technische universität
dortmund

Hybrid bayessche neuronale Netze

Masterarbeit

Berat Özdemir
1. Oktober 2021

Supervisors:
Phillipp Oberdiek, M.Sc.
Prof. Dr.-Ing. Gernot A. Fink

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

INHALTSVERZEICHNIS

1 EINLEITUNG

Neuronale Netze kommen heutzutage in einem breiten Spektrum an Anwendungsgebieten zum Einsatz. Die Relevanz zeigt sich vor allem durch das wirtschaftliche Wertschöpfungspotenzial von feedforward- und Convolutional Neural Networks, welches auf jährlich zwischen $3,5$ und $5,8$ Billionen US-Dollar geschätzt wird [CMM+18]. Mit Convolutional Neural Networks kann beispielsweise das semantische Segmentierungsproblem aus der Computer Vision angegangen werden [HKH17]. Die Lösung dieses Problems ist insbesondere für Anwendungsfälle relevant, bei denen das Verständnis von Bildinhalten eine wichtige Rolle spielt. Beispielsweise müssen autonom fahrende Fahrzeuge notwendigerweise ihre Umgebung verstehen, welche unter anderem per Kamera aufgezeichnet wird: Die Unterscheidung zwischen Gehweg und Fahrbahn oder die Erkennung von Passanten, Hindernissen und Straßenschildern ist essenziell. Ein weiteres Beispiel wären hochpräzise chirurgische Eingriffe durch Roboter, wie das Implantieren von Elektroden auf die Gehirnoberfläche. Solch ein Eingriff geschieht bei der Verpflanzung des *Neuralink*[1] durch einen Roboter: Dieser muss $4-6\mu m$ dicke Elektroden auf die Gehirnoberfläche stechen, ohne dabei Blutgefäße zu treffen [Mus19]. Beide Anwendungsfälle sind höchst sicherheitskritisch, da fehlerhafte Interpretationen von Daten beziehungsweise falsche Vorhersagen des Modells unmittelbar zu Personenschaden führen können. Daher ist es von Interesse, die Zuverlässigkeit der Modellvorhersagen zu quantifizieren. Eine weitere Eigenschaft der obigen Anwendungsfälle ist die Echtzeit-Anforderung. Die Verarbeitung der Daten und somit die Vorhersage des Modells müssen möglichst schnell geschehen. Dies bedeutet, dass die Quantifizierung der Zuverlässigkeit der Modellvorhersage unter Berücksichtigung des Rechenaufwandes geschehen muss, um praktisch relevant zu sein.

Einen beliebten stellvertretenden Wert für die Zuverlässigkeit beziehungsweise Richtigkeit einer Klassifikation stellen die Werte des Softmax-Layers bei neuronalen Netzen dar [GBC16, pp.10-12]. Diese Werte sind nicht nur bei falscher Klassifikation sehr hoch, sondern auch bei Vorhersagen auf Grundlage ungeeigneter Eingaben. Diese Art von Eingaben werden *Out-of-Distribtution* Eingaben genannt, da sie nicht der Verteilung

1 Das Neuralink ist ein sogenanntes Brain-Computer-Interface, welches eine direkte Kommunikation zwischen Gehirn und Computer ermöglichen soll

der Ursprünglich vorgesehenen Eingaben entsprechen. Zum Beispiel könnte einem Softmax basierten binären Klassifikator, der dazu trainiert wurde vorherzusagen, ob auf einem Bild ein Hund oder eine Katze zu sehen ist, das Bild einer Maus als Eingabe vorgelegt werden. Die Klassifikation würde trotz ungeigneter Eingabe mit einem hohen Softmax-Wert stattfinden. Daher führt die Nutzung der Softmax-Werte als Maß für die Richtigkeit zu einer Überkonfidenz bei Vorhersagen [HG16].
Eine geeignetere Möglichkeit zur Abschätzung der Verlässlichkeit der Vorhersage bieten *bayes'sche neuronale Netze* [KSW15, BCKW15]. Diese erweitern klassische neuronale Netze insofern, als dass Unsicherheiten in den Modellparametern berücksichtigt werden. Statt einfacher Punktschätzung der Modellparameter θ, wird die Verteilung der Modellparameter durch eine Verteilung $q(\theta)$ approximiert [BCKW15]. Die Vorhersage geschieht über ein Ensemble an neuronalen Netzen, deren Parameter aus $q(\theta)$ gezogen werden. Die bekannte U-Net-Architektur [RFB15] hat beispielsweise ca. 30 Millionen Parameter [BJ20]. Bei einem bayes'schen U-Net-Pendant müssten daher bei der Inferenz mehrere Male 30 Millionen Parameter aus $q(\theta)$ gezogen werden und anschließend, für jede Probe aus der Verteilung, die Vorhersage des entsprechenden Modells berechnet werden. Dies hat zur Folge, dass die Inferenz mit bayes'schen neuronalen Netzen um einiges rechenintensiver ist, als mit klassischen Netzen.
Dieser Rechenaufwand ist bei zeitkritischen und energiekritischen Systemen problematisch. Daher sollen in dieser Arbeit hybride neuronale Netze untersucht werden, welche nur partiell bayessch sind. Das Ziel dabei ist es die Vorteile bayes'scher neuronaler Netze beizubehalten und dabei den Rechenaufwand zu reduzieren. Dies soll anhand von empirischen Experimenten geschehen.

2

GRUNDLAGEN

Dieses Kapitel beinhaltet die theoretischen Grundlagen zum Verständnis dieser Arbeit und zugleich eine Einführung in die verwendete Notation. Die Gliederung ist folgende: Zunächst werden in Kapitel 2.1 die nötigen wahrscheinlichkeitstheoretischen Grundlagen beschrieben, welche das Fundament für die *bayes'schen neuronalen Netze* in Abschnitt 2.6 darstellen. In Kapitel 2.3 gibt eine kurze Einleitung in die Grundlagen der Mustererkennung. Daraufhin wird auf Parameterschätzung und Optimierung eingegangen. Abschließend werden neuronale Netze und bayes'sche neuronale Netze erläutert.

2.1 KULLBACK-LEIBLER DIVERGENZ

In der Mustererkennung werden oftmals statistische Modelle genutzt um einen Klassifikator zu realisieren. Aus verschiedensten Gründen, kann es sein, dass eine Wahrscheinlichkeitsverteilung approximiert werden muss. Ein Grund dafür könnte eine angestrebte Vereinfachung des Modells sein. Ein anderer Grund könnte sein, dass die gesuchte Verteilung gänzlich unbekannt und insbesondere unzugänglich ist. Sei $f(x)$ die reale den Daten zu Grunde liegende Verteilungsdichte, also die Wahrheit. In der Praxis beschreibt diese beispielsweise hochkomplexe biologische oder physikalische Prozesse, welche nie in Gänze durch ein Modell beschrieben werden können. Daher ist $f(x)$ nicht berechenbar und ist es von Belangen, zwei Wahrscheinlichkeitsverteilungen in Relation zu einer unbekannten Verteilung vergleichen zu können. Ein Maß, welches die Ähnlichkeit zweier Wahrscheinlichkeitsverteilungen angibt, ist die *Kullback-Leibler-Divergenz* [Kul59].

Definition 1 *Seien* $p(x)$ *und* $q(x)$ *zwei Wahrscheinlichkeitsdichtefunktionen. Dann ist die Kullback-Leibler-Divergenz zwischen* $p(x)$ *und* $q(x)$ *wie folgt definiert.*

$$D_{KL}(\, p(x) \parallel q(x) \,) = \int_{-\infty}^{\infty} p(x) \ln(\frac{p(x)}{q(x)}) dx = \mathbb{E}_{p(x)}[\, \ln(\frac{p(x)}{q(x)}) \,] \tag{2.1.1}$$

Die Kullback-Leibler-Divergenz ist Null, wenn $q(x) = p(x)$ gilt:

$$\begin{aligned} D_{KL}(\, p(x) \parallel p(x) \,) &= \int_{-\infty}^{\infty} p(x) \ln(\frac{p(x)}{p(x)}) dx \\ &= \int_{-\infty}^{\infty} p(x) \ln(1) dx \\ &= \int_{-\infty}^{\infty} p(x) \cdot 0 dx \\ &= 0 \end{aligned}$$

Des Weiteren ist die Kullback-Leibler-Divergenz zweier beliebiger Wahrscheinlichkeitsdichtefunktionen durch Null nach unten beschränkt, also gilt

$$D_{KL}(\, p(x) \parallel q(x) \,) \geqslant 0 \tag{2.1.2}$$

Dazu wird zunächst gezeigt, dass die Beziehung

$$\forall x > 0.\ x - 1 \geqslant \ln(x) \tag{2.1.3}$$

gültig ist.

