TECHNISCHE
UNIVERSITÄT
DRESDEN

Titel der Bachelor-Arbeit

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

vorgelegt von

VORNAME NACHNAME
geboren am XX.XX.19XX in GEBURTSORT

Institut für ??? Physik
Fakultät Physik
Bereich Mathematik und Naturwissenschaften
Technische Universität Dresden
2019

Eingereicht am xx. Monat 20xx

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

Zusammenfassung

Zusammenfassung
Deutsch

Abstract
English:

Inhaltsverzeichnis

Einleitung vii

Diphoton-Prozess ix

0.1 Partonischer Diphoton-Prozess . . . ix
0.2 Differentieller Wirkungsquerschnitt des partonischen Prozesses . . . xii
0.3 Hadronischer Diphoton Prozess . . . xiii
0.4 Reweightning zwischen PDF-Sets . . . xv

Maschinelles Lernen und tiefe neuronale Netzwerke xvii

0.5 Einführung in Maschinelles Lernen . . . xvii
0.6 Neuronale Netze . . . xviii
0.7 Training und Hyperparameter . . . xx
0.8 Transfer-Learning . . . xxi
0.9 Monte-Carlo-Integration . . . xxii

Anwendung von Maschinellem auf den Diphoton Prozess xxv

0.10 Diphoton-Prozess auf Parton-Ebene . . . xxv
0.11 Diphoton-Prozess auf Hadron-Ebene . . . xxviii
0.11.1 Allgemeines und Phasenraumschnitte . . . xxviii
0.11.2 Vorbereitung und Training . . . xxix
0.11.3 Vergleich von Hyperparametern . . . xxxi
0.12 Reweight zwischen Fits der PDF . . . xxxviii
0.13 Transfer-Learning zwischen Fits der PDF . . . xl
0.14 Monte-Carlo-Integration . . . xliv
0.14.1 Parton-Ebene . . . xliv
0.14.2 Hadron-Ebene . . . xliv

Zusammenfassung und Ausblick xlvii

0.15 Zusammenfassung . . . xlvii
0.16 Ausblick . . . xlviii

Anhang xlix

.1 Abkürzungen . . . liii
.2 Grafiken . . . liii
.3 Source-Code . . . liii

Einleitung

Maschinelles Lernen (ML) ist ein Schlagwort und Konzept, das zwar schon lange in 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.

Es wird in *Kapitel* 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. **??** beschäftigt sich zunächst mit den Konzepten hinter Maschinellem Lernen und speziell Deep-Learning mit tiefen neuronalen Netzen (DNN). Am Ende des Kapitels gehen wir auf die Grundlagen von einer Monte-Carlo-Integration (MC-Integration) ein. Die Anwendung der DNN folgt in **??**, wobei wir zunächst die differentiellen Wirkungsquerschnitte des Diphoton-Prozesses nähern. Anschließend untersuchen wir das Lernen von Gewichten zur Umgewichtung von Ergebnissen für unterschiedliche Fits der Partondichtefunktionen (PDF) und die Eignung von Transfer-Learning (TL).

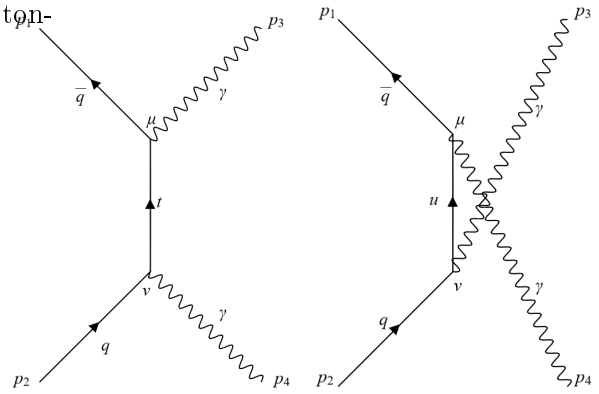
In dieser Arbeit werden gegebenfalls Abkürzungen verwendet, die bei Bedarf hier **??** nachgelesen werden können.
Wir verwenden durchweg natürliche Einheiten, sprich $\hbar = c = 1$. Vektoren werden mit Fett gedruckten Kleinbuchstaben (Bsp. $\mathbf{x}$) und Matrizen oder Tensoren mit Fett gedruckten Großbuchstaben (Bsp. $\mathbf{M}$) notiert. Speziell Dreiervektoren werden mit einem Pfeil gekennzeichnet (Bsp. $\vec{p}$). Vierervektoren ergeben sich aus dem Kontext.

Der gesamte Python-Code, der während dieser Arbeit verwendet wurde, kann unter `https://github.com/andiw9` 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 2.4.1 genutzt, wobei die Skripte auch mit TensorFlow 2.... getestet wurden.

Diphoton-Prozess

0.1 Partonischer Diphoton-Prozess

Zunächst wird der differentielle Wirkungsquerschnitt des partonischen Diphoton-Prozesses $q\overline{q} \to \gamma\gamma$ 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

Abbildung 0.1: Feynman-Diagramme des Diphoton-Prozesses führender Ordnung

In *Abbildung 0.1* sind die Feynman Diagramme führender Ordnung gezeigt. Hieraus können die Matrixelemente aus *Gleichung 0.1* abgeleitet werden. Es werden die Notation $\gamma^\mu p_\mu = \not{p}$, die Mandelstam-Variablen, sowie $\epsilon_\mu(p_i) \equiv \epsilon_{\mu,\lambda_i}$ verwendet, um die Matrixelemente zu vereinfachen (siehe *Gleichung 0.2*). λ_i beschreibt die Polarisation des Photons.

$$\begin{aligned}
\mathcal{M}_t &= \overline{\nu}\left(p_1\right)\left(-iQ_q e\gamma^\mu\right)\epsilon^*_\mu\left(p_3\right)\left(\frac{\gamma^\alpha\left(p_{1,\alpha}-p_{3,\alpha}\right)}{\left(p_1-p_3\right)^2}\right)\left(-iQ_q e\gamma^\nu\right)\epsilon^*_\nu\left(p_4\right)u\left(p_2\right)\\
\mathcal{M}_u &= \overline{\nu}\left(p_1\right)\left(-iQ_q e\gamma^\rho\right)\epsilon^*_\rho\left(p_4\right)\left(\frac{\gamma^\beta\left(p_{1,\beta}-p_{4,\beta}\right)}{\left(p_1-p_4\right)^2}\right)\left(-iQ_q e\gamma^\sigma\right)\epsilon^*_\sigma\left(p_3\right)u\left(p_2\right)
\end{aligned} \tag{0.1}$$

$$\begin{aligned}
\mathcal{M}_t &= -\frac{Q_q^2 e^2}{t}\left[\overline{\nu}(p_1)\gamma^\mu\epsilon^*_{\mu,\lambda_3}\left(\not{p}_1-\not{p}_3\right)\gamma^\nu\epsilon^*_{\nu,\lambda_4}u(p_2)\right]\\
\mathcal{M}_u &= -\frac{Q_q^2 e^2}{u}\left[\overline{\nu}(p_1)\gamma^\rho\epsilon^*_{\rho,\lambda_4}\left(\not{p}_1-\not{p}_4\right)\gamma^\sigma\epsilon^*_{\sigma,\lambda_3}u(p_2)\right]
\end{aligned} \tag{0.2}$$

Die Vierervektoren sind wie in **??** gewählt und in **??** aufgeführt. Die Mandelstam-Variablen ergeben sich zu:

$$t = -4p^2\cos^2\left(\frac{\theta}{2}\right) \quad \text{und} \quad u = (p_1-p_4)^2 = -4p^2\sin^2\left(\frac{\theta}{2}\right)\ . \tag{0.3}$$

$$p_1 = \begin{pmatrix} p \\ 0 \\ 0 \\ p \end{pmatrix} \quad p_2 = \begin{pmatrix} p \\ 0 \\ 0 \\ -p \end{pmatrix} \quad p_3 = \begin{pmatrix} p \\ \sin(\theta)p \\ 0 \\ \cos(\theta)p \end{pmatrix} \quad p_4 = \begin{pmatrix} p \\ -\sin(\theta)p \\ 0 \\ -\cos(\theta)p \end{pmatrix} \tag{0.4}$$

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

$$\begin{aligned} \mathcal{M} = \mathcal{M}_u + \mathcal{M}_t &= \mathcal{F}\left[\overline{\nu}(p_1)\left(\frac{\Gamma_t}{a} + \frac{\Gamma_u}{b}\right)u(p_2)\right] \\ &= \mathcal{F}\left[\overline{\nu}(p_1)\Gamma u(p_2)\right] \ , \end{aligned} \tag{0.5}$$

wobei die Ersetzungen ?? gewählt wurden.

$$\begin{aligned} \Gamma_t = \gamma^\mu \epsilon^*_{\mu,\lambda_3}(\not{p}_1 - \not{p}_3)\gamma^\nu \epsilon^*_{\nu,\lambda_4} \quad &\text{und} \quad \Gamma_u = \gamma^\rho \epsilon^*_{\rho,\lambda_4}(\not{p}_1 - \not{p}_4)\gamma^\sigma \epsilon^*_{\sigma,\lambda_3} \\ \text{sowie} \quad \mathcal{F} = \frac{Q_q^2 e^2}{4p^2} \quad &\text{und} \quad \Gamma = \frac{\Gamma_t}{\cos^2\left(\frac{\theta}{2}\right)} + \frac{\Gamma_u}{\sin^2\left(\frac{\theta}{2}\right)} \\ \cos^2\left(\frac{\theta}{2}\right) = a \quad &\text{und} \quad \sin^2\left(\frac{\theta}{2}\right) = b \end{aligned} \tag{0.6}$$

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:

$$\left\langle |\mathcal{M}|^2 \right\rangle = \frac{1}{12} \sum_{s_3,s_4} \sum_{\lambda_3,\lambda_4} |\mathcal{M}|^2 \ . \tag{0.7}$$

Um die Summe über die Helizitäten auszuführen, verwenden wir Casimirs Trick:

$$\sum_{Hel.} |\mathcal{M}|^2 = \mathcal{F}^2 \sum_{Hel.} \left[\overline{\nu}(p_1)\Gamma u(p_2)\right]\left[\overline{\nu}(p_1)\Gamma u(p_2)\right]^* = \mathcal{F}^2 \ \mathrm{Tr}\left[\Gamma \not{p}_2 \overline{\Gamma} \not{p}_1\right] \tag{0.8}$$

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:

$$\overline{\Gamma}_t = \gamma^\nu \epsilon_{\nu,\lambda_4}(\not{p}_1 - \not{p}_3)\gamma^\mu \epsilon_{\mu,\lambda_3} \quad \text{und} \quad \overline{\Gamma}_u = \gamma^\sigma \epsilon_{\sigma,\lambda_3}(\not{p}_1 - \not{p}_4)\gamma^\rho \epsilon_{\rho,\lambda_4} \ . \tag{0.9}$$

?? wird damit zu:

$$\mathrm{Tr}\left[\Gamma \not{p}_2 \overline{\Gamma} \not{p}_1\right] = \mathrm{Tr}\left[\frac{1}{a^2}\Gamma_t \not{p}_2 \overline{\Gamma}_t \not{p}_1 + \frac{1}{ab}\Gamma_t \not{p}_2 \overline{\Gamma}_u \not{p}_1 + \frac{1}{ba}\Gamma_u \not{p}_2 \overline{\Gamma}_t \not{p}_1 + \frac{1}{b^2}\Gamma_u \not{p}_2 \overline{\Gamma}_u \not{p}_1\right] \ . \tag{0.10}$$

Bei Einsetzen von ?? in ?? ergeben sich Terme in folgendem Schema:

$$T_{ij} = \frac{1}{12} \sum_{\lambda_3,\lambda_4} \mathcal{F}^2 \ \mathrm{Tr}\left[\frac{1}{ij}\Gamma\,(i)\,\not{p}_2 \overline{\Gamma}\,(j)\,\not{p}_1\right] \quad \text{mit} \quad i,j \in \{a,b\} \ . \tag{0.11}$$

Hierbei wird $\Gamma(a) = \Gamma_t$ und $\Gamma(b) = \Gamma_u$ identifiziert. Zunächst wird in **??** der Fall $i = j$ evaluiert, wobei diverse Spur-Methoden und Identitäten der Dirac-Matrizen verwendet werden.

