Die Trainingspunkte werden zufällig nach einer, wenn nicht explizit anders angegebenen, uniformen Verteilung generiert. Es werden im Weiteren nur Netze mit der gleichen Anzahl an Neuronen in jedem Layer verwendet.

Modell für $\frac{\mathrm{d}\sigma}{\mathrm{d}\eta}$: Der differentielle Wirkungsquerschnitt in Abhängigkeit der Pseudo-Rapidität ist eine gutartige Funktion ohne Pol- oder Sprungstellen. *Gl. 2.20* reduziert sich von der Form auf eine $\tanh^2$-Funktion, deren Wertebereich sich über $[0, 1)$ erstreckt und damit schon von vornherein auf einen geeigneten Wertebereich normiert ist. Der Vorfaktor wird mit einer Skalierung der Funktionswerte behandelt, auf die später weiter eingegangen wird. Für diese einfache Aufgabe werden die Hyperparameter wie in *Tabelle 4.1* gewählt und keine weiteren Optimierungen, wie für die folgenden Modelle, durchgeführt. Es werden in Zukunft standardmäßig der Kernel-Initializer HeNormal verwendet und das Bias auf Null zu initialisieren. Das Training an sich wird für alle folgenden Modelle von den, in

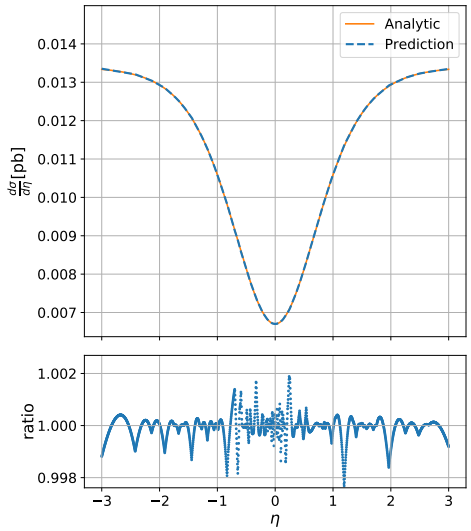
Hyperparameter	Wert
Anzahl Layer	2
Anzahl Units	64
Loss-Funktion	MAE
Optimizer	Adam
Aktivierungsfunktion	ReLU
Kernel-Initializer	HeNormal
Bias-Initializer	Zeros
Learning-Rate	0.005
Batch-Größe	128
Max. Epochen	300
Anzahl Trainingspunkte	10000

Tabelle 4.1: Hyperparameter des Modells $\frac{\mathrm{d}\sigma}{d\eta}$

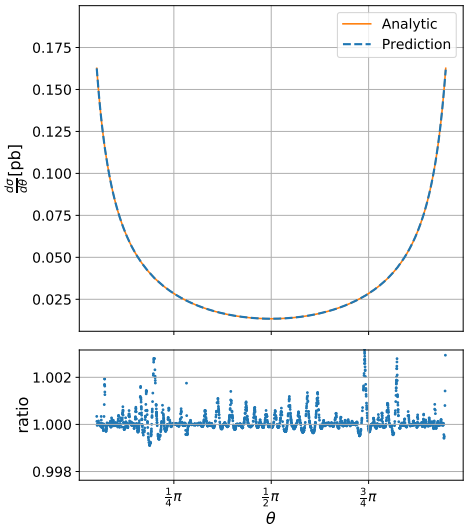
Keras implementierbaren, `Callbacks` geregelt:

- `LearningRateScheduler`: Ein Ablaufplan wird festgelegt, der für jede Epoche die zu verwendende Learning-Rate bestimmt.
- `ReduceLROnPlateau`: Erzielt das Training bezogen auf eine bestimmte Metrik nicht einen Mindestfortschritt, wird die Learning-Rate reduziert.
- `EarlyStopping`: Erzielt das Training bezogen auf eine bestimmte Metrik für eine gewisse Epochenanzahl keinen Mindestfortschritt, wird das Training gestoppt.

Die Wahl der genauen Konfiguration der Callbacks ist in *Tabelle A.4* festgehalten. Die gelernte Funktion im Vergleich mit den analytischen Werten ist in *Abb. 4.1 (a)* gezeigt. Die Werte überlagern sich gut, sodass auf den ersten Blick kein Unterschied festzustellen ist. Bei Betrachtung des Verhältnisses wird deutlich, dass sich der Unterschied auf ca. 0.1% beläuft.



(a) Vergleich DNN - analytische Werte für $\frac{\mathrm{d}\sigma}{\mathrm{d}\eta}$



(b) Vergleich DNN - analytische Werte für $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ auf $\theta \in [\epsilon, \pi - \epsilon]$ für $\epsilon = 0.163$

Abb. 4.1: Vorhersage der neuronalen Netze für den Wirkungsquerschnitt des partonischen Diphoton-Prozess

Modell für $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$: Der Wirkungsquerschnitt in Abhängigkeit von θ unterscheidet sich vom vorherigen funktionalen Zusammenhang durch seine Polstellen. Da numerisch nicht mit Polstellen umgegangen werden kann, muss der Trainingsbereich auf $[\epsilon, \pi - \epsilon]$ eingeschränkt werden. Aus physikalischer Sicht ist das legitim, da die Polstellen im Strahlengang des Speicherrings liegen und damit nicht messbar sind. Detektoren, wie ATLAS [7] und CMS [9], können Photonen mit Pseudo-Rapiditäten bis zu ca. $|\eta| \leq 2.5$ messen, was einem $\epsilon \approx 0.163$ entspricht. Durch Normierung der Labels auf das Intervall $[-1, 1]$, kann dem Modell der Umgang mit den Polstellen erleichtert werden. Da gute Modelle hier nicht mehr trivial gefunden werden können, wird auf eine automatische, zufällige Suche zurückgegriffen (Random-Search, siehe Abschnitt 3.3). Die Such-Parameter mit Ergebnis sind in *Tabelle A.5* festgehalten.

Es fällt auf, dass die Architektur des Modells um ein vielfaches komplizierter ist, als die des Modells für $\frac{\mathrm{d}\sigma}{\mathrm{d}\eta}$. Einerseits ist dies aufgrund der Polstellen zu erwarten und andererseits darin begründet, dass

komplexere Modelle Probleme generell präziser lösen können und daher aus der Suche hervorgehen. Die Performance des Modells ist in *Abb. 4.1, (b)* gezeigt. Die Präzision ist trotz der komplizierteren Funktion mit *Abb. 4.1, (a)* vergleichbar. Durch den Random-Search konnte ein passendes Modell gefunden werden.

Es werden zwei weitere Modelle mit den gleichen Hyperparametern auf einem Intervall $[\epsilon', \pi - \epsilon']$ mit $\epsilon' < \epsilon$ trainiert, wobei die Trainingsdaten des einen Modells nach der Verteilung in *Abb. 4.2* (Importance Sampling) generiert sind, um die Leistung der drei Modelle zu vergleichen.

Die Vergleiche sind in Abb. 4.3 für $\epsilon' = 0.01$ gezeigt. Wie zu erwarten weicht das Modell, das auf dem Intervall $[\epsilon, \pi - \epsilon]$ trainiert wurde, außerhalb seines Trainingsintervalls stark von der analytischen Funktion ab. Es trifft die Steigung der analytische Funktion am Startpunkt der Extrapolation, höhere Ableitungen vernachlässigt das Modell jedoch. Aufgrund ihres größeren Trainingsintervalls, zeigen die beiden anderen Modelle akzeptable Leistung auch nahe an den Polstellen. In Abb. 4.3 (a) kann man an einigen Stellen am Verhältnis sehen, dass das mit IS-generierten Trainingsdaten trainierte Netz an den Polstellen besser und im Zentrum schlechter angepasst ist. In Abb. 4.3 (c) ist der Definitionsbereich näher an die Polstelle gelegt, um den Verbesserung des IS-gesampelten Modells besser aufzulösen. Anhand von Abb. 4.3 (d) wird deutlich, dass der Effekt noch größer ist, wenn keine Überfülle an Trainingsdaten vorhanden ist. Ist also der Umfang an verfügbaren Trainingsdaten klein oder wenig Rechenleistung vorhanden, kann auf Importance Sampling zurückgegriffen werden, um Modelle mit Fokus auf wichtige Bereiche zu trainieren. Dabei ist zu beachten, dass hier ein Kompromiss zwischen Verlässlichkeit nahe der Polstelle und Verlässlichkeit im Zentrum des Definitionsbereiches eingegangen wird.

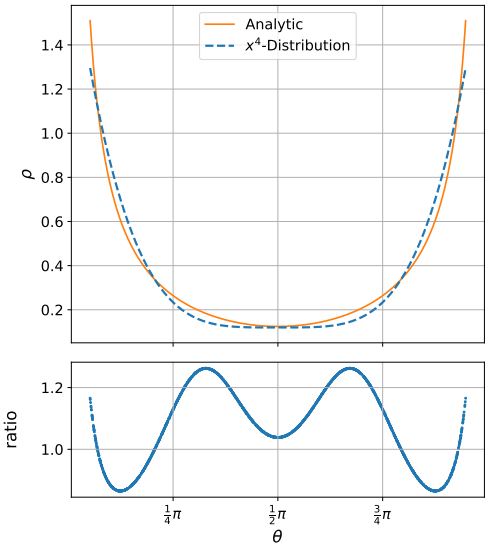
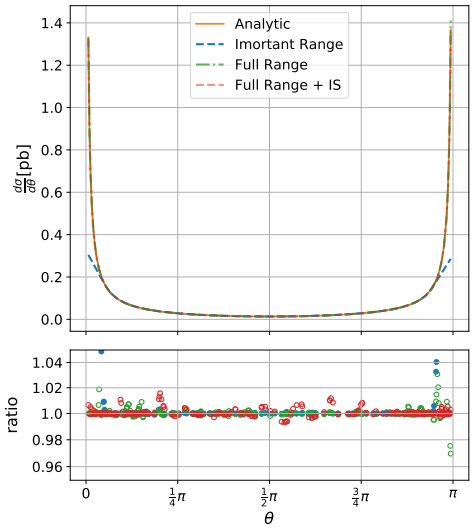
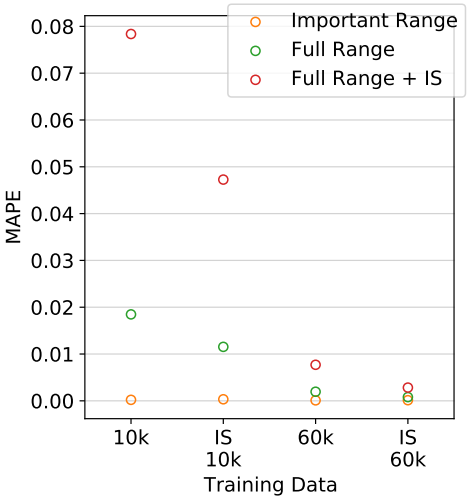


Abb. 4.2: Importance Sampling, das analytische Funktion annähert (siehe Gl. A.3)

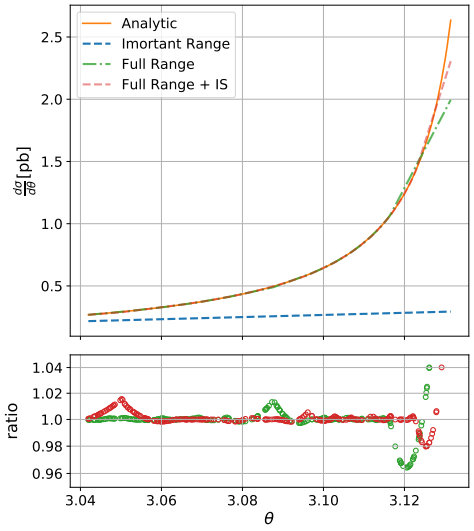
In Abb. 4.3 (b) ist noch einmal der MAPE (Mean-Absolute-Percentage-Error) der verschiedenen Modelle für verschiedene Testdatensets gezeigt. Hier wird erneut deutlich, dass das Importance Sampling vor allem nützlich ist, wenn Bereiche mit hohen Funktionswerten besonders wichtig sind. Bei der Berechnung des totalen Wirkungsquerschnittes aus dem differentiellen Wirkungsquerschnitt ist genau dies der Fall.



