IEKP-Bachelor-KA/2017-03

Trennung von Signal und Untergrund mit TensorFlow in einer Suche nach $t\bar{t}H$-Produktion am CMS-Experiment

Separation between signal and background using TensorFlow in a search for $t\bar{t}H$-production at the CMS-Experiment

Bachelorarbeit

Maximilian Welsch

An der Fakultät für Physik
Institut für Experimentelle Kernphysik (IEKP)

Erstgutachter: Prof. Dr. Ulrich Husemann
Zweitgutachter: Dr. Matthias Schröder

Karlsruhe, 17. März 2017

Diese Arbeit wurde vom Erstgutachter der Bachelorarbeit akzeptiert.

Karlsruhe, den 17. März 2017

. . . . . .

(**Prof. Dr. Ulrich Husemann**)

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, den 17. März 2017

.....

(**Maximilian Welsch**)

Inhaltsverzeichnis

1 Einleitung **1**

2 Das Standardmodell der Teilchenphysik **3**

3 Experimentelle Grundlagen **5**
3.1 Das CMS-Experiment am Large-Hadron-Collider 5
3.2 $t\bar{t}H$-Analyse 7

4 Neuronale Netze **9**
4.1 Aufbau eines FeedForward-Netzes 9
4.2 Training mit Fehlerrückführung 11

5 Aufbau des Analyseframeworks NNFlow basierend auf TensorFlow **15**

6 Studien zur Parameterwahl beim Einsatz von neuronalen Netzen unter Verwendung von NNFlow **19**
6.1 Umgang mit gewichteten Trainingsereignissen 19
6.2 Anzahl der Neuronen und verborgenen Schichten 24
6.3 Wahl der Ereignisvariablen 30

7 Fazit und Ausblick **33**

Literaturverzeichnis **35**

Anhang **39**
A Verwendete Variablen zur Studie zum Umgang mit gewichteten Ereignissen 39
B BDT-Variablen in der $\geq$ 6, 3-btag Kategorie 40
C Evt+Jet-Variablen in der $\geq$ 6, 3-btag Kategorie 47

1 Einleitung

Der Large-Hadron-Collider (LHC) beschleunigt Protonen auf Kreisbahnen in entgegengesetzter Richtung auf eine Energie von bis zu 7 TeV. Dabei untersuchen verschiedene Experimente, wie das Compact-Muon-Solenoid-Experiment (CMS), Kollisionen dieser hochenergetischen Teilchen um die grundlegenden Gesetze der Natur zu verstehen. Der bisher größte Erfolg der Experimente am LHC ist die Entdeckung des Higgs-Bosons durch die CMS- [1] und ATLAS-Kollaboration [2] im Jahr 2012. Aktuelle Forschungsgebiete bei CMS sind die Vermessung der grundlegenden Eigenschaften des Higgs-Bosons, wie z.B. die Yukawa-Kopplung an Fermionen, oder die Suche nach neuer Physik. Bei einer Schwerpunktsenergie von bis zu $\sqrt{s} = 14\,\text{TeV}$ finden bis zu 10^9 Proton-Proton-Kollisionen pro Sekunde statt. Die enormen Datenmengen, die der Detektor dabei aufzeichnen würde, können unmöglich vollständig ausgelesen werden. Zudem finden die meisten Kollisionen in einem so niedrigen Energiebereich statt, dass sie keine neuen Erkenntnisse mit sich bringen würden. Deshalb werden verschieden Trigger-Systeme eingesetzt, um nur die interessanten Ereignisse auszuwählen, wodurch nur einige 100 Ereignisse pro Sekunde gespeichert werden. Bei der Suche nach neuen Phänomenen fällt nur ein kleiner Teil dieser Ereignisse in die Signalkategorie, also genau die Prozesse, die tatsächlich untersucht werden sollen. Die restlichen Ereignisse können in die Untergrundkategorie eingeordnet werden. Oft ist man daran interessiert, dass Signal-zu-Untergrund-Verhältnis zu vergrößern, wobei z.B. nur bestimmte Bereiche einzelner Ereignisvariablen betrachtet werden. Zudem werden in der Teilchenphysik Algorithmen des Maschinellen Lernens zur Klassifikation von Signal- und Untergrundereignissen verwendet. Dabei werden diese Algorithmen mit Ereignissen bekannter Klassifikation trainiert und lernen dadurch Zusammenhänge aus den vorhandenen Daten, die sie dazu befähigen neue, unbekannte Ereignisse zu klassifizieren.

Für die Suche nach der Produktion eines Higgs-Bosons in Assoziation mit einem Top-Quark-Antiquark-Paar ($\text{t}\bar{\text{t}}\text{H}$) werden bei CMS insbesondere Boosted-Decision-Trees (BDTs) zur Signal- und Untergrundtrennung eingesetzt. BDTs haben den Vorteil, dass sie schnell zu trainieren sind und darüber hinaus eine einfach Interpretation ihrer Funktionsweise erlauben. Neben BDTs können auch neuronale Netze (NN) zur Signal- und Untergrund-Trennung benutzt werden. Die Tatsache, dass sogenannte tiefe neuronale Netze (engl. Deep Neural Network, DNN) in den letzten Jahren erfolgreich in der Bild- und Spracherkennung eingesetzt werden, macht diese auch für den Einsatz in der Teilchenphysik interessant. Dabei sorgt der Einsatz von GPU-beschleunigten Berechnungen für eine deutlich kürzere Trainingsdauer der NN, welche zuvor einer der großen Nachteile dieser Technik war. Mit der Veröffentlichung der Software-Bibliothek TensorFlow steht nun ein nützliches Werkzeug

bereit um NN mittels GPU-Beschleunigung zu trainieren. Ziel dieser Arbeit ist es ein Framework rund um TensorFlow aufzubauen, um möglichst einfach NN für die Klassifikation von $t\bar{t}H$-Signal- und $t\bar{t}$-Untergrundereignissen einzusetzen. Weiter wird mit Hilfe dieses Frameworks untersucht, wie gut sich ein NN zur Signal- und Untergrund-Trennung für die $t\bar{t}H$-Analyse eignet. Dafür wird überprüft, wie man gewichtete Ereignisse zum Training NN einsetzen kann und wie sich die Netzgröße sowie die Wahl der Ereignisvariablen auf die Klassifikation auswirken.

In Kapitel 2 und 3 werden zunächst theoretische und experimentelle Grundlagen erläutert. Weiter werden in Kapitel 4 alle erforderlichen Grundlagen zur Funktionsweise neuronaler Netze eingeführt. Der Arbeitsablauf mit dem TensorFlow-Framework NNFlow wird in in Kapitel 5 vorgestellt. Anschließend sind in Kapitel 6 Studien zur Verwendung von gewichteten Ereignissen beim Training von NN und zum Einfluss der Netzwerkgröße auf Leistungsfähigkeit von NN bei der $t\bar{t}H$-Signal-Trennung unter Verwendung von NNFlow aufgeführt. Zum Abschluss der Arbeit ist in Kapitel 7 eine Zusammenfassung sowie ein Ausblick zu finden.

2 Das Standardmodell der Teilchenphysik

Das Standardmodell der Teilchenphysik (SM) beschreibt das gegenwärtige Verständnis der Elementarteilchen und deren Wechselwirkungen. Im Folgenden werden die grundlegenden Eigenschaften der verschiedenen Teilchen und Wechselwirkungen erläutert, wobei sich die Darstellung an [3] und [4] orientiert.

Bei Elementarteilchen unterscheidet man zwischen Fermionen und Bosonen. Zu den Fermionen gehören sechs Leptonen und sechs Quarks, die in drei Familien angeordnet werden (vgl. Tab. 2.1). Zu jedem Fermion gibt es ein entsprechendes Anti-Fermion gleicher Masse aber konjugierter elektrischer Ladung. Zu den Bosonen gehören vier Eichbosonen (vgl. Tab. 2.2), die der Vermittlung der verschiedenen Wechselwirkungen (elektromagnetisch, schwach und stark) dienen, sowie das Higgs-Boson.

Die Wechselwirkungen des Standardmodells werden durch Quantenfeldtheorien beschrieben:

- Die elektromagnetische Wechselwirkung kann durch die Quantenelektrodynamik (QED) beschrieben werden. In dieser Theorie dient das Photon γ als Austauschteilchen, das die Wechselwirkung zwischen elektrisch geladenen Teilchen vermittelt. Photonen können nicht mit sich selbst wechselwirken, da sie keine elektrische Ladung tragen.

- Die starke Wechselwirkung wird durch die Quantenchromodynamik (QCD) beschrieben. Die starke Wechselwirkung wird durch das Gluon g vermittelt. Diese koppelt an Teilchen, die Farbladung tragen. Zu diesen Teilchen zählen Quarks, Anti-Quarks und die Gluonen selbst. Die QCD erlaubt die Existenz von einzelnen freien farbgeladenen Objekten nicht. Man kann stattdessen nur gebundene Zustände von Quarks, sogenannte Hadronen, beobachten.

- Die schwache Wechselwirkung, die z.B. für den β-Zerfall oder den Zerfall von schweren Quarks und Leptonen verantwortlich ist, wird durch die massiven $W^{\pm}$- und Z^0-Bosonen vermittelt. Alle links händigen Fermionen und rechts händigen Anti-Fermionen können schwach wechselwirken. Die Eichbosonen können auch untereinander wechselwirken.

Dabei werden die elektromagnetische und schwache Wechselwirkung im Standardmodell durch eine vereinheitlichte Theorie, nämlich die elektroschwache Vereinheitlichung, beschrieben. Dabei führen die Eigenschaften des Z^0-Bosons zu einer Vermischung der elektromagnetischen und schwachen Wechselwirkung. In dieser vereinheitlichten Theorie werden das Photon und das Z^0-Boson als zueinander orthogonale Linearkombinationen der

Tabelle 2.1: Die 12 Fermionen des Standardmodells, aufgeteilt in Quarks und Leptonen sowie in Familien angeordnet. Alle aufgeführten Teilchen sind Spin-$\frac{1}{2}$-Teilchen. Die Zahlenwerte der Massen sind [3] entnommen.

Familie	Teilchen	el. Ladung in e	Masse in GeV
I	Down-Quark (d)	$-\frac{1}{3}$	$0,005$
	Up-Quark (u)	$\frac{2}{3}$	$0,003$
	Elektron (e)	-1	$0,0005$
	Elektron-Neutrino (ν_e)	0	$< 10^{-9}$
II	Strange-Quark (s)	$-\frac{1}{3}$	$0,1$
	Charm-Quark (c)	$\frac{2}{3}$	$1,3$
	Myon (μ)	-1	$0,106$
	Myon-Neutrino (ν_μ)	0	$< 10^{-9}$
III	Bottom-Quark (b)	$-\frac{1}{3}$	$4,5$
	Top-Quark (t)	$\frac{2}{3}$	174
	Tau (τ^-)	-1	$1,78$
	Tau-Neutrino (ν_τ)	0	$< 10^{-9}$

Tabelle 2.2: Die Eichbosonen des Standardmodells. Alle aufgeführten Teilchen sind Spin-1-Teilchen. Die Zahlenwerte der Massen sind [3] entnommen.

Wechselwirkung	Teilchen	el. Ladung in e	Masse in GeV
Elektromagnetisch	Photon (γ)	0	0
Stark	Gluon (g)	0	0
Schwach	W-Boson ($\mathrm{W}^\pm$)	± 1	$80,4$
	Z-Boson (Z^0)	0	$91,2$

Zustände B^0 und W^0 beschrieben, wobei der Weinberg-Winkel θ_W die Mischung beider Zustände angibt [5].

Durch das Higgs-Bosons, das 2012 durch die Experimente ATLAS und CMS am Large-Hadron-Collider (LHC) entdeckt wurde, wird das Standardmodell komplementiert. Das Higgs-Boson hat eine Masse von ungefähr 125 GeV [1, 2] und ist das einzige fundamentale Spin-0-Teilchen im Standardmodell. Die Entdeckung des Higgs-Bosons ist gleichzeitig eine Bestätigung des Higgs-Mechanismus, durch den die Eichbosonen ihre Massen erhalten. Auch die Fermionen erhalten ihre Masse durch eine Kopplung an das Higgs-Feld. Diese Kopplung wird Yukawakopplung genannt und ist proportional zur Masse des Fermions, weshalb sich zur Untersuchung der Yukawakopplung Prozesse anbieten, in denen Top-Quarks als die massivsten fundamentalen Teilchen an das Higgs-Boson koppeln.

3 Experimentelle Grundlagen

Im ersten Abschnitt des Kapitels wird der Aufbau des CMS-Experiments erläutert. Anschließend findet eine Beschreibung der $t\bar{t}H$-Analyse statt, bei der vor allem die für diese Arbeit relevanten Signal- und Untergrundprozesse sowie kurz die allgemeine Analysestrategie erklärt werden.

3.1 Das CMS-Experiment am Large-Hadron-Collider

Der Large-Hadron-Collider (LHC) [6] am CERN ist mit einem Umfang von 27 km der derzeit größte Teilchenbeschleuniger der Welt. Im LHC werden Protonen in entgegengesetzter Richtung auf eine Schwerpunktsenergie von bis zu $\sqrt{s} = 14\,\text{TeV}$ bei einer Luminosität von $L = 10^{34}\,\text{cm}^{-2}\,\text{s}^{-1}$ beschleunigt. Die Motivation für den Bau des LHC war die Suche nach dem Higgs-Boson und Physik jenseits des Standardmodells. Dafür werden die beschleunigten Teilchen an vier Punkten zur Kollision gebracht, wobei sich an diesen Orten die vier Experimente ALICE [7], LHCb [8], ATLAS [9] und CMS [10] befinden. Im folgenden wird nur der Aufbau des Compact-Muon-Solenoid (CMS) Experiments näher erläutert.

Der CMS-Detektor ist ein Vielzweckdetektor, der 100 m unter der Erde am LHC Punkt 5, in der Nähe des französischen Dorfes Cessy, betrieben wird. Der Zusatz „Compact“ im Namen des Detektors kommt daher, da der Detektor bei einem Gewicht von 14 000 t trotzdem nur 21,6 m lang ist und einen Durchmesser von 14,6 m hat. Um den Anforderungen des LHC-Physikprogramms gerecht zu werden, wurde beim Design des Detektors besonders auf eine genaue Myon-Identifikation und -Impulsmessung, sowie auf eine genaue Energiemessung im elektromagnetischen Kalorimeter (ECAL) geachtet. Außerdem wurde auch auf eine genaue Spurmessung geladener Teilchen nahe dem Wechselwirkungspunkt, zur Identifikation von τ-Leptonen und b-Jets, Wert gelegt. Zusätzlich deckt das hadronische Kalorimeter (HCAL) nahezu den gesamten Raumwinkel von 4π ab, um eine genaue Energiemessung der Jets zu gewährleisten. Ein Querschnitt des CMS-Detektors ist in Abb. 3.1 zu sehen. Das CMS-Koordinatensystem hat seinen Ursprung in der Mitte des Detektors. Die x-Achse zeigt radial nach innen zum Zentrum des LHC und die y-Achse zeigt senkrecht nach oben. Die z-Achse des Koordinatensystems zeigt entlang der Strahlrichtung gegen den Uhrzeigersinn. Der Azimutalwinkel ϕ wird ausgehend von der x-Achse in der x-y-Ebene und der Polarwinkel θ ausgehend von der z-Achse gemessen. Die Pseudorapidität ist definiert als $\eta = -\log(\tan(\theta/2))$.

Im Inneren des Detektors befindet sich der Spurdetektor, der ein Volumen eines Zylinders mit 5,8 m Länge und 2,6 m Durchmesser hat und dabei einen Pseudorapiditätsbereich von

$|\eta| < 2,5$ abdeckt. Der Spurdetektor besteht aus zehn Lagen Silizium-Streifendetektoren und zusätzlich drei Lagen Silizium-Pixeldetektoren nahe des Wechselwirkungspunkts. Mit dem Spurdetektor werden die Spuren geladener Teilchen gemessen, wobei aus der Spurkrümmung und dem Magnetfeld der Teilchenimpuls bestimmt werden kann. Zusätzlich können aus den rekonstruierten Teilchenspuren die Positionen von sekundären Vertizes bestimmt werden, um z.B Jets, die aus Bottom-Quarks hervorgehen, zu identifizieren. Beim Design des ECAL im CMS-Detektor wurde besonders auf eine gute Energieauflösung geachtet, um das Higgs-Boson im Zerfallskanal $H \rightarrow \gamma\gamma$ nachzuweisen. Deshalb ist das ECAL des CMS-Detektors als ein homogenes Kalorimeter, bestehend aus Bleiwolframat-Kristallen ($PbWO_4$), realisiert. Dabei deckt das ECAL einen Pseudorapiditätsbereicht von $|\eta| < 3$ ab. Das ECAL dient der Energiemessung und Teilchenidentifikation von elektromagnetisch wechselwirkenden Teilchen wie Elektronen, Positronen und Photonen. Die Messmethode beruht auf der Rekonstruktion des elektromagnetischen Schauers, wobei die ionisierenden Teilchen des Schauers den Kristall anregen. Diese Anregungen im Festkörper werden schließlich in Licht umgewandelt, das über Photodetektoren nachgewiesen werden kann. Auf das ECAL folgt das HCAL, das der Energiemessung und Teilchenidentifikation von Hadronen dient. Das HCAL ist als Sampling-Kalorimeter realisiert. Es ist aus abwechselnden Lagen von Absorber- und Szintillatormaterial aufgebaut, wobei beim CMS-HCAL Messing-Absorber und Plastikszintillatoren verwendet werden. Die Messmethode beruht auf der Rekonstruktion des hadronischen Schauers. Dabei lösen die Hadronen im Absorbermaterial durch Kernreaktionen einen Schauer aus, der schließlich im Szintillator Licht erzeugt, das wie beim ECAL mittels Photodetektoren nachgewiesen wird. Im mittleren Bereich des Detektors deckt das HCAL eine Pseudorapidität von $|\eta| < 3$ ab, wobei in den Endkappen des Detektors das sogenannte Vorwärtskalorimeter einen Bereich von bis zu $|\eta| < 5,2$ abdeckt. Spurdetektor und Kalorimeter sind umgeben von einer supraleitenden Magnetspule. Die Spule hat dabei eine Länge von 12,5 m und einen Durchmesser von 6,3 m. Das starke

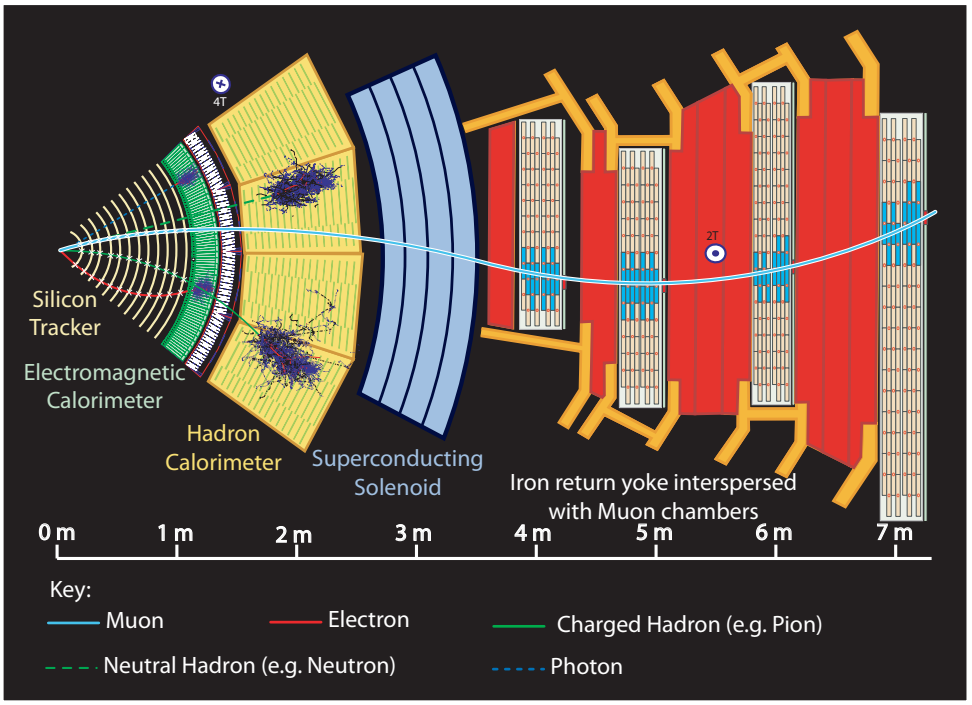


Abbildung 3.1: **Querschnitt des CMS-Detektors.** Der Aufbau des Detektor besteht aus dem Spurdetektor, dem ECAL, dem HCAL, dem supraleitenden Magneten und schließlich dem Rückführjoch, in dem das Myonsystem eingebaut ist. Entnommen aus [11].

Magnetfeld von bis zu 4 T ist dabei ein besonderes Merkmal des CMS-Detektors. Durch das Magnetfeld werden geladene Teilchen auf eine Kreisbahn gelenkt, aus deren Krümmung der Impuls der Teilchen gemessen wird. Wie im Namen des Detektors impliziert, spielt die Detektion von Myonen eine zentrale Rolle im CMS-Experiment. Der Myon-Nachweis ist dabei ein hilfreiches Werkzeug zur Rekonstruktion interessanter Signalprozesse, wie dem Zerfall $H \rightarrow ZZ \rightarrow 4 \times \ell$. Dabei sind die Myon-Detektoren in das eiserne Rückführjoch eingebaut. Der Detektor ist so entworfen, dass er Myonen über die gesamte kinematische Bandbreite, die der LHC erlaubt, nachweisen kann. Dafür werden drei verschiedene Arten von Myon-Detektoren verwendet, nämlich Driftröhren-, Kathodenstreifen- und Widerstandsplattenkammern. Weitere wichtige Teile des Experiments sind verschiedene Trigger-Systeme sowie das Computing-Grid, die im Detail in [10] beschrieben sind.

3.2 $t\bar{t}H$-Analyse

Die $t\bar{t}H$-Produktion bietet die Möglichkeit die Kopplung des Higgs-Bosons an Top-Quarks zu untersuchen. Die Betrachtung dieses Prozesses stellt einen Test für die Theorie der Fermionenmassen im SM dar. Eine Entdeckung würde die Yukawa-Kopplung des Higgs-Bosons an Fermionen weiter bestätigen [12]. Dieser Abschnitt gibt einen kurzen Überblick über die in dieser Arbeit relevanten Signal- und Untergrundprozesse und die allgemeine Analysestrategie bei der Suche nach der $t\bar{t}H$-Produktion, wobei hauptsächlich die für diese Arbeit relevanten Schritte näher erläutert werden.