Beweis 1 *Sei* $f(x) = x - 1 - \ln(x)$*, die Differenz zwischen* $x - 1$ *und* $\ln(x)$*. Dann sind die Ableitungen durch* $f'(x) = 1 - \frac{1}{x}$ *und* $f''(x) = \frac{1}{x^2}$ *gegeben. Des weiteren ist die einzige Nullstelle von* f' *an der Stelle* $x = 1$*. Da* $f(x)$ *eine konvexe Funktion ist und* $f''(1) = 1$ *gilt, folgt daraus, dass* $f(x)$ *an der Stelle* $x = 1$ *sein absolutes Minimum mit dem Wert* $f(1) = 0$ *hat. Folglich gilt* $\forall x > 0$*:*

$$f(x) = x - 1 - \ln(x) \geqslant 0 \quad \Leftrightarrow \quad x - 1 \geqslant \ln(x)$$

□

Beweis 2 *Beweis von 2.1.2. Die Kullback-Leibler-Divergenz zwischen den Wahrscheinlichkeitsverteilungen* $p(x)$ *und* $q(x)$ *ist wie in 2.1.1 beschrieben durch* $D_{KL}(\, p(x) \parallel q(x)\,) = \int_{-\infty}^{\infty} p(x) \ln(\frac{p(x)}{q(x)})dx$ *gegeben. Aufgrund des Logarithmus lässt sich die Umformung*

$$\int_{-\infty}^{\infty} p(x) \ln(\frac{p(x)}{q(x)})dx = -\int_{-\infty}^{\infty} p(x) \ln(\frac{q(x)}{p(x)})dx$$

vornehmen. Durch 2.1.3 und $\forall x.p(x) \geqslant 0$ *gilt folgende Ungleichung.*

$$\begin{aligned}
& -\int_{-\infty}^{\infty} p(x)(\frac{q(x)}{p(x)} - 1)dx && \leqslant -\int_{-\infty}^{\infty} p(x) \ln(\frac{q(x)}{p(x)})dx \\
\Leftrightarrow\ & -\int_{-\infty}^{\infty} q(x) - p(x)dx && \leqslant -\int_{-\infty}^{\infty} p(x) \ln(\frac{q(x)}{p(x)})dx \\
\Leftrightarrow\ & -(\underbrace{\int_{-\infty}^{\infty} q(x)dx}_{=1} - \underbrace{\int_{-\infty}^{\infty} p(x)dx}_{=1}) && \leqslant -\int_{-\infty}^{\infty} p(x) \ln(\frac{q(x)}{p(x)})dx \\
\Leftrightarrow\ & 0 && \leqslant -\int_{-\infty}^{\infty} p(x) \ln(\frac{q(x)}{p(x)})dx \\
\Leftrightarrow\ & 0 && \leqslant \int_{-\infty}^{\infty} p(x) \ln(\frac{p(x)}{q(x)})dx \\
\Leftrightarrow\ & 0 && \leqslant D_{KL}(\, p(x) \parallel q(x)\,)
\end{aligned}$$

□

Trotz dessen, dass D_{KL} die Ähnlichkeit zweier Wahrscheinlichkeitsdichtefunktionen angibt, ist sie keine Distanzfunktion im streng mathematischen Sinne. Sie erfüllt die Anforderung der Dreiecksungleichung $d(a,b) + d(b,c) \geqslant d(a,c)$ nicht. Des Weiteren ist D_{KL} nicht Symmetrisch, das heißt $D_{KL}(\, p(x) \parallel q(x)\,) \neq D_{KL}(\, q(x) \parallel p(x)\,)$. Trotz dessen ist die Interpretation erlaubt, dass $q(x)$ ähnlicher zu $p(x)$ ist als $q'(x)$ zu $p(x)$, wenn $D_{KL}(\, p(x) \parallel q(x)\,) < D_{KL}(\, p(x) \parallel q'(x)\,)$ gilt. Eine intuitive Begründung hierfür

ist der Aufbau der Kullback-Leibler-Divergenz. Der Quotient $\frac{q(x)}{p(x)}$ ist für $p(x) \approx q(x)$ nahe Eins und somit ist $\ln(\frac{q(x)}{p(x)})$ nahe Null. Das heißt, je ähnlicher die Werte sind, desto kleiner ist der Logarithmusterm. Im Umkehrschluss gilt natürlich, dass der Term größer wird je unähnlicher beide sind. Somit bleibt der Term $\int_{-\infty}^{\infty} p(x) \ln(\frac{p(x)}{q(x)}) dx$ klein, wenn $p(x) \approx q(x)$ für möglichst viele relevante[1] x gilt. Diese Eigenschaft ist wichtig um ein möglichst gutes Modell auswählen zu können. Sei $f(x)$ die reale den Daten zu Grunde liegende Verteilungsdichte und $g(x|\Theta)$ ein Modell, parametrisiert durch Θ, welches $f(x)$ approximieren soll. Wie oben beschrieben lässt sich $f(x)$ nicht berechnen. Trotzdem kann die KL-Divergenz $D_{KL}(\ f(x) \parallel g(x|\Theta)\ )$ genutzt werden, um $g(x|\Theta)$ zu optimieren, indem sie minimiert wird. Dazu kann D_{KL} wie folgt umgeformt werden.

$$\begin{aligned} D_{KL}(\ f(x) \parallel g(x|\Theta)\ ) &= \int_{-\infty}^{\infty} f(x) \ln(f(x)) dx \ - \int_{-\infty}^{\infty} f(x) \ln(g(x|\Theta)) dx \\ &= \underbrace{\mathbb{E}_{f(x)}[\ln(f(x))]}_{\text{Konst. bzgl. } \Theta} \ - \mathbb{E}_{f(x)}[g(x|\Theta)] \\ &= \qquad C \qquad\qquad - \mathbb{E}_{f(x)}[g(x|\Theta)] \end{aligned}$$

Da C unabhängig von Θ ist, ist er für den Vergleich zweier Modelle $g(x|\Theta_1),\ g(x|\Theta_2)$ nicht von Relevanz. Dazu reicht Beispielsweise eine Schätzung von $-\mathbb{E}_{f(x)}[g(x|\Theta)]$ auf Grundlage von empirischen Daten $D \sim f(x)$ [Aka73].
Für diskrete Wahrscheinlichkeitsverteilungen $P : X \to [0, 1]$ und $Q : X \to [0, 1]$ mit $X \subset \mathbb{N}$ wird die KL-Divergenz wie folgt berechnet.

$$D_{KL}(\ P(x) \parallel Q(x)\ ) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)}) \tag{2.1.4}$$

Im diskreten Fall sind die obigen Eigenschaften, analog zum stetigen Fall, ebenfalls gültig.

1 Mit Relevanz ist die Gewichtung durch die Wahrscheinlichkeitsdichte $p(x)$ gemeint. Da x mit einer höheren Wahrscheinlichkeitsdichte mehr ins Gewicht fallen.

2.2 UNSICHERHEITEN

Einen wichtigen Vorteil der bayes'schen neuronalen Netze stellen die zugänglichen Modellunsicherheiten dar, welche in den Experimenten dieser Arbeit genutzt werden. Daher soll in diesem Kapitel dieses Thema durchleuchtet und ein grundlegendes Verständnis von Unsicherheiten in statistischen Modellen geschaffen werden. Unsicherheit ist eine Bezeichnung für das Maß an Ungewissheit einer gemessenen oder durch ein Modell bestimmten Größe. Informationen über Unsicherheiten im Modell können sehr wertvoll sein. Dies wird vor allem dann ersichtlich, wenn Modelle in sicherheitskritischen Systemen, wie autonome Fahrzeuge oder medizinische Geräte, betrachtet werden, da Fehlentscheidungen der Modelle unmittelbar zu Personenschaden führen kann. Diese Gefahr ist nicht nur theoretischer Natur, wie anhand eines Unfalls im Jahre 2016 klar wird. Bei diesem fuhr ein Autonomer Personenkraftwagen unter ein Lastkraftwagen, welches zur Kollision führte, bei dem der Fahrer des PKW ums Leben kam. Der Hersteller begründete das Verhalten des Fahrzeugs mit einer extrem seltenen Verkehrssituation, bestehend aus einer ungewöhnlichen Höhe des LKW in Kombination mit einer ungewöhnlichen Verkehrskonstellation [Unf]. Daher ist es sinnvoll die Unsicherheit bezüglich der Vorhersage solcher Systeme zu quantifizieren, um Entscheidungen auf Basis dieser sicherer gestalten zu können.

2.2.1 *Quellen von Unsicherheit*

Es gibt verschiedene Quellen von Unsicherheit, welche in ein Modell einfließen können. Eine Quelle sind die gesammelten Daten, die zum Training und auch zur Vorhersage genutzt werden, welche selbst unsicherheitbehaftet sind. Zum einen könnte das Gemessene inhärent stochastisch sein und zum anderen könnte die Messung selbst ungenau sein. Eine Begründung weshalb die Genauigkeit von Messungen nicht nur technisch, also aufgrund der Qualität des Sensors, begrenzt ist, bietet die *Heisenbergsche Unschärferelation*. Sie besagt beispielsweise, dass je genauer der Ort eines Elektrons gemessen wird, desto ungenauer sein Impuls bestimmt werden kann und umgekehrt [Hei27]. Eine weitere mögliche Quelle der Unsicherheit in den Daten ist die Digitalisierung von Messungen. Zur Digitalisierung müssen Analoge, kontinuierliche Signale mittels diskreter endlicher Werte abgebildet werden. Dazu wird das Signal an äquidistanten Stellen abgetastet und die gemessenen Werte quantisiert. Somit geht Information verloren und das aufgezeichnete Messergebnis ist Fehlerbehaftet. Selbst ein gänzlich deterministisches System kann die Quelle von Unsicherheit sein, wenn dessen Beobachtbarkeit beschränkt ist. Ein Beispiel hierfür ist eine Studie, bei dem

Hirnforscher die Funktionen eines Mikroprozessors, bestehend aus 3510 Transistoren, mit Hilfe der Methoden der Hirnforschung ergründen sollten.[JK16]. Die Einblicke und Eingriffe begrenzten sich daher auf die Aktivitäten von Transistoren beziehungsweise der Verschaltung dieser. Diese Beschränkung der Beobachtbarkeit führte zu großen Unsicherheiten in den Modellen, wodurch die Aufgabe, den Prozessor zu verstehen scheiterte. Die letzte Quelle ist das Modell selbst: zum einen könnte die Auswahl des Modells Unsicherheit in die Vorhersagen bringen und zum anderen könnten die Modellparameter unsicherheitbehaftet sein.