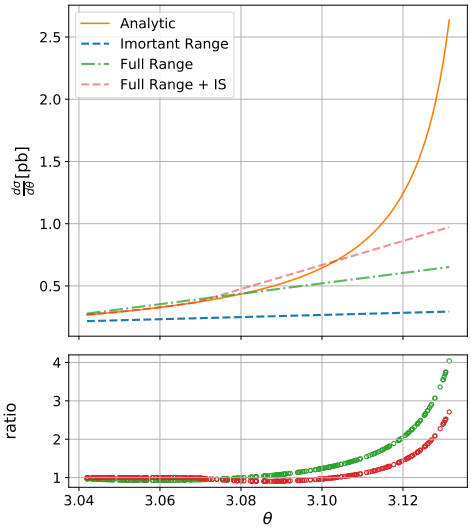
(a) $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ auf $\theta \in [\epsilon', \pi - \epsilon']$ für $\epsilon' = 0.01$



(b) MAPE für verschiedene Test-Datensets



(c) Modelle mit 60000 Traningspunkten



(d) Modelle mit 10000 Trainingspunkten

Abb. 4.3: Vergleich der Performance auf unterschiedlichen Intervallen von verschiedenen Theta-Modellen, die mit verschiedenen Datenmengen trainiert wurden.

4.2 Diphoton-Prozess auf Hadron-Ebene

4.2.1 Allgemeines und Phasenraumschnitte

Im Gegensatz zu den bisher diskutierten Prozessen, ist die Reaktion $pp \to \gamma\gamma$ beobachtbar und messbar. Der Wirkungsquerschnitt soll am Beispiel einer Messung im ATLAS-Detektor behandelt werden. Es werden einige Beschränkungen auf die Events angewendet, die sich an einer realen Messung orientieren und durch die Detektorgeometrie motiviert sind, da dieser nicht den gesamten Raumwinkelbereich abdecken kann. Die generierten Phasenraumpunkte werden selektiert und der Algorithmus nur an solchen Messpunkten trainiert, die auch praktisch detektierbar wären. Die verwendeten Selektionen sind angelehnt an ATLAS Collaboration [8] und aufgelistet in Tabelle 4.3. Dabei sind γ und γ' die Bezeichnungen für die beiden Photonen und $p_{T,\gamma} = p_{T,\gamma'} = p_T$ beschreibt den Impuls der produzierten Photonen transversal zum Strahlengang. Gl. 2.34 ist, aufgrund der Abhängigkeit der PDF, für große x_1, x_2 extrem klein und fällt zusätzlich stark ab, es werden daher außerdem Ereignisse mit $x > 0.7$ herausgenommen. Dieser Schnitt vernachlässigt den Phasenraumbereich mit extrem kleinen Labels und erleichtert dem DNN den Lernprozess.

Da sich das Laborsystem vom Schwerpunktsystem der kollidierenden Quarks unterscheidet, sichergestellt werden, dass beide Photonen die η-Selektion erfüllen (siehe Abb. 2.2). Werden η_γ und $\eta_{\gamma'}$ in entgegengesetzte Richtungen gemessen, berechnen sich diese aus η' der Photonen im Schwerpunktsystem der Quarks nach:

$$\eta_\gamma = \eta' - \frac{1}{2}\ln\left(\frac{x_2}{x_1}\right) \quad \text{sowie} \quad \eta_{\gamma'} = \eta' + \frac{1}{2}\ln\left(\frac{x_2}{x_1}\right) . \tag{4.1}$$

$\eta_{\gamma'}$ kann somit in Abhängigkeit von η_γ dargestellt werden als:

$$\eta_{\gamma'} = \eta_\gamma + \frac{1}{2}\ln\left(\frac{x_2^2}{x_1^2}\right) \tag{4.2}$$

Lorentztransformation von Gl. 2.3 liefert für den Transversalimpuls:

$$p_T = \sqrt{x_1 x_2 \left[1 - \tanh^2\left(\eta_\gamma + \frac{1}{2}\ln\left(\frac{x_2}{x_1}\right)\right)\right]}E , \quad \text{wobei} \quad p = \frac{1}{2}\sqrt{s} = \sqrt{x_1 x_2}E . \tag{4.3}$$

Für die Berechnung von Gl. 2.34 verwenden wird das PDF-Set CT14nnlo von LHAPDF [4] verwendet, wobei positive Beiträge von Charm- und Strange-Quark berücksichtigt und Top- und Bottom-Quark vernachlässigt werden. Die Strahlenergie beträgt $E = 6500\text{GeV}$.

Typ	Selektion
Photon-Energie	$\lvert p_T\rvert > 40$ GeV
Photon Winkel	$\lvert\eta_{\gamma,\gamma'}\rvert < 2.37$ ohne $1.37 < \lvert\eta_{\gamma,\gamma'}\rvert < 1.52$
Impulsbruchteil	$x_{1,2} < 0.7$

Tabelle 4.2: Event-Selektion für den hadronischen Diphoton-Prozess in Anlehnung an Messung von ATLAS [8]

4.2.2 Schwierigkeiten und Lösungsansätze

Schwierigkeiten: Im Vergleich zu den vorhergegangenen Prozessen hat sich nun die Dimensionalität des Problems, sowie die Gutartigkeit der Funktion verändert. Die Partondichtefunktionen, die den dreidimensionalen Wirkungsquerschnitt bestimmen, fallen exponentiell mit ihren Impulsbruchteilen x_1 und x_2 ab und besitzen Polstellen für $x \to 0$[1]. Die Labels variieren daher von 10^3 pb bis zu 10^{-20} pb. Wie sich herausgestellt hat, lässt sich diese Form und exponentielle Sensitivität für Veränderungen in x mit der linearen Natur der Operationen eines DNN nur begrenzt reproduzieren. Außerdem haben die stark variierenden Labels zur Folge, dass eine gängige Loss-Funktion wie der Mean-Squared- oder Mean-Absolute-Error lediglich Punkte mit hohen Wirkungsquerschnitten berücksichtigen und damit die Vorhersagen schon ab einem kleinen x unbrauchbar werden würden.

Erschwerend kommt hinzu, dass die quantitativen Werte des Wirkungsquerschnittes in standardmäßig verwendeten Einheiten, sprich $1/\text{GeV}^2$ und pb, viel kleiner als eins sind. Dies führt einerseits dazu, dass die Loss-Funktion *Mean-Squared-Logarithmic-Error* (MSLE), die Phasenraumbereiche mit kleinen Funktionswerten nicht vernachlässigen würde, nicht anwendbar ist und verstärkt andererseits das Dying-ReLU-Problem (siehe Abschnitt 3.3). Es ist aufgefallen, dass quantitativ kleine Labels dazu führen können, dass durch starke Überschätzung der Labels direkt nach der Initialisierung des Netzes, viele Neuronen bereits nach den ersten Batches absterben. Es ist entscheidend, dies zu verhindern, um brauchbare Ergebnisse zu erhalten. Während den Untersuchungen wurde beobachtet, dass es sehr mühsam ist, eine gute Initialisierung zu finden, die das Problem verhindert. Zusammenfassend sind die zu lösenden Probleme:

- Starke Variation der Labels inklusive Polstellen, stellt nur schwierig zu erlernende Form für das neuronale Netz dar und erfordert eine nicht-triviale Kostenfunktion, die nicht nur exklusiv den Bereich um die Polstelle berücksichtigt.

- Quantitativ kleine Labels verstärken das Dying-ReLU Problem und fürhen dazu, dass MSLE nicht ohne Weiteres anwendbar ist.

Lösungsansätze: Um das Problem der vernachlässigten Phasenraumbereiche zu lösen, wurde die bereits erwähnte Loss-Funktion MSLE getestet (siehe *Gl. 4.4*), da es hier ausschließlich auf das Verhältnis zwischen Label und Vorhersage. Um nicht mit negativen Werten zu arbeiten, wird generell $y \to y + 1$ transformiert. Da für die vorliegenden Labels jedoch $y \ll 1$ und damit $\ln(1 + y) \approx y$ gilt, würde so lediglich ein ineffizienterer MSE implementiert werden.

$$C\left(\mathbf{M}, \mathbf{b}\right) = \frac{1}{N} \sum_{i=1}^{N} \left(\ln\left(y^{(i)}\right) - \ln\left(\tilde{y}^{(i)}\right) \right)^2 = \frac{1}{N} \sum_{i=1}^{N} \left(\ln\left(\frac{y^{(i)})}{\tilde{y}^{(i)}} \right) \right)^2 \tag{4.4}$$

Daher wird die, in den bereits diskutierten Modellen verwendete, Skalierung implementiert. Wie sich im Laufe der Arbeit mit den künstlichen neuronalen Netzen gezeigt hat, beugt die Skalierung das Dying-ReLU-Problem vor und lässt Gebrauch des MSLE machen.

[1] wobei die PDFs numerisch ab $x_{\min} \approx 10^{-9}$ eingefroren werden.

Andere Versuche um mit dem Dying-ReLU-Problem umzugehen sind die Verwendung von Aktivierungsfunktionen mit nicht-verschwindender Ableitung wie `Leaky-ReLU` oder `ELU` (siehe *Abb. A.1*), die den Neuronen ermöglichen soll, sich zu regenerieren, oder die Übergabe eines *Clipvalue/Clipnorm*-Parameters an den Optimizer, der den Gradienten reguliert.

Die Skalierung wird im Folgenden mit einem Transformator-Objekt realisiert, der die vorliegenden Labels nach $\tilde{y} \to \tilde{y}/\tilde{y}_{min}$ anpasst, wobei $\tilde{y}_{min}$ das kleinste Label im Trainigsdatensatz bezeichnet. Der Transformator wird mit seinen Variablen abgespeichert und kann bei Bedarf wieder initialisiert werden, um die Ausgaben des DNN auf reale Werte zurückzuskalieren. Da bereits ein Transformator verwendet wird, kann der Logarithmus bereits vor der Loss-Funktion angewendet werden, was einerseits flexibler in der Verwendung der Loss-Funktionen und andererseits rechnerisch effizienter ist[2]. Die Label-Normalisierung aus dem vohergehenden Modell wurde praktisch auch im Transformator umgesetzt.

Durch die Anwendung des Logarithmus ist nun sichergestellt, dass lediglich das Verhältnis zwischen den rücktransformierten Labels und Vorhersagen für das Training relevant ist und nicht nur ein Phasenraumbereich ausschließlich gelernt wird. Da dennoch der Bereich um die Polstelle und Phasenraumbereiche mit größeren Labels wichtiger sind, wird an dieser Stelle erneut das Importance Sampling zur Generierung der Trainingspunkte verwendet. Die Verteilungen sind in *Gl. 4.5* gezeigt.

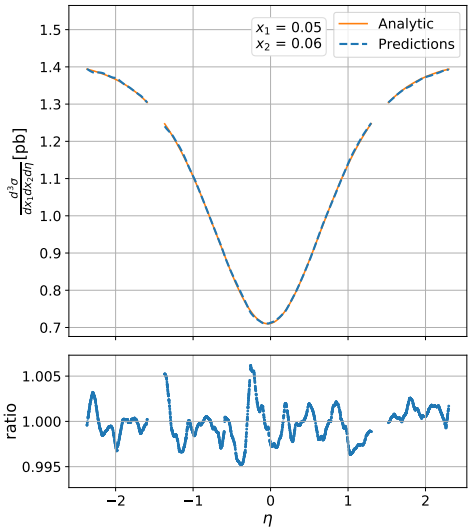
$$\begin{aligned} \rho(x) &= \frac{1}{(x+\alpha)\ln\left(\frac{x_{\max}+\alpha}{x_{\min}+\alpha}\right)} \qquad \text{mit} \qquad \alpha = 0.005 \\ \rho(\eta) &= \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\eta-\eta_{\max})^2}{2\sigma^2}\right) & \text{für} \quad 0 \le \eta \le \eta_{\max} \\ \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\eta+\eta_{\max})^2}{2\sigma^2}\right) & \text{für} \quad -\eta_{\max} \le \eta < 0 \end{cases} \quad \text{mit} \quad \sigma = 1.5 \end{aligned} \tag{4.5}$$

Bei gegebenen Parametern und Nach Anwendung der Selektionen(*Tabelle 4.3*) beläuft sich die Sample-Effizienz auf bei gegebenen Parametern auf $\approx 40\%$. Die im Weiteren angegeben Zahlen an Trainingsdaten sind die generierten Punkte bevor die Schnitte angewendet wurden.

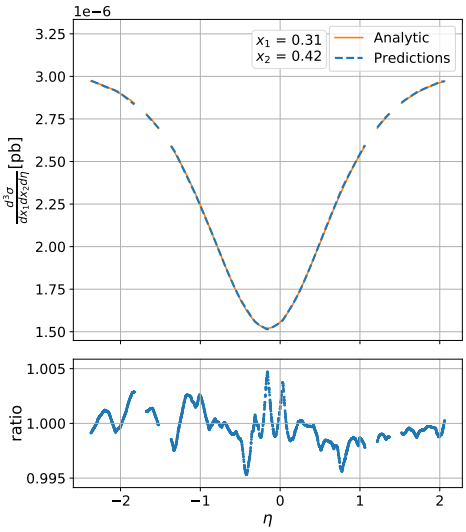
Training und Ergebnis: Zur Hyperparameteroptimierung wird erneut ein Random Search verwendet. Auch mit der intensiven Behandlung vorliegenden Situation hat nicht jede Hyperparameter-Suche akzeptable Ergebnisse geliefert. Unser Problem ist stark abhängig von den verwendeten Parametern. Die Hyperparameter einer erfolgreichen Suche sind in Tabelle A.6 aufgelistet.