In dieser Arbeit sollen Ereignisse, in denen ein Higgs-Boson in Assoziation mit einem Top-Quark-Antiquark-Paar ($t\bar{t}$) produziert wird, klassifiziert werden. Top-Quarks zerfallen fast ausschließlich in Bottom-Quarks und W-Bosonen. Der Zerfall eines Top-Quark-Paares kann deshalb durch den Zerfall der W-Bosonen charakterisiert werden. Zerfallen beide W-Bosonen in ein Quark-Antiquark-Paar ($q\bar{q}$), so wird der Zerfallskanal hadronisch genannt. Zerfällt ein W-Boson in ein $q\bar{q}$-Paar und das andere in ein geladenes Lepton und ein Neutrino, so wird der Zerfallskanal semileptonisch genannt und entsprechend heißt der Zerfallskanal dileptonisch, wenn beide W-Bosonen jeweils in ein geladenes Lepton und ein Neutrino zerfallen. Als Signal werden dabei die $t\bar{t}H$-Ereignisse berücksichtigt, in denen das Higgs-Boson in ein Bottom-Quark-Antiquark-Paar ($b\bar{b}$) und das $t\bar{t}$-Paar semileptonisch zerfallen. Der Higgs-Zerfall $H \rightarrow b\bar{b}$ ist dabei aufgrund seines großen Verzweigungsverhältnisses interessant [13]. In führender Ordnung befinden sich im Endzustand des Prozesses ein geladenes Lepton, ein Neutrino und sechs Quarks, von denen vier Bottom-Quarks sind. Ein Feynman-Diagramm des Prozesses ist in Abb. 3.2a gezeigt. Das isolierte Lepton erlaubt eine Unterscheidung zu QCD-Untergrundprozessen, in denen nur Jets vorhanden sind. Da das Neutrino nur schwach wechselwirkt, kann es im Detektor nicht direkt nachgewiesen werden. Durch das Auftreten von fehlender transversaler Energie kann allerdings auf das Neutrino zurückgeschlossen werden. Die Quarks im Endzustand hadronisieren und werden als Jets im Detektor sichtbar. Jets, die aus Bottom-Quarks entstanden sind, können durch die Verwendung von b-tagging-Algorithmen als sogenannte b-Jets identifiziert werden. Dazu wird der kombinierte Sekundärvertex-Algorithmus (engl. combined secondary vertex, CSV) verwendet [14].

Es gibt mehrere Untergrundprozesse die einen ähnlichen Endzustand wie der $t\bar{t}H(b\bar{b})$-Prozess haben. Eine Reduktion des Untergrundes ist möglich, indem nur Ereignisse berücksichtigt werden, die gewisse Anforderungen an die Zahl der Jets und b-tags erfüllen. In dieser Arbeit wird als Untergrund nur die Produktion von $t\bar{t}$-Paaren im semileptonischen Zerfallskanal betrachtet. Durch Abstrahlung und Aufspaltung von Gluonen können zusätzliche Jets entstehen. Dabei stellt die Produktion von $t\bar{t}$-Ereignissen in Assoziation mit einem $b\bar{b}$-Paar eine irreduzible Komponente des Untergrunds dar, da dieser Prozess einen identischen Endzustand wie der $t\bar{t}H$-Prozess hat (vgl. Abb. 3.2b). Eine Unterscheidung

zwischen $t\bar{t}H(b\bar{b})$- und $t\bar{t}+b\bar{b}$-Ereignissen ist somit nur möglich, indem z.B. kinematische Größen verglichen werden. Hinzu kommt, dass der Wirkungsquerschnitt des $t\bar{t}+b\bar{b}$-Prozess nach theoretischen Berechnungen in nächstführender Ordung bei einer Schwerpunktsenergie von $\sqrt{s} = 14\,\text{TeV}$ fast acht mal so groß wie der $t\bar{t}H(b\bar{b})$-Wirkungsquerschnitt ist [13, 15]. Eine weitere nicht zu vernachlässigende Untergrundkomponente ist die Produktion von $t\bar{t}$-Paaren mit zusätzlichen Jets, da diese Jets fälschlicherweise als b-Jets identifiziert werden können. Der Wirkungsquerschnitt für die $t\bar{t}$-Produktion ist um drei Größenordungen größer als der $t\bar{t}H(b\bar{b})$-Wirkungsquerschnitt [16].

Der kleine Wirkungsquerschnitt des $t\bar{t}H(b\bar{b})$-Prozesses im Vergleich zu den Untergrundprozessen sowie die zum Teil irreduziblen Untergründe sorgen dafür, dass sich die Suche nach $t\bar{t}H(b\bar{b})$-Ereignissen am LHC als anspruchsvoll herausstellt. Weiter ist es schwierig einzelne Ereignisvariablen mit genügend Trennkraft zwischen Signal und Untergrund zu finden. Deshalb ist der Einsatz multivariater Analysemethoden (MVA-Methoden) wie Boosted-Decision-Trees oder neuronaler Netze erforderlich, um besser zwischen Signal- und Untergrundereignissen trennen zu können.

Zu Beginn der $t\bar{t}H(b\bar{b})$-Analyse findet eine Rekonstruktion und Selektion der Ereignisse statt, wobei für eine detaillierte Beschreibung dieser Arbeitschritte bezüglich Anforderungen an Ereignisvariablen oder Rekonstruktionstechniken auf die aktuelle $t\bar{t}H(b\bar{b})$-Analyse [12] verwiesen wird. Anschließend werden die Ereignisse in verschiedene Kategorien eingeteilt, wobei zwischen der Zahl der Jets und b-tags unterschieden wird. Im semileptonischen Zerfallskanal der $t\bar{t}$-Paares wird dabei zwischen vier Kategorien unterschieden: $\geq$ 6-Jets, 3-b-tags; 4-Jets, 4-b-tags; 5-Jets, 4-b-tags und $\geq$ 6-Jets, $\geq$ 4-b-tags. Da die Trennkraft einzelner Ereignisvariablen nicht ausreicht, werden schließlich MVA-Methoden angewandt um verschiedene Variablen zu einer besser trennenden Diskrimanten zu kombinieren. Schließlich wird diese Diskriminante genutzt um die Ausgabeverteilung der Signal- und Untergrundprozesse an die Daten anzupassen um letztendlich ein Limit auf den $t\bar{t}H$-Wirkungsquerschnitt zu erhalten. Details zu den verwendeten statistischen Methoden finden sich z.B. in [17].

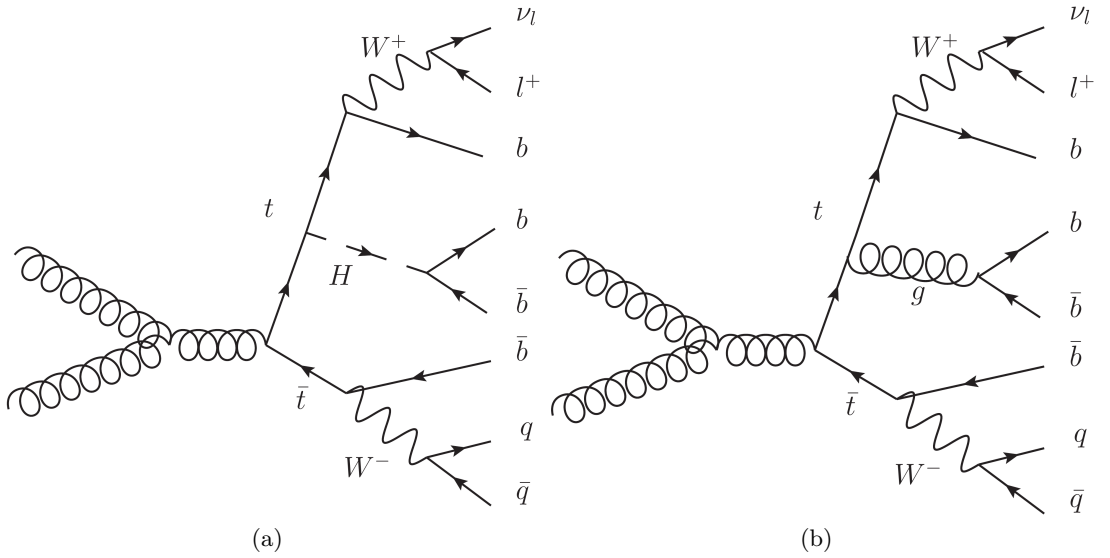


Abbildung 3.2: Feynman-Diagramme für den Prozess $t\bar{t}H$, $H \rightarrow b\bar{b}$ mit semileptonischen $t\bar{t}$-Zerfall (a) und den $t\bar{t}+b\bar{b}$-Prozess (b). Beide Prozesse haben den gleichen Endzustand bestehend aus einem Lepton und Neutrino sowie sechs Quarks, von denen vier Bottom-Quarks sind. Entnommen aus [17]

.

4 Neuronale Netze

Hauptthema dieser Arbeit ist die Klassifizierung von Ereignissen als Signal- oder Untergrundereignisse des $t\bar{t}H(b\bar{b})$-Prozesses mit Hilfe neuronaler Netze (NN). In diesem Kapitel werden die Grundlagen von FeedForward-Netzen eingeführt, wobei sich dieses Kapitel an [18, S. 168 ff] orientiert.

Bei der Trennung von Signal- und Untergrundereignissen handelt es sich um ein binäres Klassifikationsproblem. Ein Ereignis $E(\vec{x}, y)$ wird dabei durch einen Vektor $\vec{x}$, dessen Komponenten die Ereignisvariablen enthalten, und einen Skalar y, der die Klasse festlegt, zu der das Ereignis $E(\vec{x}, y)$ gehört, beschrieben. Dabei gilt z.B. für Signalereignisse $y = 1$ und für Untergrundereignisse $y = 0$. Wenn die Ereignisse der verschiedenen Klassen nicht mehr durch lineare Grenzen im Raum der Ereignisvariablen $\vec{x}$ getrennt werden können (siehe Abb. 4.1), werden zur Klassifikation der Ereignisse nicht lineare Verfahren, wie neuronale Netze, benötigt.

Ziel des Trainings eines neuronalen Netzes ist es, das NN dazu zu befähigen, dass es neue unbekannte Ereignisse, mit denen es nicht trainiert wurde, möglichst korrekt klassifiziert. Ein NN beschreibt dabei eine Abbildung $\hat{y} = f(\vec{x}, \boldsymbol{W}, \boldsymbol{b})$, wobei die Komponenten der Tensoren $\boldsymbol{W}$ und $\boldsymbol{b}$ die freien Parameter sind, die im Laufe eines Trainingsprozesses so angepasst werden, dass die Trennung zwischen den beiden Klassen optimal wird. Die Ausgabe des Netzes $\hat{y}$ ist dabei ein Schätzwert der wahren Klasse y.

Abschnitt 4.1 behandelt den Aufbau von NN, insbesondere von FeedForward-Netzen. In Abschnitt 4.2 wird der Ablauf des Trainingsprozesses sowie dessen Bewertung mittels Receiver-Operator-Characteristic-Kurven (ROC-Kurven) erklärt.

4.1 Aufbau eines FeedForward-Netzes

Der Grundbaustein neuronaler Netze ist das Neuron, das für jedes einzelne Ereignis einen Vektor $\vec{x}$ als Eingabe erhält und diesen auf eine Ausgabe y abbildet. Dabei wird zuerst eine gewichtete Summe $z = \sum_i w_i x_i + b$ gebildet, wobei b ein zusätzlicher Parameter (sogenannter Bias-Term) ist. Mit dem Bias-Term b kann die gewichtete Summe der Eingabewerte zusätzlich verschoben werden. Anschließend wird eine nicht lineare Aktivierungsfunktion $g(z)$ angewendet. Die Ausgabe $y = g(z)$ wird schließlich über gewichtete Verbindungen wieder als Eingabe an nachfolgende Neuronen weitergegeben (siehe Abb. 4.2).

Ein FeedForward-Netz besteht im wesentlichen aus Neuronen, die in Schichten angeordnet sind. Ein schematischer Aufbau eines FeedForward-Netzes ist in Abb. 4.3 gezeigt. Die

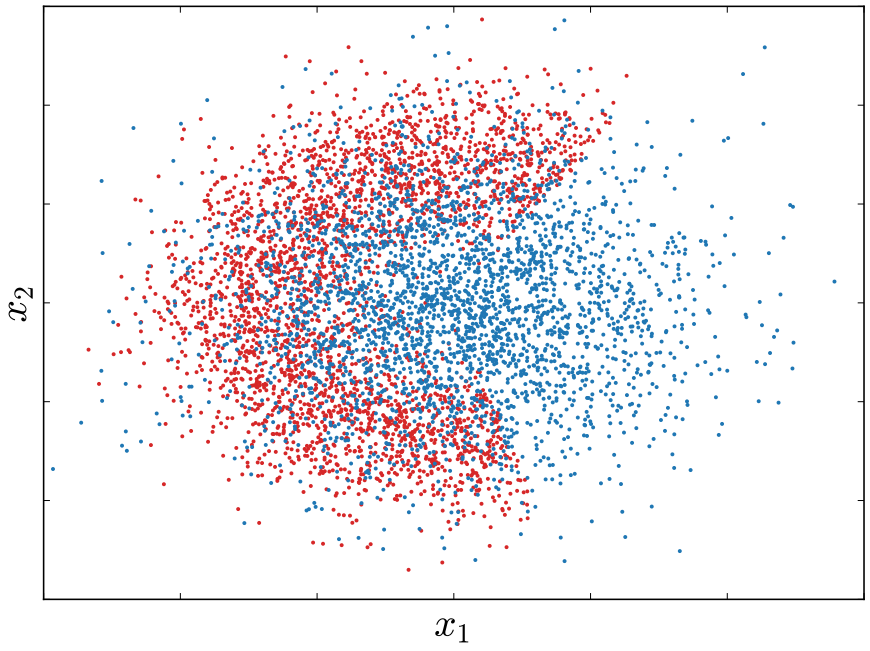


Abbildung 4.1: **Beispiel einer Verteilung von Ereignissen $E(\vec{x}, y)$, die durch zwei Variablen x_1 und x_2 beschrieben werden.** Die Untergrundereignisse (rot) sind in einem Kreisbogen um die Signalereignisse (blau) angeordnet. Es gibt keine lineare Grenze im Variablenraum, die eine passende Trennung zwischen Signal und Untergrund ermöglicht.

Eingabeschicht besteht aus einem n-dimensionalen Vektor $\vec{x}$, dessen Komponenten die n Ereignisvariablen enthalten, die das NN als Eingabe erhält. Die Neuronen in den verborgenen Schichten erhalten als Eingabevektor die Ausgabe aller Neuronen aus der vorherigen Schicht. Grundlegend für den Aufbau eines FeedForward-Netzes ist, dass Information nur von einer Schicht zur nächsten weitergegeben wird und dass keinerlei Verbindung zwischen den Neuronen innerhalb einer Schicht besteht. Die letzte Schicht des Netzwerks dient der Ausgabe des Schätzwertes der Klasse des Ereignisses.

Die Wahl der Aktivierungsfunktionen der Neuronen in den verborgenen Schichten ist ein aktuelles Forschungsfeld [19], [20]. Gängige Funktionen sind in Abb. 4.4 dargestellt. Nach [21] und [22] wird die Funktion $g(z) = \max\{0, z\}$ (Relu, engl. rectifier linear unit) als Standardwahl empfohlen, da diese eine höhere Geschwindigkeit des Trainingsprozesses ermöglicht. Allerdings kann im Voraus nicht gesagt werden, mit welcher Aktivierungsfunktion die beste Klassifikation erzielt wird. Dies muss durch Ausprobieren ermittelt werden. Für das Neuron in der Ausgabeschicht fällt die Wahl für die Aktivierungsfunktion jedoch nachfolgend immer auf die Sigmoid-Funktion $g(z) = 1/(1 + \exp(-z))$, da das Netzwerk einen Wert für $\hat{y} \in [0, 1]$ ausgeben soll (vgl. [18, S. 182 ff]).

Die Propagation des Eingabevektors $\vec{x}$ durch ein Netzwerk mit L Schichten beruht letztendlich auf mehrfacher Matrixmultiplikation und ist gegeben durch

$$x_i^k = g^k\left(z_i^k\right), \tag{4.1}$$

$$z_i^k = \sum_j W_{ij}^k x_j^{k-1} + b_i^k, \tag{4.2}$$

wobei x_i^k für das i-te Neuron in der k-ten Schicht steht und für $k = 1, \ldots, L$ gilt. W_{ij}^k ist das Gewicht der Verbindung des i-ten Neuron in der k-ten Schicht mit dem j-ten Neuron

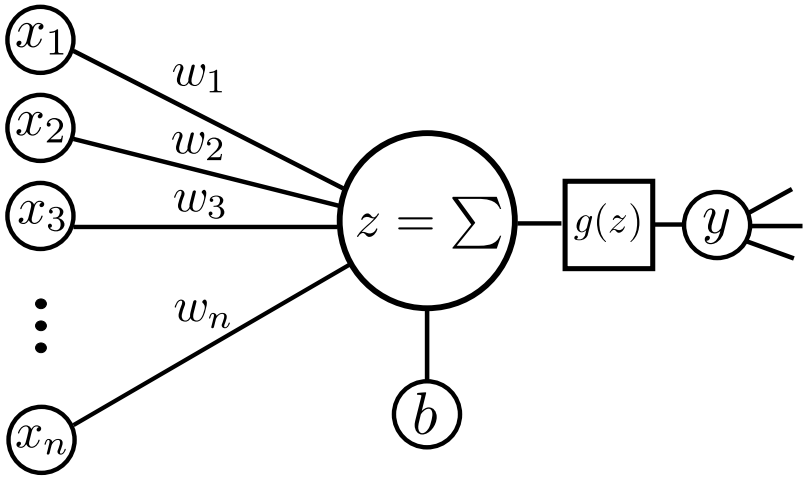


Abbildung 4.2: **Schematische Darstellung der Funktionsweise eines Neurons.** Für die Ausgabe y des Neurons gilt $y = g(z)$, $z = \sum_i w_i x_i + b$. Mit dem Bias-Term b kann der Wert der Summe zusätzlich verschoben werden.

in der vorherigen Schicht und b^k steht für die Biasvektoren der einzelnen Schichten. Der Einfachheit halber wurde für das Ausgabeneuron $\hat{y} = x^L$ gesetzt.

4.2 Training mit Fehlerrückführung

Für das Training eines neuronalen Netzes werden Ereignisse $E(\vec{x}, y)$ mit bekannter Klassifikation y_n benötigt. Um die Parameter $\boldsymbol{W}$ und $\boldsymbol{b}$ so anzupassen, dass die Ausgabe $\hat{y}(\vec{x})$ des Netzes die Klassifikation y approximiert, muss zunächst eine Fehlerfunktion $L(\hat{y}, y)$ definiert werden, die die Abweichung der Ausgabe $\hat{y}$ vom wahren Wert y quantifiziert. Eine mögliche Wahl ist die mittlere Kreuzentropie aller zum Training verwendeten Ereignisse

$$L\left(\hat{y}, y\right) = -\frac{1}{N}\sum_{n=1}^{N} y_n \log(\hat{y}_n) + (1 - y_n)\log(1 - \hat{y}_n), \tag{4.3}$$

wobei N für die Anzahl der Trainingsereignisse steht [23, Kap. 3]. Die Aufgabe für das Training des NN besteht nun darin, die Fehlerfunktion L in Bezug auf die Parameter $\vec{W}$ und $\vec{b}$ zu minimieren. Ein gängiger Algorithmus zur Lösung von Optimierungsproblemen ist das Gradientenverfahren. Dieses Verfahren nutzt die Gradienten einer Funktion $h(\vec{x}, \vec{\theta})$ bezüglich der freien Parameter $\vec{\theta}$, um die Parameter durch iteratives Aktualisieren nach dem Schema

$$\vec{\theta}^{k+1} = \vec{\theta}^{k} - \eta \nabla_{\vec{\theta}} h(\vec{x}, \vec{\theta}), \tag{4.4}$$

zu minimieren [23, Kap. 1]. Dabei wird ein zufälliger Anfangswert für die Parameter $\vec{\theta}$ gewählt. Die Lernrate $\eta > 0$ ist der Wert, der die Größe eines einzelnen Lernschrittes angibt. Es gilt die Lernrate η sorgfältig zu wählen, da ein zu kleiner Wert eine langsame Konvergenz und ein zu großer Wert Oszillationen um lokale Minima zur Folge hat.

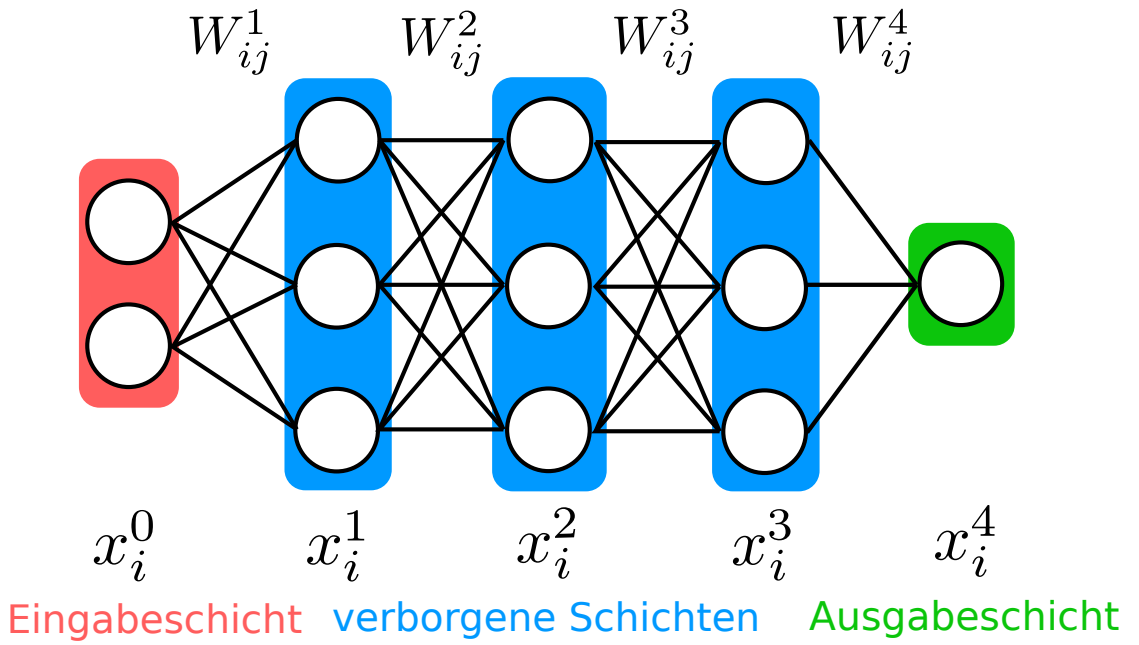


Abbildung 4.3: **Schematischer Aufbau eines FeedForward-Netzes.** Gezeigt ist ein neuronales Netz mit zwei Eingabevariablen, drei verborgenen Schichten mit jeweils drei Neuronen und einem Ausgabeneuronen. Zur Vereinfachung sind die Bias-Neuronen nicht abgebildet. Die schwarzen Linien zwischen den einzelnen Neuronen repräsentieren die Gewichte W_{ij}^k. Für das i-te Neuron in der k-ten Schicht gilt $x_i^k = g(\sum_j W_{ij}^k x_j^{k-1} + b_i^k)$, wobei x_j^{k-1} das j-te Neuron in der vorherigen Schicht ist.