$$\begin{aligned}
T_{aa} &= \frac{\mathcal{F}^2}{12a^2} \sum_{\lambda_3,\lambda_4} \operatorname{Tr}\left[\gamma^\mu \epsilon^*_{\mu,\lambda_3}(\not{p}_1 - \not{p}_3)\gamma^\nu \epsilon^*_{\nu,\lambda_4}\not{p}_2\gamma^{\nu'}\epsilon_{\nu',\lambda_4}(\not{p}_1 - \not{p}_3)\gamma^{\mu'}\epsilon_{\mu',\lambda_3}\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{12a^2} \sum_{\lambda_3,\lambda_4} \epsilon^{*\mu}_{\lambda_3}\epsilon^{\mu'}_{\lambda_3}\epsilon^{*\nu}_{\lambda_4}\epsilon^{\nu'}_{\lambda_4} \operatorname{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_3)\gamma_\nu\not{p}_2\gamma_{\nu'}(\not{p}_1 - \not{p}_3)\gamma_{\mu'}\not{p}_1\right] \\
&\overset{(2.15)}{=} \frac{\mathcal{F}^2}{12a^2}\, g^{\mu\mu'}g^{\nu\nu'} \operatorname{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_3)\gamma_\nu\not{p}_2\gamma_{\nu'}(\not{p}_1 - \not{p}_3)\gamma_{\mu'}\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{12a^2} \operatorname{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_3)\gamma_\nu\not{p}_2\gamma^\nu(\not{p}_1 - \not{p}_3)\gamma^\mu\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{12a^2} \operatorname{Tr}\left[-2\gamma^\mu(\not{p}_1 - \not{p}_3)\not{p}_2(\not{p}_1 - \not{p}_3)\gamma^\mu\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{3a^2} \operatorname{Tr}\left[(\not{p}_1 - \not{p}_3)\not{p}_2(\not{p}_1 - \not{p}_3)\not{p}_1\right] \\
&= \frac{8\mathcal{F}^2}{3a^2}(p_3\cdot p_2)(p_3\cdot p_1)
\end{aligned} \tag{0.12}$$

Wobei die Vollständigkeitsrelation für reale Photonen **??** verwendet wurde. Es folgt analog:

$$T_{bb} = \frac{8\mathcal{F}^2}{3b^2}(p_4\cdot p_2)(p_4\cdot p_1) \tag{0.13}$$

$$\sum_{\lambda=1}^{2} \epsilon^\mu_\lambda \epsilon^{*\nu}_\lambda = -g^{\mu\nu} \tag{0.14}$$

Für $i \neq j$ ergibt sich:

$$\begin{aligned}
T_{ab} &= \frac{\mathcal{F}^2}{12ab} \operatorname{Tr}\left[\gamma_\mu(\not{p}_1 - \not{p}_4)\gamma_\nu\not{p}_2\gamma^\mu(\not{p}_1 - \not{p}_3)\gamma^\nu\not{p}_1\right] \\
&= \frac{\mathcal{F}^2}{12ab} \operatorname{Tr}\left[-2\not{p}_2\gamma_\nu(\not{p}_1 - \not{p}_4)(\not{p}_1 - \not{p}_3)\gamma^\nu\not{p}_1\right] \\
&= \frac{4\mathcal{F}^2}{3ab}\left[(p_1\cdot p_2)\left[-2\,(p_1\cdot p_4) + (p_3\cdot p_4)\right] - (p_1\cdot p_3)(p_2\cdot p_4) + (p_2\cdot p_3)(p_1\cdot p_4)\right]
\end{aligned} \tag{0.15}$$

und analog:

$$T_{ba} = \frac{4\mathcal{F}^2}{3ab}\left[(p_1\cdot p_2)\left[-2\,(p_1\cdot p_3) + (p_3\cdot p_4)\right] - (p_1\cdot p_4)(p_2\cdot p_3) + (p_1\cdot p_3)(p_2\cdot p_4)\right] \tag{0.16}$$

Beim Einsetzen der expliziten Vierervektoren aus **??**, fällt auf, dass $T_{ab} + T_{ba} = 0$. Wir haben nun die

Summe über die Helizitäten ausgeführt und können damit **??** umschreiben zu:

$$\begin{aligned}\left\langle|\mathcal{M}|^2\right\rangle &= \frac{8}{3}\mathcal{F}^2\left(\frac{1}{a^2}(p_3\cdot p_2)(p_3\cdot p_1)+\frac{1}{b^2}(p_4\cdot p_2)(p_4\cdot p_1)\right)\\ &= \frac{2}{3}Q_q^4e^4\left[\frac{1-\cos^2(\theta)}{\cos^4\left(\frac{\theta}{2}\right)}+\frac{1-\cos^2(\theta)}{\sin^4\left(\frac{\theta}{2}\right)}\right]\\ &= \frac{4}{3}Q_q^4e^4\frac{1+\cos^2(\theta)}{\sin^2(\theta)} = \frac{4}{3}Q_q^4e^4\cosh(2\eta)\ .\end{aligned} \tag{0.17}$$

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

0.2 Differentieller Wirkungsquerschnitt des partonischen Prozesses

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

$$\sigma = \frac{1}{64\pi^2 s}\int\left\langle|\mathcal{M}|^2\right\rangle\mathrm{d}\Omega = \frac{1}{32\pi s}\int\left\langle|\mathcal{M}|^2\right\rangle\sin(\theta)\mathrm{d}\theta \tag{0.18}$$

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

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\theta} = \frac{1}{2}\cdot\frac{Q_q^4e^4}{24\pi s}\frac{1+\cos^2(\theta)}{\sin(\theta)}\ . \tag{0.19}$$

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

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\eta} = \frac{\mathrm{d}\theta}{\mathrm{d}\eta}\frac{\mathrm{d}\sigma}{\mathrm{d}\theta} = \frac{Q_q^4e^4}{48\pi s}(1+\tanh^2(\eta))\ . \tag{0.20}$$

0.3 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 Photonen in einem Beschleuniger, beispielsweise dem `LHC` zusammen, 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.

Das Schwerpunktsystem der Quarks unterscheidet sich im Allgemeinen vom Schwerpunktsystem der Protonen, welches im Fol-

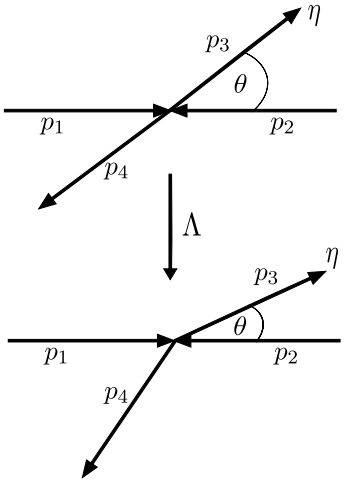


Abbildung 0.2: Kinematik der Stoßprozesse im Laber- und Schwerpunkt

genden als Laborsystem bezeichnet wird. Der Impulsbruchteil eines Quarks innerhalb eines Hadrons ist nicht fest definiert, sodass ihm zunächst einen unbestimmten Bruchteil ξ 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 geschrieben werden kann als:

$$\mathbf{p}_q = (\xi E, 0, 0, \xi E) = \xi \mathbf{p}_p \ . \tag{0.21}$$

Findet bei einer Interaktion ein Impulsübertrag $\mathbf{q}$ statt, so geht der Vierervektor des Partons $\xi\mathbf{p}_p \to (\xi\mathbf{p}_p + \mathbf{q})$ über. Bei Betrachtung der invarianten Masse beider Zustände **??** kann nach dem Impusbruchteil ξ aufgelöst werden (siehe **??**).

$$\left(\xi\mathbf{p}_p\right)^2 = m_q^2 \quad \text{und} \quad \left(\xi\mathbf{p}_p + q\right)^2 = \left(\xi\mathbf{p}_p\right)^2 + 2\xi\mathbf{p}_p \cdot \mathbf{q} + \mathbf{q}^2 = m_q^2 \tag{0.22}$$

$$2\xi\mathbf{p}_p \cdot \mathbf{q} + \mathbf{q}^2 = 0 \quad \Rightarrow \quad \xi = \frac{-\mathbf{q}^2}{2\mathbf{p}_p \cdot \mathbf{q}} = x \tag{0.23}$$

Das x in **??** 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) $f_{i,h}(x, Q^2)$. Diese PDFs beschreiben die Wahrscheinlichkeitsdichte, bei einer Energieskala $Q^2 = -\mathbf{q}^2$, das entsprechende Parton i mit dem Impulsbruchteil x im Hadron h 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 $pp \to \gamma\gamma$ 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 totale Wirkungsquerschnitt $\sigma_{i,j}$ für die Reaktion der Partonen i und j bei dem Zusammenstoß von zwei Protonen berechnet werden. In **??** ist die Kennzeichnung des Hadrons vernachlässigt.

$$\sigma_{i,j} = \int f_i(x_1, Q^2) f_j(x_2, Q^2) \tilde{\sigma}_{i,j}(x_1, x_2, Q^2) \mathrm{d}x_1 \mathrm{d}x_2 \tag{0.24}$$

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

$$\sigma = \sum_q \left(\sigma_{q,\overline{q}} + \sigma_{\overline{q},q}\right) \ . \tag{0.25}$$

In Abschnitt ?? wurde bereits der differentielle Wirkungsquerschnitt für den partonischen Prozess σ_p im Schwerpunktsystem der Konstituenten berechnet. $\tilde{\sigma}_{q,\overline{q}}(x_1, x_2, Q^2)$ kann nach ?? berechnet werden.

$$\tilde{\sigma}_{q,\overline{q}}(x_1, x_2, Q^2) = \int \frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta}\left(x_1, x_2, Q^2\right) \mathrm{d}\eta \tag{0.26}$$

?? 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 ??, wobei E die Strahlenergie bezeichnet.

$$\mathbf{p}_q = (x_1 E, 0, 0, x_1 E) \quad \text{und} \quad \mathbf{p}_{\overline{q}} = (x_2 E, 0, 0, -x_2 E) \tag{0.27}$$

Mit ?? ergibt sich die Schwerpunktenergie zu:

$$\sqrt{s} = 2\sqrt{x_1 x_2} E \ . \tag{0.28}$$

Im folgenden werden Variablen im Laborsystem ungestrichen und Variablen im Schwerpunktsystem der Partonen gestrichen benannt. DIe Pseudo-Rapidität, die sich additiv bei Inertialsystemwechsel verhält, transformiert sich, wenn sich das Schwerpunktsystem der Partonen mit der Geschwindigkeit β zum Laborsystem bewegt, nach ??.

$$\eta' = \eta + \frac{1}{2}\ln\left(\frac{1-\beta}{1+\beta}\right) \quad \Rightarrow \quad \frac{\mathrm{d}\eta'}{\mathrm{d}\eta} = 1 \tag{0.29}$$

Damit ergibt sich für den Wirkungsquerschnitt im Laborsystem:

$$\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta} = \frac{\mathrm{d}\eta'}{\mathrm{d}\eta}\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta'} = \frac{Q_q^4 e^4}{48\pi s}(1 + \tanh^2(\eta')). \tag{0.30}$$

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

$$\beta = \frac{|\vec{p}_q + \vec{p}_{\overline{q}}|}{m_q + m_{\overline{q}}} = \frac{(x_1 - x_2)E}{(x_1 + x_2)E} = \frac{x_1 - x_2}{x_1 + x_2} \tag{0.31}$$

Durch Einsetzen der gefunden Ausdrücke für s, η', und β in ??, ergibt sich mit $Q^2 = 2x_1 x_2 E^2$ insgesamt für die differentiellen Wirkungsquerschnitt im Laborsystem:

$$\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta}\left(x_1, x_2, Q^2, q\right) = \frac{Q_q^4 e^4}{96\pi Q^2}\left(1 + \tanh^2\left(\eta + \frac{1}{2}\ln\left(\frac{x_2}{x_1}\right)\right)\right) \ . \tag{0.32}$$

Schließlich können mithilfe von ??, ?? und ?? die Ausdrücke ?? für den totalen und ?? für den dreifach differentiellen Wirkungsquerschnitt gefunden werden.

$$\sigma = \sum_q \int \left[f_q(x_1, Q^2) f_{\overline{q}}(x_2, Q^2) + f_{\overline{q}}(x_1, Q^2) f_q(x_2, Q^2)\right] \frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta} \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}\eta \tag{0.33}$$

$$\frac{\mathrm{d}^3\sigma}{\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}\eta} = \sum_q \left[f_q(x_1,Q^2)f_{\overline{q}}(x_2,Q^2) + f_{\overline{q}}(x_1,Q^2)f_q(x_2,Q^2)\right]\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta} \tag{0.34}$$

0.4 Reweightning zwischen PDF-Sets

Das quantitative Ergebnis von **??** hängt von dem verwendeten Fit an die Partondichtefunktionen ab. Je nach Messung und Anpassung ergeben sich dabei kleine Unterschiede zwischen den verschiedenen Sets. Das *Reweight* entspricht einem Umrechnungsfaktor zwischen differentiellen Wirkungsquerschnitten, die mit verschiedenen Fits von Partondichtefunktionen berechnet wurden. Aus **??** kann, bis auf die Quarkladung, $\frac{\mathrm{d}\sigma_p}{\mathrm{d}\eta}$ aus der Summe herausgezogen werden, sodass sich für die Gewichte zwischen den PDF-Sets $f^{(1)}$ und $f^{(2)}$ **??** ergibt.