In Abb. 4.4 sind Schnitte des dreidimensionalen Wirkungsquerschnittes an verschiedenen Phasenpunkten gezeigt. Im Vergleich mit Abb. **??** zeigt sich, dass die intensive Behandlung der Hyperparameter und der Daten-Transformationen Erfolge zeigt. Die maximale Abweichung beträgt an den meisten Stellen lediglich 0.5%. Für Ausnahmefälle erreicht sie bis zu $\approx 1\%$. Es lässt sich leicht der Moment erkennen, ab dem die x-Werte gefiltert wurden. Wie schon im Modell für $\frac{d\sigma}{d\theta}$, verläuft die Vorhersage des Modells linear weiter und entfernt sich somit von den analytischen Werten. Für große x wird das Modell den Wirkungsquerschnitt gegebenenfalls um Größenordnungen überschätzen. Da jedoch nur der integrierte Wirkungsquerschnitt über x_1, x_2 überhaupt messbar ist und dieser sich nicht merk-

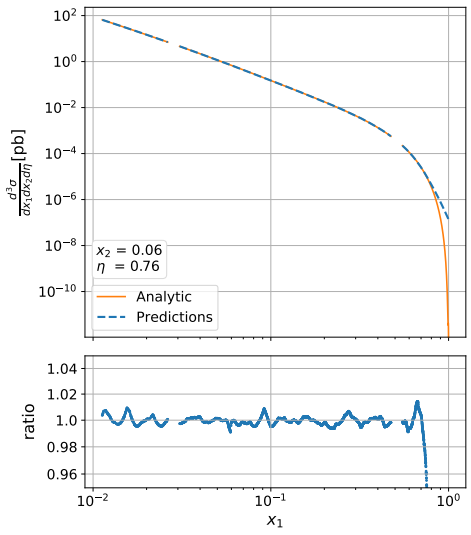
[2] als den Logarithmus einzeln für jeden Batch zu berechnen



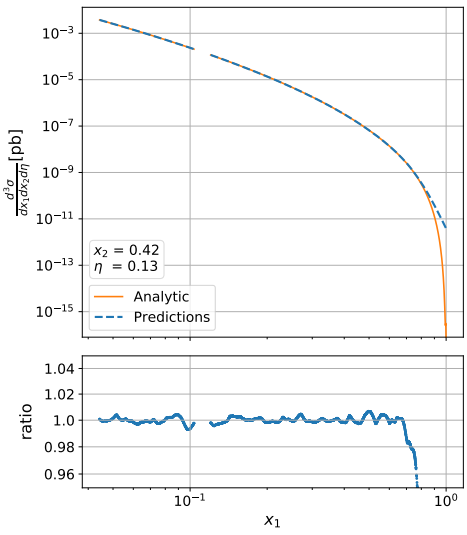
(a) Schnitt in η, $x_1 \approx x_2$



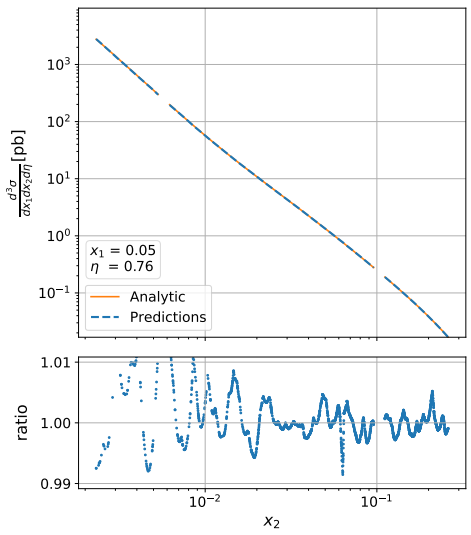
(b) Schnitt in η, $x_1 \neq x_2$



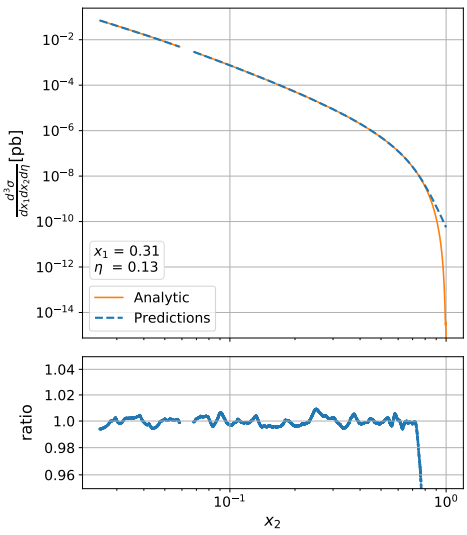
(c) Schnitt in x_1, x_2 klein



(d) Schnitt in x_1, x_2 groß



(e) Schnitt in x_2, x_1 klein



(f) Schnitt in x_2, x_1 groß

Abb. 4.4: Vergleich DNN - analytische Werte für $\frac{\mathrm{d}\sigma}{\mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}\eta}$ für verschiedene Phasenraumbereiche. Fehlende Werte sind durch die Selektionen (Tabelle 4.3) ausgeschlossen.

lich durch diese Abweichung beeinflussen lässt, ist die große Differenz in diesem Phasenraumbereich vernachlässigbar.

4.2.3 Vergleich von Hyperparametern

Es soll nun direkt die Abhängigkeit der Performance von den Hyperparameter untersucht werden. Dafür wird die Konfiguration benutzt, die im Vorhergehenden durch den Random Search gefunden wurde und in jedem Training ein Hyperparameter variiert. Da die Anzahl an Neuronen und Layern stark korreliert ist, werden zusätzlich verschiedene Kombinationen an Layern und Units . Die Modelle werden nach dem MAPE eines Sets an Testdaten beurteilt, das genauso gesampelt ist wie die Trainingsdaten (siehe *Gl. 4.5*). Jedes Modell wird fünf mal mit zufälligen Initialisierungen an zwei Millionen gesampelten Phasenraumpunkten trainiert, um die statistische Schwankung der Güte des Modells einschätzen zu können. Die eingezeichneten Fehlerbalken (z.B. *Abb. 4.5*) sollen die Schwankung verdeutlichen und sind kein Maß dafür, welchen MAPE das Netz in der Praxis erreichen kann. Ausreißer sind aus den Messungen herausgefiltert, um die Skalierung in sinnvollem Rahmen zu halten.

Loss-Funktion: In Abb. 4.8 (a) ist der Vergleich von drei verschiedenen Kostenfunktionen gezeigt. Für Probleme mit stark variierenden Labels setzt sich der Mean-Absolute-Error durch, da dieser weniger sensitiv auf Ausreißer oder, in unserem Fall, Polstellen ist. Der Huber-Loss [16], der eine Kombination des linearen Fehlers und des quadratischen Fehlers darstellt, schlägt sich insgesamt besser als der reine quadratische Fehler, kann insgesamt jedoch nicht mit dem linearen Fehler mithalten.

Optimizer: Der Vergleich des Optimizers Abb. 4.8 (c) überrascht, da generell der Adam-Optimizer [19] erfahrungsgemäß die besten Ergebnisse zeigt und als Weiterentwicklung des RMSprop [15] gilt. RMSprop lierfert hier konstant etwas bessere Ergebnisse, wobei man jedoch beachten muss, dass bei solchen kleinen Abweichungen fünf Trainingsläuft nicht genug sind, um zu beurteilen welcher Optimizer nun besser beeignet ist. Der normale Stochastic-Gradient-Descent erzielt signifikant schlechtere Ergebnisse.

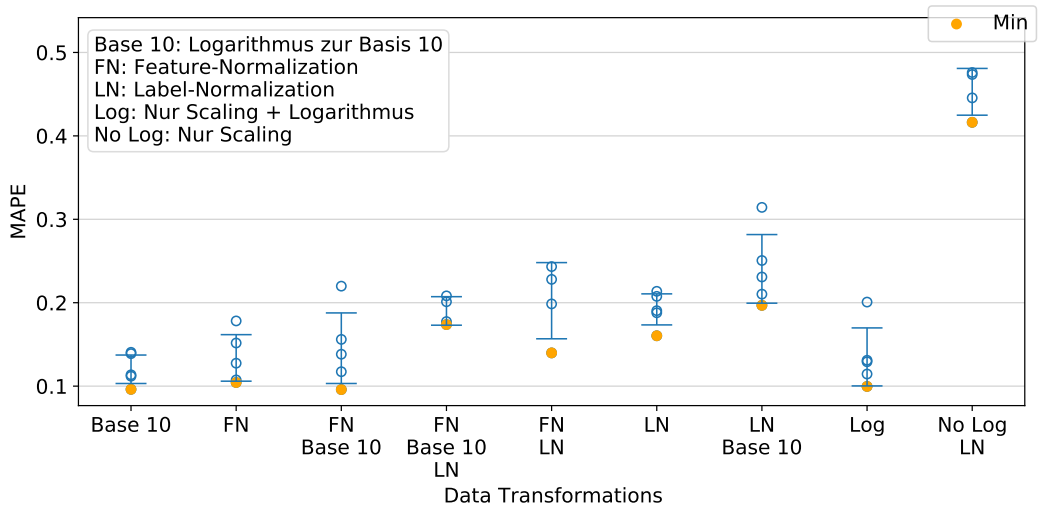


Abb. 4.5: Vergleich des MAPE von Daten-Transformationen (siehe *Tabelle A.3*) für eine Standardkonfiguration

Trainingsdaten: Die Größe des Sets an Trainingsdaten ist in Abb. 4.8 (e) verglichen. Wie erwartet nimmt der Fehler des Modells mit der Zahl an vorhandenen Trainingsdaten ab. Das gleiche gilt voraussichtlich für die Unsicherheit auf dem Ergebnis, auch wenn in unserem Fall das Modell, das mit den meisten Trainingsdaten einen Ausreißer zeigt. Man erkennt jedoch auch, dass die Performance des Modells konvergiert und mehr als vier Millionen gesampelte Daten zu keiner signifikanten Verbesserung führen.

Learning-Rate: An dem Vergleich der Learning-Rates, der in Abb. 4.8, (f) gezeigt ist, kann man ein interessantes Verhalten des Netzes erkennen. Für eine Learning-Rate von $1 \cdot 10^{-3}$ verändert sich der mittlere Fehler lediglich unwesentlich, woraus geschlossen werden kann, dass unabhängig der Initialisierung das selbe lokale Minimum gefunden wird. Die Unsicherheit der Performance wird danach mit der Learning-Rate größer, es werden höhere und tiefere lokale Minima gefunden. Bei einer zu kleinen anfänglichen Lernrate, kommt der Lernvorgang bereits in einem hohen Minimum zum Erliegen. Es kann daher gut sein kann, seine anfängliche Learning-Rate etwas größer zu initialisieren, da man so diversere lokale Minima finden kann. Beachte jedoch, dass dieser Ansatz nur in Kombination mit einem Zeitplan zur Reduzierung der Lernrate funktioniert und dass "Dying-ReLU„-Problem verstärken oder sogar auslösen kann.