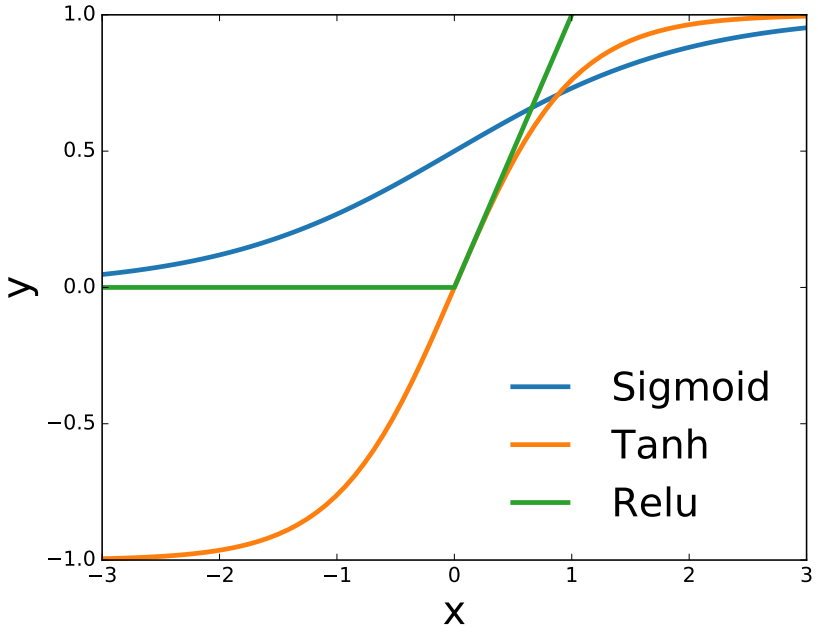


Abbildung 4.4: **Gängige Aktivierungsfunktionen für Neuronen in neuronalen Netzen nach [21, 22].** Gezeigt sind die logistische Sigmoid-Funtion $y(x) = 1/(1 + \exp(-x))$, der Tangens-Hyperbolicus $y(x) = \tanh(x)$ und die Relu-Funktion $y(x) = \max\{0, x\}$.

Der Backpropagation-Algorithmus bietet eine effiziente Methode, die Gradienten der Parameter der Fehlerfunktion eines neuronalen Netzes zu berechnen [24]. Ausgehend von Gleichung 4.1 und 4.2 lässt sich der Algorithmus durch vier Formeln beschreiben, die es ermöglichen, den Gradienten bezüglich der Parameter in den einzelnen Schichten des neuronalen Netzes iterativ zu berechnen:

$$\delta_i^L = \frac{\partial L}{\partial z_i^L} = \frac{\partial L}{\partial x_i^L} g'^L\left(z_i^L\right), \tag{4.5}$$

$$\delta_i^k = \frac{\partial L}{\partial z_i^k} = \sum_j W_{ji}^{k+1} \delta_j^{k+1} g'^k\left(z_i^k\right), \tag{4.6}$$

$$\frac{\partial L}{\partial b_i^k} = \delta_i^k, \tag{4.7}$$

$$\frac{\partial L}{\partial W_{ij}^k} = x_j^{k-1} \delta_i^k. \tag{4.8}$$

Mit den δ_i sind Hilfsgrößen definiert, die für die eigentliche Berechnung der Ableitungen der Fehlerfunktion L bezüglich der freien Parameter $\boldsymbol{W}$ und $\boldsymbol{b}$ genutzt werden. Dabei bezeichnet δ_i^L die Hilfsgröße für das i-te Neuron in der Ausgabeschicht und δ_i^k, $k \in \{1, 2, ..., L-1\}$, die Hilfsgröße für das i-te Neuron in der k-ten verborgenen Schicht des NN. Eine ausführlichere Beschreibung des Algorithmus findet sich in [23, Kap. 2]. In der Praxis wird häufig das sogenannte stochastische Gradientenverfahren verwendet. Dabei wird der Gradient der Fehlerfunktion bezüglich der Parameter nicht mit allen zur Verfügung stehenden Trainigsdaten berechnet, sondern der Trainingsdatensatz wird in kleinere Pakete (engl. Batches) aufgeteilt, die jeweils z.B. nur 100 Ereignisse enthalten. Mit den Batches kann dann jeweils ein approximierter Wert des wahren Gradienten berechnet wird. Dieses Verfahren ist bevorzugt, da es bei großen Trainingsdatensätzen eine Beschleunigung des Trainingsprozesses erzielt [18, S. 277 ff]. Eine Iteration über den gesamten Trainingsdatensatz wird auch Epoche genannt, wobei ein Training in der Regel aus vielen Epochen besteht.

Um das Training zu beschleunigen ist es hilfreich, alle Ereignisvariablen im gesamten Trainingsdatensatz zu normieren [25]. Dazu wird jede Variable x_i auf Mittelwert $\overline{x}_i = 0$ und Standardabweichung $\sigma_{x_i} = 1$ transformiert:

$$x_i' = \frac{x_i - \overline{x}_i}{\sigma_{x_i}}, \tag{4.9}$$

wobei $\bar{x}_i$ für den Mittelwert und σ_{x_i} für die Standardabweichung einer Variablen im Trainingsdatensatz steht. Es ist zu beachten, dass die Normierung (4.9) auch bei künftigen, noch zu klassifizierenden, Ereignissen vorgenommen werden muss.

Beim Training besteht die Gefahr, dass sich das NN zu gut an die Trainingsdaten anpasst und somit neue, vom Trainingsdatensatz abweichende Ereignisse nicht mehr so gut klassifizieren kann. Dies wird auch Generalisierungsfehler (engl. Overfitting) genannt. Um festzustellen, ob ein Generalsierungsfehler gemacht wird, benötigt man einen weiteren unabhängigen Validierungsdatensatz, mit dem das NN nicht trainiert wird und mit dem die Generalisierungsfähigkeit des NN überprüft wird. Während des Trainings wird die Fehlerfunktion dann nicht nur mit den Trainingsdaten, sondern auch mit den Validierungsdaten ausgewertet. Das Auftreten des Overfitting wird daran erkannt, dass beim batchweisen Traininieren und Validieren der Wert der Fehlerfunktion mit den Validierungsdaten größer und mit den Trainingsdaten kleiner wird. Um diesen Generalisierungsfehler zu verringern stehen verschiedene Regularisierungsmethoden zur Wahl. Die in dieser Arbeit verwendeten Methoden sollen kurz eingeführt werden, wobei für eine Erläuterung der Ideen hinter diesen Methoden auf die angegebene Literatur verwiesen wird:

L2-Regularisierung: Bei dieser Methode wird der Fehlerfunktion L ein weiterer Term hinzugefügt, der dafür sorgt, dass die Gewichte $\vec{W}$ des NN betragsmäßig klein bleiben [23, Kap. 3]. Es gilt:

$$L' = L + \underbrace{\frac{\lambda}{2} \sum {W_{ij}^k}^2}_{\text{L2-Term}}, \tag{4.10}$$

Der Regularisierungsparameter λ gibt dabei den Einfluss an, den der $L2$-Term bei der Minimierung der Fehlerfunktion hat.

Dropout: In dieser neuen Methode nach [26] wird die Ausgabe eines Neurons beim Training mit einer gegebenen Wahrscheinlichkeit p auf null gesetzt.

Early-Stopping: Hier werden die Parameter des Netzwerks nur verändert, falls sich der Fehler mit den Validierungsdaten im Vergleich zur vorherigen Trainingsepoche verringert hat. Findet für eine gegebene Anzahl an Epochen keine Verringerung des Fehlers mit den Validierungsdaten statt, wird das Training beendet.

Als Maß für die Qualität des Trainings eines NN zur Trennung von Signal- und Untergrundereignissen wird in dieser Arbeit hauptsächlich das Integral der Receiver-Operator-Characteristic-Kurve (ROC-Kurve) betrachtet. Zur Berechnung der ROC-Kurve werden für verschiedene Arbeitspunkte der Ausgabe des neuronalen Netzes jeweils die Richtig-Positiv-Rate (RPR) und Falsch-Positiv-Rate (FPR) bestimmt. Dabei wird als Arbeitspunkt ein fester Wert der Ausgabe des NN definiert, ab dem ein Ereignis als Signal klassifiziert wird. Liegt die Ausgabe des NN für ein Ereignis über (unter) dem Arbeitspunkt, so wird es als Signal (Untergrund) klassifiziert. Es gilt

$$RPR = \frac{RP}{N_{\text{Signal}}}, \tag{4.11}$$

$$FPR = \frac{FP}{N_{\text{Untergrund}}}, \tag{4.12}$$

wobei RP (Richtig Positiv) die Anzahl der durch das NN richtig als Signal und FP (Falsch Positiv) die Anzahl der durch das NN falsch als Signal klassifizierten Signal- und Untergrundereignisse ist. N_{Signal} ($N_{\text{Untergrund}}$) ist die Anzahl der Signalereignisse (Untergrundereignisse) im verwendeten Datensatz. Üblicherweise wird in der Teilchenphysik die Untergrundablehnung ($1 - FPR$) auf der y-Achse und die Signaleffizienz (RPR) auf der x-Achse aufgetragen. Das Integral der ROC-Kurve (engl. area under curve, AUC) ist ein sinnvolles Maß für die Trennung von Signal und Untergrund beim Vergleich verschiedener Netze, da hier noch kein bestimmter Arbeitspunkt gewählt wurde. Eine größere Fläche ist gleichbedeutend mit besserer Trennung.

5 Aufbau des Analyseframeworks NNFlow basierend auf TensorFlow

TensorFlow [27] ist eine Open-Source-Software-Bibliothek für numerische Berechnungen. Ursprünglich wurde TensorFlow von Google entwickelt und wurde im November 2015 unter der Apache 2.0 Open-Source-Lizenz veröffentlicht. Mittels einer Python-Schnittstelle werden mathematische Operationen als Knoten in einem symbolischen Graphen angeordnet. In Abb. 5.1 ist ein Beispiel für einen solchen Graphen gezeigt. Weiter existiert eine C++-Schnittstelle, die es ermöglicht, trainierte Modelle außerhalb der Python-Umgebung zu nutzen. TensorFlow erlaubt es, den konstruierten Graphen in einen numerischen Algorithmus zu übersetzen und auf einer oder mehreren CPUs oder GPUs ausführen zu lassen. Dabei unterstützt TensorFlow die Implementierung und das Training von neuronalen Netzen (NN). Allerdings bietet TensorFlow nur grundlegende Funktionen zur Nutzung von NN. Deshalb sind Kenntnisse über die Funktionsweise von NN zwingend notwendig, da der Nutzer die Modelle von Grund auf selbst implementieren muss. Um die Verwendung von TensorFlow so einfach wie möglich zu machen, ist im Rahmen dieser Arbeit das Analyseframework NNFlow [28] entstanden. Es erlaubt dem Nutzer ohne Vorkenntnis über TensorFlow oder NN ein FeedForward-Netz, wie in Kapitel 4 eingeführt, zu trainieren. Im Rest dieses Kapitels wird der Arbeitsablauf unter Verwendung von NNFlow vorgestellt. In Abb. 5.2 ist dieser schematisch dargestellt.

Um ein NN zur Trennung von Signal- und Untergrundereignissen zu trainieren, werden Ereignisse mit bekannter Klassifikation benötigt. Die gewünschten Prozesse können mit Monte-Carlo Methoden [29, 30] simuliert werden, wobei die rekonstruierten Ereignisse üblicherweise in ROOT-Dateien [31] gespeichert sind. TensorFlow kann von sich aus keine ROOT-Dateien als Eingabe verarbeiten, sondern nutzt bevorzugt die in Python verfügbaren NumPy-Arrays [32]. Eine wesentliche Funktion von NNFlow ist die Verwendung des Python-Moduls root_numpy [33], um die ROOT-Dateien in NumPy-Arrays zu konvertieren. Die Dimension der Arrays entspricht dabei der Anzahl der Ereignisse und Ereignisvariablen.

Der nächste Schritt besteht in der Vorverarbeitung der Daten. Nach der Umwandlung der ROOT-Dateien liegen die Daten noch im Structured-Array-Format vor, in dem die einzelnen Ereignisvariablen noch durch ihren Namen gekennzeichnet sind. Da TensorFlow dieses Datenformat nicht verarbeiten kann, müssen die Arrays noch in zwei-dimensionale Arrays umgewandelt werden. Dabei kann eine gewisse Auswahl an Variablen getroffen werden, die dann als Eingabe für das NN dienen. Zusätzlich besteht die Möglichkeit nur Ereignisse, die in eine gewisse N-Jets, N-b-tags (vgl. Kap. 3.2) Kategorie fallen, zu berücksichtigen. Weiter

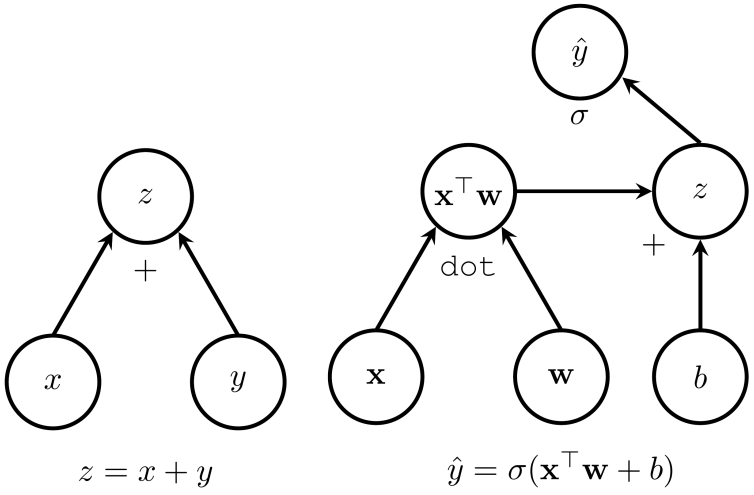


Abbildung 5.1: **Beispiel eines symbolischen Graphen, wie er in TensorFlow verwendung findet.** Der linke Graph zeigt die einfache Addition zweier Eingabevariablen x und y. Der komplexere rechte Graph zeigt die Funktionsweise eines Neurons, wie es in FeedForward-Netzen Verwendung findet. Entnommen aus [34].

übernimmt NNFlow die Berechnung der einzelnen Ereignisgewichte w. Dies geschieht nach

$$w = \prod_i w_{\mathrm{MC},i}, \tag{5.1}$$

wobei $w_{\mathrm{MC},i}$ eine Auswahl aus vorhandenen Monte-Carlo-Gewichten ist. Die Monte-Carlo-Gewichte sind Korrekturfaktoren um simulierte Daten an Messdaten anzugleichen und sind nicht mit den Parametern $\boldsymbol{W}$ des NN zu verwechseln, die üblicherweise auch als Gewichte bezeichnet werden. Darüber hinaus werden die Ereignisgewichte für Signal und Untergrund getrennt voneinander auf eins normiert. Diese Normierung sorgt für einen Ausgleich im Falle eines ungleichen Verhältnis der Anzahl von Signal- und Untergrundereignissen im Trainingsdatensatz. Wird diese nicht vorgenommen, so wird die Fehlerfunktion während dem Training zugunsten der überrepräsentierten Ereignisklasse minimiert. Falls z.B. deutlich mehr Untergrund- als Signalereignisse vorhanden sind, wird das Netz versuchen möglichst allen Untergrund korrekt zu klassifizieren und ignoriert dabei das Signal beim Training. Außerdem wird noch die Klassenkennzeichnung y =1,0 bzw. y =0,0 für Signal- bzw. Untergrundereignisse hinzugefügt. Schließlich werden die Daten in einen Trainings-, Validierungs- und Testdatensatz aufgeteilt. Der Anteil an der Gesamtzahl der Ereignisse beträgt in dieser Arbeit 50%, 10% und 40%, wobei darauf geachtet wird, dass das Verhältnis von Signal zu Untergrund in jedem Datensatz gleich ist.

Um schließlich NN unter Verwendung von NNFlow zu trainieren, kann auf die Klassen `BinaryMLP` und `Dataframe`, die auf TensorFlow und NumPy aufbauen, zurückgegriffen werden. In der Klasse `BinaryMLP` ist ein FeedForward-Netz zur binären Klassifikation implementiert, wie in Kap. 4 beschrieben. Dabei ist die Funktionsweise der Klasse an der Python-Bibliothek scikit-learn [35] orientiert, d.h. die Initialisierung und das Training des NN kann mit zwei Zeilen Code ausgeführt werden. Die Implementierung des NN nutzt dabei die Funktionalität der Klasse `DataFrame`, die für die Aufbewahrung der umgewandelten Datensätze dient. Dabei besteht die Hauptaufgabe des `DataFrame` in der Unterteilung der Daten in Batches sowie das zufällige Mischen der Ereignisse nach jeder Trainingsepoche. Zum Zeitpunkt der Initialisierung muss lediglich die Anzahl der Eingabevariablen, verborgenen Schichten und Neuronen pro Schicht sowie die Aktivierungsfunktion bekannt sein.

Zudem muss ein Verzeichnis angegeben werden, in dem die Parameter des Netzes gespeichert werden. Für den Trainingsprozess werden sowohl der Trainings- als auch der Validierungsdatensatz benötigt. Weiter können die Parameter für die Regularisierungsmethoden und den Optimierungsalgorithmus festgelegt werden. Die Normierung der Trainingsereignisse nach Gl. 4.9 wird vor dem Trainingsprozess durchgeführt. Dabei werden die Parameter der Transformation gespeichert, sodass auch weitere Ereignisse entsprechend normiert werden können. Für die Minimierung der Fehlerfunktion beim Training eines NN steht zusätzlich zum einfachen Gradientenverfahren (siehe Gl. 4.4) noch der Adam-Algorithmus [36] zur Verfügung. Dieser passt im Laufe des Trainingsprozesses die Lernrate für die Gewichte einzeln an. Die Veröffentlichung [36] zeigt, dass der Adam-Algorithmus die Geschwindigkeit des Trainings erhöht, weswegen dieser als Standardwahl genutzt wird. Anders als in Kap. 4 beschrieben, wird als Abbruchkriterium des Trainings (engl. Early-Stopping) nicht der Wert der Fehlerfunktion sondern das ROC-Integral der Validierungsdaten betrachtet, da in dieser Arbeit vor allem Letzterer als Metrik für die Leistungsfähigkeit der Klassifikation genutzt wird. Beim Training werden die Ereignisgewichte berücksichtigt, indem der Fehler jedes Ereignisses mit dessen Gewicht multipliziert wird. Genaueres zum Training von NN mit gewichteten Ereignissen wird in Kap. 6.1 erläutert.

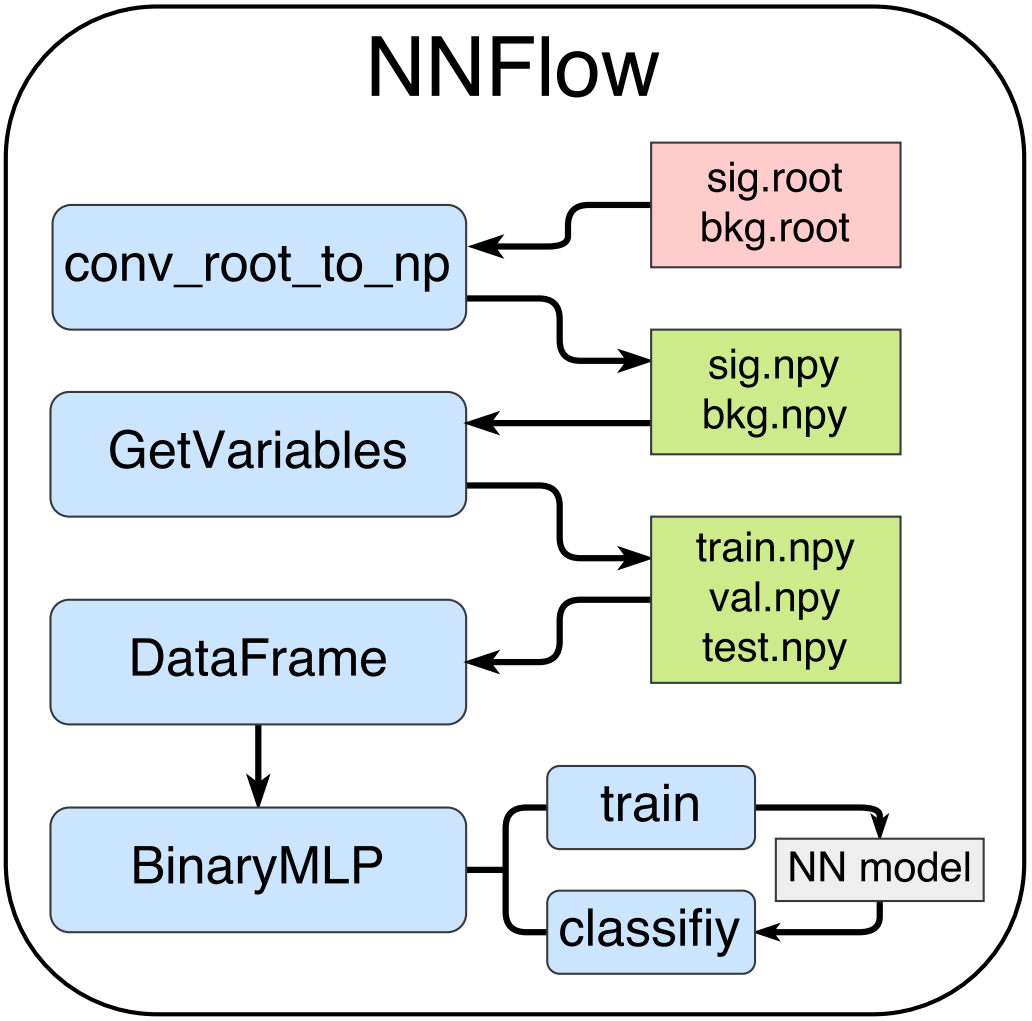


Abbildung 5.2: **Schematische Darstellung des Arbeitsablauf der Konvertierung der ROOT-Dateien in NumPy-Arrays, der Vorverarbeitung dieser Arrays sowie des Trainings von NN unter Verwendung von NNFlow.** Der erste Schritt besteht in der Konvertierung der gewünschten ROOT-Dateien in NumPy-Arrays. Anschließend können bestimmte Variablen aus den NumPy-Arrays extrahiert werden. Die Daten werden getrennt in Trainings-, Validierungs- und Testdatensätzen gespeichert. Die Klassenkennzeichnung und die Gewichte der einzelnen Ereignisse werden dabei hinzugefügt. Letztendlich können die Datensätze über den bereitgestellten `DataFrame` in das neuronale Netz eingegeben werden.