$$w\left(x_1,x_2\right) = \frac{\sum_q Q_q^4\left[f_q^{(1)}(x_1,Q^2)f_{\overline{q}}^{(1)}(x_2,Q^2) + f_{\overline{q}}^{(1)}(x_1,Q^2)f_q^{(1)}(x_2,Q^2)\right]}{\sum_q Q_q^4\left[f_q^{(2)}(x_1,Q^2)f_{\overline{q}}^{(2)}(x_2,Q^2) + f_{\overline{q}}^{(2)}(x_1,Q^2)f_q^{(2)}(x_2,Q^2)\right]} \tag{0.35}$$

Maschinelles Lernen und tiefe neuronale Netzwerke

0.5 Einführung in Maschinelles Lernen

Motivation: Die Berechnung eines differentiellen Wirkungsquerschnitts eines Prozesses aus den zugrundeliegenden Feynman-Diagrammen, kann schnell sehr kompliziert werden. Oft sind diese Aufgaben analytisch nicht oder nur noch sehr aufwändig lösbar, sodass numerische Methoden bemüht werden müssen. Diese fortgeschrittenen Methoden können in der Praxis sehr rechenintensiv sein und viele Ressourcen beanspruchen. Algorithmen, die maschinelles Lernen verwenden, können je nach Typ und Komplexität jedoch sehr effizient und im Vergleich mit herkömmlichen numerischen Methoden signifikant schneller sein. Ein ML-Algorithmus ist zwar nicht in der Lage, den differentiellen Wirkungsquerschnitt analytisch aus den zugrundeliegenden Feynman-Diagrammen in erster Instanz zu berechnen, er kann die Funktion jedoch durch die Vorarbeit eines rechenaufwändigeren Algorithmus erlernen. Der Vorteil liegt hierbei darin, die aufwändigen numerischen Methoden zur Berechnung einer ausreichenden Anzahl von Phasenraumpunkten nur einmalig durchzuführen, mit diesen das DNN zu trainieren, um anschließend eine größere Anzahl an Punkten zu generieren.
Im Folgenden werden die Möglichkeiten eines solchen Einsatzes von ML-Algortihmen untersucht und evaluiert.
Einführung: 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. Wir unterscheiden dabei die Teilgebiete:

- Klassifizierung und
- Regression

Klassifizierung ordnet Objekten ihre jeweilige Gruppe, auch genannt „Label“ zu. Dies geschieht auf Grundlage der Eigenschaften eines Objektes, den sogenannten „Features“ . Wir werden uns im Folgenden mit **Regression** beschäftigen, wobei hier anstatt einer diskreten Zuordnung eine reelle Zahl ausgegeben wird. Betrachten wir eine Funktion $f : \mathbb{R}^n \to \mathbb{R}, \mathbf{x} \mapsto f(\mathbf{x})$, bezeichnen wir die Einträge des Vektors $\mathbf{x}$ als Features und den Funktionswert $f(\mathbf{x}) \in \mathbb{R}$ als Label. Am Beispiel des hadronischen Diphoton Prozess **??**, 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 wahrend Ergebnis abgeglichen werden und das Netz seine Parameter entsprechend anpassen kann, um 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 hoch-

dimensionale Funktionen erlernen und damit die Effizienz der numerischen Berechnung von differentiellen Wirkungsquerschnitten steigern kann.

0.6 Neuronale Netze

Eine Veranschaulichung des Konzeptes eines neuronalen Netzes ist in **??** 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 oder Nodes genannt werden, können unterschiedlich stark aktiviert sein. Die Nodes 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 hierbei 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.

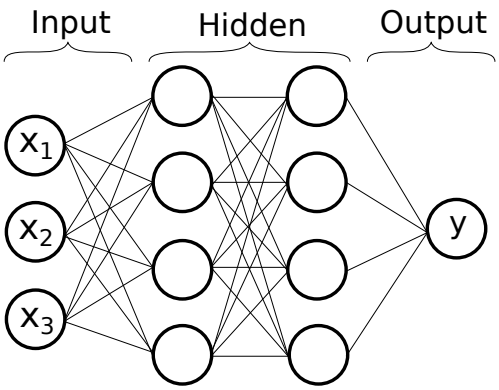


Abbildung 0.3: Konzeptzeichnung eines mehrschichtigen Perzeptron, kann noch verändert werden

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. Wir stellen die Ausgabe des n-ten Neurons der Schicht l, bezeichnet mit y_l^n, als Skalarprodukt zwischen den Gewichten der Node $\mathbf{w}_l^n$ dar, 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.

$$y_l^n = \sigma\left(\mathbf{w}_l^n \cdot \mathbf{y}_{l-1} + b_l^n\right) \tag{0.36}$$

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

$$\mathbf{y}_l = \sigma\left(\mathbf{W}_l \cdot \mathbf{y}_{l-1} + \mathbf{b}_l\right) \tag{0.37}$$

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 (Cost-Function) oder Verlustfunktion (Loss-Function) bezeichnet, wobei hier eine mögliche Wahl die mittlere quadratische Abweichung ist.

$$C(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \tilde{y}^{(i)} \right)^2 \tag{0.38}$$

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, greifen wir auf *Gradient-Descent* zurück. Hierbei wird, beim Output-Layer beginnend, der Gradient der Kostenfunktion berechnet und per Kettenregel zum vorherigen Layer fortgepflanzt. Diesen Vorgang, so für alle Nodes 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, an Trainingspunkten berechnet und auf die Gewichte angewendet, sodass man sich einem lokalen, oder auch globalen, Minimum nähern kann (Stochastic Gradient Descent, SGD).

0.7 Training und Hyperparameter

Man nennt Parameter, die der Programmierende im Vornherein festlegen muss und die nicht vom Algorithmus erlernt werden, Hyperparameter. Wir sprechen im Weiteren über folgende Hyperparameter, wobei wir nicht zwischen solchen Hyperparametern, die die Architektur des Netzes bestimmen und Trainingsparametern, die das Lernverhalten bestimmen, differenzieren:

- Anzahl der Layer und Nodes
- 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 Nodes** 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 sie ist es die letztendlich optimiert wird, um die Gradienten zu erhalten.

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. Gegebenenfalls kann sie auch dazu genutzt werden, die Ausgabe eines Neurons zu regulieren. Speziell für Aktivierungsfunktionen mit verschwindenden Ableitungen, besonders die namensgebende Aktvierung *ReLU*, kann das *Dying-ReLU-Problem* auftreten. Hierbei wurden 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.

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 verwendeten Aktivierungsfunktion, sodass sich bereits spezielle Initialisierungsmethoden für bestimmte Aktivierungen etabliert haben, wie zum Beispiel HeNormal für ReLU-Aktivierung.

Die hier besprochenen Lernalgorithmen basieren auf Gradientenabstieg. Für 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 hierbei so gewählt werden, dass das Lernen weder in einem zu hohen lokalen Maximum zum Erliegen kommt, noch zu groß ist um den Tiefpunkt des Minimums zu erreichen.

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 erstetllt. Anschließend werden im ersten Fall alle und im zweiten lediglich zufällige Gitterpunkte trainiert und evaluiert. Fortgeschrittenere Methoden der Hyperparameteroptimierung wie Bayesian-Search oder Hyperband, sollen in kürzerer Zeit bessere Parameter finden. Die Hyperparameteroptimierung kann sehr aufwändig sein.

Implementierung mit Keras und Tensorflow Die Implementierung des ML-Algorithmus wird in dieser Arbeit mit der Open-Source Python-Bibliothek `TensorFlow` und `Keras` stattfinden. 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

[1] Null speziell für ReLU

Netz komfortabler machen und das Speichern und Laden vereinfachen.

0.8 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, 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

Wir machen Nutzen von beiden Aspekten, da wir einerseits die Trainingszeit reduzieren und sich andererseits die Zeit zur Datengeneration verkürzt. Konkret wird im Laufe dieser Arbeit das Transfer-Learning verwendet, um die 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.
Im Folgenden wird kurz auf den Ablauf von Transfer-Learning für künstliche neuronale Netze eingegangen:

- 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.

0.9 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.

$$\langle f(\mathbf{X}) \rangle = \langle f \rangle = \frac{1}{\|\Omega\|} \int_\Omega f(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{0.39}$$

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

$$\langle \tilde{f} \rangle = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) \quad \text{mit} \quad \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) = \langle f \rangle \tag{0.40}$$

Da de Erwartungswert nicht exakt berechnet werden kann, weil es nicht möglich ist f an unendlich vielen Punkten zu evaluieren, wird verwendet, dass der Schätzer gegen den Erwartungswert konvergiert und genähert $\langle \tilde{f} \rangle \approx \langle f \rangle$. Die Geschwindigkeit der Konvergenz der Näherung kann erhöht werden, indem in **??** 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 (**??**)

$$I = \int_\Omega f(\mathbf{x}) \mathrm{d}x = \int_\Omega \frac{f(\mathbf{x})}{\rho(\mathbf{x})} \rho(\mathbf{x}) \mathrm{d}\mathbf{x} = \left\langle \left(\frac{f}{\rho} \right) \right\rangle_\rho \tag{0.41}$$

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

$$I \approx \left\langle \left(\frac{\tilde{f}}{\rho} \right) \right\rangle_\rho = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)} \tag{0.42}$$

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

$$\mathrm{Var}\left(\frac{f}{\rho} \right) = \left\langle \left(\frac{f}{\rho} - \left\langle \frac{f}{\rho} \right\rangle \right)^2 \right\rangle = \left\langle \left(\frac{f}{\rho} \right)^2 \right\rangle - \left\langle \frac{f}{\rho} \right\rangle^2 \approx \frac{1}{N} \sum_{i=1}^{N} \left(\frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)} \right)^2 - I^2 \tag{0.43}$$

Die Varianz minimiert sich also, wenn jeder Summand aus **??** gleich groß ist. Der Vorgang die Wahrscheinlichkeitsdichte ρ an die Form unserer zu integrierenden Funktion f anzupassen, nennt man **Importance Sampling**. 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:

$$\sigma_{\left\langle \frac{f}{\rho} \right\rangle} = \frac{1}{\sqrt{N-1}} \cdot \sqrt{\mathrm{Var}\left(\frac{f}{\rho} \right)} \tag{0.44}$$

Wir werden im Folgenden simple Monte-Carlo-Methoden und das Importance Sampling verwenden, um aus unseren differentiellen Wirkungsquerschnitten die totalen Wirkungsquerschnitte zu erhalten.

Anwendung von Maschinellem auf den Diphoton Prozess

0.10 Diphoton-Prozess auf Parton-Ebene

Zunächst werden neuronale Netze zur Regression der eindimensionalen differentiellen Wirkungsquerschnitte aus **??** benutzt. Dabei werden geeignete Wertebereiche gewählt, sprich $\theta \in [\epsilon, \pi - \epsilon]$ bzw. $\eta \in [-3, 3]$, wobei wir das Verhalten des DNN für verschiedene ϵ evaluieren werden. 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 $\sqrt{s} = 200\text{GeV}$ berechnet. Wir generieren die Trainingspunkte zufällig nach einer, wenn nicht explizit anders angegebenen, uniformen Verteilung. 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. **??** reduziert sich von der Form auf einen $\tanh^2$, dessen Wertebereich sich über $[0, 1)$ erstreckt und damit schon von vornherein auf einen geeigneten Wertebereich normiert ist. Wir behandeln den Vorfaktor mit einer Skalierung der Funktionswerte, auf die wir später noch weiter eingehen werden. Für diese einfache Aufgabe werden die Hyperparameter wie in **??** gewählt und keine weiteren Optimierungen, wie für die folgenden Modelle, durchgeführt. Wir werden in Zukunft standardmäßig den Kernel-Initializer HeNormal verwenden und das Bias zu Null initialisieren. Das Training an sich wird von den folgenden, in Keras implementierbaren, Callbacks

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

Tabelle 0.1: Hyperparameter des Modells $\frac{d\sigma}{d\eta}$

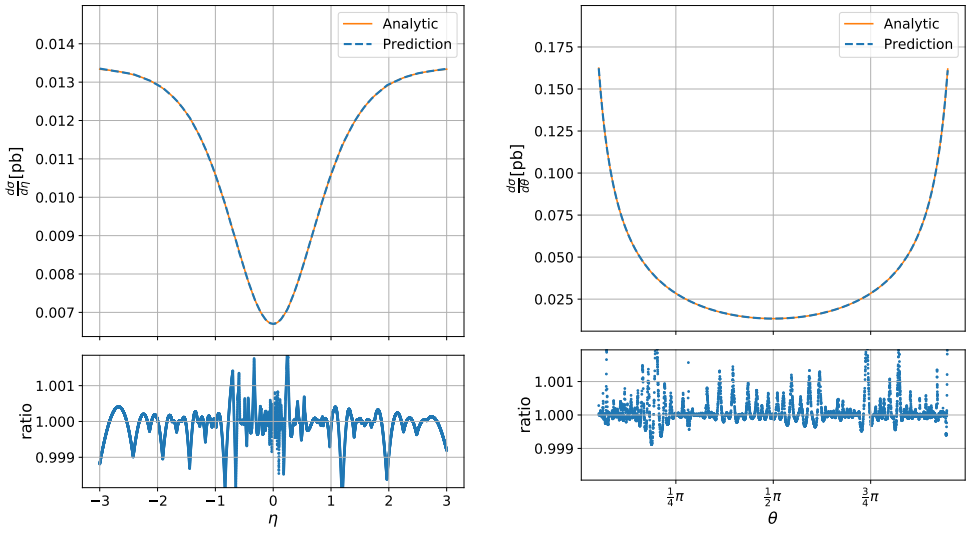
geregelt:

- `LearningRateScheduler`: Ein Ablaufplan wird festgelegt, der für jede Epoche die zu verwen-

dende 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 Zeit keinen Mindestfortschritt, wird das Training gestoppt.