Daten-Transformationen: Welche Daten-Transformationen für unser Problem funktionieren, ist in Abb. 4.5 gezeigt. An dieser Stelle soll erneut die Wichtigkeit dieser Transformationen hervorgehoben werden. Während die Literatur viel die Normalisierung oder das Reskalieren der Features behandelt (Bsp. [3, 18]), werden die Labels oft von Transformationen ausgenommen. Für spezielle Regressionsprobleme, wie es bei uns vorliegt, können diese jedoch der Schlüssel dazu sein, überhaupt konvergierende Modelle zu erhalten. Abb. 4.5 zeigt, dass verschiedene Implementationen brauchbare Ergebnisse liefern und es wichtiger ist, überhaupt die Skalierung und den Logarithmus anzuwenden. Trainingsläufe ohne Skalierung, noch Anwendung des Logarithmus sind nicht aufgeführt, da der Fehler

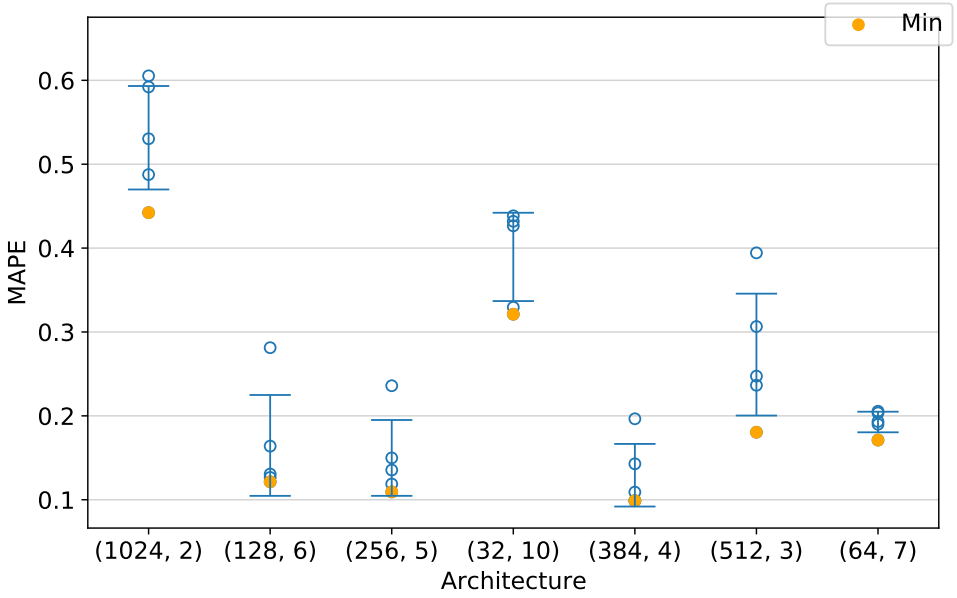


Abb. 4.6: Vergleich des MAPE für verschiedene Modell-Architekturen. x-Achsenbeschriftung in (Units, Nr of Layers)

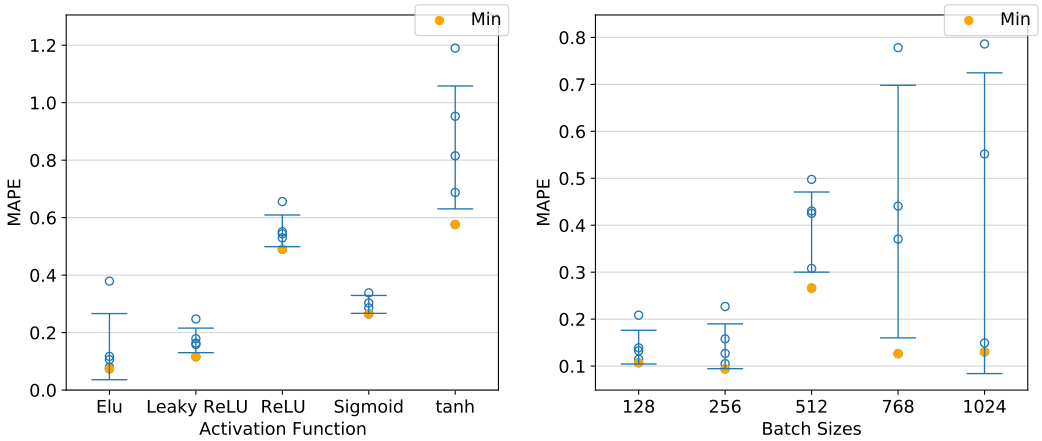
nicht vergleichbar ist.

Achitektur: Die Architektur in *Abb. 4.6* zu vergleichen ist interessant, da man sehen kann, dass eine passende Architektur zum Problem effektiver ist als die Komplexität des Modells. Das Modell mit den wenigsten zu trainierenden Parametern zeigt im Vergleich bessere Leistung als das komplexeste[3] Modell. Die Modelle (128, 6), (256,5), (384, 4) zeigen unabhängig von ihren trainierbaren Parameter sehr gute Genauigkeit. Beachte jedoch, dass die rechnerische Effizienz nicht nur von der Anzahl an freien Parametern abhängt.

Anzahl an Layer und Units pro Layer: Sowohl für die Anzahl an Layer und der Units pro Layer verläuft nach einer Kurve, die ihr Minimum bei unseren optimalen Parametern hat (siehe Abb. 4.8 (b), (d)). Der Schritt zum jeweils simpleren Modell ist klein und kann bei Bedarf einen Kompromiss zwischen Geschwindigkeit und Genauigkeit darstellen.

Aktivierungsfunktionen: Die Abwandlungen der ReLU-Funktion zeigen sehr gute Ergebnisse, einsehbar in Abb. 4.7 (a). Die gewöhnliche ReLU zeigt jedoch schlechtere Leistung. Eine plausible Erklärung hierfür liefert wiederum das "Dying ReLU"-Problem in Kombination mit unserer vergleichsweise hohen anfänglichen Lernrate.

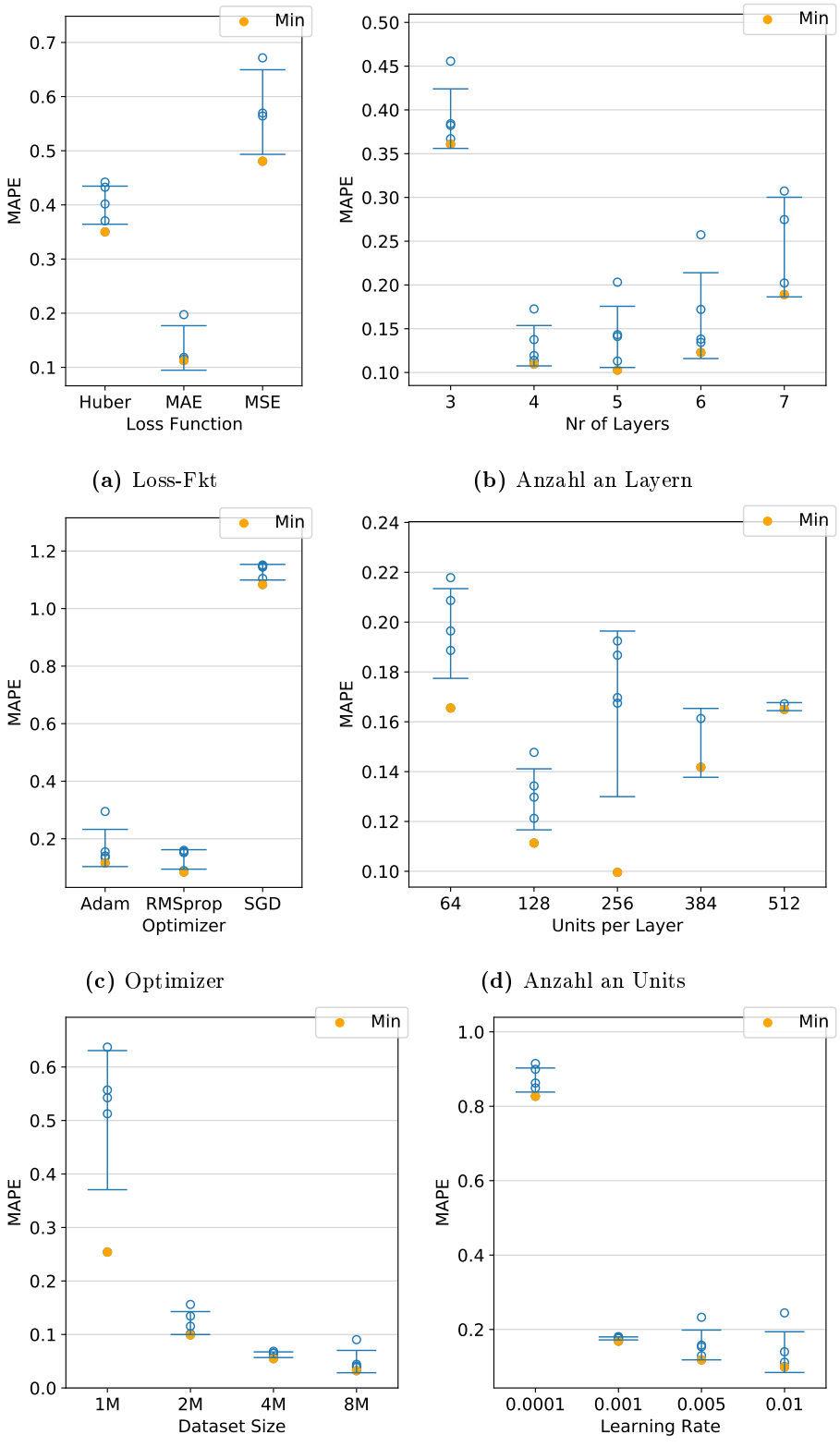
Batch-Sizes: Ein Trainingsvorgang ist bei größerer Batch-Size (siehe Abb. 4.7 (b)) mit passender Hardware zwar schneller, zeigt jedoch eindeutig größere Abweichungen. Aufgrund von zu wenigen Messdaten ist nicht klar, ob ein Modell, das mit großen Batches trainiert wird, in der gleichen Zeit, oder überhaupt, so tiefe Minima wie die kleineren erreichen können.



(a) Activation-Functions **(b)** Batch-Sizes

Abb. 4.7: Vergleich von Hyperparametern (I) am MAPE eines Testdatensets für eine Standardkonfiguration

[3] Zur Berechnung der Komplexität eines Modells siehe *Anhang A*



(a) Loss-Fkt

(b) Anzahl an Layern

(c) Optimizer

(d) Anzahl an Units

(e) Anzahl an Trainigspunkten

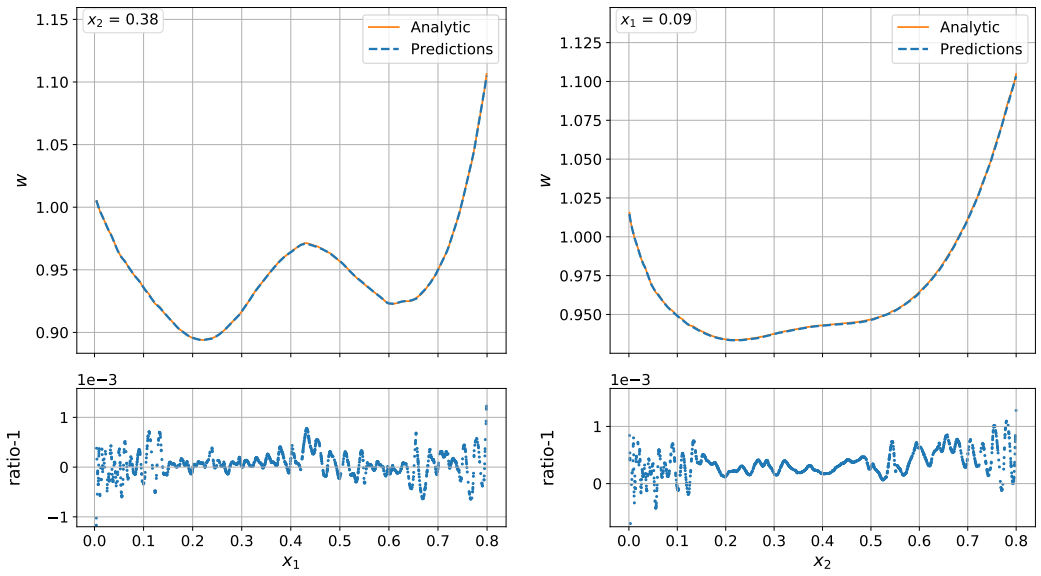
(f) Anfangs-Lernraten

Abb. 4.8: Vergleich von Hyperparametern (II) am MAPE eines Testdatensets für eine Standardkonfiguration

4.3 Umgewichtung für verschiedene PDF-Sets

Als nächstes wird ein neuronales Netz verwendet, um die Gewichte (Gl. 2.35) zwischen Wirkungsquerschnitten, die mit verschiedenen Anpassungen der PDFs berechnet wurden, zu erlernen. Konkret werden die Sets „CT14nnlo“ und „MMHT2014nnlo“ verwendet. Die Gewichte schwanken um eins und haben keine Polstellen und sind damit leichter zu modellieren als Problem im vorangegangenen Abschnitt. In Phasenraumbereichen mit großen x weichen die Sets jedoch stark voneinander ab und die Gewichte beginnen teils zu oszillieren. Angelehnt an den Phasenraumschnitt aus *Tabelle 4.3*, werden die Gewichte bis zu $x_{max} = 0.8$ trainiert, da die starken Abweichungen hier etwas später beginnen. Zusätzlich Selektieren wir Ereignisse, die unabhängig von η_γ nicht messbar wären (siehe**??**). Der Random-Search mit den besten Parametern ist in *Tabelle A.7* zu sehen.

Wie erwartet, ist die Genauigkeit des Modells sehr gut, wie in *Abb. 4.9* beobachtet werden kann. Die Abweichung beträgt generell weniger als 0.1% und ist somit kaum von den analytisch berechneten Werten zu unterscheiden. Wie die Gewichte in der Anwendung funktionieren, ist in *Abb. 4.10* dargestellt. Auch hier können nur minimale Abweichungen verzeichnet werden. Das Ratio in *Abb. 4.10 (c)* ist wie erwartet konstant, da das Gewicht nicht von η abhängt (siehe *Gl. 2.35*).