6 Studien zur Parameterwahl beim Einsatz von neuronalen Netzen unter Verwendung von NNFlow

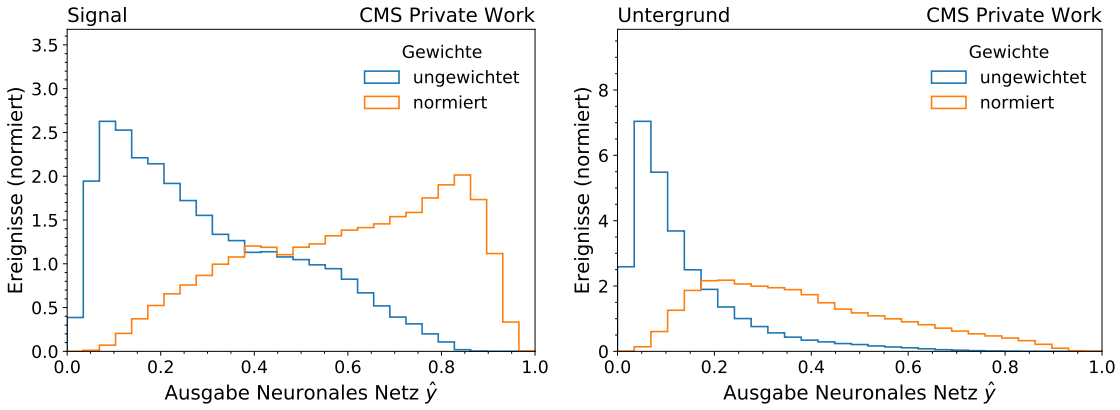
Dieses Kapitel bietet eine Übersicht über das Training neuronaler Netze (NN) mit $t\bar{t}H$-Signal- und $t\bar{t}$-Untergrundereignissen. Abschnitt 6.1 behandelt den Umgang mit gewichteten Ereignissen. Weiter wird in Abschnitt 6.2 und 6.3 untersucht, welchen Einfluss die Anzahl der verborgenen Schichten und Neuronen sowie die Wahl der Ereignisvariablen auf den Trainingsprozess von NN haben. Für die Studien werden bereits vorhandene $t\bar{t}H$- und $t\bar{t}$-Datensätze aus den Analysen [12] genutzt, wobei sich dort auch Details zur Ereignisgenerierung und -auswahl finden lassen. Für die Berechnung der ROC-Kurven und ROC-Integrale wird auf Funktionen von scikit-learn [35] zurückgegriffen.

6.1 Umgang mit gewichteten Trainingsereignissen

Es ist möglich, dass die simulierten Ereignisse die Messdaten, z.B. den Wirkungsquerschnitt oder die b-tagging-Effizienz, nicht korrekt beschreiben. Allerdings besteht die Möglichkeit, die generierten Daten zu gewichten. Dabei kann jedes simulierte Ereignis mit einem individuellen Gewicht versehen werden, dass seiner Häufigkeit in den echten Daten entspricht. Die Gewichte können von den Eigenschaften des Ereignisses abhängen. Diese Korrektur sorgt dafür, dass die Verteilungen der Ereignisvariablen der simulierten Daten die Wirklichkeit besser beschreiben. Für das Training von NN ist es essentiell, einen Trainingsdatensatz zu nutzen, der möglichst ähnlich zu den späteren Messdaten ist. Bei üblichen Implementationen von NN ist es nicht möglich gewichtete Ereignisse zum Training zu verwenden. Um die Ähnlichkeit der Trainingsdaten zu den Messdaten zu gewährleisten, muss deshalb geklärt werden, wie das Gewicht eines simulierten Ereignisses beim Trainingsprozess berücksichtigt werden soll. Die Idee ist dabei, den Beitrag eines Ereignisses zur Fehlerfunktion L mit dessen Gewicht w_n zu multiplizieren

$$L\left(\hat{y}, y\right) = -\frac{1}{N}\sum_{n=1}^{N} w_n \left[y_n \log(\hat{y}_n) + (1-y_n)\log(1-\hat{y}_n)\right]. \tag{6.1}$$

Ereignisse mit negativen Gewichten, die bei verschiedenen Monte-Carlo-Generatoren auftreten können, sollten bei dieser Methode nicht zum Training verwendet werden, da das Vorzeichen des Gewichts sonst für einen negativen Beitrag zur Fehlerfunktion führt. Es



(a) (b)

Abbildung 6.1: **Verteilung der Ausgabe zweier NN für Testdaten, wobei das Training beider Netze mit ungewichteten Ereignissen durchgeführt wurde.** Das Integral der Histogramme ist auf eins normiert. In (a) ist die Verteilung der Ausgabe für Signal- und in (b) entsprechend für Untergrundereignisse zu sehen. Beide Netze erhalten als Trainingsereignisse die gleichen Daten, wobei sich ungefähr fünf mal so viele Untergrund- wie Signalereignisse im Datensatz befinden. Dabei sind die Ereignisse für das erste Netz mit $w_n = 1{,}0$ gewichtet (blau), wohingegen die Gewichte für das zweite Netz für Signal und Untergrund getrennt voneinander normiert sind (gelb).

ist nicht klar, ob negative Gewichte berücksichtigt werden können, indem der Betrag des Gewichts $|w_n|$ verwendet wird. In der Dokumentation des TMVA-Frameworks Version 4.2.0 [37], das das Standard-Framework für Maschinelles Lernen in der Teilchenphysik ist, wird empfohlen Ereignisse mit negativen Gewichten beim Training von FeedForward-Netzen zu ignorieren, wobei das Training mit einer gewichteten Fehlerfunktion wie in Gl. 6.1 stattfindet. Beim Training mit dem Gradientenverfahren (beschrieben in Gl. 4.4) sorgt das Gewicht dafür, dass der Beitrag eines Ereignisses zum Trainingsprozess entsprechend skaliert wird. Höher gewichtete Ereignisse führen dabei zu einer größeren effektiven Lernrate η, wodurch das Aktualisieren der NN-Parameter durch diese Ereignisse stärker beeinflusst wird.

Die Gewichtung einzelner Ereignisse ist zu Beginn dieser Arbeit dazu verwendet worden, um das ungleiche Verhältnis der Zahl von Signal- zu Untergrundereignissen auszugleichen. Dabei sind alle Ereignisse mit $w_n = 1{,}0$ gewichtet, wobei zusätzlich eine getrennte Normierung der Gewichte von Signal und Untergrund vorgenommen wird. Da sich in den vorhandenen Datensätzen ungefähr fünf mal so viele Untergrund- wie Signalereignisse befinden, gilt für das Verhältnis von Signal- zu Untergrundgewicht $w_{\text{Signal}}/w_{\text{Untergrund}} \approx 5$. In Abb. 6.1 ist die Ausgabe von zwei NN für einen Testdatensatz gezeigt. Beide Netze haben eine verborgene Schicht mit 200 Neuronen, wobei die Neuronen die Relu-Aktivierungsfunktion nutzen. Das eine NN wurde mit ungewichteten Trainingsdaten trainiert und bei dem anderen Netz wurden die Gewichte, wie oben beschrieben, entsprechend normiert. Das Training findet mit dem Adam-Algorithmus bei einer Lernrate $\eta = 0{,}001$, sowie einer Batch-Größe von 128 für maximal 100 Epochen statt. Als Regularisierungsparameter werden eine Dropout-Rate von 0,5, L2-Regularisierung $\lambda = 1 \cdot 10^{-10}$ und als Early-Stopping Kriterium 10 Epochen gewählt. Vor der Normierung der Gewichte wird das Training der NN durch die hohe Zahl der Untergrundereignisse dominiert, sodass es bei der Minimierung der Fehlerfunktion günstiger ist, die Ausgabe des Netzes für alle Ereignisse gegen Null zu schieben. Mit

getrennt normierten Gewichten nehmen die Signalereignisse einen größeren Einfluss auf den Trainingsprozess. Dies ist daran erkennbar, dass das Netz nun auch für Signalereignisse Werte ausgibt, die näher an der richtigen Klassenkennzeichung $y_n = 1{,}0$ liegen.

Motiviert durch den Erfolg bei der Gewichtung mit getrennt normierten Gewichten von Signal- und Untergrundereignissen zum Ausgleich des ungleichen Verhältnisses der Anzahl von Signal- und Untergrundereignissen, soll nun der Effekt mit Monte-Carlo-Gewichten untersucht werden. Dazu wird ein vereinfachtes Beispiel betrachtet, bei dem das Netz als Eingabe lediglich drei Ereignisvariablen erhält, die jedoch gut zwischen Signal und Untergrund trennen:

- Evt_CSV_Average - mittlere Ausgabe des b-tagging-Algorithmus für alle Jets
- Evt_Deta_Jets_Average - mittlere Pseudorapiditätsdifferenz $\Delta\eta$ zwischen zwei Jets
- NJets - Anzahl der Jets

Histogramme der Variablen sind in Anhang A gezeigt. Zusätzlich werden nicht die richtigen Monte-Carlo-Gewichte, sondern selbst festgelegte Gewichte verwendet. Dabei wird zwischen drei verschiedenen Arten von Gewichten unterschieden:

- keine Gewichtung, $w_n = 1{,}0$ für Signal und Untergrundereignisse, wobei die Gewichte für Signal und Untergrund getrennt auf eins normiert sind.
- Gewichte unkorreliert zu den Ereignisvariablen. Dazu sind, jeweils für Signal und Untergrund getrennt, eine Hälfte der Ereignisse mit $w_n = 0{,}2$ und die andere Hälfte mit $w_n = 0{,}7$ gewichtet.
- Gewichte korreliert zu NJets. Ereignisse mit NJets > 4 sind mit $w_n = 0{,}7$, der Rest mit $w_n = 0{,}2$ gewichtet.

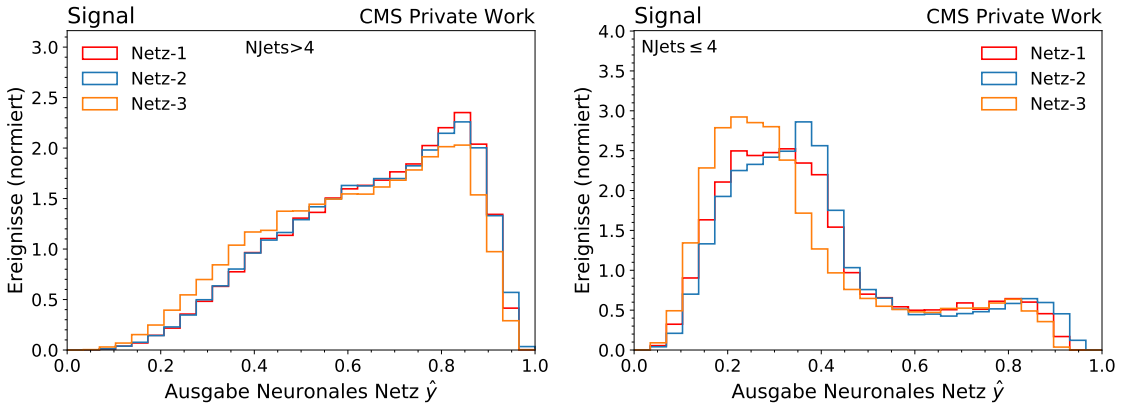
Für die drei verschiedenen Gewichtungen wird jeweils ein NN trainiert, das wie folgt bezeichnet werden sollen:

- **Netz-1** - NN trainiert mit Ereignissen mit $w_n = 1{,}0$ gewichtet und für Signal bzw. Untergrund getrennt normiert.
- **Netz-2** - NN trainiert mit unkorreliert gewichteten Ereignissen.
- **Netz-3** - NN trainiert mit korreliert gewichteten Ereignissen.

Jedes Netz hat dabei eine verborgenen Schicht mit 200 Neuronen. Als Aktivierungsfunktion wird die Relu-Funktion verwendet. Das Training findet mit dem Adam-Algorithmus bei einer Lernrate $\eta = 0{,}001$ sowie einer Batch-Größe von 128 für maximal 100 Epochen statt. Als Regularisierungsparameter werden eine Dropout-Rate von 0,5, L2-Regularisierung $\lambda = 1 \cdot 10^{-10}$ und als Early-Stopping Kriterium 10 Epochen gewählt. Eine Übersicht über die Größe und Zusammensetzung des verwendeten Datensatzes findet sich in Tab. 6.1.

Tabelle 6.1: **Anzahl Ereignisse der in dieser Studie verwendeten Datensätze.** Alle Zahlenwerte sind mit 10^3 zu multiplizieren. Das Verhältnis von Signal zu Untergrund enstpricht in etwa eins zu fünf.

	Training		Validierung		Test	
	Untergrund	Signal	Untergrund	Signal	Untergrund	Signal
Insgesamt	714,8	143,6	143,0	28,7	571,9	114,9
NJets$\leq$4	311,9	27,8	62,1	5,7	248,6	22,2
NJets>4	402,9	115,8	80,9	23,0	323,3	92,7

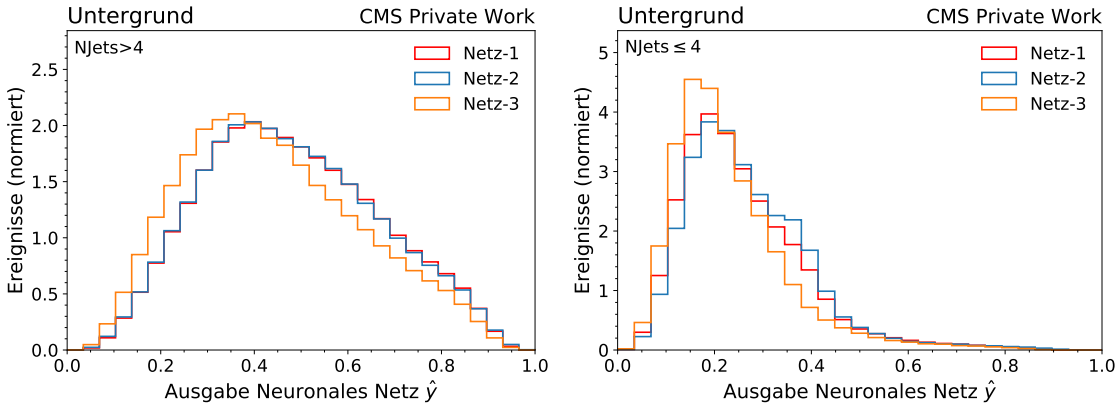


(a) (b)

Abbildung 6.2: **Verteilungen der Ausgabe von Netz-1, Netz-2 und Netz-3 für Signalereignisse des Testdatensatzes.** Das Integral der Histogramme ist auf eins normiert. In (a) ist die Verteilung für Ereignisse mit mehr als vier Jets gezeigt (bei den korrelierten Gewichten entspricht dies $w = 0{,}7$). In (b) ist entsprechend die Verteilung für Ereignisse mit vier oder weniger Jets gezeigt (bei den korrelierten Gewichten entspricht dies $w = 0{,}2$). In (a) ist die Ausgabe für korrelierte Gewichte etwas mehr gegen 0 gerückt. Für unkorellierte Gewichte ist die Verteilung der Ausgabe ähnlich zu der ohne Gewichte. Auch in (b) rückt die Verteilung der Ausgabe für korrelierte Gewichte etwas gegen 0.

In Abb. 6.2 sind die Verteilungen der Ausgabe der drei trainierten NN für Signalereignisse des Testdatensatzes gezeigt. Dabei ist in Abb. 6.2a der Vergleich zwischen den NN-Ausgaben für Ereignisse mit mehr als vier Jets zu sehen. Alle Verteilung haben ihr Maximum um $\hat{y} = 0{,}85$. Zwischen den Verteilungen der Ausgabe $\hat{y}$ von Netz-1 und Netz-2 ist kaum ein Unterschied zu erkennen. Dagegen ist die Ausgabe des NN Netz-3 zu etwas niedrigeren Werten verschoben. In Abb. 6.2b ist derselbe Vergleich für Ereignissen mit weniger oder gleich vier Jets zu sehen. Die Maxima der drei Verteilungen liegen im Bereich zwischen $0{,}2 \leq \hat{y} \leq 0{,}4$. Jedoch beinhalten alle Verteilungen auch Ereignisse im Bereich zwischen $0{,}8 \leq \hat{y} \leq 0{,}95$. Die Form der Verteilung lässt darauf schließen, dass viele Ereignisse mit NJets ≤ 4 sehr untergrundartig sind und deshalb einen niedrigen Ausgabewert $\hat{y}$ erhalten. Das Maximum der Ausgabeverteilung von Netz-3 ist im Vergleich zu den anderen zwei Verteilungen zudem etwas gegen null verschoben. Eine mögliche Ursache dieser Verschiebung ist das Training mit Ereignissen, die mit $w_n = 0{,}2$ gewichtet sind. Durch das kleine Gewicht werden diese Art von Signalereignissen beim Training eher vernachlässigt und es wird nicht versucht die Ausgabe für diese Ereignisse zu verbessern, d.h. in Richtung $\hat{y} = 1$ zu schieben. Im Bereich der NN-Ausgabe $\hat{y} > 0{,}6$ ist der Unterschied der drei Verteilungen jedoch wieder sehr gering. Möglicherweise ist es für ein NN einfach, diese Ereignisse als Signalereignisse zu klassifizieren, wodurch die Gewichtung der Ereignisse weniger Einfluss auf das Training nimmt.

In Abb. 6.3 sind die Verteilungen der Ausgabe der drei trainierten NN für Untergrundereignisse des Testdatensatzes gezeigt. Dabei ist in Abb. 6.3a die Verteilung der NN-Ausgabe für Untergrundereignisse mit mehr als vier Jets zu sehen. Die Ausgabe für Netz-1 und Netz-2 sind sich wieder sehr ähnlich, wobei das Maximum beider Verteilungen ungefähr bei $\hat{y} = 0{,}4$ befindet. Dagegen ist das Maximum der Verteilung für die Ausgabe von Netz-3 etwas gegen null verschoben, sodass das Maximum hier etwa bei $\hat{y} = 0{,}35$ liegt. Im Vergleich liegt die Verteilung der Ausgabe von Netz-3 auch deutlich über den anderen zwei Verteilungen,



(a) (b)

Abbildung 6.3: **Verteilungen der Ausgabe von Netz-1, Netz-2 und Netz-3 für Untergrundereignisse des Testdatensatzes.** Das Integral der Histogramme ist auf eins normiert. In (a) ist die Verteilung für Ereignisse mit vier oder weniger Jets gezeigt (bei den korrelierten Gewichten entspricht dies $w = 0{,}2$). In (b) ist entsprechend die Verteilung für Ereignisse mit mehr als vier Jets gezeigt (bei den korrelierten Gewichten entspricht dies $w = 0{,}7$). Sowohl in (a), als auch in (b), sind die Verteilung für unkorrelierte und keine Gewichte sehr ähnlich. Im Vergleich dazu ist die Verteilung für Gewichte, die korreliert zu NJets sind, etwas gegen 0 verschoben.

sodass man hier darauf schließen kann, dass die höhere Gewichtung der Ereignisse mit NJets > 4 von $w_n = 0{,}7$ beim Training von Netz-3 dazu führt, dass diese Ereignisse mehr berücksichtigt werden im Vergleich zum Training von Netz-1 bwz. Netz-2. Abb. 6.3b zeigt die Ausgabeverteilungen der drei NN für Untergrundereignisse mit NJets ≤ 4. Das Maximum der Verteilung liegt für Netz-1 und Netz-2 bei $y_n \approx 0{,}2$, wohingegen das Maximum für Netz-3 etwas unterhalb von $y_n = 0{,}2$ zu finden ist. Die Verteilungen unterscheiden sich weiter im Anstieg und Abfall des Histogramms, wobei die Ausgabe von Netz-3 am steilsten ansteigt und wieder abfällt. Obwohl Ereignisse mit NJets ≤ 4 mit $w_n = 0{,}2$ gewichtet sind, verbessert sich in diesem Fall sogar die Ausgabe des NN, indem generell kleinere $\hat{y}$ für diese Art von Untergrund ausgegeben werden.

In Abb. 6.4 ist zum Vergleich die Ausgabe von Netz-1 und Netz-3, ohne Unterscheidung nach NJets, für den Testdatensatz gezeigt. Das Training mit gewichteten Ereignissen bei Netz-3 hat keinen Einfluss auf das ROC-Integral, das für beide Netze gleich ist. Allerdings hat sich die Form der Verteilung für Netz-3 geändert. Wie es scheint, ist beim Training dieses NN mehr darauf geachtet worden, Untergrundereignisse besser zu klassifizieren. Die Ausgabe für Signalereignisse scheint bei Netz-3 dagegen etwas gegen Null verschoben.

Zusammenfassend ist zu sagen, dass das Training von NN durch die Gewichtung des Beitrags eines Ereignisses in der Fehlerfunktion wie in Gl. 6.1 beeinflusst werden kann. Am deutlichsten ist der Effekt der Gewichtung in Abb. 6.1 zu sehen. Das Beispiel, das in dieser Studie untersucht wird, zeigt, dass auch verschieden gewichtete Ereignisse einen Einfluss auf den Trainingsprozess nehmen können. Trotz der Tatsache, dass bisher noch nicht geklärt ist wie und ob Ereignisse mit negativen Gewichten beim Training von NN berücksichtigt werden könne, scheint der hier gewählte Ansatz richtig. Die Trennung zwischen Signal- und Untergrundereignissen bleibt bezüglich des ROC-Integrals gleich und gleichzeitig kann die Form der Verteilung der Ausgabe eines NN positiv beeinflusst werden.