Die Wahl der genauen Konfiguration der Callbacks ist in **??** festgehalten. Die gelernte Funktion im Vergleich mit den analytischen Werten ist in **??** gezeigt. Die Werte überlagern sich gut, sodass man auf den ersten Blick keinen Unterschied feststellen kann. Betrachtet man das Verhältnis, erkennt man, 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 $\epsilon = 0.163$

Abbildung 0.4: Predictions der neuronalen Netze für den Wirkungsquerschnitt des Diphoton-Prozess auf partonischer Ebene

Modell für $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$: Der Wirkungsquerschnitt in Abhängigkeit von θ unterscheidet sich vom vorherigen Modell durch seine Polstellen. Da wir numerisch nicht mit Polstellen umgehen können, müssen wir den Trainingsbereich auf $[\epsilon, \pi - \epsilon]$ einschränken. Aus physikalischer Sicht ist das legitim, da die Polstellen im Strahlengang des Speicherrings liegen und damit nicht messbar sind. Viele Detektoren, wie ATLAS, können Photonen mit Pseudo-Rapiditäten bis zu $|\eta| \leq 2.5$ 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). Die Such-Parameter mit Ergebnis sind in **??** 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 **??** gezeigt. Die Präzision ist trotz der komplizierteren Funktion mit **??** 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 **??**(*Importance Sampling(IS)*) generiert sind, um die Leistung der drei Modelle zu vergleichen.

Die Vergleiche sind in **??** und in **??** 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 **??** kann man an einigen Stellen am Verhältnis sehen, dass das mit importance gesampelten Trainingsdaten trainierte Netz an den Polstellen besser und im Zentrum schlechter angepasst ist. In **??** a) ist der Definitionsbereich näher an die Polstelle gelegt, um den Verbesserung des IS-gesampelten Modells besser aufzulösen. Anhand von **??** b) 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.

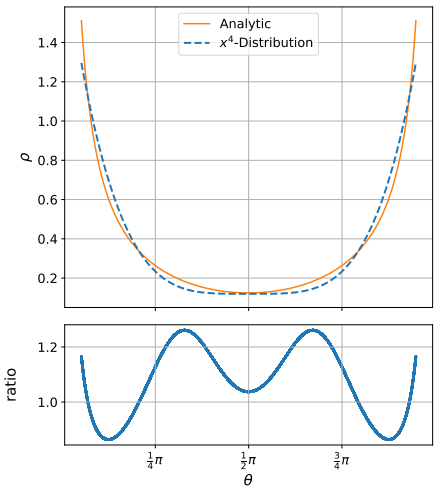
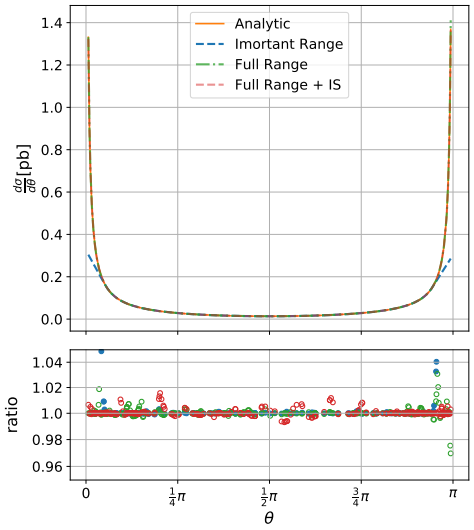
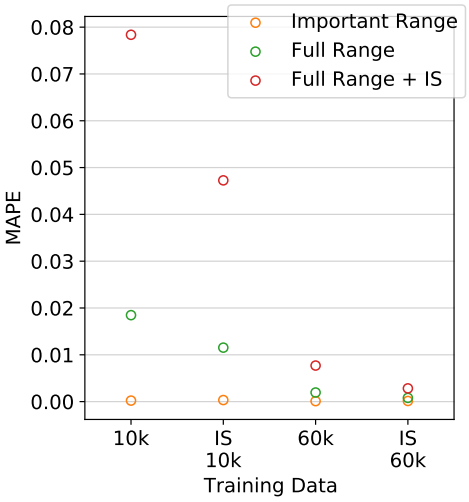


Abbildung 0.5: simples Importance Sampling, das die analytische Funktion annähern soll hier verweis auf Funktion in anhang

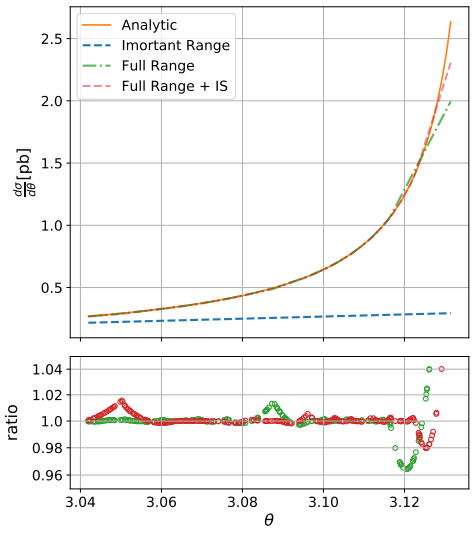
In **??** sind noch einmal der MAPE (Mean-Absolute-Percentage-Error) der verschiedenen Modelle für verschiendene Testdatensets gezeigt. Hier wird erneut deutlich, dass das Importance Sampling vor allem nützlich ist, wenn Bereiche mit hohen Funktionswerten besonders wichtig sind. Bei der Berechnung des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



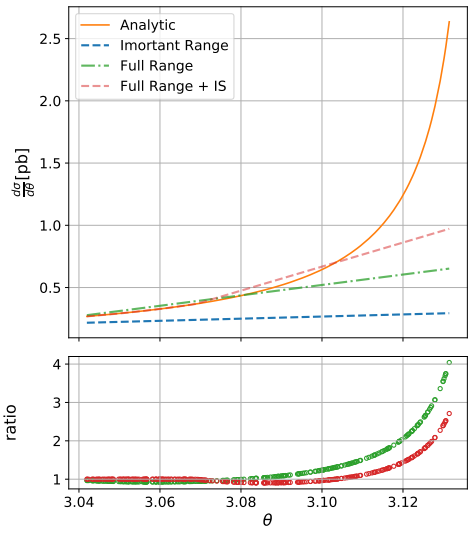
(a) Vergleich DNN-analytische Werte für $\frac{\mathrm{d}\sigma}{\mathrm{d}\theta}$ auf $\theta \in [\epsilon', \pi - \epsilon']$ für $\epsilon' = 0.01$



(b) MAPE für verschiedene Test-Datensets



(c) Modelle mit 60000 Tranings-punkten



(d) Modelle mit 10000 Trainings-punkten

Abbildung 0.6: Vergleich der Performance auf unterschiedlichen Intervallen von verschiedenen Theta-Modellen, die mit verschiedenen Datenmengen trainiert wurden.

0.11 Diphoton-Prozess auf Hadron-Ebene

0.11.1 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 Se-

lektionen sind angelehnt an **??** und aufgelistet in **??**. 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. **??** 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, müssen wir sicherstellen, dass beide Photonen die η-Selektion erfüllen. Werden η_γ und $\eta_{\gamma'}$ in entgegengesetzte Richtungen gemessen, berechnen sich diese aus η' der Photonen im Schwerpunktsystem der Quarks nach:

$$\eta_\gamma = \eta' - \frac{1}{2}\ln\left(\frac{x_2}{x_1}\right) \quad \text{sowie} \quad \eta_{\gamma'} = \eta' + \frac{1}{2}\ln\left(\frac{x_2}{x_1}\right) . \tag{0.45}$$

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

$$\eta_{\gamma'} = \eta_\gamma + \frac{1}{2}\ln\left(\frac{x_2^2}{x_1^2}\right) \tag{0.46}$$

Für die Berechnung von **??** verwenden wir das PDF-Set CT14nnlo von LHAPDF, wobei wir positive Beiträge von Charm- und Strange-Quark berücksichtigen und Top- und Bottom-Quark vernachlässigen. Die Strahlenergie beträgt $E = 6500\text{GeV}$.

0.11.2 Vorbereitung und Training

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 $x \to 0$[2]. 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. Hinzu kommt, 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. Es wurde Lösung in der oft für ähnliche Probleme verwendeten Loss-Funktion *Mean-Squared-Logarithmic-Error*(MSLE) gesucht (siehe **??**), da es hier ausschließlich auf das Verhältnis zwischen Label und Vorhersage ankommt und damit gesichert ist, dass keine Phasenraumbereiche vernachlässigt werden. Um nicht mit negativen Werten zu arbeiten, wird generell

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

Bereich	Cut
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 0.2: Event-Selektion für den hadronischen Diphoton-Prozess in Anlehnung an Messung von ATLAS[Ref]

$y \to y+1$ transformiert. Da jedoch der Großteil der quantitativen Werte des Wirkungsquerschnittes in standardmäßig verwendeten Einheiten, sprich $1/\mathrm{GeV}^2$ und pb, viel kleiner als eins ist und damit $\ln(1+y) \approx y$ gilt würde so nur ein ineffizienterer MSE implementiert werden.

$$C(\mathbf{M}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N}\left(\ln\left(y^{(i)}\right) - \ln\left(\tilde{y}^{(i)}\right)\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\ln\left(\frac{y^{(i)})}{\tilde{y}^{(i)}}\right)\right)^2 \tag{0.47}$$