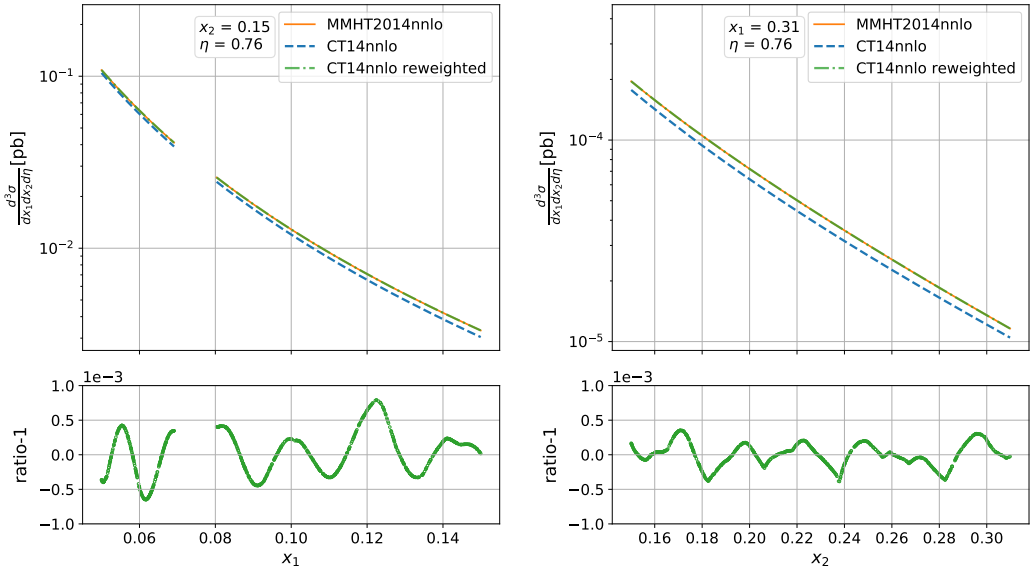


(a) Schnitt in x_1 **(b)** Schnitt in x_2

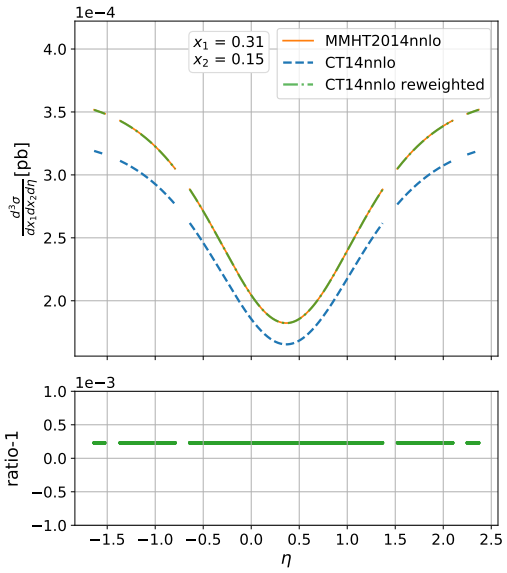
Abb. 4.9: Vergleich DNN - analytische Werte für Schnitte der Gewichte w in verschiedenen Phasenraumbereichen

Typ	Selektion
Photon-Energie	$\lvert p_T\rvert > 40$ GeV $\Rightarrow$ $\sqrt{x_1 x_2 E} > 40$ GeV
Photon Winkel	$\lvert\eta_{\gamma,\gamma'}\rvert < 2.37$ $\Rightarrow$ $\lvert\frac{1}{2}\ln(x_2^2/x_1^2)\rvert < 4.74$
Impulsbruchteil	$x_{1,2} < 0.8$

Tabelle 4.3: Event-Selektion für die Umgewichtung des hadronischen Diphoton-Prozess in Anlehnung an Messung mit ATLAS [8]



(a) Schnitt in x_1 **(b)** Schnitt in x_2



(c) Schnitt in η

Abb. 4.10: Umgewichtung von CT14nnlo auf MMHT2014nnlo mittels gelernten Weights

4.4 Transfer-Learning zwischen PDF-Sets

Eine weitere Möglichkeit, den Wirkungsquerschnitt, der mit einem anderen PDF-Set berechnet wurde, zu ermitteln, ist Transfer-Learning (siehe *Abschnitt 3.4*). Mit Transfer-Learning kann der, in den relevanten Phasenraumbereichen kleine, Unterschied ausgeglichen und so mit wenig Aufwand gute Modelle für andere PDF-Sets erhalten werden. Es kommt erneut ein Random-Search zum Einsatz, um gute Hyperparameter für den Transfer zu finden. Diese können in *Tabelle A.8* nachgelesen werden. Das Ergebnis der Suche zeigt jedoch identische Genauigkeit mit Austausch des Output-Neurons und anschließendem Training[4] mit den Trainingsparametern aus Tabelle A.6, sodass das simplere Modell weiterverwendet wird.

In *Abb. 4.11* sind Schnitte des differentiellen Wirkungsquerschnitts des besten transferierten Modells, des Modells das als Quelle gedient hat, und den analytischen Werten gezeigt. Es ist zu beobachten, dass die Genauigkeit des transferierten Modells fast identisch mit der des Source-Modells in *Abb. 4.4* ist. Weder verliert das Modell beim Transfer an Genauigkeit, noch kann eine Verbesserung der Performance festgestellt werden.

Es kann in *Tabelle 4.4* eindeutig beobachtet werden, dass sich die Menge an Lerndaten und damit auch die Trainingsdauer signifikant verringert. Es konnte mit einfachen Mitteln eine Reduktion um den Faktor vier an Trainingspunkten erreicht werden. Sowohl das Transfer-Learning, als auch das Reweighting sind also legitime Methoden um den Wirkungsquerschnitt von einem PDF-Set auf das nächste zu übertragen. Ein visueller Vergleich folgt in *Abb. 4.12*, wobei sich hier nicht erkennen lässt, welche Methode die besseren Leistungen zeigt. Interessant zu sehen ist, dass sich die Form des Ratios beider Modelle ähnelt. Der Grund hierfür liegt darin, dass das transferierte Modell vom Source-Modell abstammt und sich die Gewichte der Neuronen ähneln.

In *Tabelle 4.4* sind einige Kenndaten der Modelle gegenübergestellt. Es ergibt sich, dass das präziseste und schnellste Modell die Umgewichtung der analytisch berechneten Werte ist. Steht eine große Anzahl an Werten von differentiellen Wirkungsquerschnitten zur Verfügung, dann ist dies das optimale Modell. Ist jedoch über ein vollständiges Modell benötigt, dass nicht auf die analytische Berechnung von differentiellen Wirkungsquerschnitten angewiesen ist, schneidet das Modell „Transfer + FT“ am Besten ab. Es übertrifft die Alternative „Reweight + Source“ in den bedeutenden Kriterien. Da im zweiten Fall sowohl die Gewichte, als auch die Source-Wirkungsquerschnitten mit neuronalen Netzen berechnet werden, verdoppelt sich hier die Berechnungszeit für 10^6 Punkte (Time per Million: TPM) im Vergleich

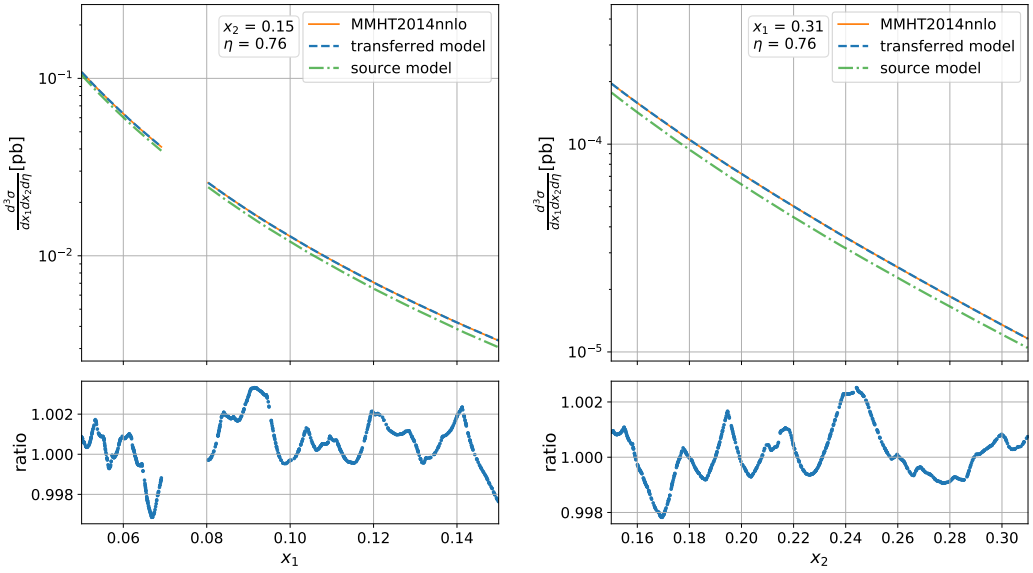
[4] einschließlich Fine-Tuning

Modell	MAPE	Training[s]	Punkte	TPM[s]
Reweight + Source	0.076	145.38	1M	30.60
Reweight + Analy.	0.017	145.38	1M	13.84
Transfer	0.204	68.61	1M	15.73
Transfer + FT	0.064	162.13	1M	15.65
Source-Model	0.229	860.30	4M	15.81

Tabelle 4.4: Vergleich von Reweight- und Transfer-Modellen TPM: Time per Million, Berechnungszeit für 10^6 Punkte. Zeiten aufgenommen auf Tabelle A.10

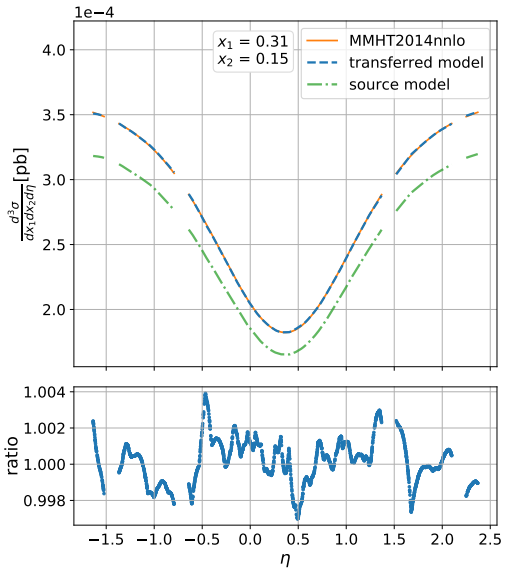
zu den restlichen Modellen. Wird das Source-Modell geeignet transformiert, passt sich das Netz gut an die neuen Daten an und der MAPE bleibt minimal. Werden die Ergebnisse des Source-Models neu gewichtet, pflanzen sich beide Ungenauigkeiten fort und die Unsicherheit steigt etwas.

Es kann geschlussfolgert werden, dass das Transfer-Learning eine generell bessere Methode für den angesprochenen Zweck ist und das Erlernen der Gewichte zwar gut funktioniert, jedoch nur nützlich ist, wenn speziell die Gewichte benötigt werden.