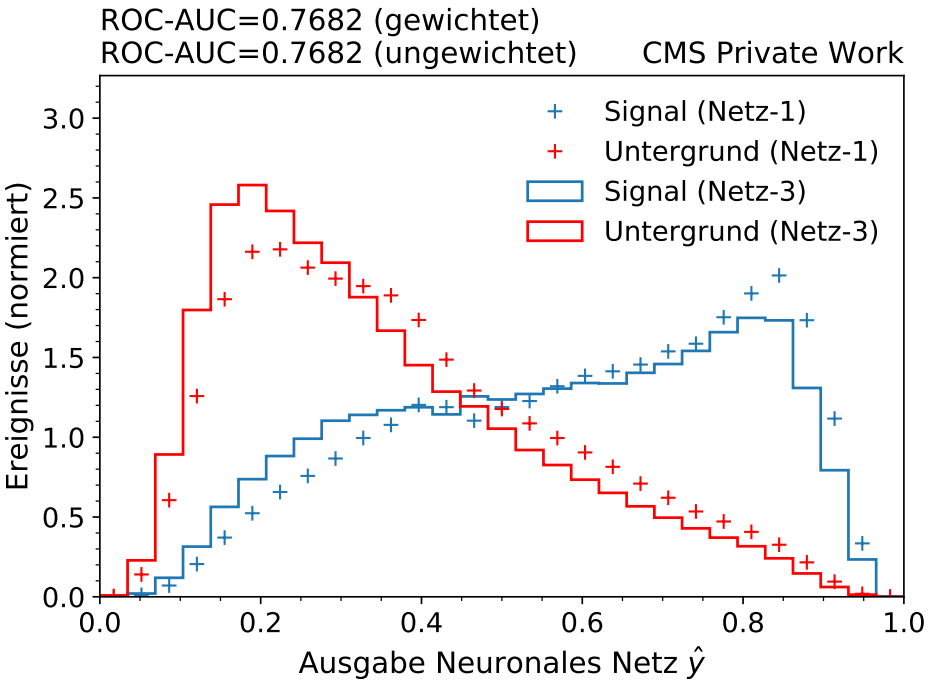


Abbildung 6.4: **Vergleich der Ausgabeverteilung für Signal- und Untergrund von Netz-1 und Netz-3.** Das Integral der Histogramme ist auf eins normiert. Zu sehen ist der Einfluss, den das Training mit gewichteten Ereignissen im Vergleich zum Training mit ungewichteten Ereignissen hat. Dabei ist zu erkennen, dass die Verteilungen von Netz-3 etwas mehr gegen 0 verschoben ist.

6.2 Anzahl der Neuronen und verborgenen Schichten

Seit einiger Zeit sind NN mit vielen verborgenen Schichten (engl. Deep Neural Network (DNN)) sehr beliebt. Dabei erzielen vor allem Convolutional NN [18, S. 330 ff] große Erfolge im Bereich der Bilderkennung [38]. Desweiteren gibt es auch theoretische Hinweise, dass NN mit mehreren verborgenen Schichten Vorteile gegenüber Netzen mit nur einer verborgenen Schicht bieten können [39]. Dabei besteht die Hoffnung, dass das NN durch die zusätzlichen Schichten weitere Zusammenhänge aus den Variablen lernen kann. Deshalb wird im Folgenden der Einfluss der Anzahl der verborgenen Schichten und Neuronen auf die Leistungsfähigkeit der Klassifikation bei der Trennung von $t\bar{t}H$-Signal- und $t\bar{t}$ -Untergrundereignissen untersucht. Dazu werden verschiedene NN trainiert, wobei die Anzahl der verborgenen Schichten zwischen Eins und Acht liegt. Zusätzlich wird die Zahl der Neuronen pro Schicht zwischen 15, 50, 100, 300 und 500 variiert. Für alle Netze wird die Relu-Funktion als Aktivierungsfunktion gewählt. Im folgenden wird die Netzkonfiguration mit $A \times B$ abgekürzt, wobei A für die Zahl der verborgenen Schichten und B für die Neuronen pro Schicht steht. Jede Netzkonfiguration wird schließlich fünfmal trainiert, wobei jeweils die Startwerte der NN-Gewichte $\boldsymbol{W}$ und $\boldsymbol{b}$ zufällig initialisiert werden. Letztendlich wird der Mittelwert sowie die Standardabweichung des ROC-Integrals, ermittelt mit dem Testdatensatz, verwendet um die Leistungsfähigkeit einer Netzkonfiguration bei der Trennung von Signal- und Untergrundereignissen zu beurteilen.

Als Eingabevariablen werden die Variablen verwendet, die auch als Eingabe für Boosted Decision Trees (BDT) verwendet werden, wie sie in der aktuellen $t\bar{t}H$-Analyse [12] zum Einsatz kommen. Da hier nicht der Einfluss einzelner Variablen auf die Leistungsfähigkeit der Klassifikation untersucht werden soll, erhält das NN alle zur Verfügung stehenden

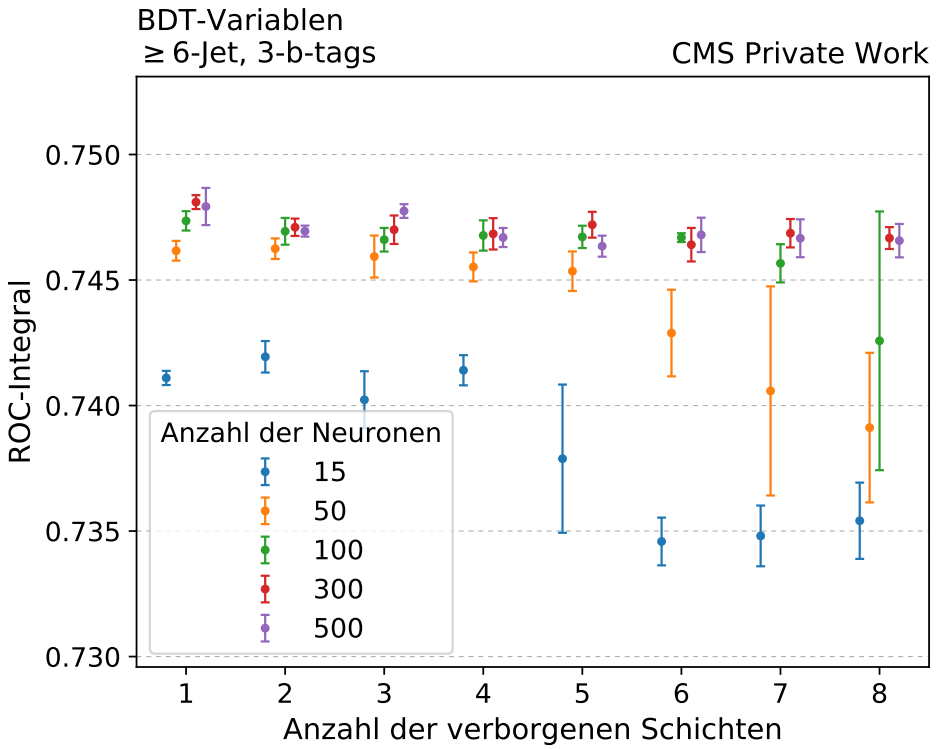
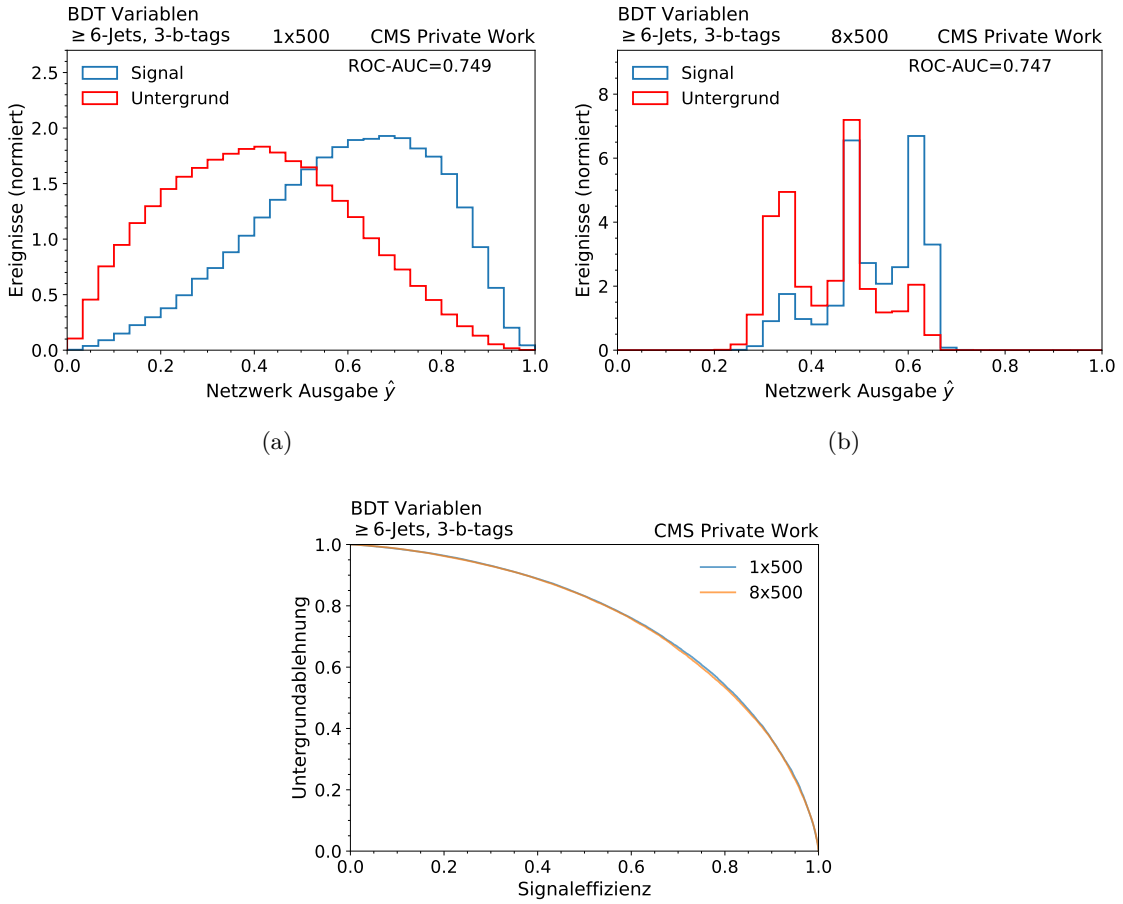


Abbildung 6.5: **Abhängigkeit des mittleren ROC-Integrals von der Anzahl der verborgenen Schichten und Neuronen.** Zur besseren Übersicht sind die Einträge für eine verborgene Schicht jeweils etwas versetzt nebeneinander angeordnet. Die Fehlerbalken geben die Standardabweichung vom Mittelwert an. Alle anderen Parameter sind für jede Netzkonfiguration identisch gewählt und werden im Text beschrieben.

Variablen als Eingabe. Dazu zählen allgemeine Ereignis-, b-tagging-, Winkel-, Massen- sowie Variablen, die durch die Anwendung von Jet-Substruktur-Algorithmen berechnet werden. Eine detaillierte Beschreibung der Variablen findet sich z.B. in [17]. Im Folgenden wird diese Auswahl von Variablen als „BDT-Variablen“ bezeichnet. Plots der verwendeten Größen befinden sich in Anhang B. Zunächst werden exemplarisch nur Ereignisse berücksichtigt, die in die $\geq$ 6-Jets, 3-b-tags Kategorie fallen. Die Einordnung der Ereignisse in die verschiedenen Kategorien erfolgt analog zur $t\bar{t}H$-Analyse [12]. Für die Gewichtung der Ereignisse werden Standardgewichte angewendet, die die Verteilungen der simulierten Daten so modifizieren, dass die Daten beschrieben werden. Die Zusammensetzung des Trainings-, Validierungs und Testdatensatzes kann aus Tab. 6.2 entnommen werden. Das Training jedes NN findet mit dem Adam-Algorithmus bei einer Lernrate $\eta = 0{,}001$ sowie einer Batch-Größe von 128 für maximal 100 Epochen statt. Als Regularisierungsparameter werden eine Dropout-Rate von 0,5, L2-Regularisierung $\lambda = 1 \cdot 10^{-10}$ und als Early-Stopping Kriterium 10 Epochen gewählt.

In Abb. 6.5 ist der Einfluss der Anzahl der verborgenen Schichten und Neuronen auf das ROC-Integral der jeweiligen Netzgrößen zu sehen. Dabei sind der Mittelwert sowie die Standardabweichung des ROC-Integrals für den Testdatensatz von jeweils fünf NN pro Netzkonfiguration abgebildet. Die Einträge für die Zahl der verborgenen Schichten sind, zur besseren Übersicht, etwas versetzt nebeneinander angeordnet. Es ist zu erkennen, dass eine höhere Zahl an verborgenen Schichten zu keiner Erhöhung des ROC-Integrals führt. In diesem Fall erzielen die Netze mit nur einer verborgenen Schicht sogar die beste Trennung, auch wenn der Unterschied zu den anderen NN-Konfigurationen gering ausfällt.



(a) (b)

(c)

Abbildung 6.6: **Vergleich der Ausgabewerte der** 1×500 **bzw.** 8×500 **Netze mit den Testdaten.** In (a) bzw. (b) ist die Verteilung der Ausgabewerte für ein 1×500 bzw. 8 Netz zu sehen. Für das 8×500 Netz nimmt die Ausgabe nur Werte zwischen 0,2 und 0,7 an, wohingegen die Ausgabe des 1×500 Netz von 0 bis 1 verläuft. Die ROC-Kurven beider Netze in (c) unterscheiden sich jedoch kaum von einander.

Lediglich die Anzahl der Neuronen pro Schicht nimmt Einfluss auf das ROC-Integral. Dabei bleibt festzustellen, dass sich eine größere Zahl von Neuronen positiv auf das ROC-Integral auswirkt. Interessant ist, dass das ROC-Integral, für eine bestimmte Neuronenzahl, ab einer gewissen Zahl von verborgenen Schichten abnimmt. Zudem steigt dabei auch die Standardabweichung, was auf ein nicht stabiles Training hinweist. Zum Beispiel fällt das ROC-Integral für Netze mit nur 15 Neuronen pro Schicht schon ab 5 verborgenen Schichten ab. Bei der Zahl von 8 verborgenen Schichten scheinen lediglich die Netze mit 300 und 500 Neuronen pro Schicht noch konstante Werte für das ROC-Integral zu liefern. Beim Vergleich der Ausgabe der Netze 1×500 und 8×500 in Abb. 6.6 für den Testdatensatz fällt auf, dass das 8×500 Netz nur Werte im Bereich $0{,}2 \leq \hat{y} \leq 0{,}7$ ausgibt. Signal- und Untergrundverteilung haben ein gemeinsames Maximum bei $\hat{y} \approx 0{,}5$ und jeweils ein Maximum bei $\hat{y} \approx 0{,}3$ bzw. 0,6. Zudem unterscheidet sich die Form der Verteilung des 8×500 NN deutlich von der des 1×500 Netz. Die ROC-Kurven beider Netze unterscheiden sich jedoch kaum von einander. Die deutliche Abweichung der Verteilungsform deutet darauf hin, dass das Training eines DNN mit den verwendeten Daten deutlich schwieriger scheint.

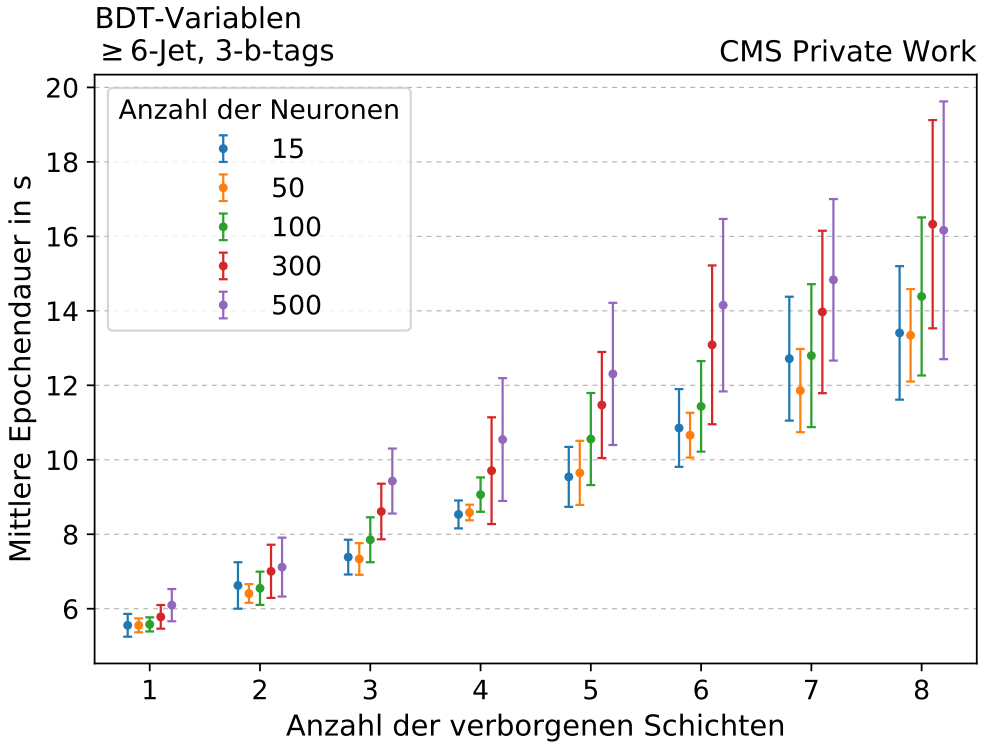


Abbildung 6.7: **Abhängigkeit des Trainingsdauer pro Epoche von der Anzahl der verborgenen Schichten und Neuronen.** Zur besseren Übersicht sind die Einträge für eine verborgene Schicht jeweils etwas versetzt nebeneinander angeordnet. Die Fehlerbalken geben die Standardabweichung vom Mittelwert an. Alle anderen Parameter sind für jede Netzkonfiguration identisch gewählt und werden im Text beschrieben.

In Abb. 6.7 ist die mittlere Trainingsdauer pro Epoche in Abhängigkeit der Netzgröße zu sehen. Die Werte stammen aus der Studie zur Abhängkeit des ROC-Integrals von der Netzgröße. Die Epochendauer steigt linear mit der Anzahl der verborgenen Schichten an. Dies entspricht den Erwartungen, da der Backpropagation-Algorithmus (vgl. Gl. 4.5), der zur Berechnung der Gradienten für das Training verwendet wird, von der Ordnung $\mathcal{O}(n)$, also linear in der Anzahl der Netzparameter n ist [18, S. 218].

Zur Kontrolle der Ergebnisse aus Abb. 6.5 wird dieselbe Studie auch noch für die 5-Jets, 3-b-tags Kategorie und zusätzlich ohne Einordnung in Kategorien durchgeführt, wobei alle Ereginisse betrachtet werden. Die Zusammensetzung der Datensätze ist in Tab. 6.2 zu entnehmen. Für die 5-Jets, 3-b-tags Kategorie werden die gleichen Trainingsparameter wie für die $\geq$ 6-Jet, 3-b-tags Kategorie gewählt. Beim Training mit den Daten ohne Kategorisierung ändert sich nur die Batchgröße von 128 auf 1048. Die Vergrößerung der Batchgröße wird vorgenommen, da sich herausstellt, dass bei der momentanen Implementierung des FeedForward-Netzes in NNFlow die Epochendauer trotz kleinerer Batchgröße nicht weiter verringert. Bei der Betrachtung der Ergebnisse in der 5-Jets, 3-b-tags Kategorie in Abb. 6.8a lassen sich kaum Veränderungen zur $\geq$ 6-Jet, 3-b-tags Kategorie feststellen.

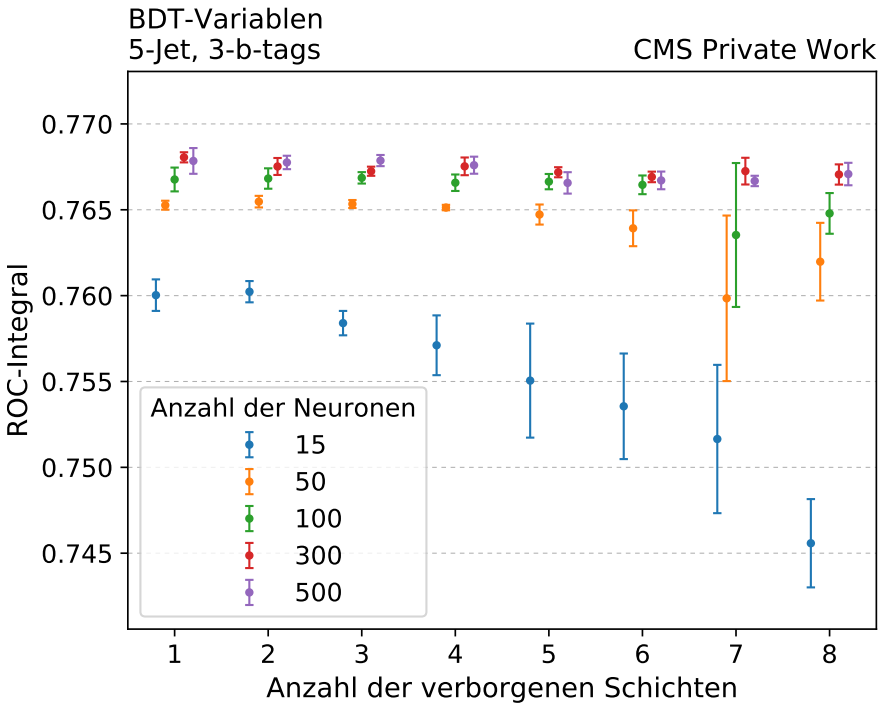
Tabelle 6.2: **Anzahl Ereignisse der in dieser Studie verwendeten Datensätze.** Alle Zahlenwerte sind mit 10^3 zu multiplizieren.