2.2.2 *Epistemische und Aleatorische Unsicherheit*

Die oben genannten Quellen für Unsicherheit können in zwei Kategorien aufgeteilt werden: **epistemische** und **aleatorische** Unsicherheit[KG17]. Ersteres bezeichnet die Unsicherheit mit dem Ursprung im Modell und der Modellparameter, welche theoretisch minimiert werden könnte, wenn mehr Information gegeben wäre. Letzteres ist die Unsicherheit, welche den Daten zu Grunde liegt. Insgesamt ist die Unsicherheit des Modells die Summe beider Unsicherheiten:

Definition 2 *Sei ϵ die Unsicherheit eines Modells. Dann ist sie die Summe der aleatorischen Unsicherheit ϵ_a und der epistemischen Unsicherheit ϵ_e*

$$\epsilon = \epsilon_a + \epsilon_e \tag{2.2.1}$$

2.2.3 *Maß für Unsicherheit*

Sei $p(y|\underline{x})$ ein statistisches Modell, welches für das Muster $\underline{x}$ die Wahrscheinlichkeit für die Zugehörigkeit zur Klasse y angibt. Dann ist dieses Modell maximal unsicher in seiner Vorhersage, wenn die Verteilung $p(y|\underline{x})$ der Gleichverteilung entspricht und somit $p(y|\underline{x}) = \frac{1}{K}$ gilt, wobei K die Anzahl möglicher Klassen ist. Minimale Unsicherheit und somit maximale Sicherheit wäre gegeben, wenn nur für eine Klasse k, $p(y = k|\underline{x}) = 1$ und alle anderen Klassen $j \neq k$, $p(y = j|\underline{x}) = 0$ gilt. Daraus kann die Intuition abgeleitet werden, dass das Modell unsicherer ist, je mehr Ähnlichkeit der Vorhersage zur Gleichverteilung besteht.
Ein Maß für Unsicherheit, welche ihren Ursprung in der Informationstheorie hat, ist die Entropie nach *Claude Elwood Shannon* [Sha48]:

$$H(P(x)) = -\sum_{x \in X} P(x) \cdot \log(P(x)) \tag{2.2.2}$$

Die Entropie H erreicht ihr Maximum, wenn die Wahrscheinlichkeitsverteilung $p(\cdot)$ der Gleichverteilung entspricht. Eine mögliche Interpretation dieser Gleichung lässt sich in der KL-Divergenz finden. Sei $U(x) : \{1, \dots, K\} \to \mathbb{R}$ die diskrete Gleichverteilung mit K Ausprägungen der Zufallsvariable. Dann ist die KL-Divergenz zwischen einer Wahrscheinlichkeitsverteilung $P : \{1, \dots, K\} \to \mathbb{R}$ und $U(x)$ folgende.

$$
\begin{aligned}
D_{KL}(\, P(x) \parallel U(x) \,) &= \sum_{x \in \{1,\dots,K\}} P(x) \log\left(\frac{P(x)}{U(x)}\right) \\
&= \sum_{x \in \{1,\dots,K\}} P(x) \log(K \cdot P(x)) \\
&= \sum_{x \in \{1,\dots,K\}} P(x)(\log(P(x)) + \log(K)) \\
&= \left(\sum_{x \in \{1,\dots,K\}} P(x) \log(P(x)) \right) + \log(K) \cdot \underbrace{\sum_{x \in \{1,\dots,K\}} P(x)}_{=1} \\
&= \underbrace{\left(\sum_{x \in \{1,\dots,K\}} P(x) \log(P(x)) \right)}_{\text{negative Entropie}} + \underbrace{\log(K)}_{\text{Konstante}}
\end{aligned}
$$

So ist zu sehen, dass die KL-Divergenz zwischen einer Wahrscheinlichkeitsverteilung $P : \{1, \dots, K\} \to \mathbb{R}$ und $U(x)$ der negativen Entropie von $P(X)$ plus einer Konstanten entspricht. Somit könnte die Entropie als ein Maß angesehen werden, welche angibt, wie ähnlich die Gleichverteilung zur Verteilung P ist und daher, wie oben erwähnt, die Unsicherheit bemisst.

2.3 MUSTERERKENNUNG

Kerngegenstand der Mustererkennung ist die Nachbildung der Wahrnehmungsleistung des Menschen. Dabei werden im weitesten Sinne die mathematisch-technischen Aspekte der Perzeption durch Maschinen umfasst [Nie13]. Ziel ist es das Ergebnis menschlicher Perzeption nachzubilden ohne dabei die Prozesse zu kopieren. Im folgenden werden die formalen Grundlagen der Mustererkennung gegeben und erläutert.

Definition 3 *Die Umwelt* U *wird durch die Menge aller messbaren physikalischen Größen* ${}^{\rho}\underline{b}(\underline{x}) : X(\rho) \to B(\rho)$ *dargestellt.*

$$U = \{{}^{\rho}\underline{b}(\underline{x}) : X(\rho) \to B(\rho) \mid \rho = 1, 2, \dots\} \tag{2.3.1}$$

Die physikalischen Größen werden durch mehrdimensionale Funktionen dargestellt, welche für jeden Punkt im Raum und/oder Zeit, eine charakteristische Größe angeben. Dabei kann für jedes ρ sowohl die Definitions- als auch die Zielmenge unterschiedliche Dimensionen und Grundmengen haben, je nach dem, welche Größe dargestellt werden soll. Somit kann jedes Objekt und auch jedes Ereignis durch eine Teilmenge von U dargestellt werden. Aufgrund der Allgemeinheit und der damit verbundenen Mächtigkeit von U beschränken sich Mustererkennungssysteme auf eine Teilmenge von U, welche Problemkreis genannt wird.

Definition 4 *Ein Problemkreis* Ω *ist eine echte Teilmenge der Umwelt* U*. Er beinhaltet Objekte eines begrenzten Anwendungsgebiets, welche durch geeignete Sensoren erfasst wurden. Es gilt:*

$$\Omega = \{{}^{\rho}\underline{f}(\underline{x}) : \; X \to F \mid \rho = 1, 2, \dots\} \subset U \tag{2.3.2}$$

Zu bemerken ist, dass die Definitionsmenge X und die Zielmenge F der Elemente eines Problemkreises konstant sind.

Definition 5 *Die Elemente* $\underline{f}(\underline{x})$ *eines Problemkreises* Ω *werden Muster genannt. Es gilt:*

$$\underline{f}(\underline{x}) = \begin{cases} f_1(x_1, \dots, x_n) \\ \dots \\ f_m(x_1, \dots, x_n) \end{cases} \tag{2.3.3}$$

Mit dieser flexiblen Definition lassen sich unterschiedlichste Objekte in einfacher Weise darstellen. Bespielsweise könnten 32×32 Pixel große RGB-Bilder durch die Menge der Funktionen $\underline{f}(\underline{p}) : \{0,1,\ldots,31\}^2 \to \{0,1,\ldots,255\}^3$ dargestellt werden. Hierbei würde $\underline{p}$ die Position eines Pixels angeben und $\underline{f}(\underline{p})$ die drei RGB-Werte an dieser Position:

$$\underline{f}(\underline{p}) = \begin{cases} f_r(x,y) \\ f_g(x,y) \\ f_b(x,y) \end{cases}$$

Die Muster eines Problemkreises können in verschiedene Klassen aufgeteilt werden.

Definition 6 *Die Untermengen Ω_κ des Problemkreises Ω , welche sich durch Partitionierung ergeben, werden Musterklassen genannt. Es gilt also:*

$$\Omega_\kappa \neq \emptyset \quad \forall \kappa \tag{2.3.4}$$

$$\Omega_\kappa \cap \Omega_\lambda = \emptyset \quad \forall \lambda \neq \kappa \tag{2.3.5}$$

$$\bigcup_\kappa \Omega_\kappa = \Omega \tag{2.3.6}$$

Die Gesamtheit der Partitionen $\{\Omega_\kappa\}_{\kappa=1,2,\ldots,K}$ wird Klassensystem genannt.

Mithilfe dieser Musterklassen kann das *Klassifikationsproblem* formuliert werden.