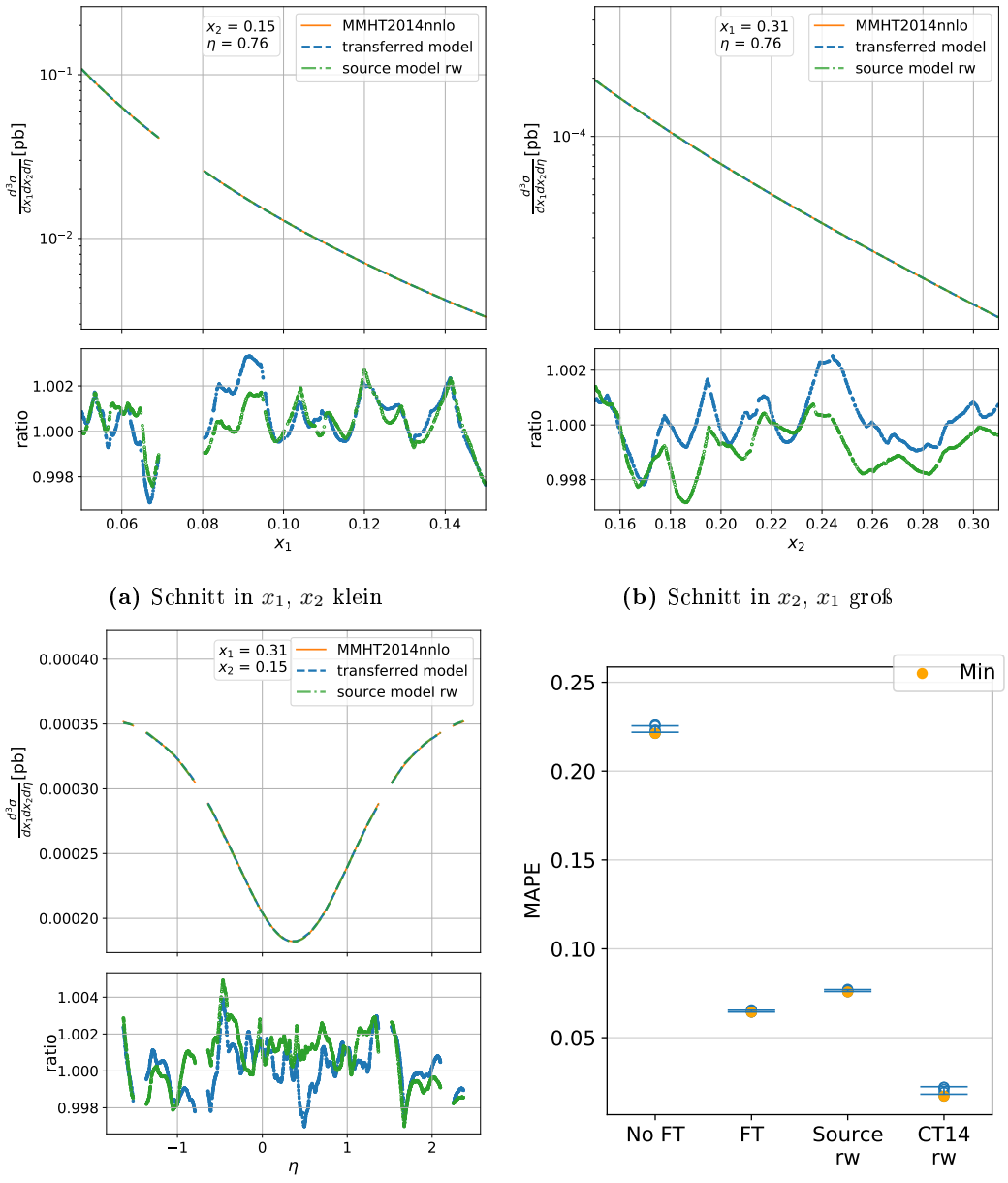
(a) Schnitt in x_1, x_2 klein

(b) Schnitt in x_2, x_1 groß



(c) Schnitt in η, $x_1 \neq x_2$

Abb. 4.11: Vergleich transferiertes Modell - Source Model für verschiedene Phasenraumbereiche



(a) Schnitt in x_1, x_2 klein

(b) Schnitt in x_2, x_1 groß

(c) Schnitt in η, $x_1 \neq x_2$

(d) Vergleich am MAPE

Abb. 4.12: Vergleich transferiertes Modell - umgewichtetes Source Model für verschiedene Phasenraumbereiche. rw: reweighting

4.5 Monte-Carlo-Integration

4.5.1 Parton-Ebene

Es werden Monte-Carlo-Methoden zur Integration von *Gl. 2.19*, *Gl. 2.20* und deren zugehörigen Machine Learning Modellen verwendet. Zur Integration von *Gl. 2.19* wird das Importance Sampling aus *Abb. 4.2* genutzt, um die Konvergenz des Integrals zu beschleunigen. Der Prozess $qq \to \gamma\gamma$ ist zwar nicht messbar, es muss dennoch ein Schnitt in η festgelegt werden, da der totale Wirkungsquerschnitt sonst divergiert. Es werden die Beschränkungen *Gl. 4.6* verwendet.

$$|\eta| \leq 2.5 \qquad \Rightarrow \qquad \theta \in [\epsilon, \pi - \epsilon] \quad \text{mit} \quad \epsilon = 0.1638 \tag{4.6}$$

Die Unsicherheit der Monte-Carlo-Integration wird aus *Gl. 3.9* bestimmt. Die Integrationen wird mit 1000 Stützstellen durchgeführt und 100 mal wiederholt. In *Tabelle 4.5* sind die erhaltenen Ergebnisse mit dem analytischen Wert verglichen. Es ist zu sehen, dass die neuronalen Netze so präzise sind,

Integrand	σ_{tot} [pb]	
analytische Stammfunktion	0.053793	$\pm$ 0
$\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ analytisch + IS	0.05382	$\pm$ 0.00006
$\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ analytisch	0.05389	$\pm$ 0.00015
$\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ ml + IS	0.05386	$\pm$ 0.00005
$\frac{\mathrm{d}\sigma}{\mathrm{d}\eta}$ analytisch	0.053796	$\pm$ 0.000034
$\frac{\mathrm{d}\sigma}{\mathrm{d}\eta}$ ml	0.053801	$\pm$ 0.000034

Tabelle 4.5: Monte-Carlo-Integration des partonischen Diphoton Prozesses

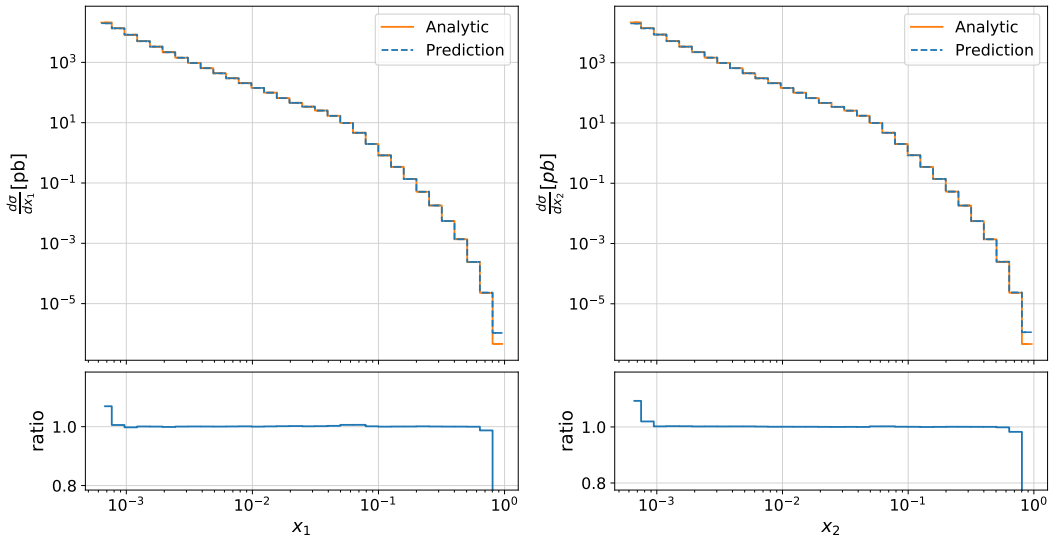
dass ihre Abweichung in der Unsicherheit der Monte-Carlo-Integration untergeht. Das sind gute Voraussetzungen für die Anwendbarkeit von neuronalen Netzen auch bei höherdimensionalen Prozessen. Das simple Importance-Sampling bringt eine signifikante Varianz-Verringerung mit sich.

4.5.2 Hadron-Ebene

Auch für den hadronischen Diphoton-Prozess wird Importance-Sampling genutzt. Für die Generation der Impulsbruchteile x wird die Verteilung aus *Gl. 4.5* verwendet. Aufgrund der Selektionen leisten Phasenraumpunkte mit kleinem η einen Größeren Beitrag zum messbaren σ_{tot}, daher werden die Pseudo-Rapiditäten aus einer Gaußverteilung um Null (*Gl. 4.7*) gezogen.

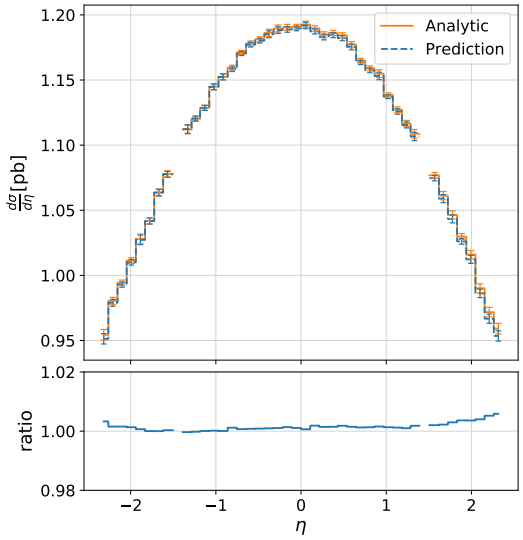
$$\rho(\eta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\eta^2}{2\sigma^2}\right) \quad \text{mit} \quad \sigma = 2 \tag{4.7}$$

Zunächst werden die Wirkungsquerschnitte über zwei Freiheitsgrade integriert und in Abhängigkeit von x_1, x_2 und η betrachtet. Dazu werden 10.000.000 Punkte generiert und der Prozess zwanzig mal wiederholt. Die Ergebnisse sind in *Abb. 4.13* dargestellt. Es ist zu beobachten, dass sich die integrierten Wirkungsquerschnitte für die analytischen Werte und die Vorhersagen des DNN an vielen Phasenpunkten überdecken. Lediglich für große x überschätzt die Vorhersage wie erwartet den eigentlichen Wert. Diese Abweichung kann jedoch vernachlässigt werden, da am Ratio von *Abb. 4.13 (c)* zu sehen



(a) Integration über x_2, η

(b) Integration über x_1, η



(c) Integration über x_1, x_2

Abb. 4.13: MC-Integrationen über zwei Freiheitsgrade. Unsicherheiten in logarithmischer Darstellung nicht sichtbar

ist, dass die analytischen Werte insgesamt unterschätzt werden. Der Grund hierfür liegt vermutlich in der Polstelle an $x = 0$. Die Selektionen nehmen viele Phasenraumpunkte um $x_1 = x_2 = 0$ mit großem Wirkungsquerschnitt heraus, die die p_T-Hürde nicht erfüllen und führen voraussichtlich zu einer geringfügigen Unterschätzung des Wirkungsquerschnittes an diesen einflussreichen Stellen.

Zur Integration über alle Freiheitsgrade werden die gleichen Daten wie im vorherigen Abschnitt verwendet. Die Ergebnisse sind in *Gl. 4.8* aufgeführt.

$$\begin{aligned} \sigma_{\text{tot}}^{\text{analytic}} &= (5.1661 \pm 0.0023)\ \text{pb} \\ \sigma_{\text{tot}}^{\text{ml}} &= (5.1588 \pm 0.0022)\ \text{pb} \end{aligned} \tag{4.8}$$

Dies entspricht einer Abweichung von 0.14%, wobei die Ergebnisse außerhalb ihrer Unsicherheiten liegen. Es kann abschließend gesagt werden, dass die Präzision des DNN sehr gut und die Näherung

erfolgreich gelungen ist.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Arbeit wurde der Diphoton Prozess als $q\bar{q} \to \gamma\gamma$ und $pp \to \gamma\gamma$ in führender Ordnung behandelt und analytische Ausdrücke für die jeweiligen differentiellen Wirkungsquerschnitte hergeleitet. An diesen Beispielen wurde anschließend die Eignung von tiefen neuronalen Netzwerken zur Näherung des Integranden überprüft. Dabei mussten verschiedene Schwierigkeiten, wie quantitativ kleine Labels, bedacht und behandelt werden. In diesem Kontext wurde die Wichtigkeit von Label-Transformationen deutlich. Im Anschluss wurden die Gewichte zwischen den Sets der Partondichtefunktionen CT14nnlo und MMHT2014nnlo erlernt und angewendet. Schließlich wurde die Möglichkeit der Anwendung von Transfer-Learning zur Umgewichtung eines Modells überprüft. Mithilfe von Monte-Carlo-Methoden wurden die analytischen und vorhergesagten differentiellen Wirkungsquerschnitte integriert.

Wie erwartet, haben die neuronalen Netze keine Probleme mit einfachen Regressionsaufgaben, wie die Wirkungsquerschnitte des Prozesses $q\bar{q} \to \gamma\gamma$ darstellen. Die Funktionswerte können mit ausgezeichneter Genauigkeit ($\approx 0.1\%$ Abweichung) und wenig Aufwand vorhergesagt werden.

Dagegen ist der differentielle Wirkungsquerschnitt des Prozesses $pp \to \gamma\gamma$ nicht trivial. Hier müssen Hürden wie das Dying-ReLU-Problem und die passende Wahl der Loss-Funktion überwunden werden. Hier kann es helfen, kaum beitragende Phasenraumbereiche zu vernachlässigen, um die Spanne an Größenordunungen, über die sich die Wirkungsquerschnitte verteilen, zu verkleinern. Letztendlich können mit Feingefühl und Erfahrung im Umgang mit neuronalen Netzen jedoch passende Modelle gefunden werden, die gute Genauigkeit ($\approx 0.5\%$ Abweichung) zeigen.

Das Erlernen der Gewichte stellt aus Sicht des neuronalen Netzes kein Schwierigkeit dar, solange ein geeigneter Phasenraumbereich gewählt wird. Das Netz kann die Funktionswerte mit exzellenter Genauigkeit ($< 0.1\%$ Abweichung) vorhersagen.