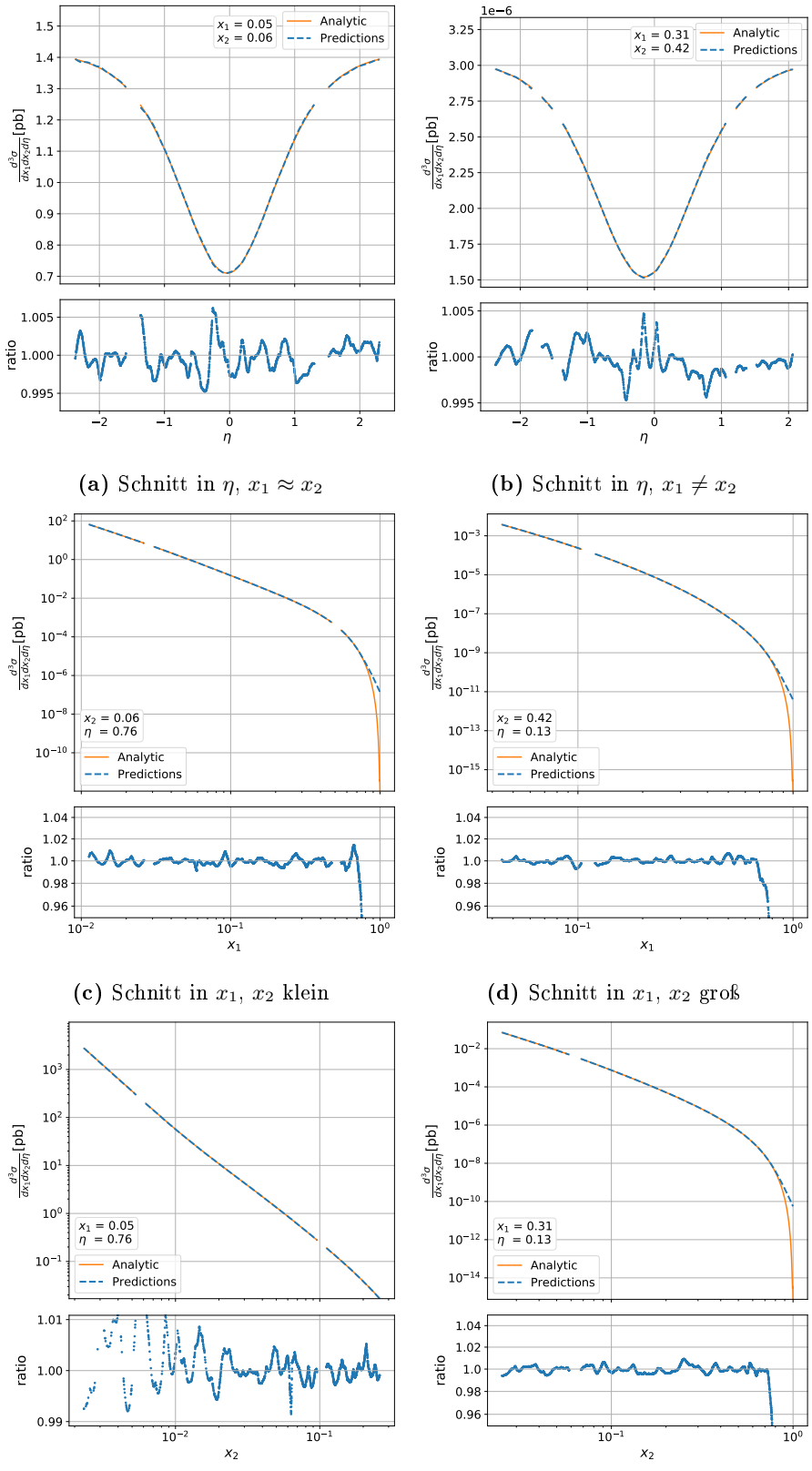
An dieser Stelle wurde ursprünglich das erste mal die Skalierung implementiert, die bereits in den diskutierten Modellen verwendet wurde. Wie sich im Laufe der Arbeit mit den künstlichen neuronalen Netzen gezeigt hat, beugt die Skalierung außerdem das Dying-ReLU-Problem (siehe **??**) vor. Es ist aufgefallen, 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. Schließlich wurde auf die Skalierung der Funktionswerte zurückgegriffen. Andere Versuche um mit dem Dying-ReLU-Problem umzugehen sind die Verwendung von Aktivierungsfunktionen mit nicht-verschwindender Ableitung wie Leaky-ReLU oder ELU (siehe **??**), die den Nodes ermöglichen soll, sich zu regenerieren, oder die Übergabe eines *Clipvalue/Clipnorm*-Parameters an den Optimizer, der den Gradienten reguliert.
Im Folgenden wird die Skalierung 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 ist. 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[3]. Die Label-Normalisierung aus dem vohergehenden Modell wurde praktisch auch im Transformator umgesetzt.
Zur Hyperparameteroptimierung verwenden wir wieder einen Random Search. Mit anderen Suchmethoden wie Bayesian Search oder Hyperband konnte keine Verbesserung der Hyperparametersuche festgestellt werden. Die Hyperparameter einer erfolgreichen Suche sind in **??** aufgelistet. Hierbei muss man beachten, dass die Trainingspunkte zufällig generiert sind nach den Verteilungen:

$$\begin{aligned}
\rho(x) &= \frac{1}{(x+\alpha)\ln\left(\frac{x_{max}+\alpha}{x_{min}+\alpha}\right)} \qquad \text{mit} \qquad \alpha = 0.005 \\
\rho(\eta) &= \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\eta-\eta_{max})^2}{2\sigma^2}\right) & \text{für} \quad 0 \le \eta \le \eta_{max} \\ \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\eta+\eta_{max})^2}{2\sigma^2}\right) & \text{für} \quad -\eta_{max} \le \eta < 0 \end{cases} \qquad \text{mit} \quad \sigma = 1.5
\end{aligned} \tag{0.48}$$

Bei gegebenen Parametern und Nach Anwendung der Cuts(**??**) beläuft sich die Sample-Effizienz auf bei gegebenen Parametern auf $\approx 40\%$. Wir trainieren also mit weniger Daten, als es auf den ersten Blick wirkt.

In den Abbildungen **??**, **??**,... sind Schnitte des dreidimensionalen Wirkungsquerschnittes an verschie-

[3] 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

Abbildung 0.7: Schnitte des differentiellen Wirkungsquerschnitts in x_1, x_2, η

denen Phasenpunkten gezeigt. Wir sehen, dass sich unsere intensive Behandlung der Hyperparameter und der Daten-Transformationen bezahlt gemacht hat. Wir verzeichnen an den meisten Stellen eine maximale Abweichung von lediglich 0.5%. Für Ausnahmefälle beträgt die Abweichung 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 unser Modell also den Wirkungsquerschnitt gegebenenfalls um Größenordnungen überschätzen. Da jedoch nur der integrierte Wirkungsquerschnitt über x_1, x_2 überhaupt messbar ist und dieser sich nicht merklich durch diese Abweichung beeinflussen lässt, ist die große Abweichung in diesem Phasenraumbereich vernachlässigbar.

0.11.3 Vergleich von Hyperparametern

Wir wollen nun noch einmal direkt die Auswirkungen der einzelnen Hyperparameter an unserem Problem untersuchen. Dafür benutzen wir die Konfiguration, die wir im Vorhergehenden durch den Random Search gefunden haben und variieren für jedes Training nur einen Hyperparameter. Da die Anzahl an Neuronen und Layern stark korreliert ist, vergleichen wir auch verschiedene Architekturen. Die Modelle werden nach dem MAPE eines Test-Datestets beurteilt, das genauso gesampelt ist wie die Trainingsdaten. Wir trainieren jedes Modell fünf mal mit zufälligen Initialisierungen an zwei Millionen gesampelten Phasenraumpunkten, um die statistische Schwankung der Güte des Modells einschätzen zu können. Die eingezeichneten Fehlerbalken sollen die Schwankung verdeutlichen und sind kein Maß dafür, welchen MAPE das Netz in der Praxis erreichen kann. Ausreißer sind aus den Messungen herausfiltert, um die Skalierung in sinnvollem Rahmen zu halten.

Loss-Funktion: In **??** a) ist der Vergleich von drei verschiedenen Kostenfunktionen gezeigt. Für Probleme mit stark variierenden Labels setzt sich der Mean-Absolute-Error durch, da dieser nicht so sensitiv auf Ausreißer oder, in unserem Fall, Polstellen ist. Der Huber-Loss, 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 **??** c) überrascht, da generell der Adam-Optimizer in Literatur und auch erfahrungsgemäß die besten Ergebnisse liefert. 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.

Trainingsdaten: Die Größe des Sets an Trainingsdaten ist in **??** 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 keinen signifikanten Beitrag mehr leisten.

Learning-Rate: An dem Vergleich der Learning-Rates, der in **??** gezeigt ist, kann man ein interessantes Verhalten des Netzes erkennen. Für eine Learning-Rate von $1 \cdot 10^{-3}$ verändert sich der mittlere Fehler so gut wie überhaupt nicht, woraus wir schließen können, dass wir unabhängig von unserer Initialisierung bei dieser Learning-Rate das gleiche lokale Minimum finden. Die Unsicherheit der Performance wird danach mit der Learning-Rate größer, wir finden nun höhere und tiefere lokale Minima. Bei einer zu kleinen anfänglichen Lernrate, bleiben wir schon früh in einem hohen Minima stecken.

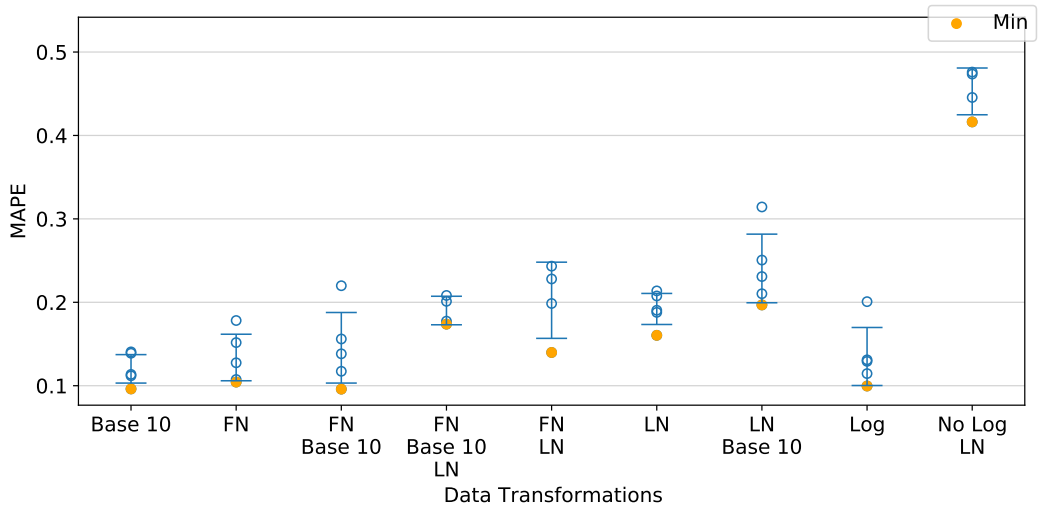


Abbildung 0.8: wichtig: die Daten-Transformationen ohne die überhaupt nichts geht

Es kann daher gut sein kann, seine anfängliche Learning-Rate etwas größer zu initialisieren, als man intuitiv für richtig halten würde, 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 **??** gezeigt. Ich möchte an dieser Stelle noch einmal die Wichtigkeit dieser Transformationen hervorheben. Während man in der Literatur viel über die Normalisierung oder das Reskalieren der Features liest, 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. **??** zeigt, dass verschiedene Implementationen brauchbare Ergebnisse liefern und es wichtiger ist, überhaupt die Skalierung und den Logarithmus anzuwenden. Trainingsläufe ohne Skalierung, noch Logarithmus sind nicht aufgeführt, da der Fehler nicht vergleichbar ist.

Achitektur: Die Architektur in **??** 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 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. 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 **??**. 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 **??**) mit passender Hardware

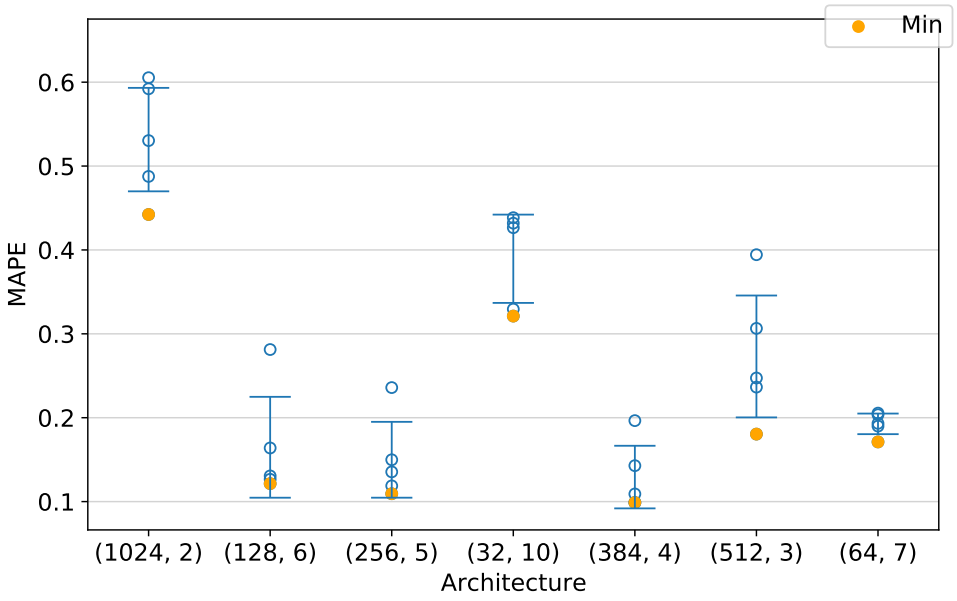
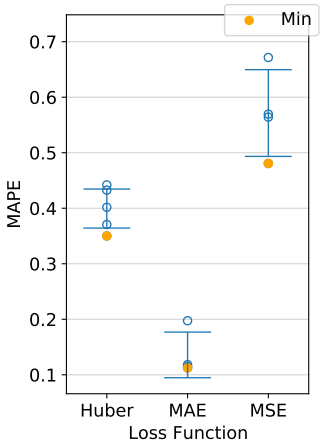
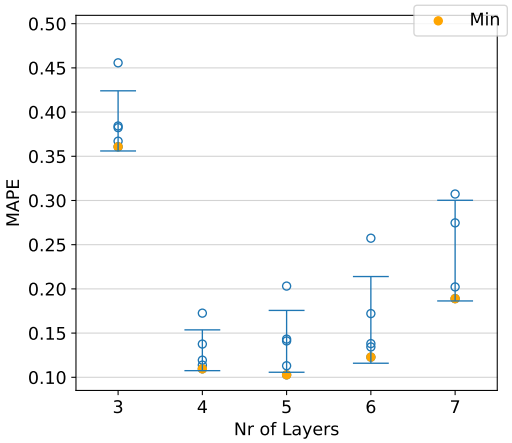


Abbildung 0.9: Vergleich für verschiedene Modell-Architekturen. x-Labels in (Units, Nr of Layers)

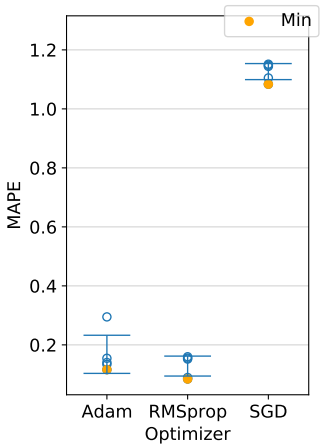
zwar schneller, zeigt jedoch eindeutig größere Abweichungen. Aufgrund von zu wenigen Messdaten ist nicht klar, ob ein Modell, das mit großen Batches trainiert wird, in der gleichen Zeit, oder überhaupt, so tiefe Minima wie die kleineren erreichen können.