Definition 7 *Klassifikationsproblem: Finde eine Funktion $g : \Omega \to \{1,2,\ldots,K\}$, gegeben eines Problemkreises Ω mit dem Klassensystem $\Omega_1, \Omega_2, \ldots, \Omega_K \subset \Omega$ bestehend aus K Klassen, sodass*

$$\forall \underline{f} \in \Omega.\ g(\underline{f}) = \kappa \Leftrightarrow \underline{f} \in \Omega_\kappa \tag{2.3.7}$$

Die Funktion g wird *Klassifikator* genannt und weist jedem Muster eine Klasse der K Klassen zu. Er erfüllt in der Praxis die Definition 2.3.7 nicht für alle $\underline{f} \in \Omega$. In den folgenden Kapiteln werden Kriterien vorgestellt, welche die Güte eines Klassifikators messen. Um einen geeigneten Klassifikator zu finden, ist eine repräsentative Stichprobe $\omega' \subset \Omega$ von Nöten. Die Repräsentativität der Stichprobe ω' ist maßgeblich für die Generalisierung von Beobachtungen, da anhand der Beobachtungen in der Stichprobe Erkenntnisse über Elemente von Ω gewonnen werden sollen. Dazu wird meist neben den Mustern ${}^\rho\underline{f}(\underline{x})$ zusätzliche Informationen y_ρ gegeben, sodass die annotierte Stichprobe ω der Größe N wie folgt definiert wird:

$$\omega = \{({}^\rho\underline{f}(\underline{x}),\ y_\rho) \mid {}^\rho\underline{f}(\underline{x}) \in \omega'\} \text{ mit } N = |\omega|$$

Die zusätzlichen Informationen können verschiedener Art sein. Diese Arbeit beschränkt sich auf das binäre und multinomiale Klassifikationsproblem, bei der y_ρ lediglich angibt, zu welcher Klasse das entsprechende Muster zugehörig ist. Beim binären Klassifikationsproblem besteht das Klassensystem aus zwei Klassen, welche häufig *negative* und *positive* Klasse genannt werden und daher $y_\rho \in \{-1, 1\}$ gewählt wird. Bei der multinomialen Klassifikation gilt $K > 2$, sodass $y_\rho \in \{1, 2, \ldots, K\}$ gewählt wird. In der Regel wird ein parametrisiertes Modell $g(\underline{f}|\ \Theta)$ mit Parametervektor $\Theta = (\theta_1, \ldots, \theta_n)$ genutzt um einen Klassifikator zu finden, welcher bestimmte Güte-Kriterien erfüllt.

2.4 PARAMETERSCHÄTZUNG UND OPTIMIERUNG

In diesem Kapitel werden die wichtigsten Prizipien zur Parameterschätzung und Optimierungsmethoden vorgestellt.

2.4.1 *Das Maximum-Likelihood-Prinzip*

Ein solides Prinzip, um die optimalen Parameter Θ^* eines Modells, auf Grundlage einer repräsentativen Stichprobe $\omega = \{{}^\rho\underline{c}|\ \rho = 1, \ldots, N\}$ des betrachteten Problemkreises Ω zu bestimmen, ist das Maximum-Likelihood-Prinzip [Nie13, GBC16]. Eine Grundvoraussetzung für dieses Prinzip ist die statistische Unabhängigkeit der Elemente von Ω. Prinzipiell werden hier jene Parameter Θ des Modells gesucht, welche die Wahrscheinlichkeit für die Beobachtungen der Stichprobe maximieren. Das heißt, es wird Θ gesucht, sodass die Wahrscheinlichkeit $p(\omega|\ \Theta)$, welche *Likelihoodfunktion* genannt wird, maximal ist. Aufgrund der statistischen Unabhängigkeit der Elemente kann diese Wahrscheinlichkeit wie folgt dargestellt werden.

$$p(\omega|\ \Theta) = \prod_{\rho=1}^{N} p({}^\rho\underline{c}|\ \Theta)$$

Letztlich ergibt sich dadurch folgendes Optimierungsproblem.

Definition 8 *Maximum-Likelihood-Schätzung.*

$$\Theta^*_{ML} = \underset{\Theta}{argmax} \prod_{\rho=1}^{N} p({}^\rho\underline{c}|\ \Theta) = \underset{\Theta}{argmax} \sum_{\rho=1}^{N} \ln(p({}^\rho\underline{c}|\ \Theta)) \tag{2.4.1}$$

In der Praxis wird der natürliche Logarithmus auf die Likelihoodfunktion angewandt, welche *Loglikelihoodfunktion* genannt wird, um aus dem Produkt eine Summe zu

machen, da diese einfacher zu handhaben ist. Auf Grund der Monotonie-Eigenschaft des Logarithmus wird das Ergebnis der Schätzung nicht beeinflusst.
Um das Prinzip besser interpretieren zu können, wird die Likelihoodfunktion durch N, der Stichprobengröße, geteilt, sodass wir folgendes erhalten.

$$\Theta^*_{ML} = \underset{\Theta}{\operatorname{argmax}} \frac{\sum_{\rho=1}^{N} \ln(p({}^{\rho}\underline{c}|\,\Theta))}{N}$$

Mit dem *Gesetz der großen Zahlen* lässt sich der Ausdruck zum Erwartungswert hinsichtlich der, der Stichprobe zu Grunde liegenden Verteilung $p(\underline{c})$, umformen[2]:

$$\Theta^*_{ML} = \underset{\Theta}{\operatorname{argmax}}\ \mathbb{E}_{p(\underline{c})}[\ln(p(\underline{c}|\,\Theta))] \tag{2.4.2}$$

Dieser Ausdruck entspricht einem Teil folgender Kullback-Leibler-Divergenz:

$$D_{KL}[p(\underline{c})\ \|\ p(\underline{c}|\Theta)] = \underbrace{\mathbb{E}_{p(\underline{c})}[\ln p(\underline{c})]}_{\text{Unabhängig von }\Theta} - \underbrace{\mathbb{E}_{p(\underline{c})}[\ln p(\underline{c}|\Theta)]}_{\text{Abhängig von }\Theta} \tag{2.4.3}$$

Da $\mathbb{E}_{p(\underline{c})}[\ln p(\underline{c})]$ unabhängig von Θ ist, wird durch Maximierung von 2.4.2, die Kullback-Leibler-Divergenz in 2.4.3 minimiert. Da die Kullback-Leibler-Divergenz ein Maß für die Ähnlichkeit zweier Verteilungen darstellt, lässt dies die Interpretation zu, dass durch das Maximum-Likelihood-Prinzip, das geschätzte Modell $p(\underline{c}|\Theta)$ an die unbekannte, der Stichprobe zu Grunde liegende, Verteilung $p(\underline{c})$ genähert wird[GBC16, s.131ff].

2.4.2 *Das Maximum-a-posteriori-Prinzip*

Beim *Maximum-a-posteriori-Prinzip* wird die a-posteriori Wahrscheinlichkeit der Parameter, gegeben der Stichprobe, $p(\Theta|\,\omega)$ maximiert [GBC16, Nie13].

Definition 9 *Maximum-a-posteriori-Schätzung:*

$$\begin{aligned}
\Theta^*_{MAP} &= \underset{\Theta}{argmax}\ p(\Theta|\,\omega) \\
&= \underset{\Theta}{argmax}\ \frac{p(\omega|\,\Theta)p(\Theta)}{p(\omega)} \\
&= \underset{\Theta}{argmax}\ p(\omega|\,\Theta)p(\Theta) \\
&= \underset{\Theta}{argmax}\ \ln(p(\omega|\,\Theta)) + \ln(p(\Theta))
\end{aligned} \tag{2.4.4}$$

2 Hierbei sei zu beachten, dass N hinreichend groß sein muss

Die Schätzung nach dem Maximum-a-posteriori-Prinzip besteht aus der Summe der Loglikelihoodfunktion $\ln(p(\omega|\,\Theta))$ und der *Prior-Verteilung* $\ln(p(\Theta))$. Letztere ermöglicht es Vorwissen über die Parameter Θ in die Schätzung miteinzubeziehen. Dieser Prior entspricht einer Regularisierung der Modellparameter. Wird beispielsweise ein lineares Regressionsmodell mit dem Prior $p(\Theta) = \mathcal{N}(\Theta \mid \underline{0}, \frac{1}{\lambda}\mathbf{I}^2)$ verwendet, ist der Term $\ln(p(\Theta))$ proportional zu $\lambda\Theta^T\Theta$, welches dem *Weight Decay*[KH91] Regularisierungsterm entspricht [GBC16, S.138ff].

2.4.3 *Optimierung durch Gradientenabstieg*

Die Maximum-Likelihood- und Maximum-a-posteriori-Schätzung geben das Grundgerüst zur Findung der Modellparameter, ohne dabei eine Aussage darüber zu treffen, wie die Parameter zu finden sind. Zum berechnen des argmax stehen, je nach Anwendungsgebiet, verschiedenste Methoden zur Verfügung. Im Kontext neuronaler Netze, welche Gegenstand diese Arbeit sind, ist der *Gradientenabstieg* die Methode zur Optimierung der Modellparameter.
Der Gradientenabstieg ist eine lokale Optimierungsmethode ohne Nebenbedingungen[Nie13, S.35ff]. Sie ist für Funktionen mit stetigen ersten und zweiten Ableitungen geeignet. Im Folgenden wird das Problem der Findung eines lokalen Minimums untersucht.