	Training		Validierung		Test	
Kategorie	Untergrund	Signal	Untergrund	Signal	Untergrund	Signal
$\geq$ 6-Jets, 3-b-tags	160,0	46,3	32,0	9,3	127,8	37,1
5-Jets, 3-b-tags	211,4	34,8	42,3	7,0	169,1	27,8
keine Kategorisierung	714,9	143,6	143,0	28,7	571,9	114,9

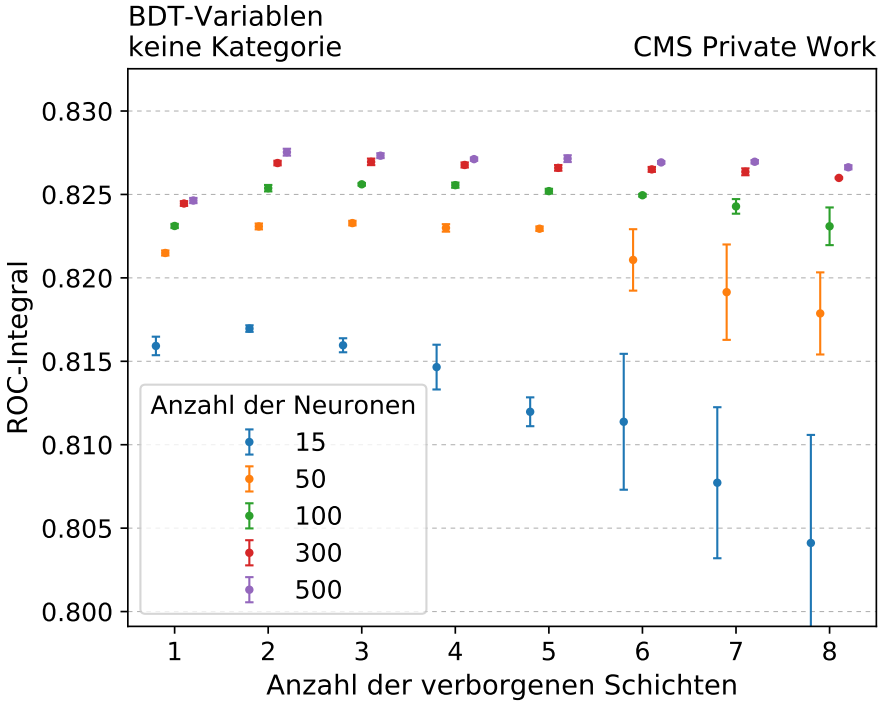
Wieder erzielen die NN mit nur einer verborgenen Schicht die besten ROC-Integrale. Auch ist der Abfall des ROC-Integrals mit zunehmender Zahl der verborgenen Schichten und kleiner Neuronenzahl zu beobachten.

Jedoch verbessert sich das ROC-Integral beim Training ohne Kategorisierung, zu sehen in Abb. 6.8b, bei Netzen mit zwei verborgenen Schichten im Vergleich zu einer Schicht. Danach fällt dieser allerdings auch wieder ab. Auffällig ist außerdem, dass das Training mit allen Ereignissen, für NN mit einer hohen Zahl an Neuronen, äußerst stabil ist, da die ROC-Integrale pro Netz hier kaum streuen. Die größeren ROC-Integrale bei der Trennung von Signal- und Untergrund ohne Einordnung in eine Kategorie im Vergleich zur $\geq$ 6-Jet, 3-b-tags oder 5-Jets, 3-b-tags Kategorie, könnten dadurch erklärt werden, dass der Datensatz mit allen Ereignissen auch $t\bar{t}$-Untergrund mit Jets aus leichten Quarks enthält. Da diese sich mehr von den $t\bar{t}H$-Signalereignissen unterscheiden, fällt es dem Netz leichter, diese Untergründe besser vom Signal zu trennen. Der Effekt, dass sich die Verteilung der NN-Ausgabe mit einer höheren Zahl verborgener Schichten wie in Abb. 6.6 verändert, kann auch für die NN der 5-Jets, 3-b-tags Kategorie sowie ohne Kategorie beobachtet werden.

Dass sich die Trennung von $t\bar{t}H$-Signal- und $t\bar{t}$-Untergrundereignissen mit steigender Zahl verborgener Schichten nicht verbessert, kann verschiedene Ursachen haben. Zum einen ist es möglich, dass einfach nicht genügend Trainingsereignisse zur Verfügung stehen. Dies sieht man daran, dass sich das ROC-Integral bei Netzen mit zwei verborgenen Schichten im Vergleich zu einer Schicht verbessert, wenn für das Training alle Ereignisse, ohne Einteilung in Kategorien, verwendet werden. Eine andere Möglichkeit besteht darin, dass das Klassifikationsproblem beim $t\bar{t}H$-Prozess am besten durch NN mit einer verborgenen Schicht gelöst wird. Das kann bedeuten, dass ein NN im Falle des $t\bar{t}H$-Prozesse nicht wie erhofft neue Zusammenhänge aus den Variablen lernt und dass deshalb weiterhin die Eingabevariablen sorgfältig ausgewählt werden müssen. Abschließend bleibt zu sagen, dass das Training von NN mit vielen Schichten oft nicht einfach ist, das sehr viele Faktoren, wie z.B. Initialisierung der Parameter, den Erfolg des Trainings beeinträchtigen können. Eine ausführliche Beschreibung der allgemeinen Probleme beim Training von DNN findet sich in [40].



(a)



(b)

Abbildung 6.8: **Abhängigkeit des ROC-Integrals von der Anzahl der verborgenen Schichten und Neuronen.** Zur besseren Übersicht sind die Einträge für eine verborgene Schicht jeweils etwas versetzt nebeneinander angeordnet. Die Fehlerbalken geben die Standardabweichung vom Mittelwert an. Alle anderen Parameter sind für jede Netzkonfiguration identisch gewählt und werden im Text beschrieben. In (a) ist die Abhängigkeit bei NN, die mit Ereignisse in der 5-Jet, 3-b-tag Kategorie trainiert sind, zu sehen. Entsprechend ist in (b) die Abhängigkeit für NN, die mit Ereignissen ohne Kategorisierung trainiert sind, zu sehen.

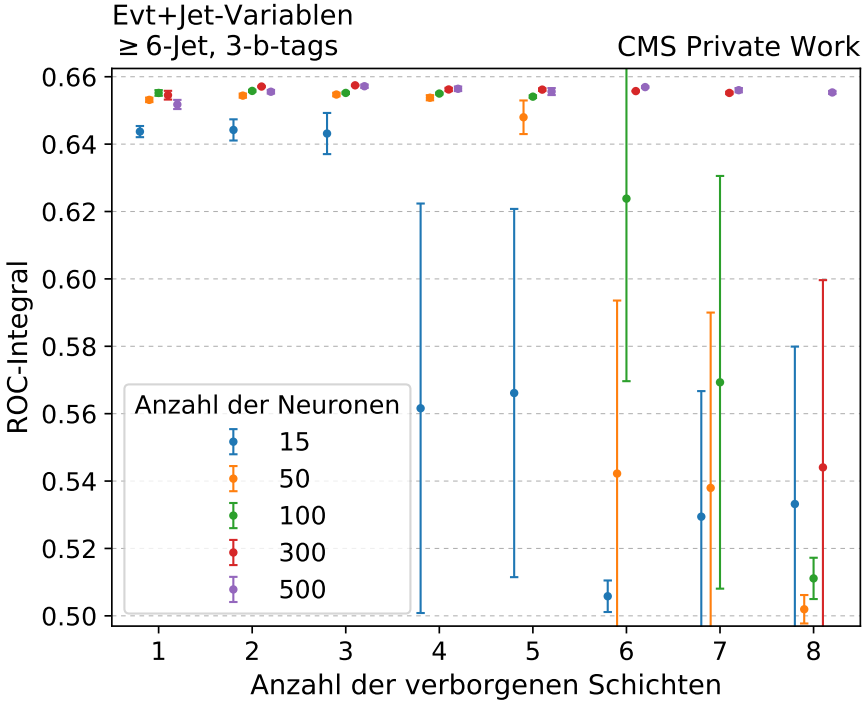
6.3 Wahl der Ereignisvariablen

Die Variablen, die für die Studien in Kap. 6.2 verwendet werden, sind so ausgewählt, dass sie eine gute Trennung zwischen Signal und Untergrund bieten. Die bisher als BDT-Variablen bezeichneten Größen sind dabei aus den Vierer-Impulsen der Jets, der Leptonen, der fehlenden Transversalenergie und den b-tagging-Informationen berechnet. Im Folgenden soll untersucht werden, ob eine Klassifikation von $t\bar{t}H$- und $t\bar{t}$-Ereignissen mit einem FeedForward-Netz auch direkt mit einfachen Variablen, aus denen die BDT-Variablen konstruiert sind, möglich ist. Dazu werden nun anstatt der BDT-Variablen die ihnen zugrundeliegenden Größen verwendet. Im folgenden wird diese Auswahl von Variablen als „Evt+Jet-Variablen“ bezeichnet.

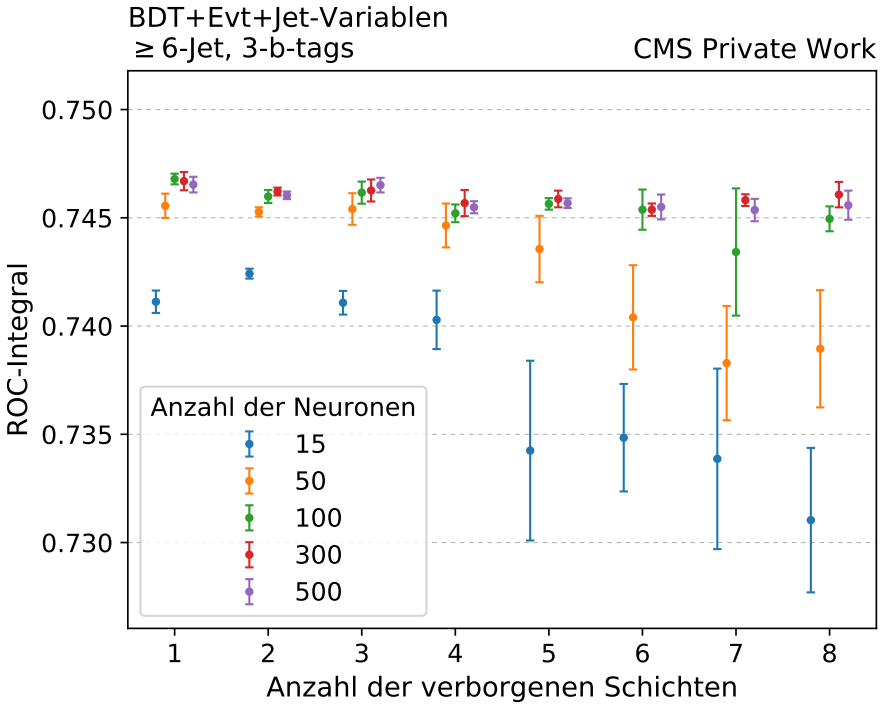
Zum Vergleich wird dafür die Studie aus Kap. 6.2 nun einmal mit den Variablen, die in Anhang C gezeigt sind, und einmal mit beiden, also „BDT+Evt+Jet-Variablen“, in der 6-Jet, 3-b-tags Kategorie wiederholt. Die Gewichtung der Ereignisse erfolgt genauso wie in Kap. 6.2. Auch die Parameter des Trainingsprozess werden nicht verändert.

In Abb. 6.9a sind die Ergebnisse der Studie für die Evt+Jet-Variablen zusammengefasst. Die besten Werte der ROC-Integrale der Netze, deren Training mit den Evt+Jet-Variablen durchgeführt wird, sind im Vergleich zu den Netzen, die mit den BDT-Variablen trainiert sind, um etwa 11,3% kleiner. Der Abfall der ROC-Integrale bei einer höheren Anzahl an verborgenen Schichten ist auch hier vorhanden. Zudem streuen diese Werte mehr als in der in der vorherigen Studie. Das ROC-Integral von Netzen mit vielen verborgenen Schichten ist zudem mit Werten knapp über 0,5 sehr schlecht, da ein Klassifikator mit einem ROC-Integral von 0,5 keine Trennung zwischen Signal und Untergrund liefert. Diese Ergebnisse zeigen, dass einfache Variablen nicht zur Trennung von $t\bar{t}H$- und $t\bar{t}$-Ereignissen mit einem FeedForward-Netz geeignet sind. Abb. 6.9b zeigt die ROC-Integrale der Netze, die mit den Evt+Jet- und BDT-Variablen trainiert sind. Die ROC-Integrale sind minimal kleiner als für Netze, die nur mit BDT-Variablen trainiert sind (vgl. Abb. 6.5). Ansonsten sind keine Unterschiede zwischen beiden Schaubildern festzustellen.

Diese Studie zeigt, dass die Klassifikation von $t\bar{t}H$-Ereignissen mit FeedForward-Netzen, die mit einfachen Variablen, wie sie hier verwendet werden, schlechter ist als mit NN, die mit gut trennenden Variablen trainiert sind. Deshalb sollten die Variablen, die zur Trennung von $t\bar{t}H$-Signal und $t\bar{t}$-Untergrund mit FeedForward-Netzen verwendet werden, weiterhin sorgfältig ausgewählt werden.



(a)



(b)

Abbildung 6.9: **Abhängigkeit des ROC-Integrals von der Anzahl der verborgenen Schichten und Neuronen.** Zur besseren Übersicht sind die Einträge für eine verborgene Schicht jeweils etwas versetzt nebeneinander angeordnet. Die Fehlerbalken geben die Standardabweichung vom Mittelwert an. Alle anderen Parameter sind für jede Netzkonfiguration identisch gewählt und werden im Text beschrieben. In (a) sind die Ergebnisse für das Training mit den einfachen Evt+Jet-Variablen zu sehen. (b) zeigt die Ergebnisse für das Training mit BDT- und Evt+Jet-Variablen.

7 Fazit und Ausblick

In dieser Arbeit werden die Grundlagen zur Signal- und Untergrundtrennung mit TensorFlow in einer Suche nach $t\bar{t}H$-Produktion am CMS-Experiment erarbeitet. Für die Studien wurden simulierte Ereignisse verwendet. Die Klassifikation der Ereignisse findet mit neuronalen Netzen statt. Dazu ist das Analyseframework NNFlow basierend auf Tensorflow entwickelt worden. Dabei ist besonderer Wert auf eine einfache Bedienung des Frameworks gelegt worden, sodass es möglich ist, neuronale Netze ohne Vorwissen zu TensorFlow zu trainieren. Zusätzlich erlaubt NNFlow die Umwandlung von ROOT-Dateien in NumPy-Arrays sowie diverse Vorverabeitungsschritte wie die Variablenauswahl oder Berechnung der Ereignisgewichte. Unter Verwendung von NNFlow wird untersucht, wie gewichtete Ereignisse beim Training neuronaler Netze berücksichtigt werden können. Dafür wird gezeigt welche Auswirkungen die Multiplikation des Gewichtes eines Ereignisses an die Fehlerfunktion, die zum Training neuronaler Netze minimiert wird, auf die Verteilung der Ausgabewerte des Netzes hat. Weiter wird ermittelt, welchen Einfluss die Netzgröße, also die Zahl der verborgenen Schichten und Neuronen pro Schicht, und die Wahl der Variablen auf das ROC-Integral des trainierten Netzes haben. Es stellt sich heraus, dass, bei Einteilung der Ereignisse in die bei der $t\bar{t}H$-Analyse üblichen Kategorien, neuronale Netze mit nur einer verborgenen Schicht und vielen Neuronen das größte ROC-Integral erzielen. Wird jedoch keine Einteilung in Kategorien vorgenommen, erzielt ein Netz mit zwei verborgenen Schichten das größte ROC-Integral. Bei der Wahl der Variablen zeigt sich, dass weiter auf eine sorgfältige Variablenauswahl geachtet werden muss. Netze, die mit gut trennenden Variablen trainiert werden, erzielen deutlich höhere ROC-Integrale als Netze, bei denen das Training nur mit einfachen Variablen stattfindet.

Die mit NNFlow trainierten neuronalen Netze können in entsprechenden Modell-Dateien gespeichert werden. Um diese Modelle zukünftig in der $t\bar{t}H$-Analyse einsetzen zu können, muss auf die C++-Schnittstelle zu TensorFlow zurückgegriffen werden. Der dafür benötigte Code sowie Anleitungen zur Kompilierung der entsprechenden Bibliotheken existiert und ist auf einem lokalen Computer getestet worden. Leider ist es bisher noch nicht geglückt die Kompilierung der TensorFlow-Bibliotheken zu automatisieren, sodass eine Nutzung von TensorFlow in der $t\bar{t}H$-Analyse derzeit nicht möglich ist. Ansonsten kann das NNFlow-Framework für zukünftige Studien im Rahmen weiteren Studien verwendet werden. Dabei bietet sich vor allem ein Vergleich zu den in der $t\bar{t}H$-Analyse verwendeten Boosted-Decision-Trees oder eine Studie zur Variablenauswahl für NN an. Weiter muss geklärt werden, wie die Trennung von Signal und Untergrundereignissen mit neuronalen Netzen genau funktioniert. Offene Fragen sind u.a.: Welche Korrelationen lernt das Netz aus den Daten? Findet im

NN z.B. von selbst eine Kategorisierung nach der Anzahl der Jets und b-tags statt, falls diese Kategorisierung der Ereignisse nicht selbst vorgenommen wird?

Literaturverzeichnis

[1] CMS Collaboration, „Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC," Physics Letters B, Vol. 716, Nr. 1, S. 30 – 61, 2012.

[2] ATLAS Collaboration, „Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC," Physics Letters B, Vol. 716, Nr. 1, S. 1 – 29, 2012.

[3] M. Thomson, Modern Particle Physics, 6. Aufl. Cambridge u. a.: Cambridge University Press, 2013.

[4] B. Povh, Teilchen und Kerne : Eine Einführung in die physikalischen Konzepte, 9. Aufl. Berlin, Heidelberg: Springer Spektrum, 2014.

[5] S. Weinberg, „A Model of Leptons," Phys. Rev. Lett., Vol. 19, S. 1264–1266, 1967. http://link.aps.org/doi/10.1103/PhysRevLett.19.1264 Abgerufen am 07.03.2017.

[6] O. Brüning et al., LHC Design Report. Geneva: CERN, 2004. https://cds.cern.ch/record/782076

[7] ALICE Collaboration, „The ALICE experiment at the CERN LHC," Journal of Instrumentation, Vol. 3, Nr. 08, 2008.

[8] LHCb Collaboration, „The LHCb Detector at the LHC," Journal of Instrumentation, Vol. 3, Nr. 08, 2008.

[9] ATLAS Collaboration, „The ATLAS Experiment at the CERN Large Hadron Collider," Journal of Instrumentation, Vol. 3, Nr. 08, 2008.

[10] CMS Collaboration, „The CMS experiment at the CERN LHC," Journal of Instrumentation, Vol. 3, Nr. 08, 2008.

[11] D. Barney, „CMS slice raw illustrator files, CMS Document 5581-v1, Version vom 04.10.2011," 2011. https://cms-docdb.cern.ch/cgi-bin/PublicDocDB/ShowDocument?docid=5581 Abrufdatum: 20.02.2017.

[12] CMS Collaboration, „Search for $\mathrm{t\bar{t}H}$ production in the $\mathrm{H} \to \mathrm{b\bar{b}}$ decay channel with 2016 pp collision data at $\sqrt{s} = 13$ TeV," CMS-PAS-HIG-16-038, 2016. http://cds.cern.ch/record/2231510

[13] „LHC Higgs cross section working group." https://twiki.cern.ch/twiki/bin/view/LHCPhysics/LHCHXSWG Abgerufen am 26.02.2017.

[14] CMS Collaboration, „Identification of b quark jets at the CMS Experiment in the LHC Run 2," CMS-PAS-BTV-15-001, 2016. http://cds.cern.ch/record/2138504

[15] G. Bevilacqua et al., „Assault on the NLO wishlist: $pp \to t\bar{t} + b\bar{b}$," Journal of High Energy Physics, Vol. 2009, Nr. 09, S. 109, 2009. http://stacks.iop.org/1126-6708/2009/i=09/a=109 Abgerufen am 26.02.2017.

[16] „NNLO+NNLL top-quark-pair cross sections." https://twiki.cern.ch/twiki/bin/view/LHCPhysics/TtbarNNLO#Top_quark_pair_cross_sections_at Abgerufen am 26.02.2017.

[17] K. El Morabit, „A study of the multivariate analysis of Higgs boson production in association with a top quark-antiquark pair in the boosted regime at the CMS experiment," Masterarbeit, Karlsruher Institut für Technologie, Nr. IEKP-KA/2015-19, 2015.

[18] I. Goodfellow et al., Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

[19] K. He et al., „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," arXiv:1502.01852, Version vom 06.02.2015, 2015. https://arxiv.org/abs/1502.01852 Abrufdatum: 04.01.2017.

[20] I. Goodfellow et al., „Maxout networks." arXiv:1302.4389, Version vom 18.02.2013, 2013. https://arxiv.org/abs/1302.4389 Abrufdatum: 04.01.2017.

[21] A. Krizhevsky et al., „Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, S. 1097–1105.

[22] A. Karpathy, „CS231n: Convolutional Neural Networks for Visual Recognition," 2016. http://cs231n.github.io/neural-networks-1/ Abrufdatum: 04.01.2017.

[23] M. A. Nielsen, Neural networks and Deep Learning. Determination Press, 2015. http://neuralnetworksanddeeplearning.com Abrufdatum: 05.01.2017.

[24] D. Rumelhart et al., „Learning representations by back-propagating errors," Nature, Vol. 323, S. 533–536, 1986.

[25] Y. LeCun et al., „Efficient backprop," in Neural networks: Tricks of the trade. Springer, 2012, S. 9–48. http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf Abrufdatum: 04.01.2017.

[26] N. Srivastava et al., „Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research, Vol. 15, Nr. 1, S. 1929–1958, 2014.

[27] M. Abadi et al., „TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. http://tensorflow.org/

[28] M. Welsch, „NNFlow, Version: BachelorThesisFinal," 2017. https://github.com/welschma/NNFlow Abrufdatum 07.03.2017.

[29] A. Buckley et al., „General-purpose event generators for LHC physics," Phys. Rept., Vol. 504, S. 145–233, 2011.

[30] B. Webber, „Parton shower Monte Carlo event generators," Scholarpedia, Vol. 6, Nr. 12, 2011. http://www.scholarpedia.org/article/Parton_shower_Monte_Carlo_event_generators Abgerufen am 28.02.2017.

[31] R. Brun und F. Rademakers, „ROOT - An Object Oriented Data Analysis Framework," Nucl. Inst. & Meth. in Phys. Res. A, Vol. 389, S. 81–86, 1996. http://root.cern.ch/

[32] S. van der Walt et al., „The NumPy Array: A Structure for Efficient Numerical Computation," Computing in Science Engineering, Vol. 13, Nr. 2, S. 22–30, 2011.

[33] N. Dawe et al., „rootnumpy: 4.3.0," 2015. https://rootpy.github.io/root_numpy/index.html

[34] P. Goldsborough, „A Tour of TensorFlow," arXiv:1610.01178, Version vom 01.10.2016, 2016. https://arxiv.org/abs/1610.01178 Abrufdatum: 23.02.2017.

[35] F. Pedregosa et al., „Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, Vol. 12, S. 2825–2830, 2011.

[36] D. Kingma und J. Ba, „Adam: A Method for Stochastic Optimization," arXiv:1412.6980, Version vom 23.07.2015, 2014. https://arxiv.org/abs/1412.6980 Abrufdatum: 30.01.2017.