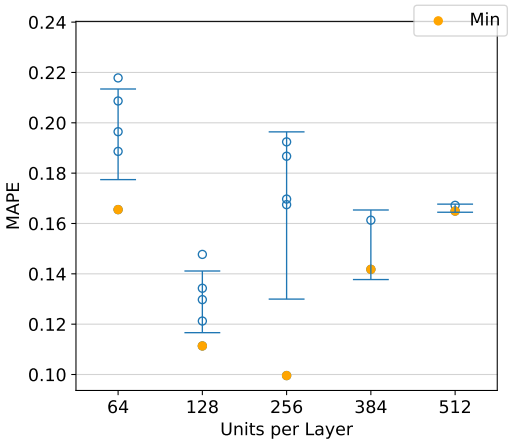
(a) Vergleich für verschiedene Loss-Funktionen



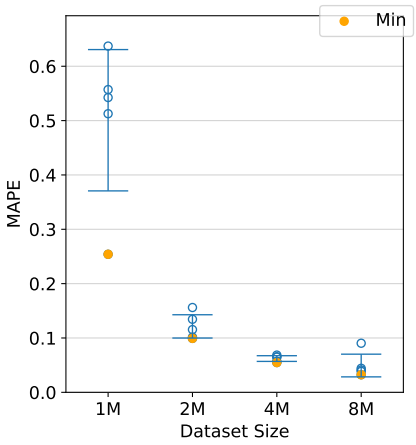
(b) Vergleich für verschiedene Zahlen an Layern



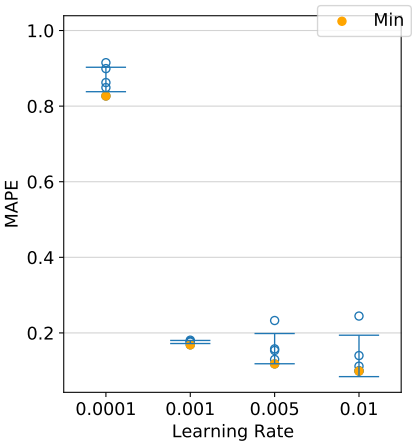
(c) Vergleich für verschiedene Optimizer



(d) Vergleich für verschiedene Zahlen an Units

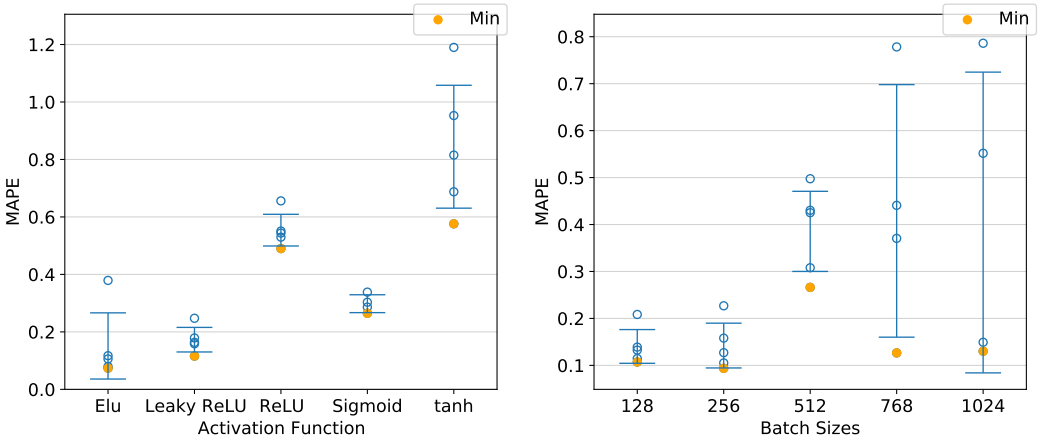


(e) Vergleich für verschiedene Anzahl an Trainigspunkten



(f) Vergleich für verschiedene Anfangs-Lernraten

Abbildung 0.10: Vergleich von Hyperparametern (I)



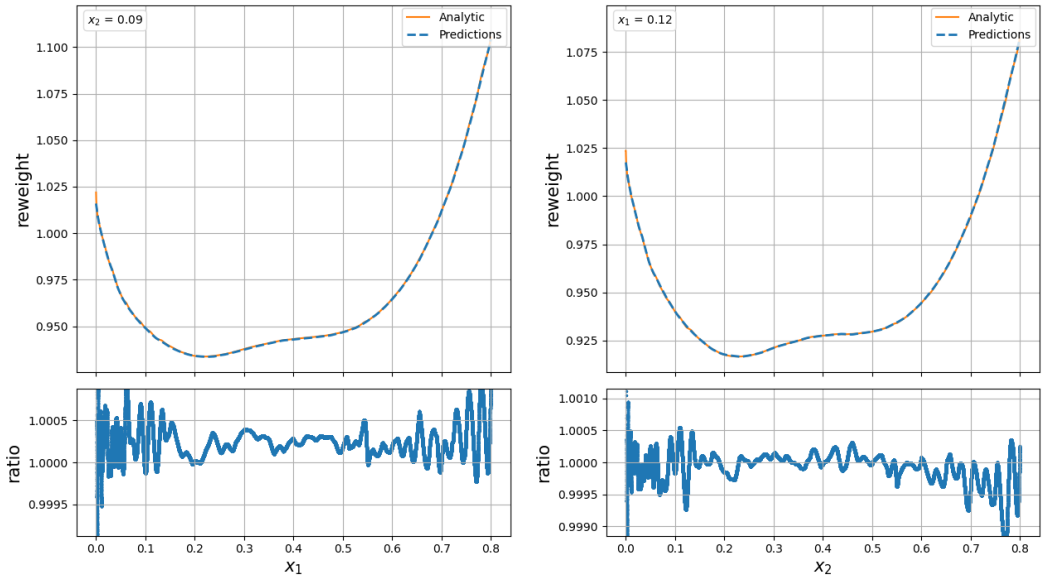
(a) Vergleich für verschiedene Activation-Functions

(b) Vergleich für verschiedene Batch-Sizes

Abbildung 0.11: Vergleich von Hyperparametern (II)

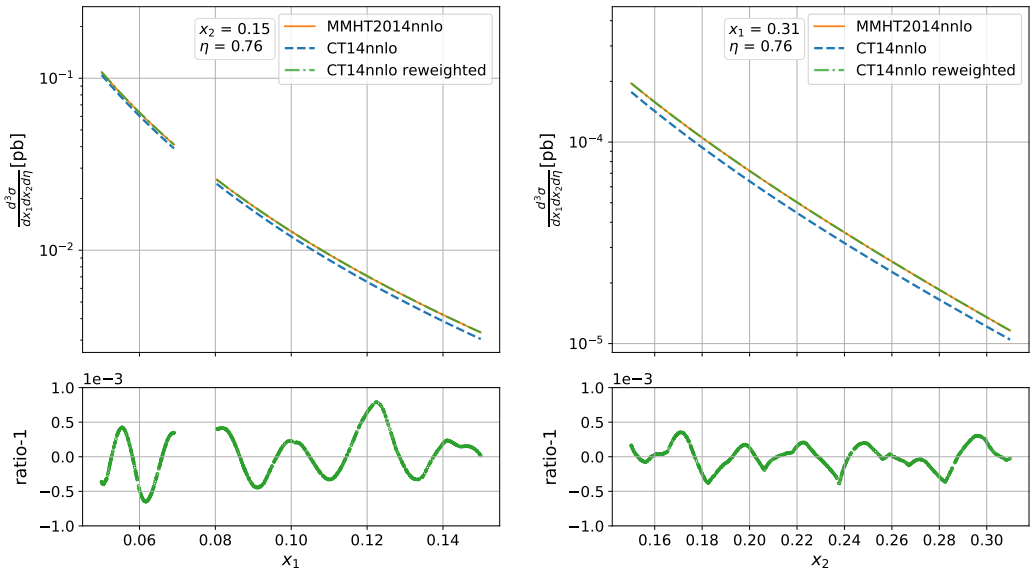
0.12 Reweight zwischen Fits der PDF

Als nächstes möchten wir ein neuronales Netz verwenden, um die Reweights (**??**) zwischen Wirkungsquerschnitten, die mit verschiedenen Anpassungen der PDFs berchnet wurden, zu lernen. Explizit verwenden wir im Folgenden die Fits glqq CT14nnlo“ und „MMHT2014nnlo“ . Die Reweights schwanken um eins und haben keine Polstellen oder Ähnliches und sind damit bereits numerisch in einem geeigneten Bereich. In Phasenraumbereichen mit großen x weichen die Fits jedoch stark voneinander ab und die Gewichte beginnen teils zu oszillieren. Angelehnt an den Phasenraumschnitt aus **??**, trainieren wir die Reweights bis zu $x_{max} = 0.8$, da die starken Abweichungen etwas später beginnen.
Der Random-Search mit den besten Parametern ist in **??** zu sehen.
Wie erwartet, ist die Genauigkeit des Modells sehr gut, wie wir in **??** beobachten können. Die Abweichung beträgt generell weniger als 0.1% und ist somit kaum von den analytisch berechneten Werten zu unterscheiden. Wie das gelernte Reweight in der Praxis funktioniert, ist in **??** dargestellt. Auch hier können wir nur minimale Abweichungen verzeichnen. Das Ratio in **??** c) ist wie erwartet konstant, da das Reweight nicht von η abhängt (siehe **??**).

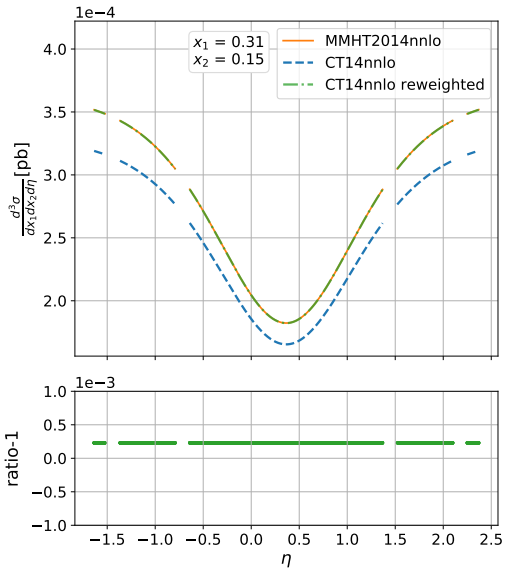


(a) Schnitt in x_1 **(b)** Schnitt in x_2

Abbildung 0.12: Vlt noch 2 3D-plots, einmal die analytischen Reweights, gelernte Reweights werden aber gleich aussehen bei der geringen abweichung



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



(c) Schnitt in η

Abbildung 0.13: Reweight von CT14nnlo auf MMHT2014nnlo mittels gelernten Weigths

0.13 Transfer-Learning zwischen Fits der PDF

Eine weitere Möglichkeit, den Wirkungsquerschnitt, der mit einem anderen PDF-Set berechnet wurde, zu ermitteln, ist Transfer-Learning (siehe **??**). Mit Transfer-Learning können wir den, in den relevanten Phasenraumbereichen kleinen, Unterschied ausgleichen und mit wenig Aufwand gute Modelle für andere PDF-Fits erhalten. Wir nutzen wieder einen Random-Search, um gute Hyperparameter für den Transfer zu finden. Diese können in **??** nachgelesen werden.