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\ g(\Theta) \tag{2.4.5}$$

Das Maximierungsproblem kann hierbei aus dem Minimierungsproblem abgeleitet werden, da die folgende Beziehung gilt.

$$\underset{\Theta}{\operatorname{argmax}}\ g(\Theta) = \underset{\Theta}{\operatorname{argmin}}\ -g(\Theta)$$

Folgende Bedingungen sind hinreichend für ein lokales Minimum von g [Nie13][Fleoo, S.12ff].

$$\nabla_\Theta g(\Theta^*) = \begin{pmatrix} \frac{\partial g}{\partial \theta_1} \\ \vdots \\ \frac{\partial g}{\partial \theta_n} \end{pmatrix} = 0, \tag{2.4.6}$$

$$\underline{y}^T\ \nabla^2_\Theta\ g(\Theta^*)\underline{y}\ =\ \underline{y}^T \begin{pmatrix} \frac{\partial^2 g}{\partial \theta_1 \partial \theta_1} & \cdots & \frac{\partial^2 g}{\partial \theta_1 \partial \theta_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 g}{\partial \theta_n \partial \theta_1} & \cdots & \frac{\partial^2 g}{\partial \theta_n \partial \theta_n} \end{pmatrix} \underline{y} > 0, \quad \forall \underline{y} \neq \underline{0} \tag{2.4.7}$$

Sind diese Bedingungen erfüllt, kann wie folgt Θ^* iterativ berechnet werden.

Definition 10 *Gradientenabstieg*

$$\Theta^{(i+1)} = \Theta^{(i)} - \beta \nabla_\Theta g(\Theta^{(i)}) \text{ beziehungsweise } \theta_j^{(i+1)} = \theta_j^{(i)} - \beta \frac{\partial g(\Theta^{(i)})}{\partial \theta_j} \quad (2.4.8)$$

Die Intuition hinter dem Verfahren ist, dass die Suche, ausgehend von einem initialen Wert von $\Theta^{(0)}$, in die steilste abfallende Richtung, gegeben durch $\nabla_{\Theta^{(i)}} g(\Theta)$, fortgeführt wird. Dabei gibt der Skalar β an, wie weit der Schritt in diese Richtung gehen soll. Wie oben beschrieben wird per Gradientenabstieg ein lokales Minimum gefunden. Dies wird im Beispiel der zwei dimensionalen Funktion in Abbildung 2.4.1 deutlich: Je nach dem welcher Startpunkt für $\Theta^{(0)}$ gewählt wird, konvergieren die Parameter gegen ein anderes Minimum. Wird s_1 oder s_2 gewählt, so würden die Parameter gegen m_1 konvergieren. Wird $\Theta^{(0)} = s_3$ gewählt, so konvergieren die Parameter gegen m_3. Dies zeigt auch, dass die Wahl der initialen Werte das Ergebnis der Optimierung stark beeinflussen kann.

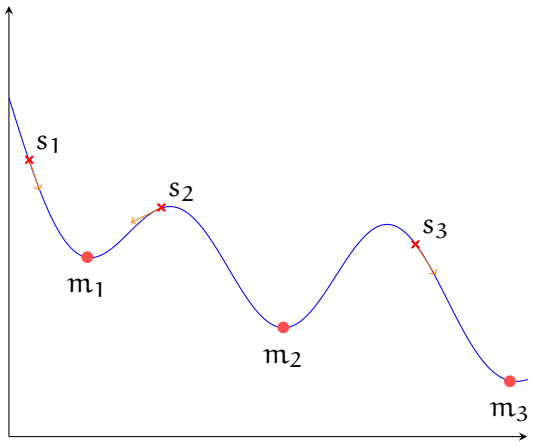


Abbildung 2.4.1: Zwei dimensionale Funktion.

2.5 NEURONALE NETZE

Künstliche neuronale Netze stellen ein weit verbreitetes und sehr erfolgreiches Konzept der Mustererkennung dar und haben ihre Inspiration in den natürlichen neuronalen Netzen des Gehirns.[Ros57] In der Theorie kann eine beliebige stetige Funktion durch ein ein-schichtiges neuronales Netz approximiert werden [HSW89]. Im folgenden werden *künstliche neuronale Netze*, ohne das Adjektiv *künstlich*, *neuronale Netze* genannt.

2.5.1 *Einfaches Neuron*

Die Grundeinheiten neuronaler Netze werden *Neuronen* genannt. Prinzipiell berechnet das Neuron die gewichtete Summe seiner Eingabe, welche anschließend durch eine nichtlineare *Aktivierungsfunktion* $\sigma(\cdot)$ verarbeitet wird, welches in Abbildung 2.5.1 visualisiert wird. Die Eingabe des Neurons wird durch den Vektor $\underline{x} = (x_1, \ldots, x_d)^T$ dargestellt. Die zu erlernenden Parameter Θ_w sind die Gewichte $w_1, \ldots, w_d$. Der Skalar b, welcher es ermöglicht die Eingabe zu verschieben, stellt hierbei den *bias* dar und muss auch erlernt werden. Dieser deckt dabei die Anteile der Berechnung ab, welche unabhängig von den Eingabedaten sind.

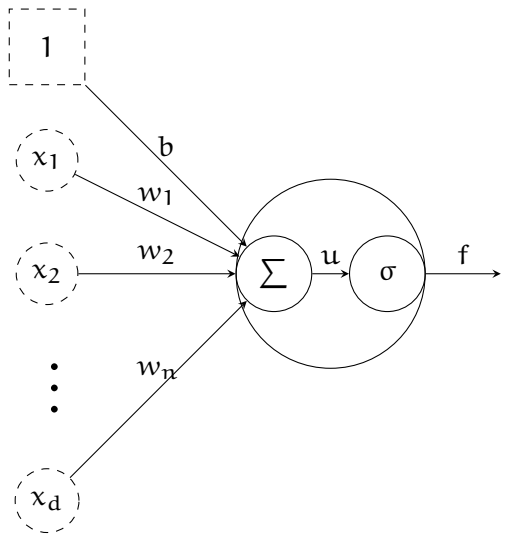


Abbildung 2.5.1: Einfaches Neuron

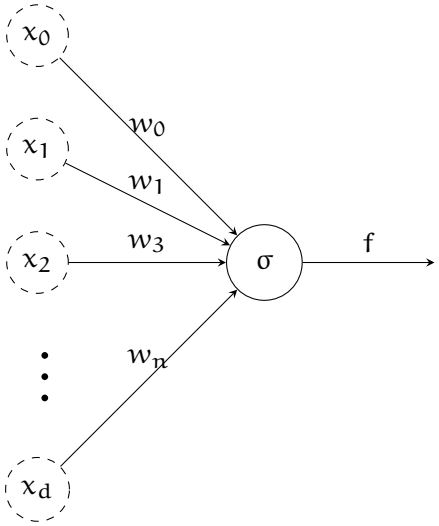


Abbildung 2.5.2: Vereinfachte Darstellung des Neurons

Die Erregung u des Neurons berechnet sich wie folgt.

$$u = b + \sum_{i=1}^{n} w_i x_i = \sum_{i=0}^{n} w_i x_i \tag{2.5.1}$$

Die Ausgabe berechnet sich in dem die Aktivierungsfunktion auf die Erregung angewendet wird.

$$f = \sigma(u) = \sigma(b + \sum_{i=0}^{n} w_i x_i) \tag{2.5.2}$$

Die Aktivierungsfunktion $\sigma(\cdot)$ ist eine nichtlineare, stetige, differenzierbare Funktion. Beispiele für Aktivierungsfunktionen sind die *sigmoid-*, *tanh-* und *ReLU*-Funktion.

$$\sigma_{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.5.3}$$

$$\sigma_{tanh}(x) = \tanh(x) \tag{2.5.4}$$

$$\sigma_{ReLu}(x) = \max(0, x) \tag{2.5.5}$$

$$\sigma_{hlim}(x) = \begin{cases} 1, & \text{falls } x \geqslant 0 \\ 0, & \text{falls } x < 0 \end{cases} \tag{2.5.6}$$

Diese sind in Abbildung 2.5.3 inklusive ihrer ersten Ableitung veranschaulicht. Die Nichtlinearität und Differenzierbarkeit sind notwendig, um das Netz, wie in Kapitel 2.4.3 beschrieben, optimieren zu können. Des Weiteren beeinflussen die Eigenschaften der Aktivierungsfunktionen die Optimierung per Gradientenabstieg, da ihre Ableitungen unmittelbar den Gradienten beeinflussen. Wird beispielsweise $\sigma_{hlim}(\cdot)$ gewählt, ist eine Optimierung per Gradientenabstieg nicht möglich, da $\sigma'_{hlim}(\cdot) = 0$ und folglich $\frac{\partial}{\partial w_i} y = 0$.

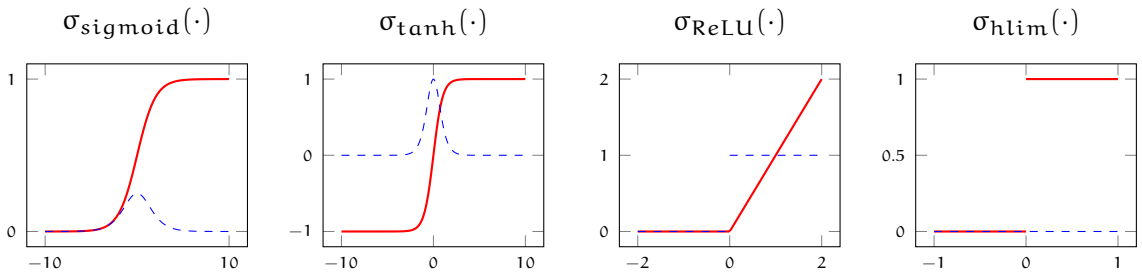


Abbildung 2.5.3: Aktivierungsfunktionen(rot) und ihre Ableitungen(blau, gestrichelt)

Um im Folgenden den Überblick wahren zu können, wird die Darstellung des Neurons wie in Abbildung 2.5.2 vereinfacht, in dem der Bias b als Gewicht w_0 definiert, die Eingabe des Neurons um eine nullte Komponente $x_0 = 1$ erweitert und die Summation nicht mehr graphisch dargestellt wird.