[37] A. Hocker et al., „TMVA - Toolkit for Multivariate Data Analysis," PoS, Vol. ACAT, S. 40, 2007.

[38] C. Szegedy et al., „Going DeDeep with Convolutions," arXiv:1409.4842, Version vom 27.09.2014, 2014. https://arxiv.org/abs/1409.4842 Abrufdatum: 15.02.2017.

[39] Y. Bengio, „Learning Deep Architectures for AI," Foundations and trends® in Machine Learning, Vol. 2, Nr. 1, S. 1–127, 2009. http://www.iro.umontreal.ca/~bengioy/papers/ftml_book.pdf Abrufdatum: 15.02.2017.

[40] X. Glorot und Y. Bengio, „Understanding the difficulty of training deep feedforward neural networks," in Aistats, Vol. 9, 2010, S. 249–256. http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf Abrufdatum: 15.02.2017.

Anhang

A Verwendete Variablen zur Studie zum Umgang mit gewichteten Ereignissen

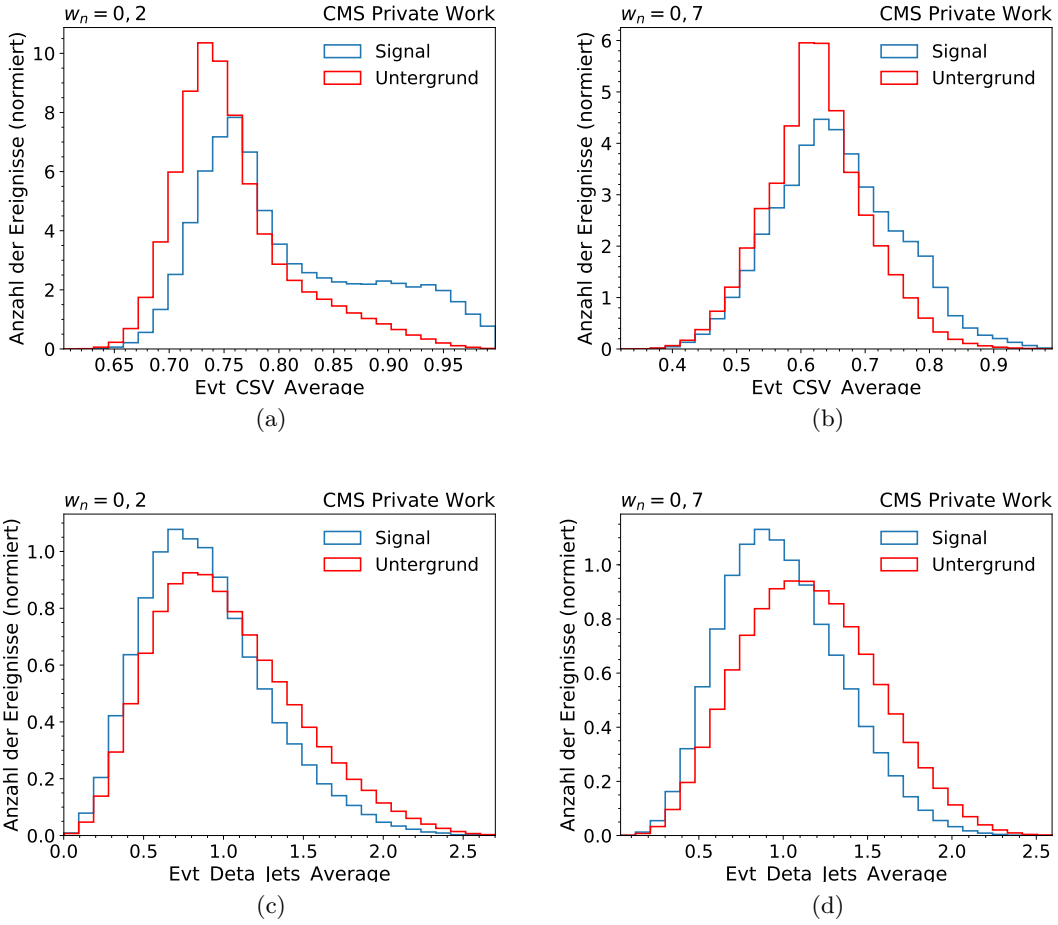
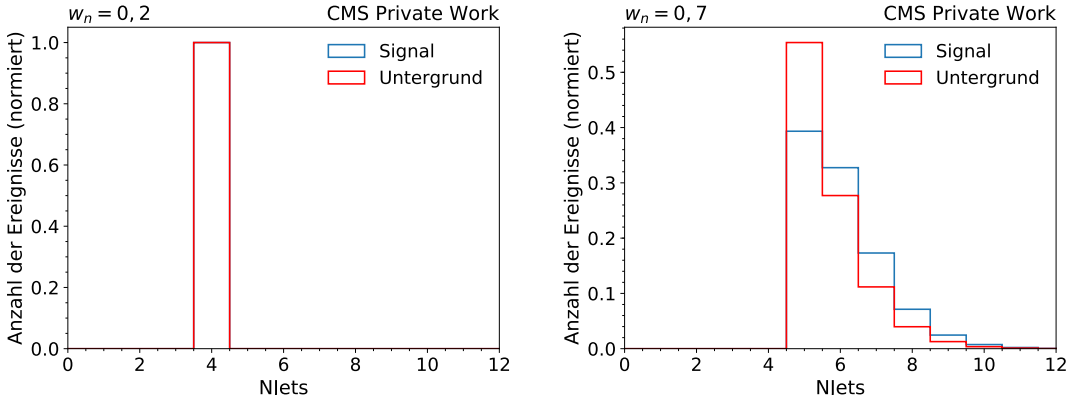


Abbildung A.1: **Die ersten beiden Variablen, die in der Studie 6.1 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert. Die Variablen sind für verschieden gewichtete Ereignisse getrennt gezeigt.

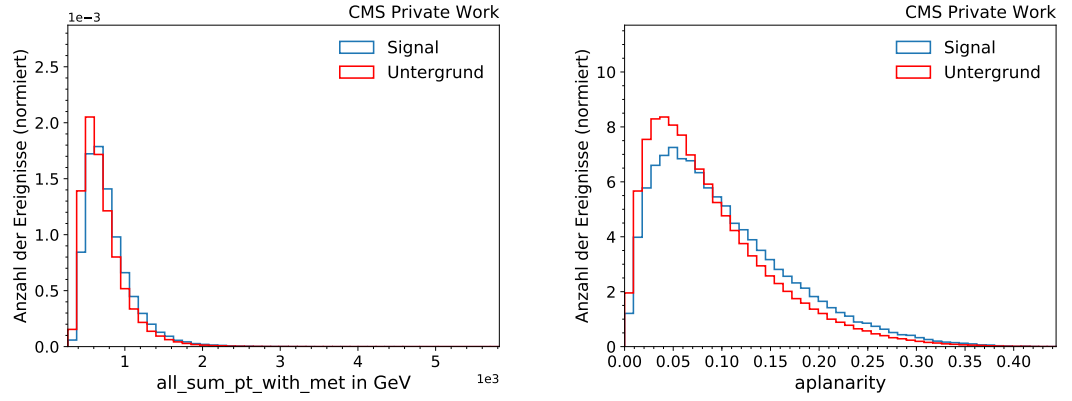


(a) (b)

Abbildung A.2: **Die verbleibende Variable, die in der Studie 6.1 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert. Die Variablen sind für verschieden gewichtete Ereignisse getrennt gezeigt.

B BDT-Variablen in der $\geq$ 6, 3-btag Kategorie

Im folgenden sind alle Ereignisvariablen, die in der Studie zum Einfluss der Netzgröße verwendet werden, in der $\geq$ 6, 3-btag Kategorie gezeigt. Eine physikalische Beschreibung der einzelnen Größen findet sich in [17].



(a) (b)

Abbildung B.1: **Variablen, die in der Studie 6.2 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

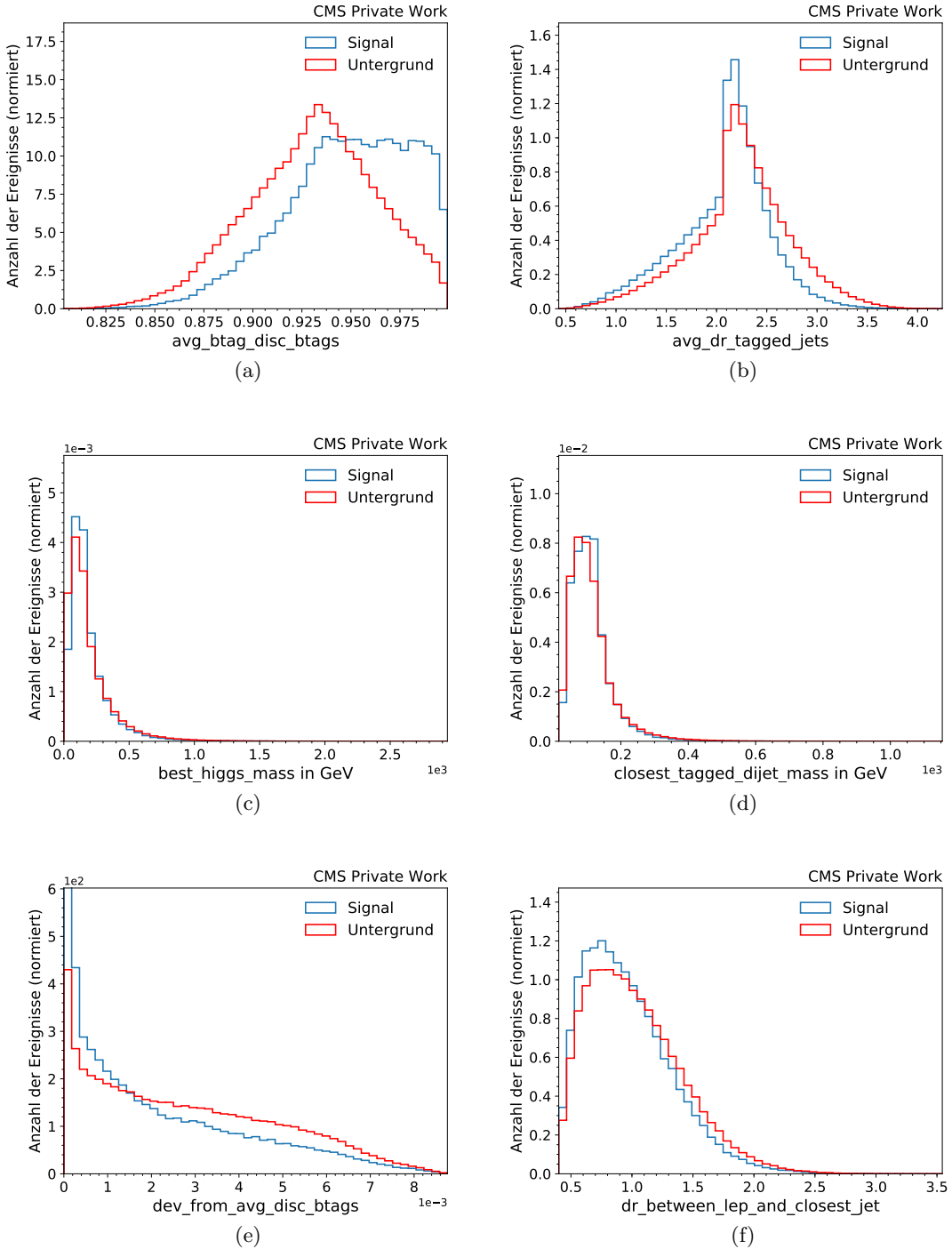


Abbildung B.2: **Fortsetzung der Variablen, die in der Studie 6.2 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

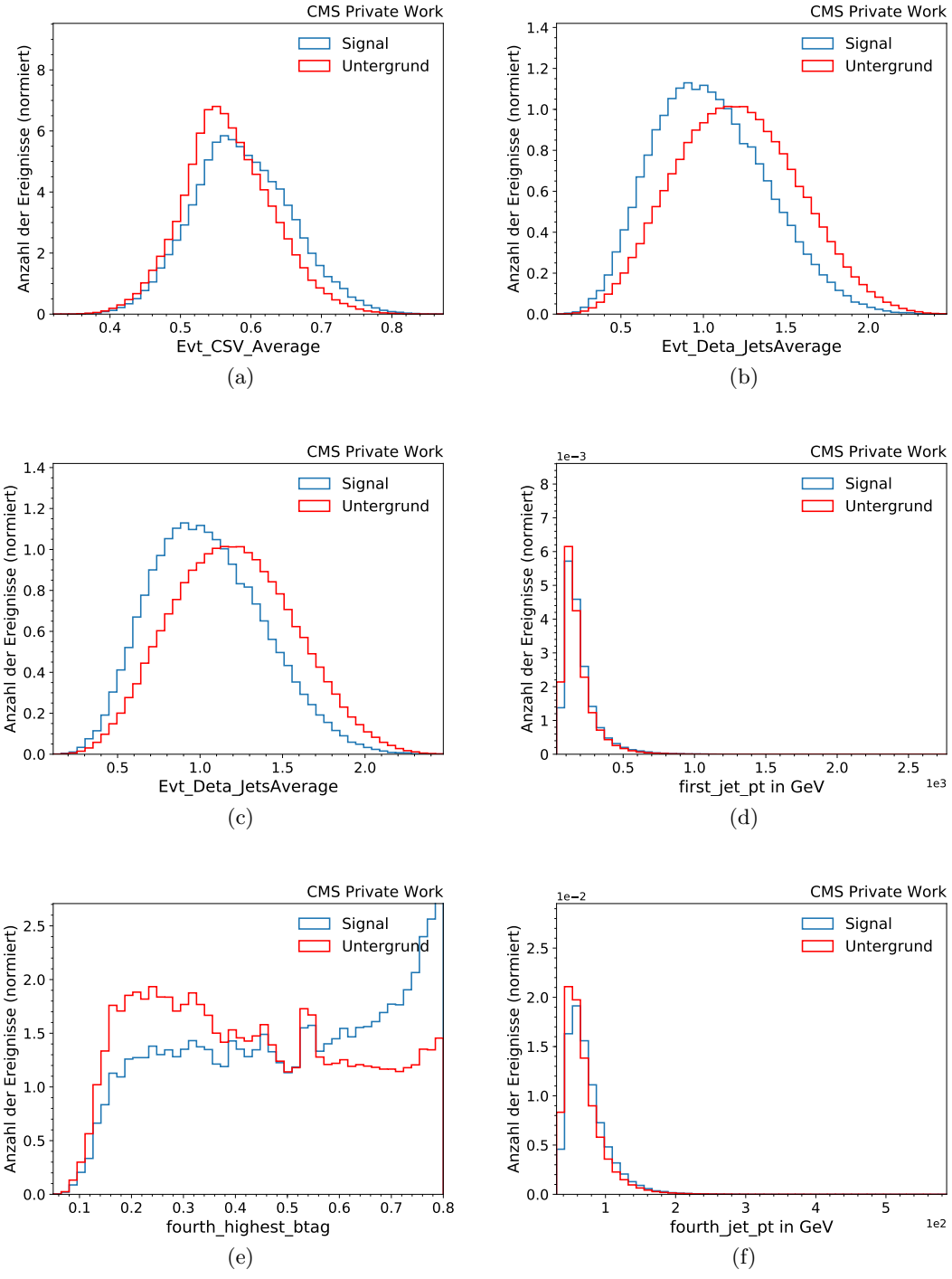


Abbildung B.3: **Fortsetzung der Variablen, die in der Studie 6.2 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

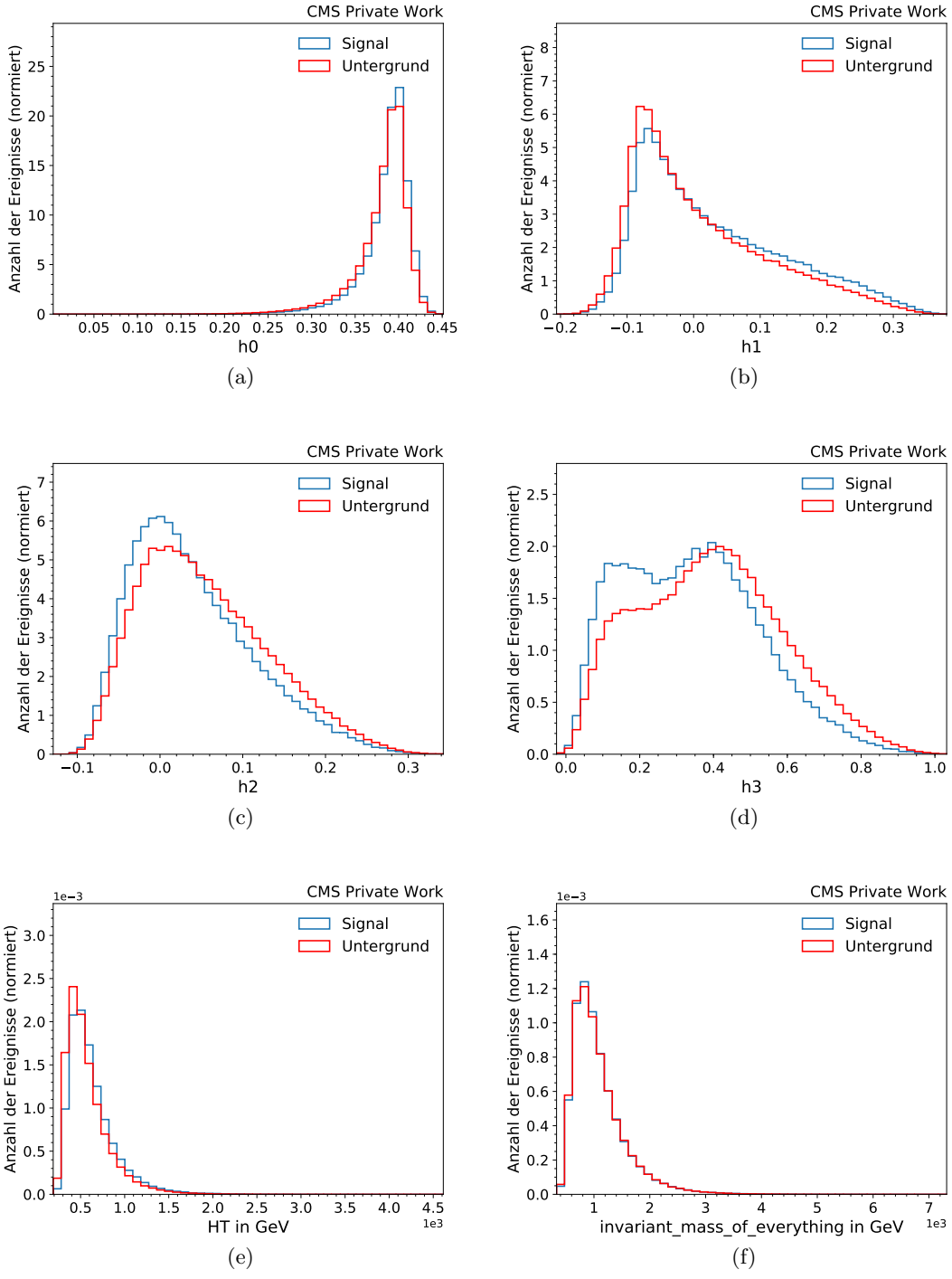


Abbildung B.4: **Fortsetzung der Variablen, die in der Studie 6.2 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

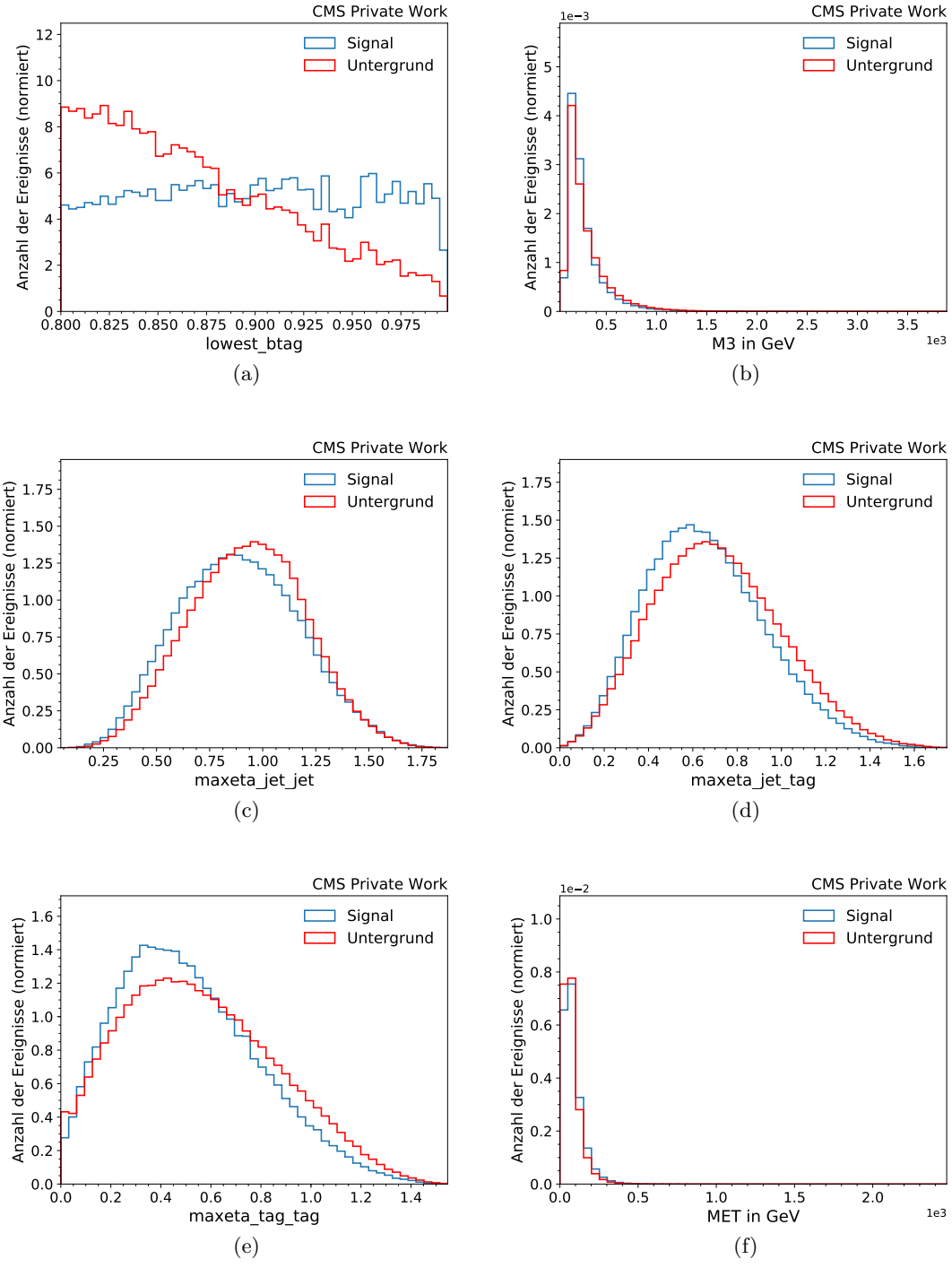


Abbildung B.5: **Fortsetzung der Variablen, die in der Studie 6.2 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