Transfer-Learning stellt sich als eine gute Möglichkeit heraus, aus einem bereits vorhandenen Modell, ein Modell für ein anders Set an PDFs zu erhalten. In puncto Berechnungsgeschwindigkeit und Genauigkeit übertrifft das Transfer-Learning die Umgewichtung eines bereits vorhandenen Source-Models.

5.2 Ausblick

Die in dieser Arbeit behandelten Methoden haben gute Ergebnisse an den einfachen Beispielen gezeigt. Als nächstes sollte nun der Test an höherdimensionalen Prozessen mit analytisch nicht mehr oder nur

aufwändig zu berechnenden Wirkungsquerschnitten folgen. Es muss noch untersucht werden, ob die neuronalen Netze ihre Genauigkeit auch in höheren Dimensionen aufrechterhalten können und wenn ja, ob dies mit einer realisierbaren Zahl an Trainingspunkten möglich ist. Anschließend muss überprüft werden, wie groß der rechentechnische Vorteil der neuronalen Netze ist. In den behandelten, einfachen Beispielen ist der analytische Weg noch um einen Faktor zwei schneller. Die in dieser Arbeit erhaltenen Ergebnisse sind jedoch gute Voraussetzungen für die Funktionstüchtigkeit im Höherdimensionalen. Zusätzlich sind neuronale Netze gut für die Arbeit mit hochdimensionalen Eingangswerten geeignet.

Auch das Transfer-Learning hat in dieser Arbeit seine Funktionalität bewiesen. Es muss jedoch nicht beim Transfer zwischen PDF-Sets bleiben. Transfer-Learning kann zwischen viel diverseren Sachverhalten eingesetzt werden. Es könnte sich lohnen, den Transfer zwischen sich ähnelnden Prozessen in der Teilchenphysik zu untersuchen. Denkbar sind hier Vorgängen die sich beispielsweise nur durch das Vorhandensein von Myonen anstatt Elektronen unterscheiden oder auch den Transfer von Leading-Order Prozessen zu höheren Ordnungen.

Abgesehen von den hier untersuchten Verwendungsmöglichkeiten gibt es noch unzählige weiter Anwendungsmöglichkeiten von Machine-Learning oder tiefen neuronalen Netzen in der Teilchenphysik. Hierunter fällt beispielsweise...

A Anhang

Berechnung der freien Parameter eines Modells: Betrachte ein DNN mit N versteckten Layern l, wobei n_l die Anzahl an Neuronen von Layer l bezeichnet, dann berechnet sich die Anzahl k an zu trainierenden Parametern nach *Gl. A.1*. Hierbei bezeichnet n_0 die Dimensionalität der Features.

$$k = \sum_{l=1}^{N} \left[(n_l \cdot n_{l-1}) + n_l \right] + (l_N + 1) \tag{A.1}$$

Gilt hierbei $n_l = n_{l+1} = n$ für $l > 0$ vereinfacht sich *Gl. A.1* zu:

$$k = n \cdot (n_0 + 1) + (N-1) \cdot n(n+1) + n + 1 = n\left(n_0 + 2 + (n+1)(N-1)\right) + 1 \ . \tag{A.2}$$

Importance Sampling: Die Verteilung, die in Abschnitt 4.1 genutzt wurde, um ein Modell für $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ zu trainieren, ist in Gl. A.3 definiert.

$$\rho(\theta) = a((x-\mu)^4 + b) \quad \text{mit} \quad b = 0.4,\ \mu = \frac{\pi}{2} \quad \text{und a, sodass} \quad \int_{\epsilon}^{\pi-\epsilon} \rho(\theta)\mathrm{d}\theta = 1 \tag{A.3}$$

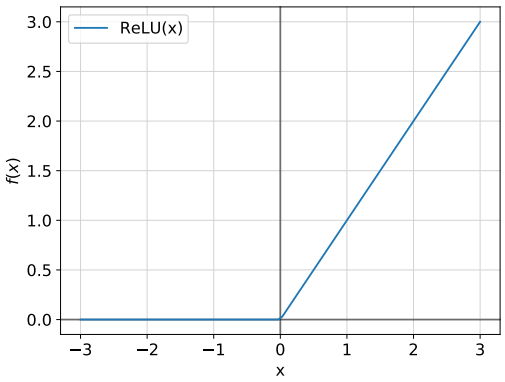
A.1 Such- und Hyperparameter

Loss-Funktionen

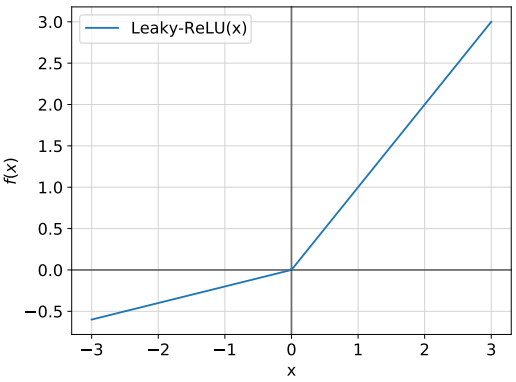
Bezeichnung	Implementierung
Mean-Absolute-Error (MAE)	$C(\mathbf{M}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \left\lvert y^{(i)} - \tilde{y}^{(i)} \right\rvert$
Mean-Squared-Error (MSE)	$C(\mathbf{M}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \tilde{y}^{(i)} \right)^2$
Mean-Absolute-Percentage-Error (MAPE)	$C(\mathbf{M}, \mathbf{b}) = \frac{100}{N} \sum_{i=1}^{N} \left\lvert y^{(i)} - \tilde{y}^{(i)} \right\rvert / \tilde{y}^{(i)}$
Mean-Squared-Logarithmic-Error (MSLE)	$C(\mathbf{M}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \left(\ln\left(y^{(i)}\right) - \ln\left(\tilde{y}^{(i)}\right) \right)^2$
Huber-Loss	$C(\mathbf{M}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} L_\delta \left(y^{(i)}, \tilde{y}^{(i)} \right)$

$$\text{mit} \qquad L_\delta \left(y^{(i)}, \tilde{y}^{(i)} \right) = \begin{cases} \frac{1}{2} \left(y^{(i)} - \tilde{y}^{(i)} \right)^2 & \text{für } \left\lvert y^{(i)} - \tilde{y}^{(i)} \leq \delta \right\rvert \\ \delta \left\lvert y^{(i)} - \tilde{y}^{(i)} \right\rvert - \frac{1}{2}\delta^2 & \text{sonst} \end{cases}$$

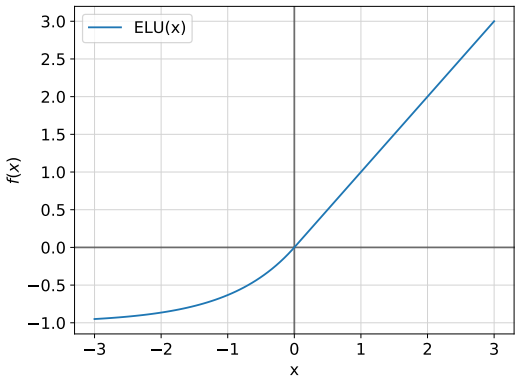
Tabelle A.1: Erwähnte und genutzte Kostenfuntkionen



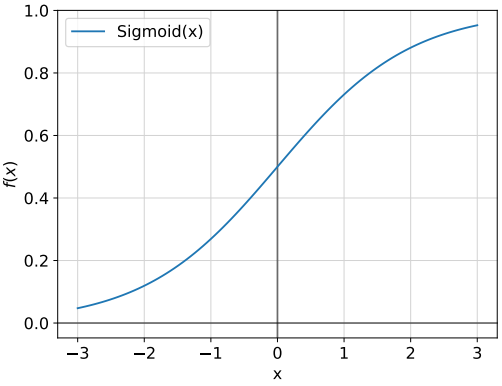
(a) Rectified Linear Unit: $f(x) = \max(0, x)$



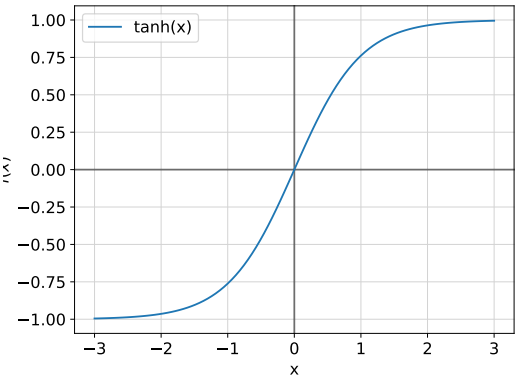
(b) Leaky-ReLU [20]: $f(\alpha, x) = \alpha x$ für $x < 0$



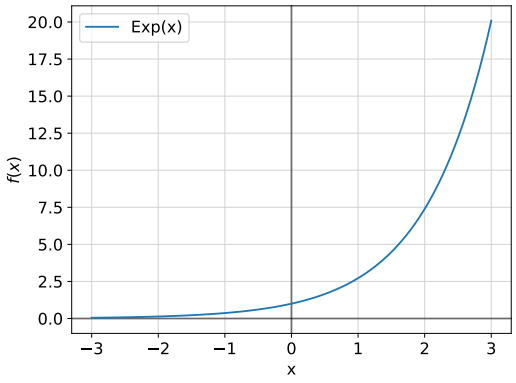
(c) ELU [6]: $f(\alpha, x) = \alpha(e^x - 1)$ für $x < 0$



(d) Sigmoid: $f(x) = \frac{e^x}{e^x+1}$



(e) tanh : $f(x) = \tanh(x)$



(f) Exp: $f(x) = e^x$

Abb. A.1: verwendete Activation-Funkctions

Optimizer

Bezeichnung	Funktionsweise
SGD	Der Gradient für einen Batch gemittelt und auf die Gewichte angewendet: $\mathbf{W} \to \mathbf{W} - \frac{\alpha}{N} \sum_{i=1}^{N} \nabla C_i(\mathbf{W})$ mit α: Learning-Rate und C_i: Loss für Punkt i
RMSprop [15]	Die Learning-Rate wird für jedes Gewicht durch ein gewichtetes Mittel von einigen vorhergehenden Gradienten geteilt.
Adam [19]	Die Learning-Rate wird für jedes Gewicht durch ein gewichtetes Mittel von einigen vorherg. Gradient geteilt und die Gradienten sind träge (*Momentum*).

Tabelle A.2: verwendete Optimizer

Daten-Transformationen

Abkürzung	Bezeichnung	Implementierung
No Log	Nur Skalierung	$\tilde{y} \to \frac{\tilde{y}}{\tilde{y}_{\min}}$
Log	Logarithmus	$\tilde{y} \to \ln\left(\frac{\tilde{y}}{\tilde{y}_{\min}}\right) =: \tilde{y}^{(\ln)}$
Base 10	Logarithmus zur Basis 10	$\tilde{y} \to \log_{10}(\frac{\tilde{y}}{\tilde{y}_{\min}})$
LN	Label-Normalization	$\tilde{y} \to \frac{2\tilde{y}^{(\ln)}}{\tilde{y}^{(\ln)}_{\max}} - 1$
FN [3]	Feature-Normalization	$x \to (x - \overline{x})/\sigma_x$

Tabelle A.3: Verwendete Daten-Transformationen, $\tilde{y}$ bezeichnet die Labels, x ein Feature, $\overline{x}$ den Mittelwert und σ_x die Standardabweichung des Features

Callbacks

Bezeichnung	Implementation
LearningRateScheduler	nach einer Verzögerung von 10 Epochen, wird die Learning-Rate nach jeder Epoche um 5% reduziert, bis diese auf $5 \cdot 10^{-8}$ abgefallen ist.
ReduceLROnPlateau	Fällt der Loss nach einer Epoche nicht um mindestens $2 \cdot 10^{-6}$, wird die Learning-Rate um 50% reduziert.
EarlyStopping	Fällt der Loss in drei aufeinanderfolgenden Epochen nicht um $2 \cdot 10^{-7}$ ab, wird der Trainingsvorgang gestoppt.