2.5.2 *Mehrschichtiges Vollvernetztes Neuronales Netzwerk*

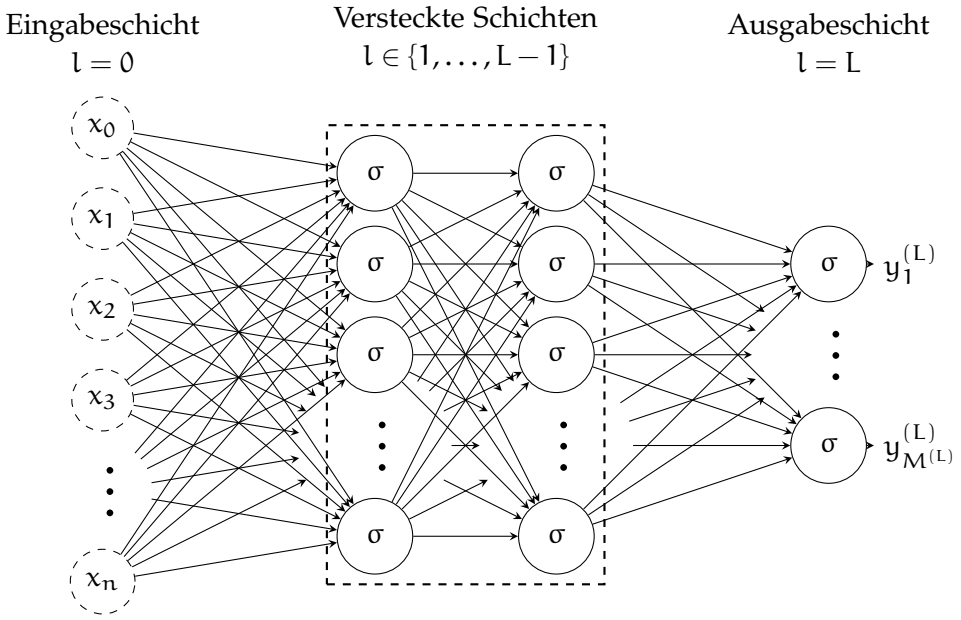


Abbildung 2.5.4: Vollvernetztes Neuronales Netz mit Zwei versteckten Schichten

Ein mehrschichtiges vollvernetztes neuronales Netz ist schichtweise strukturiert und besteht aus L Schichten. Dabei besteht jede Schicht $l \in \{0, \ldots, L\}$ aus $M^{(l)} \in \mathbb{N}$ Neuronen. Die 0-te Schicht stellt hierbei die Eingabeschicht dar, welche die Eingabedaten bereit stellt und wird daher bei der Zählung der Schichten nicht berücksichtigt. Darauf folgen versteckte Schichten, welche die Berechnungen des Modells durchführen, die nach außen hin nicht sichtbar sind. Das endgültige Ergebnis des Modells wird aus der *Ausgabeschicht* ausgelesen. Für alle Schichten gilt, dass nur aufeinander folgende Schichten miteinander verbunden sind, weswegen diese Netztopologie auch als *feedforward neural net* bezeichnet wird [GBC16]. Die Bezeichnung *vollvernetzt* rührt daher, dass jedes Neuron einer Schicht, mit jedem Neuron aus der vorherigen und

folgenden Schicht verbunden ist. Die Erregung des j-ten Neurons im $l+1$-ten Layer $u_j^{(l+1)}$ berechnet sich analog zu 2.5.1 wie folgt.

$$u_j^{(l+1)} = b_j^{(l+1)} + \sum_{i=1}^{M^{(l)}} w_{ij}^{(l+1)} f_i^{(l)} \tag{2.5.7}$$

Dabei werden die Aktivierungen der Neuronen aus der vorherigen Schicht $f_j^{(l)}$ gewichtet aggregiert. Die Parameter $w_{ij}^{(l+1)}$ stellen die Gewichte dar, welche die Aktivierung des i-ten Neurons aus der l-ten Schicht in der Erregung des j-ten Neurons in der $l+1$-ten Schicht gewichten. Zur Vereinfachung der Notation wird der Bias des j-ten Neurons aus der $l+1$-ten Schicht $b_j^{(l+1)}$ als nulltes Gewicht $w_{0j}^{(l+1)}$ und $f_0^{(l)} = 1$ definiert.

$$u_j^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{ij}^{(l+1)} f_i^{(l)} = \underline{w}_j^{(l+1)} \underline{f}^{(l)} \tag{2.5.8}$$

Anschließend ist es möglich die Berechnung mittels Skalarprodukt der Vektoren $\underline{w}_j^{(l+1)} = (w_{0j}^{(l+1)}, w_{1j}^{(l+1)}, \dots, w_{M^{(l)}j}^{(l+1)})^T$ und $\underline{f}^{(l)} = (f_0^{(l)}, f_1^{(l)}, \dots, f_{M^{(l)}}^{(l)})^T$ darzustellen. Abschließend kann die Erregung einer Schicht als Vektor $\underline{u}^{(l+1)} = (u_1^{(l+1)}, u_2^{(l+1)}, ..., u_{M^{(l+1)}}^{(l+1)})$ zusammengefasst werden und kompakt als Matrix-Vektor-Produkt beschrieben werden.

$$\underline{u}^{(l+1)} = \mathbf{W}^{(l+1)} \underline{f}^{(l)}, \qquad \text{mit } \mathbf{W}^{(l+1)} = (w_{ij}^{(l+1)})^T = (\underline{w}_0^{(l+1)}, \underline{w}_1^{(l+1)}, \dots, \underline{w}_{M^{(l+1)}}^{(l+1)})^T \tag{2.5.9}$$

Die Aktivierung $\underline{f}^{(l)}$ der l-ten Schicht, für $l \in \{1, \dots, L\}$ ist dann die komponentenweise Anwendung der Aktivierungsfunktion auf den Erregungsvektor $\underline{u}^{(l)}$.

$$\underline{f}^{(l+1)} = \sigma(\mathbf{W}^{(l+1)} \underline{f}^{(l)}) = (1, \sigma(u_1^{(l+1)}), \sigma(u_2^{(l+1)}), \dots, \sigma(u_{M^{(l+1)}}^{(l+1)}))^T \tag{2.5.10}$$

Die Aktivierung der Eingabeschicht, $l = 0$, ist der Eingabevektor $\underline{x}$ erweitert um die nullte Komponente.

$$\underline{f}^{(0)} = \text{pad}(\underline{x}) = (1, x_1, x_2, \dots, x_{M^{(0)}})^T \tag{2.5.11}$$

So lässt sich die Ausgabe eines L-schichtigen vollvernetzten neuronalen Netzes mit der Eingabe $\underline{x}$ über $\underline{f}^{(L)}$ rekursiv wie folgt berechnen.

$$\begin{aligned} \underline{f}^{(L)} = \sigma(\mathbf{W}^{(L)} \underline{f}^{(L-1)}) &= \sigma(\mathbf{W}^{(L)} \cdot \sigma(\mathbf{W}^{(L-1)} \underline{f}^{(L-2)})) \\ &= \sigma(\mathbf{W}^{(L)} \cdot \sigma(\mathbf{W}^{(L-1)} \cdot \ldots \cdot \sigma(\mathbf{W}^{(1)} \underline{f}^{(0)}) \ldots)) \\ &= \sigma(\mathbf{W}^{(L)} \cdot \sigma(\mathbf{W}^{(L-1)} \cdot \ldots \cdot \sigma(\mathbf{W}^{(1)} \text{pad}(\underline{x})) \ldots)) \end{aligned}$$

Die Berechnung der Ausgabe $\underline{\hat{y}}$ eines vier schichtigen neuronalen Netzes erfolgt beispielsweise folgendermaßen.

$$\underline{\hat{y}} = \sigma(\mathbf{W}^{(4)} \cdot \sigma(\mathbf{W}^{(3)} \cdot \sigma(\mathbf{W}^{(2)} \cdot \sigma(\mathbf{W}^{(1)} \text{pad}(\underline{x})))))$$

Im folgenden wird $\underline{f}^{(l)}(\underline{x})$ die Ausgabe der Schicht l mit Eingabevektor $\underline{x}$ bezeichnen.

2.5.3 *Gradientenabstieg bei neuronalen Netzen*

Die Parameter $\Theta_{\mathbf{W}} = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, ..., \mathbf{W}^{(L)})$ des neuronalen Netzes können mittels Gradientenabstieg, welches in Abschnitt 2.4.3 vorgestellt wurde, erlernt werden. Dazu sei die annotierte Stichprobe $\omega = \{({}^{\rho}\underline{x}, {}^{\rho}\underline{y}) | \rho = 1, \ldots, N\}$ aus dem betrachteten Problemkreis gegeben. Um nun den Gradientenabstieg anwenden zu können, muss zunächst einmal ein Ziel definiert werden. Dieses Ziel ist das Minimieren des Fehlers in der Ausgabe des neuronalen Netzes für unsere Stichprobe. Daher ist ein *Fehlermaß* von Nöten. Diese werden auch *Loss-* beziehungsweise *Verlust*-Funktion genannt. Zu unterscheiden ist hierbei zwischen der Gesamtverlustfunktion über die gegeben Stichprobe $\mathfrak{L}(\omega) : \mathfrak{P}(\Omega) \to \mathbb{R}$ und dem Verlust für ein Stichprobenelement $\mathcal{L}(\underline{\hat{y}}, \underline{y}) : Y^2 \to \mathbb{R}$, wobei $\underline{\hat{y}} = \underline{f}^{(L)}(\underline{x})$ die Vorhersage des neuronalen Netzes und $\underline{y}$ die gegebene Annotation beziehungsweise den korrekten Wert, den es zu vorhersagen galt, darstellt. Der Gesamtverlust kann hierbei beispielsweise durch den mittleren Einzelverlust berechnet werden.

$$\mathfrak{L}(\omega) = \frac{1}{|\omega|} \sum_{\rho=1}^{|\omega|} \mathcal{L}({}^{\rho}\underline{\hat{y}}, {}^{\rho}\underline{y}) \tag{2.5.12}$$

Somit können die Gewichte $w_{ij}^{(l)}$ gemäß folgender Vorschrift iterativ berechnet werden.