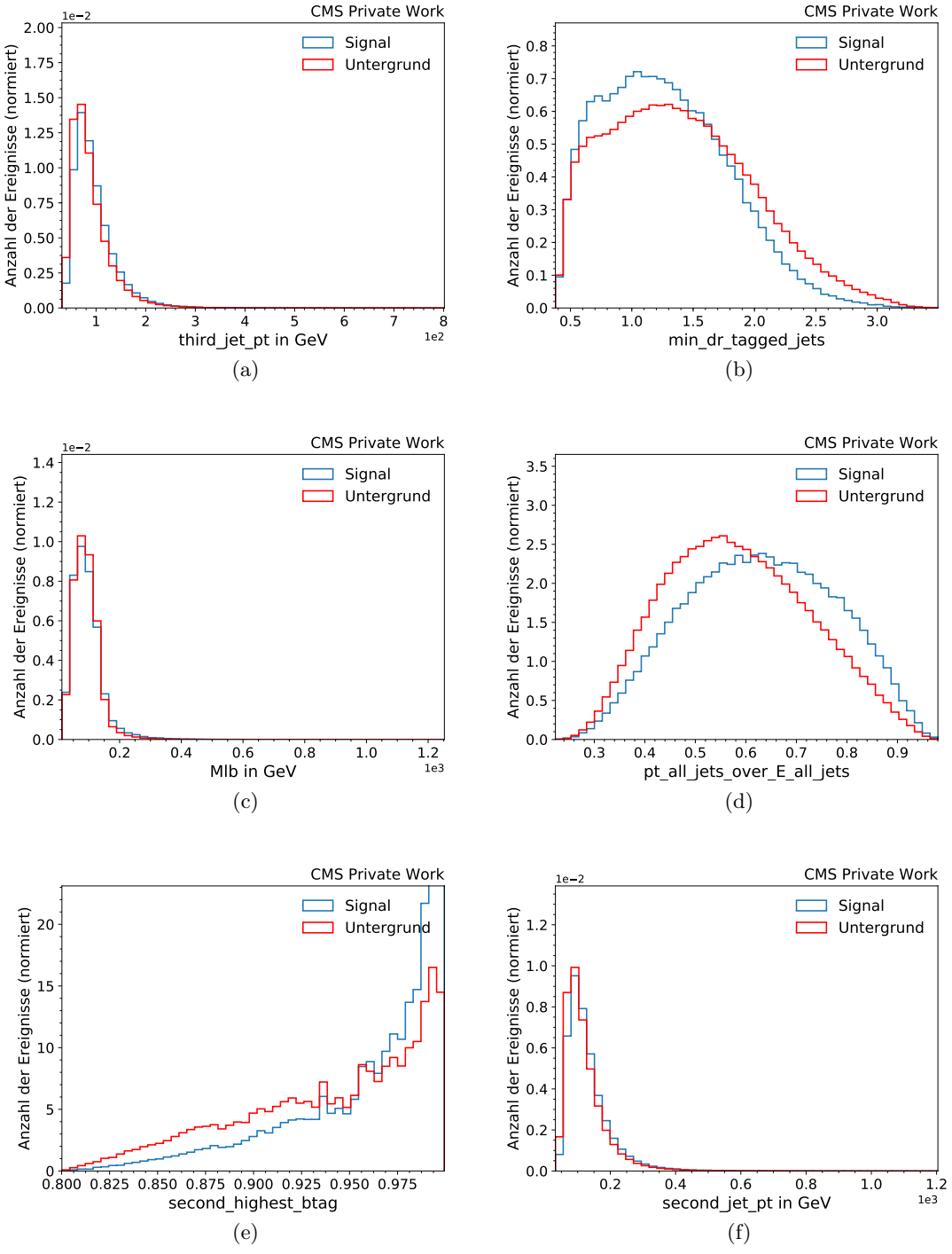


Abbildung B.6: **Fortsetzung der Variablen, die in der Studie 6.2 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

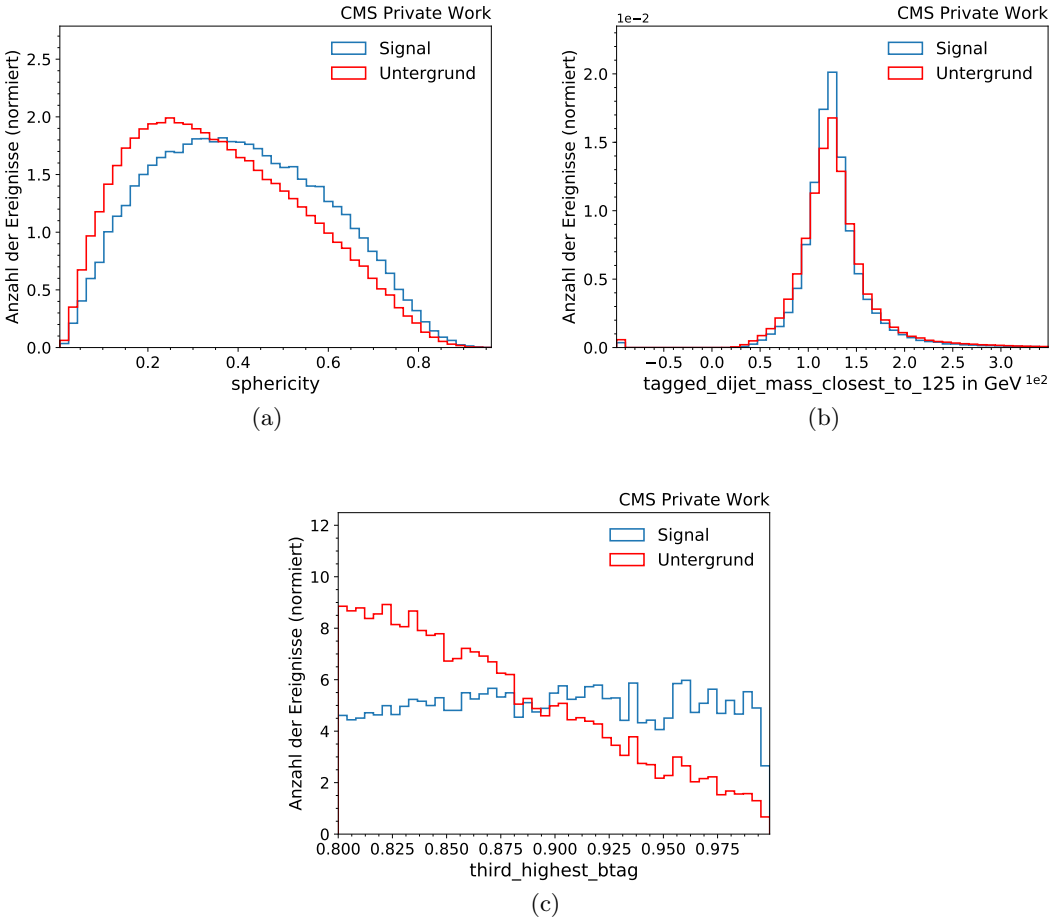


Abbildung B.7: **Fortsetzung der Variablen, die in der Studie 6.2 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

C Evt+Jet-Variablen in der ≥ 6, 3-btag Kategorie

Im folgenden sind alle Ereignisvariablen, die in der Studie zum Einfluss der Ereignisvariablen verwendet werden, in der ≥ 6, 3-btag Kategorie gezeigt. Dabei handelt es sich um kinematische Größen des isolierten Leptons, der fehlenden Transversalenergie und der vier Jets mit dem größten Transversalimpuls.

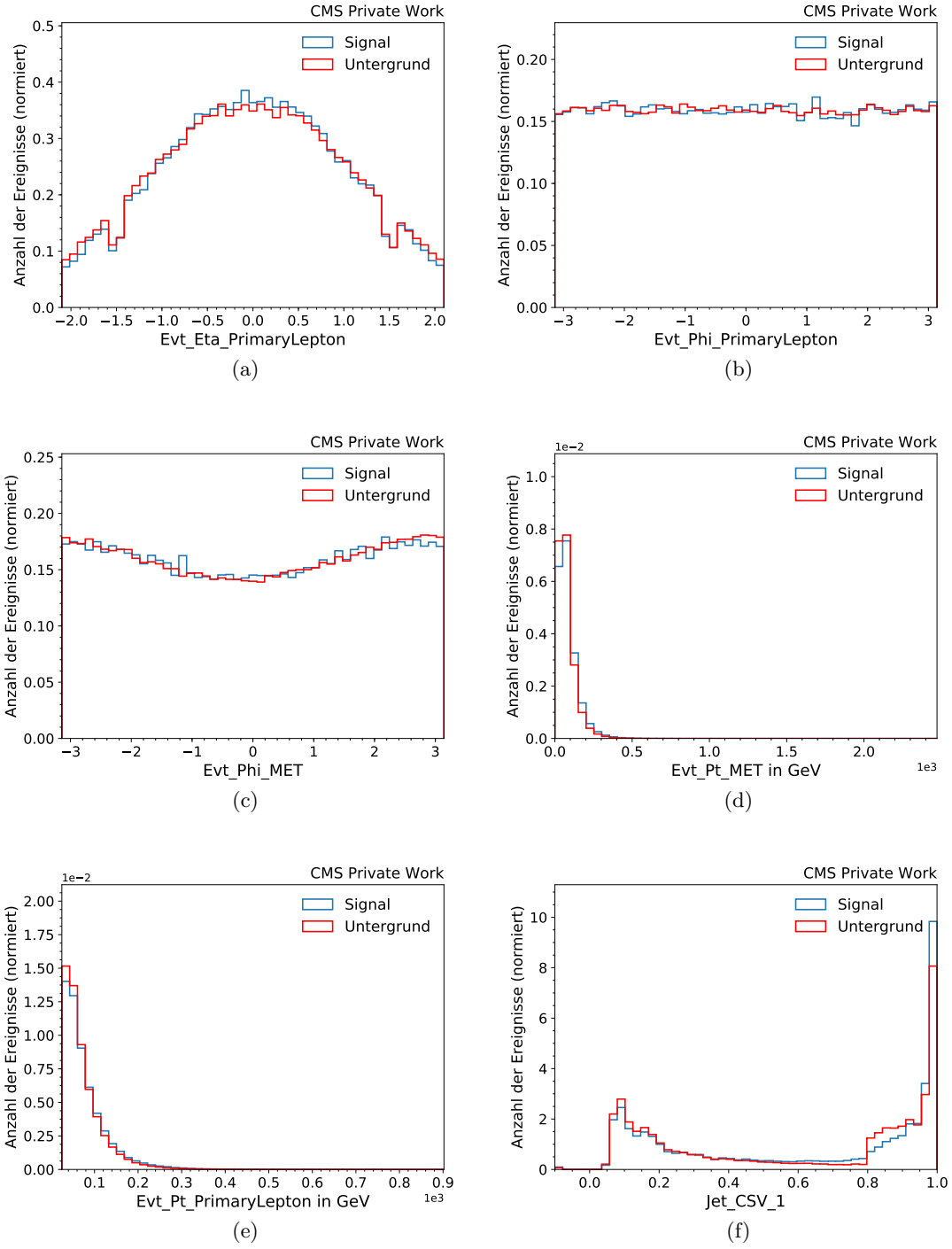


Abbildung C.1: **Variablen, die in der Studie 6.3 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

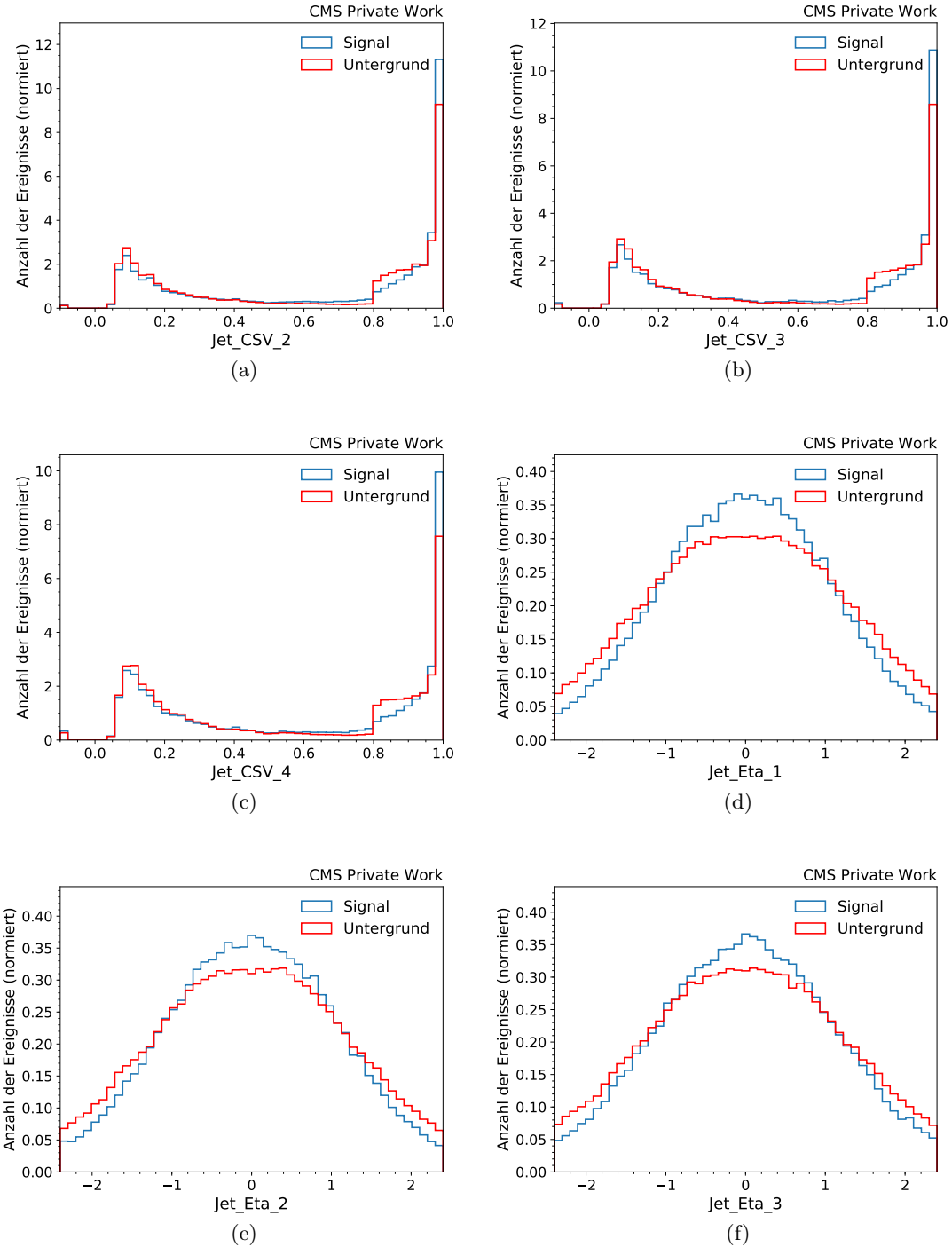


Abbildung C.2: **Fortsetzung der Variablen, die in der Studie 6.3 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

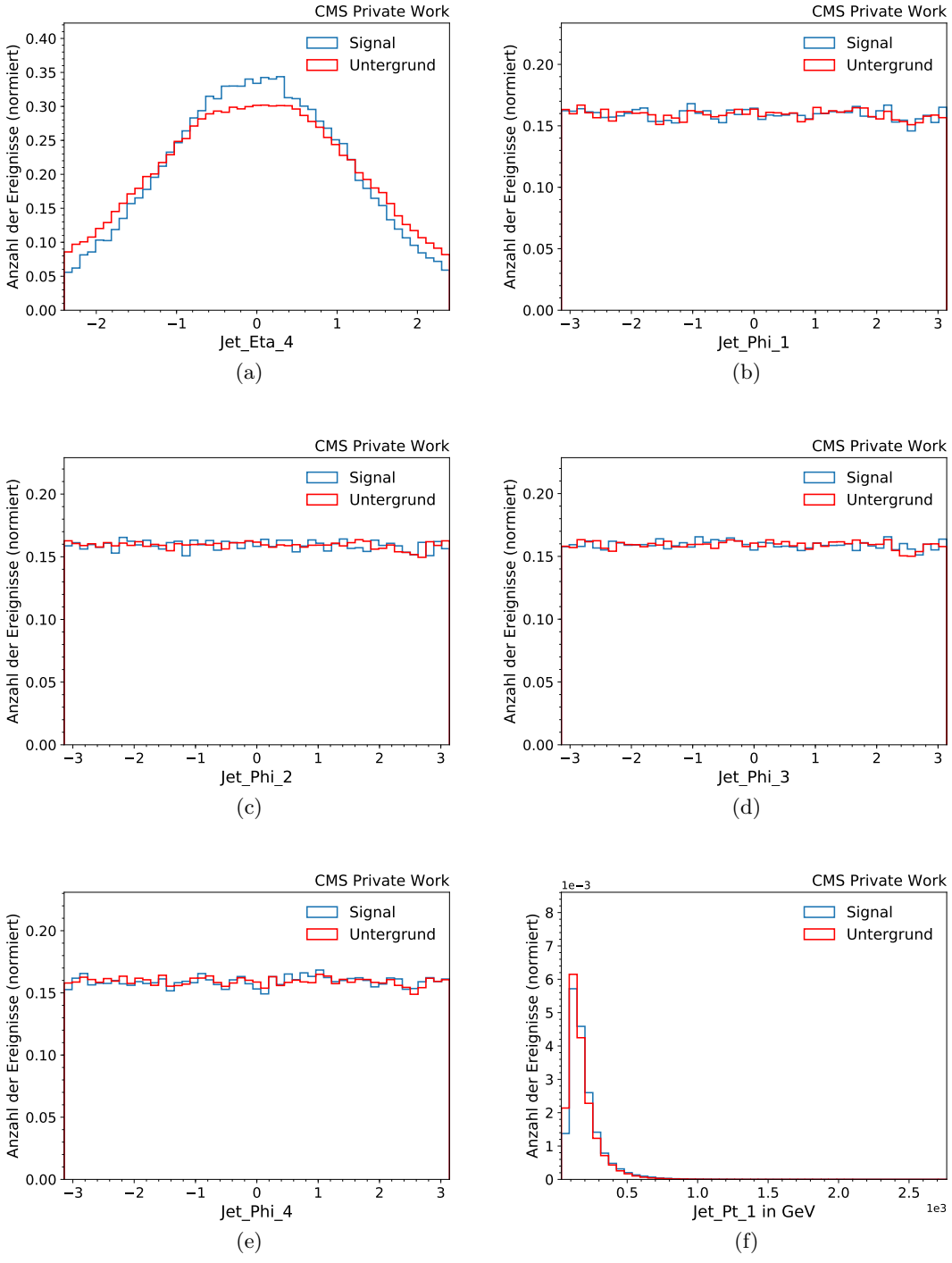


Abbildung C.3: **Fortsetzung der Variablen, die in der Studie 6.3 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.

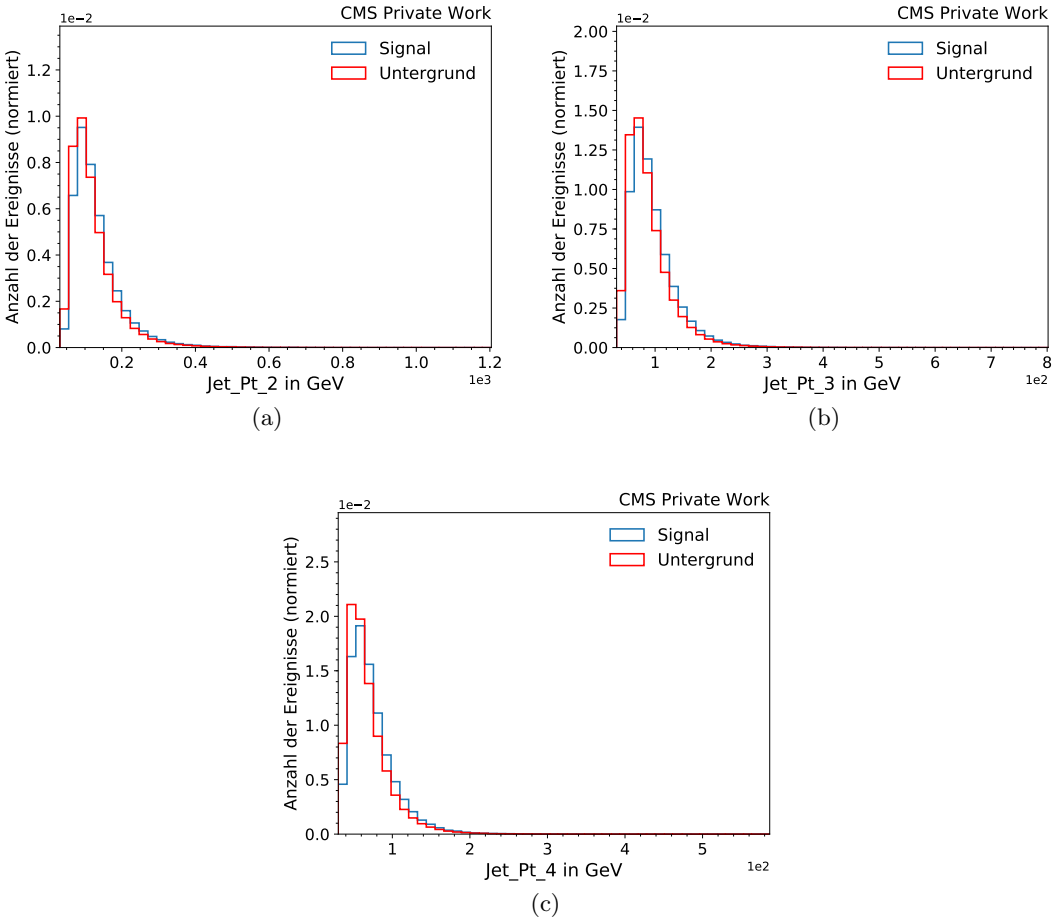


Abbildung C.4: **Fortsetzung der Variablen, die in der Studie 6.3 verwendet werden.** Die Fläche der Histogramme ist auf eins nomiert.