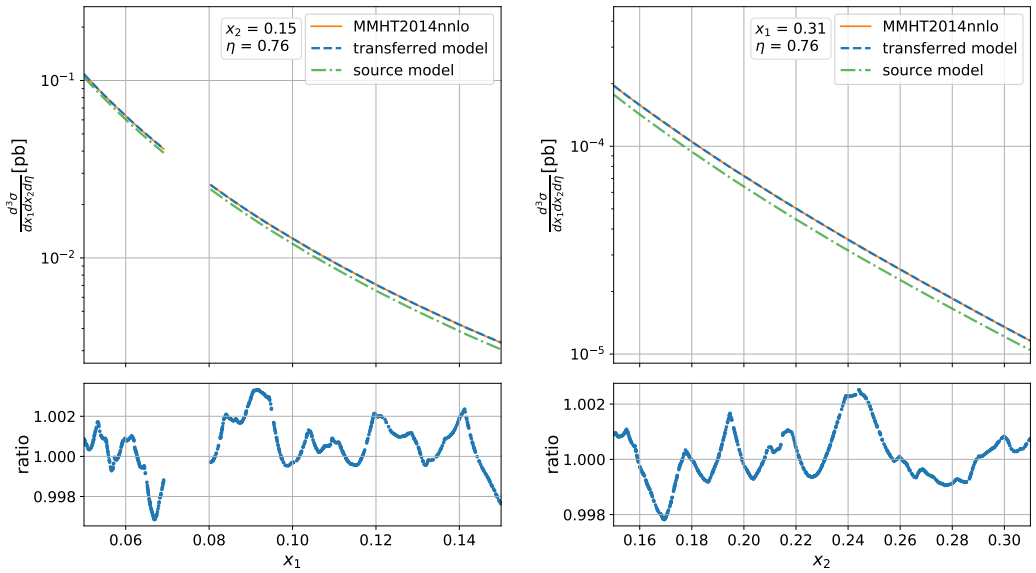
In **??** sind Schnitte des differentiellen Wirkungsquerschnitts mit transferiertem Modell, dem Modell das als Quelle gedient hat, und den analytischen Werten gezeigt. Wir können beobachten, dass die Genauigkeit des transferierten Modells fast identisch mit der des Source-Modells in **??** ist. Weder verliert das Modell beim Transfer an Genauigkeit, noch beobachten wir eine Verbesserung der Performance, die dem Transfer-Learning manchmal zugesprochen wird.
Eindeutig können wir sehen, dass sich die Menge an Lerndaten und damit auch die Trainingsdauer stark 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 **??**, 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 Nodes ähneln.

In **??** sind einige Eigenschaften gegenübergestellt. Wir sehen, dass das präziseste und schnellste Modell das Reweighting der analytisch berechneten Werte ist. Verfügt man also bereits über eine große Anzahl an Werten von differentiellen Wirkungsquerschnitten, dann ist dies das Modell der Wahl. Möchten wir jedoch über ein vollständiges Modell verfügen, dass nicht auf die analytische Berechnung von differentiellen Wirkungsquerschnitten angewiesen ist, schneidet das Modell „Transfer + FT“ am Besten ab. Es sticht seinen Konkurrenten „Reweight + Source“ in allen betrachtet Kriterien aus. Da im zweiten Fall sowohl die Reweights mit einem Netz, als auch die Source-Wirkungsquerschnitten mit neuronalen Netzen berechnet werden, verdoppelt sich hier die TPM im Vergleich zu den restlichen Modellen. Transferieren wir das Source-Model auf eine geeignete Art, passt sich das Netz gut an die neuen Daten an und der MAPE bleibt minimal. Gewichten wir jedoch die Ergebnisse des Source-Models neu, pflanzen sich beide Ungenauigkeiten fort und die Unsicherheit steigt etwas.

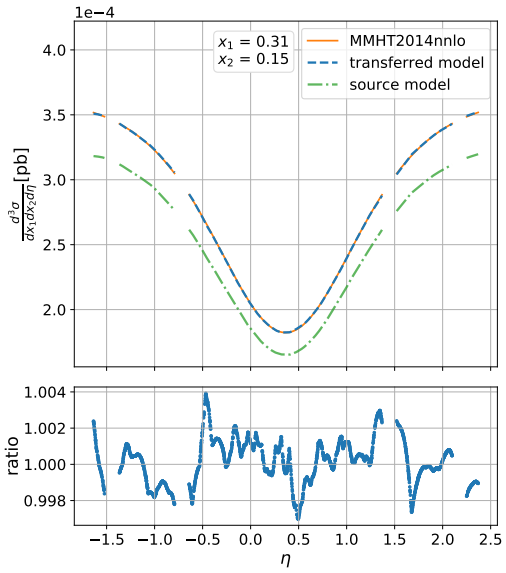
Modell	MAPE	Training[s]	Punkte	TPM[s]
Reweight + Source	0.076	243.42	1M	30.60
Reweight + Analy.	0.017	243.42	1M	13.84
Transfer	0.204	85.78	1M	15.73
Transfer + FT	0.064	164.83	1M	15.65
Source-Model	0.229	841.46	4M	15.81

Tabelle 0.3: Vergleich von Reweight- und Transfer-Modellen TPM: Time per Million, Berechnungszeit für 10^6 Punkte

Wir kommen zum Schluss, dass das Transfer-Learning eine generell bessere Methode für den angesprochenen Zweck ist und das Erlernen des Reweights zwar gut funktioniert, jedoch nur nützlich ist, wenn man für speziell die Gewichte benötigt.

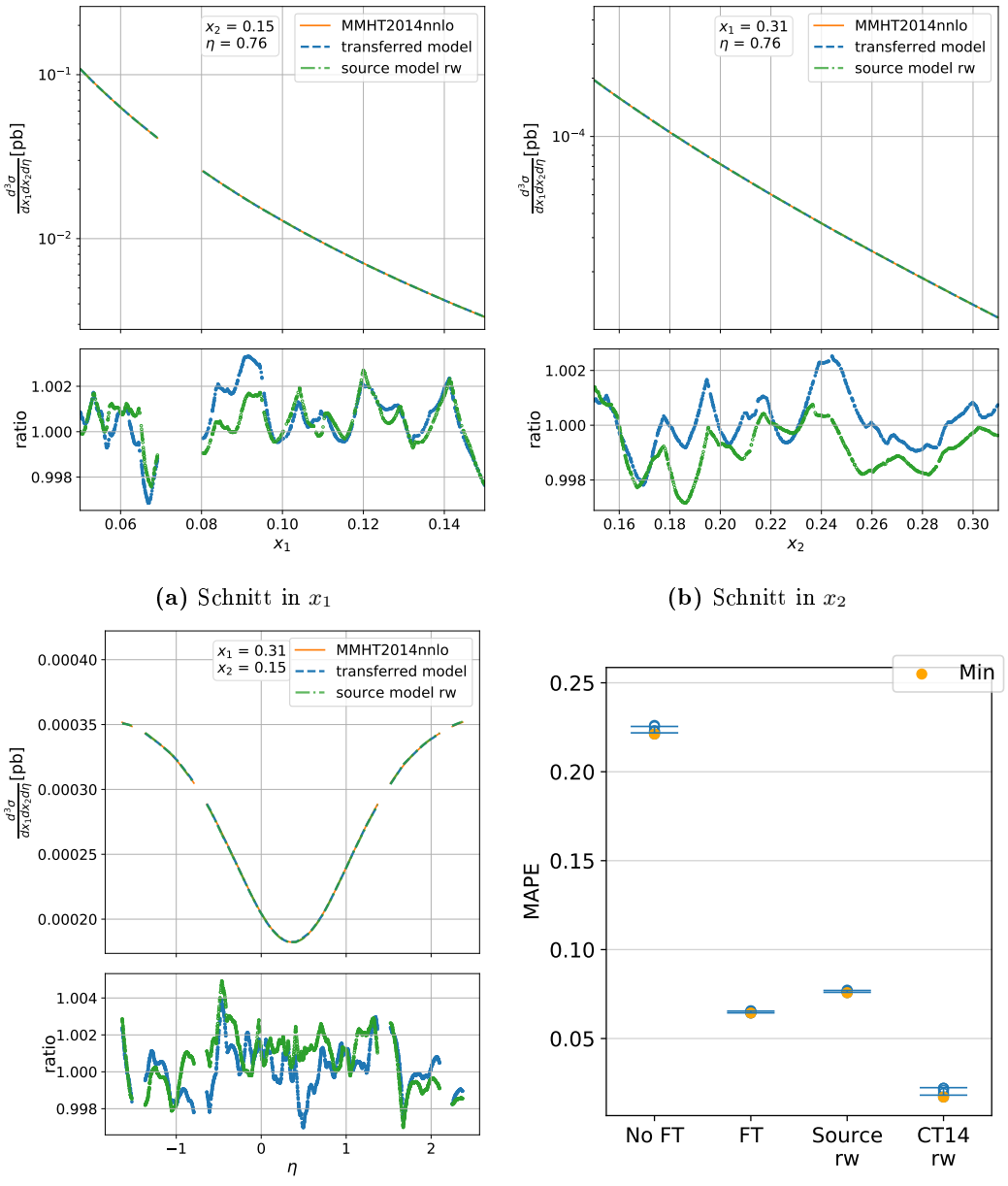


(a) Schnitt in x_1 **(b)** Schnitt in x_2



(c) Schnitt in η

Abbildung 0.14: Transferiertes Model von Source Model zum Transfer model



(a) Schnitt in x_1

(b) Schnitt in x_2

(c) Schnitt in η

(d) Vergleich

Abbildung 0.15: Vergleich transferiertes Modell, gereweightetes Source Model rw: reweighted

0.14 Monte-Carlo-Integration

0.14.1 Parton-Ebene

Wir nutzen Monte-Carlo-Methoden zur Integration von **??**, **??** und deren zugehörigen Machine Learning Modellen. Zur Integration von **??** nutzen wir das Importance Sampling aus **??**, um die Konvergenz des Integrals zu beschleunigen. Der Prozess $qq \to \gamma\gamma$ ist zwar nicht messbar, jedoch müssen wir trotzdem einen Cut in η festlegen, da der totale Wirkungsquerschnitt sonst divergiert. Wir entscheiden uns für die Beschränkungen **??**.

$$|\eta| \leq 2.5 \qquad \Rightarrow \qquad \theta \in [\epsilon, \pi - \epsilon] \quad \text{mit} \quad \epsilon = 0.1638 \tag{0.49}$$

Die Unsicherheit auf unsere Monte-Carlo-Integration bestimmen wir aus **??**. Wir führen die Integrationen mit 1000 Stützstellen durch und wiederholen die Integration 100 mal. In **??** sind die erhaltenen Ergebnisse mit dem analytischen Wert verglichen. Wir sehen, dass die neuronalen Netze so präzi-

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 0.4: Monte-Carlo-Integration des partonischen Diphoton Prozesses

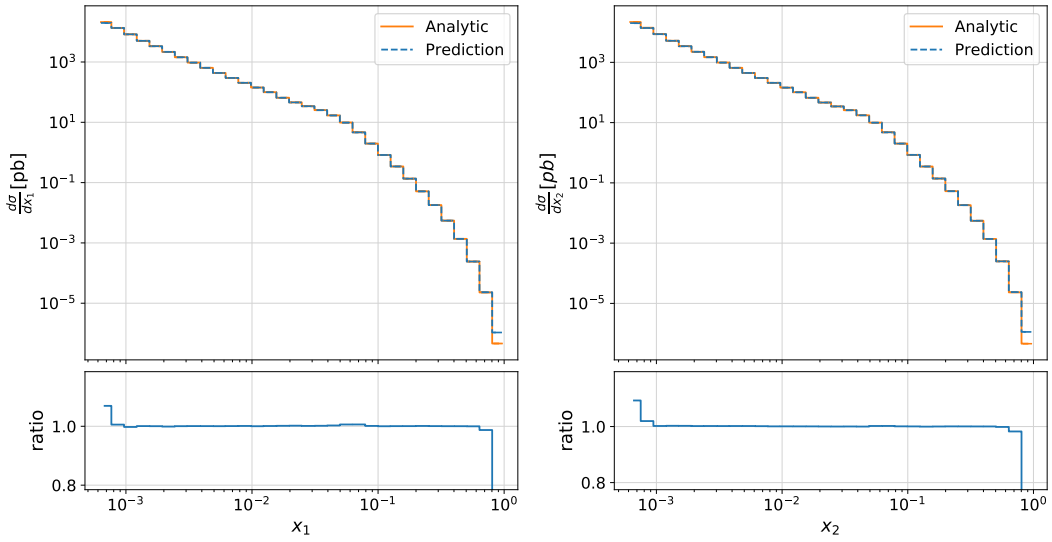
se sind, dass ihre Abweichung in der Unsicherheit der Monte-Carlo-Integration untergeht. Das sind gute Voraussetzungen für die Anwendbarkeit von neuronalen Netzen auch bei höherdimensionalen Prozessen. Das simple Importance-Sampling bringt eine signifikante Varianz-Verringerung mit sich.