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \beta \frac{\partial \mathfrak{L}(\omega)}{\partial w_{ij}^{(l)}} = w_{ij}^{(l)} - \beta \frac{1}{|\omega|} \sum_{\rho=1}^{|\omega|} \frac{\partial \mathcal{L}({}^{\rho}\underline{\hat{y}}, {}^{\rho}\underline{y})}{\partial w_{ij}^{(l)}} \tag{2.5.13}$$

Dazu wird $\frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial w_{ij}^{(l)}}$ mittels *Backpropagation*-Algorithmus[RHW86] berechnet, welcher im folgenden skizziert wird. Da $\mathcal{L}(\underline{\hat{y}}, \underline{y})$ von $\underline{\hat{y}} = \underline{f}^{(L)}$ abhängt, lässt sich durch Ver-

folgung der Abhängigkeitskette $f_j^{(L)} \to f_j^{(l)} \to u_j^{(l)} \to w_{ij}^{(l)}$ durch Anwendung der Kettenregel die partielle Ableitung wie folgt umformen.

$$\frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial w_{ij}^{(l)}} = \underbrace{\frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial f_j^{(l)}} \cdot \frac{\partial f_j^{(l)}}{\partial u_j^{(l)}}}_{\delta_j^{(l)}} \cdot \frac{\partial u_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{2.5.14}$$

Der Faktor $\frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial f_j^{(l)}}$ lässt sich berechnen, indem der Fehler der Aktivierungen $f_j^{(l+1)}$ auf die Fehler der Aktivierungen $\underline{f}^{(l)}$ zurückgeführt wird. Dies wird bei Betrachtung von 2.5.10 ersichtlich, da $f_j^{(l+1)}$ von $\underline{f}^{(l)}$ abhängig ist. Daher ergibt sich mit der verallgemeinerten Kettenregel folgende Beziehung.

$$\begin{aligned} \frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial f_j^{(l)}} &= \sum_{i=0}^{M^{(l+1)}} \frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial f_i^{(l+1)}} \frac{\partial f_i^{(l+1)}}{\partial f_j^{(l)}} = \sum_{i=0}^{M^{(l+1)}} \underbrace{\frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial f_i^{(l+1)}} \cdot \frac{\partial f_i^{(l+1)}}{\partial u_i^{(l+1)}}}_{\delta_i^{(l+1)}} \cdot \frac{\partial u_i^{(l+1)}}{\partial f_j^{(l)}} \\ &= \sum_{i=0}^{M^{(l+1)}} \delta_i^{(l+1)} \cdot \frac{\partial u_i^{(l+1)}}{\partial f_j^{(l)}} = \sum_{i=0}^{M^{(l+1)}} \delta_i^{(l+1)} \cdot w_{ji}^{(l+1)} \end{aligned} \tag{2.5.15}$$

Des weiteren lassen sich die letzten beiden Faktoren von 2.5.14 wie folgt berechnen.

$$\frac{\partial f_j^{(l)}}{\partial u_j^{(l)}} = \frac{\partial}{\partial u_j^{(l)}} \sigma(u_j^{(l)}) = \sigma'(u_j^{(l)}) \tag{2.5.16}$$

und

$$\frac{\partial u_j^{(l)}}{\partial w_{ij}^{(l)}} = \frac{\partial}{\partial w_{ij}^{(l)}} \left(\sum_{k=0}^{M^{(l-1)}} w_{kj}^{(l)} f_k^{(l-1)} \right) = f_i^{(l-1)} \tag{2.5.17}$$

So kann der Fehler des Neurons j im l-ten Layer $\delta_j^{(l)}$ mittels 2.5.16 und 2.5.15 wie folgt berechnet werden.

$$\delta_j^{(l)} = \left(\sum_{i=0}^{M^{(l+1)}} \delta_i^{(l+1)} \cdot w_{ji}^{(l+1)} \right) \cdot \sigma'(u_j^{(l)}) \tag{2.5.18}$$

Insgesamt lässt sich 2.5.14 nun vereinfacht darstellen.

$$\frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \cdot f_i^{(l-1)} \tag{2.5.19}$$

Informell zusammengefasst, wird der Fehler einer Schicht auf den Fehler der folgenden Schicht zurückgeführt, wodurch der Name des Verfahrens *Fehler-Rückführungs-Algorithmus* oder auch, wie oben genannt, *Backpropagation*-Algoritmus ist. Die Berechnung des Fehler beginnt bei der letzten Schicht L, sodass diese initial wie folgt berechnet wird.

$$\delta_j^{(L)} = \frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial f_j^{(L)}} \cdot \frac{\partial f_j^{(L)}}{\partial u_j^{(L)}} = \frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial \hat{y}_j} \cdot \sigma'(u_j^{(L)}) \tag{2.5.20}$$

Konkretisiert ergibt sich aus 2.5.13 nun folgende Regel zur iterativen Optimierung der Gewichte.

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \beta \frac{1}{|\omega|} \sum_{\rho=1}^{|\omega|} {}^{\rho}\delta_j^{(l)} \cdot^{\rho} f_i^{(l-1)} \tag{2.5.21}$$

Dieses Verfahren wird auch *Batch Gradient Descent*[Rud17] genannt. Da für jede Iteration der Parameteroptimierung jedes Stichprobenelement betrachtet wird, fällt dieses Verfahren für große Datensätze langsam aus. Eine alternative stellt das *Stochastic Gradient Descent* [Bot91] Verfahren dar. In diesem werden für jedes Stichprobenelement einzeln die Parameter per Gradientenabstieg optimiert. Die Optimierungsklausel ist also folgende.

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \beta \cdot {}^{\rho}\delta_j^{(l)} \cdot^{\rho} f_i^{(l-1)} \tag{2.5.22}$$

Da hierbei für jedes Element die Parameter aktualisiert werden, ist der Rechenaufwand pro Iteration geringer gegenüber 2.5.21 und das Verfahren konvergiert schneller gegen ein Minimum. Ein Problem jedoch ist die Varianz der berechneten Gradienten und somit in den Iterationen, sodass Minima verfehlt werden können oder keine Konvergenz stattfindet.[Rud17]. Eine Kombination beider Verfahren ist *Mini-Batch Stochastic Gradient Descent*[QK20]. Hierbei wird der Gradient über Teilmengen der Stichprobe gebildet, welche Mini-Batch genannt werden.

2.5.4 *Problem des verschwindenden Gradienten*

Bei vielschichtigen vollvernetzten neuronalen Netzen existiert das Problem des verschwindenen Gradienten, welches dazu führt, dass die Gewichte während der Opti-

mierung, aufgrund eines zu kleinen Gradienten ≈ 0, stationär werden und nicht mehr aktualisiert werden können.

$$\frac{\partial \mathcal{L}(\underline{\hat{y}}, \underline{y})}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \cdot f_i^{(l-1)} = \left(\sum_{i=0}^{M^{(l+1)}} \delta_i^{(l+1)} \cdot w_{ji}^{(l+1)} \right) \cdot \sigma'(u_j^{(l)}) \cdot f_i^{(l-1)} \tag{2.5.23}$$

$$= \left(\sum_{i=0}^{M^{(l+1)}} \left(\sum_{k=0}^{M^{(l+2)}} \delta_k^{(l+2)} \cdot w_{ik}^{(l+2)} \right) \cdot \sigma'(u_i^{(l+1)}) \cdot w_{ji}^{(l+1)} \right) \cdot \sigma'(u_j^{(l)}) \cdot f_i^{(l-1)} \tag{2.5.24}$$

Wird hierzu 2.5.23 und 2.5.24 betrachtet, fällt auf, dass bei der Fehlerrückführung für jede Schicht eine Multiplikation mit der Ableitung der Aktivierungsfunktion $\sigma'(\cdot)$ stattfindet. Für die sigmoid- und tanh-Ableitungsfunktion gilt $\sigma'_{sigmoid}(\cdot) \leqslant 0.25$ und $\sigma'_{tanh}(\cdot) \leqslant 1$, welches in Abbildung 2.5.3 gesehen werden kann. Des weiteren sind die Ableitungen beider Funktionen ≈ 0, wenn sie Gesättigt sind, welches der Fall ist, wenn die Eingabewerte sehr groß oder sehr klein sind. Da die Multiplikation vieler Zahlen, welche kleiner als Eins sind, gegen Null geht, also $a \cdot x^n \xrightarrow{n\to\infty} 0$ für $x < 1$, nähert sich der berechnete Fehler eines Neurons $\delta_j^{(l)}$ auch der Null, je mehr Schichten das Netz hat und je näher am Anfang des Netzes das Neuron plaziert ist. Somit ergibt sich für die Optimierungsregel.