Tabelle A.4: Für alle Modelle verwendete Callbakcs zur Steuerung des Trainings

Hyperparameter	Pool	Best Config
Anzahl Layer	$\{1, 2, 3, 4\}$	4
Anzahl Units	$\{32, 64, 128, 256\}$	128
Loss-Funktion	MAE, MSE, Huber	MAE
Optimizer	Adam, RMSprop, SGD	Adam
Aktivierungsfunktion	ReLU, Leaky-ReLU, Sigmoid	Leaky-ReLU
Learning-rate	$\{10^{-2}, 5 \cdot 10^{-3}, 10^{-3}, 10^{-4}\}$	$5 \cdot 10^{-3}$
Batch-Größe	$\{64, 128, 512, 768, 2048\}$	128
Label-Normalisierung	$\{\text{keine}, [-1, 1]\}$	$[-1, 1]$
Max. Epochen	200	
Anzahl Trainingspunkte	60000	

Tabelle A.5: Parameter der Random-Search für $\frac{d\sigma}{d\theta}$ mit Ergebnis

Hyperparameter	Pool	Best Config
(Units, Nr. of Layers)	$\{(256,5),(512,3),(64,7),(1024,2),(128,6)\}$	$(256,5)$
Loss-Funktion	MAE, MSE, Huber	MAE
Optimizer	Adam, RMSprop	Adam
Aktivierungsfunktion	ReLU, Leaky-ReLU, Sigmoid, ELU, tanh	Leaky-ReLU
Learning-rate	$\{10^{-2}, 5\cdot10^{-3}, 10^{-3}, 10^{-4}\}$	10^{-2}
Batch-Größe	$\{256, 128, 512, 768, 1024\}$	256
Basis 10	True, False	True
Label-Normalisierung	$\{\text{keine}, [-1,1]\}$	keine
Feature-Normal.	True, False	True
Skalierung	True	
Logarithmus	True	
Max. Epochen	100	
Trainingspunkte	4.000.000	

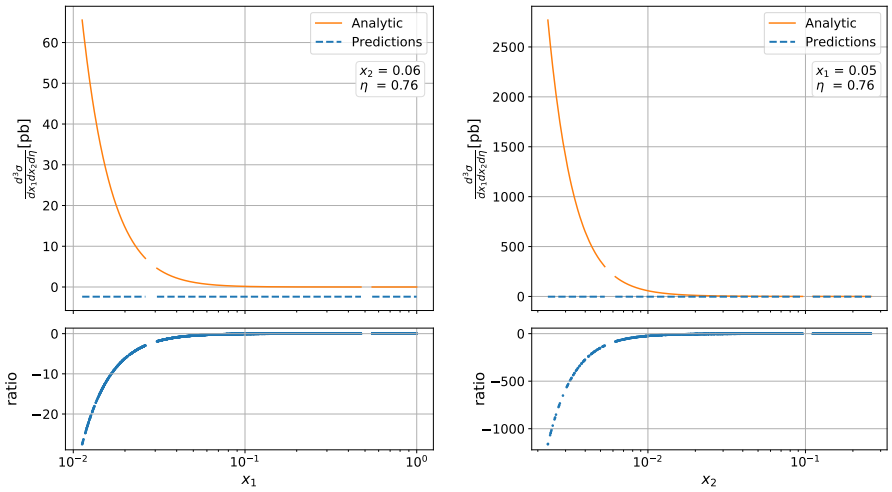
Tabelle A.6: Hyperparameter Pools eines Random-Search mit Ergebnis für $\frac{\mathrm{d}^3\sigma}{\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}\eta}$

Hyperparameter	Pool	Best Config
Anzahl Layer	$\{1,2,3,4\}$	2
Units	$\{32, 64, 128, 256\}$	256
Loss-Funktion	MAE, MSE	MAE
Optimizer	Adam, RMSprop, SGD	Adam
Aktivierungsfunktion	ReLU, Leaky-ReLU, Sigmoid	Leaky-ReLU
Learning-rate	$\{10^{-2}, 5\cdot10^{-3}, 10^{-3}, 10^{-4}\}$	$5\cdot10^{-3}$
Batch-Größe	$\{256, 128, 512, 768, 1024\}$	512
Label-Normalisierung	$\{\text{keine}, [-1,1]\}$	keine
Feature-Normal.	True, False	True
Skalierung	False	
Logarithmus	False	
Max. Epochen	100	
Trainingspunkte	1.000.000	

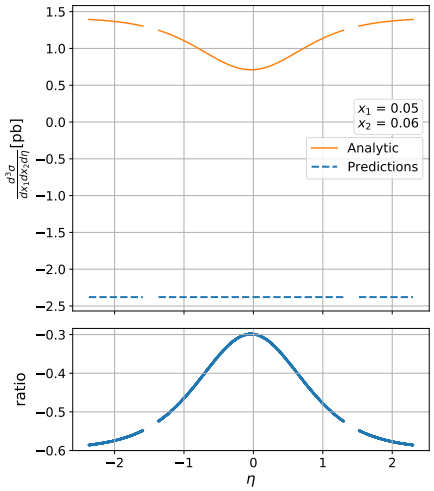
Tabelle A.7: Hyperparameter Pools eines Random-Search mit Ergebnis für die Umgewichtung des differentiellen Wirkungsquerschnitt

Hyperparameter	Pool	Best Config
Anzahl entfernte Layer	$\{1,2\}$	1
Anzahl hinzugefügte Layer	$\{0,1,2\}$	1
Units (hinzugefügte Layer)	$\{64, 128, 512\}$	128
Aktivierungsfunktion	ReLU, Leaky-ReLU, Sigmoid	ReLU
Learning-Rate	$\{10^{-2}, 5\cdot10^{-3}, 10^{-3}, 10^{-4}\}$	$5\cdot10^{-3}$
Batch-Größe	$\{128, 512, 768, 2048, 8196\}$	768
Fine-Tuning	True, False	True
Loss-Funktion	MAE	
Optimizer	Adam	
Max. Epochen	100	
Trainingspunkte	1.000.000	

Tabelle A.8: Hyperparameter Pools eines Random-Search mit Ergebnis für Transfer zwischen PDF-Sets



(a) Schnitt in x_1 **(b)** Schnitt in x_2



(c) Schnitt in η

Abb. A.2: Das Dying-ReLU-Problem in Aktion. Werden bei quantitativ sehr kleinen Labels keine Transformationen verwendet, stirbt der Großteil der Neuronen ab. Im vorliegenden Fall scheinen alle Units des letzten Layers lediglich Null zurückzugeben, woraus resultiert, dass das Output-Neuron eine Parallele zur x-Achse mit Verschiebung (hervorgerufen dessen Bias) ausgibt. Die Konfiguration des gezeigten Modells ist in Tabelle A.9 festgehalten

Hyperparameter	Wert	Hyperparameter	Wert
Anzahl Layer	5	Batch-Größe	256
Anzahl Units	256	Max. Epochen	100
Loss-Funktion	MAE	Anzahl Trainingspunkte	4M
Optimizer	Adam	Scaling	False
Aktivierungsfunktion	ReLU	Logarithmus	False
Kernel-Initializer	HeNormal	Basis 10	False
Bias-Initializer	Zeros	Label-Normalisierung	keine
Learning-rate	0.01	Feature-Normal.	False

Tabelle A.9: Hyperparameter des Modells $\frac{\mathrm{d}^3\sigma}{\mathrm{d}x_1\,\mathrm{d}x_2\,\mathrm{d}\eta}$ mit sterbenden ReLU-Aktivierungsfunktionen. Es sei angemerkt, dass dies die Konfiguration ist, die sich durch den Random-Search in Tabelle A.6 ergeben hat, mit dem Unterschied, dass keine Daten-Transformationen stattgefunden haben und Leaky-ReLU $\rightarrow$ ReLU.

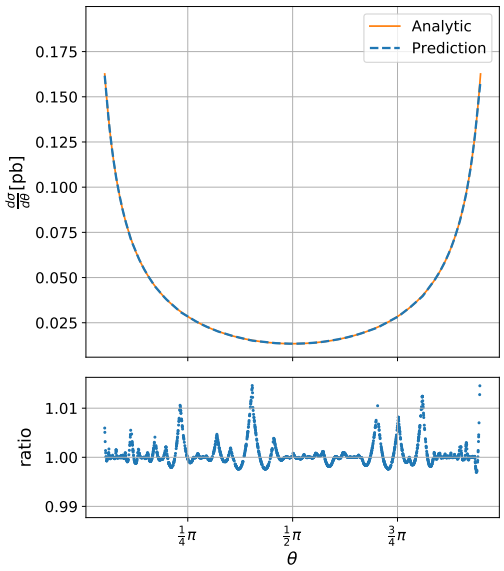


Abb. A.3: Modell für $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ mit gleicher Konfiguration wie in Tabelle 4.1 und 60000 Trainingspunkten. Trotz mehr Trainingspunkte hat die Genauigkeit im Vergleich mit Abb. 4.1 abgenommen (nachweisbar am Ratio).

Setup

Betriebssystem	Ubuntu 20.04.2.0 LTS
CPU	Intel Core i5-7300HQ
GPU	NVIDIA Geforce GTX 1050 Ti
Arbeitsspeicher	8 GB DDR4 2666 MHz
Python	3.8.5
TensorFlow	2.4.1
CUDA	11.3.0

Tabelle A.10: Verwendete Hard- und Software mit der alle Zeiten aufgenommen wurden

A.2 Abkürzungsverzeichnis

ML	Machine-Learning
TL	Transfer-Learning
DNN	Deep-Neural-Network
PDF	Partondichtefunktion
MC	Monte-Carlo
Features	Eingabewerte eines ML-Algorithmus
Labels	wahrer Funktionswert der Features
Units	Neuronen, Grundbaustein des DNN
Layer	Schicht von Neuronen
MSE	Mean-Squared-Error, mittlere quadratische Abweichung
MAE	Mean-Absolute-Error, mittlere absolute Abweichung
MAPE	Mean-Absolute-Percentage-Error
MSLE	Mean-Squared-Logarithmic-Error
SGD	Stochastic-Gradient-Descent
RMSProp	Root Mean Square Propagation
Base 10	Daten werden mit Logarithmus zur Basis 10 transformiert
FN	Feature-Normalization
LN	Label-Normalization
Log	Nur Scaling+Logarithmus
No Log	Nur Scaling
FT	Fine-Tuning
rw	Reweighting
TPM	Time per Million, Zeit zur Eval. von 10^6 Phasenraumpunkten

Tabelle A.11: Häufig genutzte Abkürzungen und Fachvokabular

B Literaturverzeichnis

[1] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, and S. Abdel Khalek. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29, 2012.

[2] Georges Aad, Alexander Kupco, Jay Chan, Timo Dreyer, Yufeng Wang, Karl Jakobs, Brian Le, Martin Spousta, Marina Cobal, Gen Tateno, et al. Search for resonances decaying into photon pairs in 139 fb^{-1} of pp collisions at $\sqrt{s}$= 13 tev with the atlas detector. Technical report, ATLAS-HIGG-2018-27-003, 2021.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Andy Buckley, James Ferrando, Stephen Lloyd, Karl Nordström, Ben Page, Martin Rüfenacht, Marek Schönherr, and Graeme Watt. Lhapdf6: parton density access in the lhc precision era. *The European Physical Journal C*, 75(3):1–20, 2015.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[7] ATLAS Collaboration. The atlas experiment at the cern large hadron collider. *JINST*, 3:S08003. 437 p, 2008. Also published by CERN Geneva in 2010.

[8] ATLAS Collaboration. Measurement of the production cross-section of isolated photon pairs in pp collisions at 13 tev with the atlas detector. Technical report, CERN, Geneva, Jul 2020.

[9] CMS Collaboration. The cms experiment at the cern lhc. the compact muon solenoid experiment. *JINST*, 3:S08004. 361 p, 2008. Also published by CERN Geneva in 2010.

[10] Haskell B Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[11] S. Chatrchyan et Al. Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. *Physics Letters B*, 716(1):30–61, 2012.

[12] Tanju Gleisberg, Stefan Höche, F Krauss, M Schönherr, S Schumann, F Siegert, and J Winter. Event generation with sherpa 1.1. *Journal of High Energy Physics*, 2009(02):007, 2009.

[13] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[15] G. Hinton. Divide the gradient by a running average of its recent magnitude. coursera neural netw. *COURSERA: Neural Networks for Machine Learning*, 6:26–31, 2012.

[16] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[17] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[18] Piotr Juszczak, D Tax, and Robert PW Duin. Feature scaling in support vector data description. In *Proc. asci*, pages 95–102. Citeseer, 2002.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[21] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier, 2019.

[22] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[23] Herbert Robbins and David Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In *Optimizing methods in statistics*, pages 233–257. Elsevier, 1971.

[24] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[25] Google Brain Team. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Danksagung

Danke an Frank Siegert, der mich während dieser Arbeit betreut und es mir möglich gemacht hat, an diesem interessanten Thema zu arbeiten.

Ich bedanke mich vielmals bei Christian Wiel für seine Unterstützung und sein offenes Ohr während meiner Arbeitsphase. Trotz Corona hatte ich so einen Ansprechpartner, der immer schnell und verlässlich Hilfe geleistet hat. Weiterhin möchte ich mich bei Heino Bülow, sowie Christian Wiel, für das Korrekturlesen meiner Arbeit bedanken.

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.

Andreas Weitzel

Dresden, Mai 2021