0.14.2 Hadron-Ebene

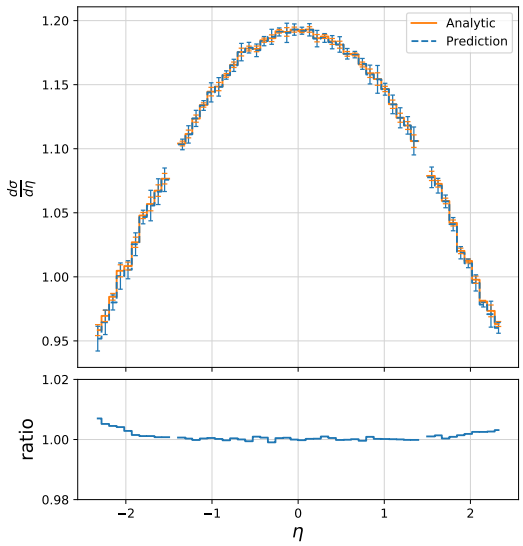
Auch für den hadronischen Diphoton-Prozess nutzen wir Importance-Sampling. Für die Generation der Impulsbruchteile x verwenden wir die Verteilung aus **??**. Aufgrund der Cuts leisten Phasenraumpunkte mit kleinem η einen Größeren Beitrag zum messbaren σ_{tot}, wir ziehen daher Punkte aus einer Gaußverteilung um Null(**??**).

$$\rho(\eta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\eta)^2}{2\sigma^2}\right) \quad \text{mit} \quad \sigma = 2 \tag{0.50}$$

Wir integrieren zunächst über zwei Freiheitsgrade und sehen uns die Wirkungsquerschnitte in Abhängigkeit von x_1, x_2 und η an. Wir samplen dazu 50.000.000 Punkte, mit gleicher Sample-Effizienz wie in **??** und wiederholen den Prozess fünf mal. Die Ergebnisse sind in **??** dargestellt. Wie wir sehen, überdecken sich die integrieren Wirkungsquerschnitte für die analytischen Werte und die Vorhersagen des neuronalen Netzes an vielen Phasenpunkten. Lediglich für große x überschätzt die Vorhersage wie erwartet den eigentlichen Wert. Diese Abweichung kann jedoch vernachlässigt werden, da wir am Ratio von **??** c) sehen, dass die analytischen Werte insgesamt unterschätzt werden. Der Grund hierfür liegt vermutlich in der Polstelle an $x = 0$. Durch die Cuts werden viele Phasenraumpunkte um



(a) Integration über x_2, η **(b)** Integration über x_1, η



(c) Integration über x_1, x_2

Abbildung 0.16: MC-Integrationen über zwei Freiheitsgrade

$x_1 = x_2 = 0$ mit großem Wirkungsquerschnitt herausgenommen, da sie die p_T-Hürde nicht erfüllen und führt voraussichtlich zu einer geringfügigen Unterschätzung des Wirkungsquerschnittes an diesen einflussreichen Stellen.

Zur Integration über alle Freiheitsgrade verwenden wir die gleichen Daten wie im vorherigen Abschnitt. Wir erhalten die Ergebnisse **??**.

$$\begin{aligned} \sigma_{\text{tot}}^{\text{analytic}} &= (5.1707 \pm 0.0038)\ \text{pb} \\ \sigma_{\text{tot}}^{\text{ml}} &= (5.1634 \pm 0.0038)\ \text{pb} \end{aligned} \tag{0.51}$$

Dies entspricht einer Abweichung von 0.14%, wobei die Unsicherheiten der Monte-Carlo-Integration zusätzlich überlappen. Wir können also abschließend sagen, dass die Präzision des DNN sehr gut und die Näherung erfolgreich gelungen ist.

Zusammenfassung und Ausblick

0.15 Zusammenfassung

In dieser Arbeit wurde der Diphoton Prozess als $q\bar{q} \to \gamma\gamma$ und $pp \to \gamma\gamma$ auf Leading-Order Ebene behandelt und Ausdrücke für die jeweiligen differentiellen Wirkungsquerschnitte hergeleitet. An dies Beispielen wurde dann die Eignung von tiefen neuronalen Netzwerken zur Näherung des Integranden überprüft. Dabei mussten verschiedene Tücken bedacht und behandelt werden. In diesem Kontext wurde die Wichtigkeit von Label-Transformationen und der Architektur des Neuronalen Netzes deutlich. Anschließend wurden die Gewichte zwischen den Fits der Partondichtefunktionen CT14nnlo und MMHT2014nnlo erlernt und angewendet. Schließlich wurde noch die Möglichkeit des Transfer-Learning zwischen Modellen, die an verschiedenen PDF-Sets trainiert wurden überprüft. Mithilfe von Monte-Carlo-Methoden wurden die differentiellen Wirkungsquerschnitte integriert.
Wir erwartet, haben die neuronalen Netze keine Probleme mit simplen Regressionsaufgaben, wie die Wirkungsquerschnitte des Prozesses $q\bar{q} \to \gamma\gamma$ darstellen. Die Funktionswerte können mit ausgezeichneter Genauigkeit und wenig Aufwand vorhergesagt werden.
Andererseits ist der differentielle Wirkungsquerschnitt des Prozesses $pp \to \gamma\gamma$ wiederum nicht trivial. Hier müssen Hürden wie das „Dying-ReLU“ Problem und die passende Wahl der Loss-Function beachtet werden. Auch kann es hier helfen kaum beitragende Phasenraumbereiche zu vernachlässigen, um die Spanne an Größenordunungen, über die sich die Wirkungsquerschnitte verteilen, zu verkleinern. Letztendlich kann mit etwas Feingefühl und Erfahrung im Umgang mit neuronalen Netzen jedoch passende Modelle gefunden werden, die gute Genauigkeit zeigen.
Das Erlernen der Reweights stellt aus Sicht des neuronalen Netzes kein Problem dar, solange ein sinnvoller Phasenraumbereich gewählt wird. Hier kann das Netz die Funktionswerte mit exzellenter Genauigkeit vorhersagen. Das neuronale Netz kann also hier gut als Interpolation zwischen den analytischen Werten dienen.
Transfer-Learning stellt sich als eine gute Möglichkeit heraus, aus einem bereits vorhandenen Modell, ein Modell für einen anderen Fit von Partondichtefunktionen zu erhalten. In puncto Berechnungs- und Trainingsgeschwindigkeit und Genauigkeit übertrifft das Transfer-Learning hier das Reweighten eines bereits vorhandenen Source-Models.

0.16 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 realistischen Zahl an Trainingspunkten möglich ist. Anschließend muss überprüft

werden, wie groß der rechentechnische Nutzen der neuronalen Netze. In den einfachen, analytischen Beispielen von uns 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 dafür bekannt mit sehr hochdimensionalen Eingangswerten umgehen zu können.

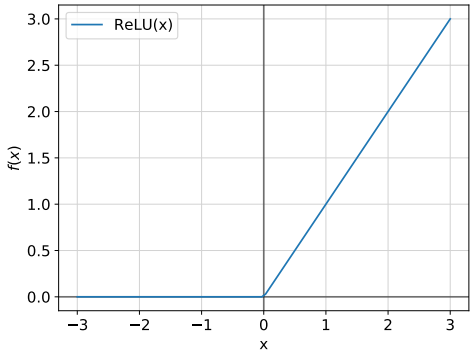
Auch das Transfer-Learning hat in dieser Arbeit seine Funktionalität bewiesen. Es muss jedoch nicht beim Transfer zwischen PDF-Sets bleiben. In der Praxis wird Transfer-Learning zwischen viel diverseren Datensets eingesetzt. Es könnte sich also lohnen, auch den Transfer zwischen sich ähnelnden Prozessen in der Teilchenphysik zu untersuchen. Ich spreche hier von 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...

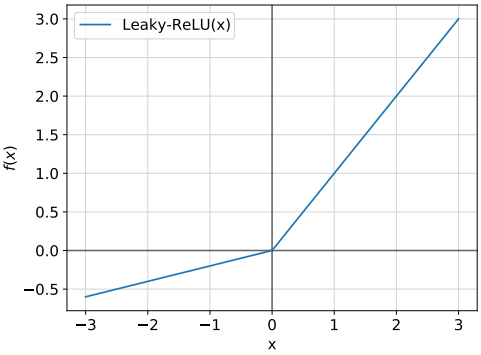
Anhang

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	

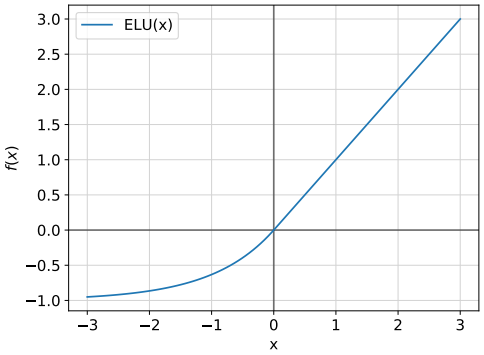
Tabelle .5: Parameter der Random-Search für $\frac{d\sigma}{d\theta}$ mit Ergebnis



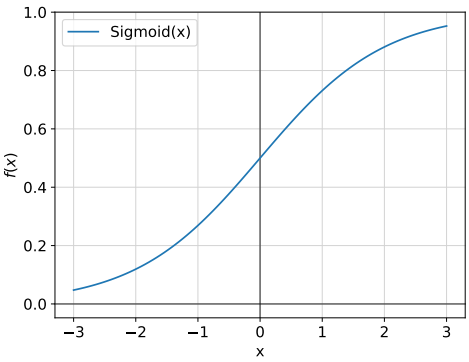
(a) Rectified Linear Unit: $f(x) = \max(0, x)$



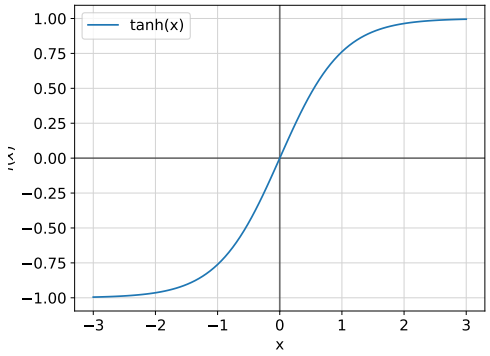
(b) Leaky-ReLU: $f(\alpha, x) = \alpha x$ für $x < 0$



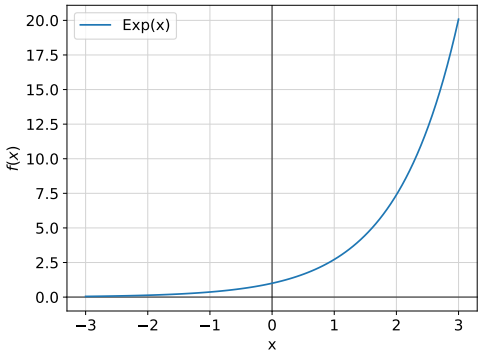
(c) ELU: $f(\alpha, x) = \alpha(e^x - 1)$ für $x < 0$



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



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



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

Abbildung .17: Activation-Funktions, die verwendet wurden

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	$\{$keine, $[-1,1]\}$	keine
Feature-Normal.	True, False	True
Skalierung	True	
Logarithmus	True	
Max. Epochen	100	
Trainingspunkte	4.000.000	

Tabelle .6: Hyperparameter Pools eines erfolgreichen Random-Search mit bester Konfiguration für den dreidimensionalen differentiellen Wirkungsquerschnitt des Diphoton Prozesses

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	$\{$keine, $[-1,1]\}$	keine
Feature-Normal.	True, False	True
Skalierung	False	
Logarithmus	False	
Max. Epochen	100	
Trainingspunkte	1.000.000	

Tabelle .7: Hyperparameter Pools eines Random-Search mit bester Konfiguration für ein Reweighting des differentiellen Wirkungsquerschnitt des Diphoton Prozesses

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
Learning-Rate Loss-Funktion	MAE	
Optimizer	Adam	
Max. Epochen	100	
Trainingspunkte	1.000.000	

Tabelle .8: Hyperparameter Pools eines Random-Search mit bester Konfiguration für Transfer-Learning zwischen Wirkungsquerschnitten verschiedener PDF-Sets

Callback	Config
LearningRateScheduler	nach einem Offset 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.

.1 Abkürzungen

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
Nodes	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
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
TPM	Time per Million, Zeit die das DNN benötigt, um 10^6 Phasenraumpunkte zu evaluieren

Tabelle .9: Häufig genutzte Abkürzungen und Fachvokabular

.2 Grafiken

.3 Source-Code

Erklärung

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

Vorname Nachname
Dresden, Monat 2019