$$w_{ij}^{(l)} - \beta \delta_j^{(l)} f_i^{(l-1)} \approx w_{ij}^{(l)} - \beta \cdot 0 \cdot f_i^{(l-1)} \approx w_{ij}^{(l)} \tag{2.5.25}$$

Eine Lösung dieses Problems ist es die *ReLU*-Aktivierungsfunktion $\sigma_{ReLU}(\cdot)$ zu verwenden, dessen Ableitung 1 für positive Argumente oder 0 für negative Argumente ist. Das Problem der Unstetigkeit der Ableitung an Stelle Null, lässt sich lösen indem $\sigma'_{ReLU}(0) = 0$ oder $\sigma'_{ReLU}(0) = 1$ definiert wird, welches jedoch kaum einen Einfluss auf das Endergebnis hat, da es numerisch sehr unwahrscheinlich ist, dass das Argument exakt 0 ist. Somit bleibt bei positiven Argumenten, der zurückgeführte Fehler bei wiederholter Multiplikation mit $\sigma'_{ReLU}(\cdot)$ unverändert, werden jedoch bei negativem Argument Null. Daher erfüllt die ReLU−Aktivierung eine *Gating* Funktion für den Gradienten. Um einen Gradienten gleich 0 zu vermeiden kann die parametrisierte

Leaky-ReLU-Funktion verwendet werden, welche durch zwei Geraden definiert ist [HZRS15].

$$\sigma_{\mathrm{LeakyReLU}}(x) = \begin{cases} x, & \text{falls } x \geqslant 1 \\ ax, & \text{falls } x < 0 \end{cases} \tag{2.5.26}$$

$$\sigma'_{\mathrm{LeakyReLU}}(x) = \begin{cases} 1, & \text{falls } x \geqslant 1 \\ a, & \text{falls } x < 0 \end{cases} \tag{2.5.27}$$

Für Aktivierungsfunktionen, dessen Ableitung $\sigma'(\cdot) > 1$ ist, wird der zurückgeführte Fehler exponentiell groß, sodass das Training sehr instabil wird, da Veränderungen der Gewichte je Iteration sehr groß werden, sodass kein Optimum mehr gefunden werden kann. Dieses Problem nennt sich *exploding gradient* Problem [BSF94]. Eine Lösung dieser Probleme ist es L klein zu halten, welches jedoch die Modellkomplexität beschränken würde. Das *exploding gradient* Problem lässt sich beispielsweise auch durch das *gradient clipping* [PMB13] lösen, welches prinzipiell den Gradienten herunterskaliert, sobald es einen bestimmten threshold überschreitet.

2.5.5 *Klassifikation mit neuronalen Netzen*

Das binäre Klassifikationsproblem 2.3.7 kann mittels eines neuronales Netz gelöst werden, dessen Ausgabeschicht die Dimension $M^{(L)}=1$ hat und die *sigmoid* Aktivierungsfunktion aus 2.5.3 nutzt. Denn dann gilt $0 \leqslant f^{(L)}(\underline{x}) \leqslant 1$ und es kann über beispielsweise einen Threshold $\tau = 0.5$ entschieden werden, zu welcher Klasse $\underline{x}$ zugeordnet werden soll. In der Praxis kann mittels mehrerer binärer Klassifikatoren eine multinomiale Klassifikation vorgenommen werden. Dazu wird beispielsweise für jede Klasse i ein Klassifikator $k_i(\underline{x})$ trainiert, für den i die positive Klasse darstellt und die restlichen Klassen zur negativen Klasse zusammengefasst werden. Zur Klassifikation wird der Eingabe $\underline{x}$ die Klasse j zugewiesen, dessen Klassifikator $k_j(\underline{x})$ den höchsten Score erzielt hat. Dieses Verfahren wird *One-vs.-Rest* [Bis06] genannt. Dies ist jedoch mit einem Overhead verbunden, da für jede Klasse ein separates Modell trainiert werden muss.
Neuronale Netze bieten die Möglichkeit mehrere Ausgaben gleichzeitig auszugeben. Dies wird genutzt um ein Modell zu trainieren, welches sich für multinomiale Klassifikation eignet. Sei $\omega = \{({}^{\rho}\underline{x}, {}^{\rho}\underline{y}) | \rho = 1, \ldots, N\}$ die annotierte Stichprobe und $\Omega_1, \ldots, \Omega_K \subset \Omega$ die betrachteten Klassen, dann muss $M^{(L)} = K$ für die Ausgabeschicht (L) gelten. Das heißt die Anzahl an Ausgabeneuronen, muss der Anzahl an

Klassen entsprechen. Für die Annotation der Stichprobenelemente wird eine 1-*aus*-k Kodierung [Nie13] genutzt. Bei dieser werden die Klassenzugehörigkeiten durch einen K-dimensionalen Vektor dargestellt, wobei jede Dimension des Vektors dem Indikator einer Klasse entspricht. Die Annotation der Stichprobenelemente ist daher ${}^{\rho}\underline{y} = ({}^{\rho}y_1, \ldots, {}^{\rho}y_K)^T$ mit

$$ {}^{\rho}y_j = \begin{cases} 1, \text{ falls } {}^{\rho}\underline{x} \in \Omega_j \\ 0, \text{ falls } {}^{\rho}\underline{x} \notin \Omega_j \end{cases} \tag{2.5.28} $$

Dadurch soll erzielt werden, dass das Netz lernt, hohe Werte bei der Ausgabe $f_j^{(L)}$ zu berechnen, wenn die Eingabe $\underline{x}$ der Klasse j zugehörig ist und alle anderen Ausgaben $f_i^{(L)}$ niedrig zu halten. Eine Möglichkeit die Ausgabe des Modells der Zielvariable anzugleichen kann beispielsweise mit der *Mean Square Error* (kurz *MSE*) Loss-Funktion erreicht werden.

$$ \mathcal{L}_{MSE}(\underline{\hat{y}}, \underline{y}) = \frac{1}{M^{(L)}}(\underline{y} - \underline{\hat{y}})^T(\underline{y} - \underline{\hat{y}}) = \frac{1}{M^{(L)}} \sum_{i=1}^{M^{(L)}} (y_i - \hat{y}_i)^2 \tag{2.5.29} $$

Wird als Gesamtverlustfunktion 2.5.12 verwendet, ergibt sich unter Nutzung der MSE Loss-Funktion folgender Gesamtverlust.

$$ \mathfrak{L}_{MSE}(\omega) = \frac{1}{|\omega|} \sum_{\rho=1}^{|\omega|} \frac{1}{M^{(L)}} ({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}})^T ({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}}) \tag{2.5.30} $$

$$ = \frac{2}{|\omega| \cdot M^{(L)}} \sum_{\rho=1}^{|\omega|} \frac{1}{2} ({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}})^T ({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}}) \tag{2.5.31} $$

$$ = \frac{2}{|\omega| \cdot M^{(L)}} \sum_{\rho=1}^{|\omega|} \ln(e^{\frac{1}{2}({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}})^T ({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}})}) \tag{2.5.32} $$

$$ = -\frac{2}{|\omega| \cdot M^{(L)}} \sum_{\rho=1}^{|\omega|} \ln(e^{-\frac{1}{2}({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}})^T ({}^{\rho}\underline{y} - {}^{\rho}\underline{\hat{y}})}) \tag{2.5.33} $$

Sei nun $\mathbf{C} \in \mathbb{R}^{K \times K}$ die Einheitsmatrix. Dann lässt sich das Optimierungsproblem mit der MSE-Verlustfunktion, nach auslassen der Konstanten, wie folgt formulieren.

$$\Theta_W^* = \underset{\Theta_W}{\operatorname{argmin}} \left(-\sum_{\rho=1}^{|\omega|} \ln(e^{-\frac{1}{2}({}^{\rho}\underline{y}-{}^{\rho}\underline{\hat{y}})^T({}^{\rho}\underline{y}-{}^{\rho}\underline{\hat{y}})}) \right) \tag{2.5.34}$$

$$= \underset{\Theta_W}{\operatorname{argmin}} \left(-\sum_{\rho=1}^{|\omega|} \ln(e^{-\frac{1}{2}({}^{\rho}\underline{y}-{}^{\rho}\underline{\hat{y}})^T C^{-1}({}^{\rho}\underline{y}-{}^{\rho}\underline{\hat{y}})}) \right) \tag{2.5.35}$$

$$= \underset{\Theta_W}{\operatorname{argmax}} \left(\sum_{\rho=1}^{|\omega|} \ln(\mathcal{N}({}^{\rho}\underline{y}|{}^{\rho}\underline{\hat{y}}, \mathbf{C})) \right) \tag{2.5.36}$$

Nun ist zu sehen, dass die Minimierung der MSE-Verlustfunktion einer Maximum-Likelihood-Schätzung (siehe 2.4.1) der Parameter unter der Annahme einer normalverteilten Zielvariable $\underline{y}$ mit Erwartungsvektor $\underline{\mu} = \underline{\hat{y}}$ und Kovarianzmatrix $\Sigma = \mathbf{C}$ gleich kommt. Da die 1-aus-k kodierte Zielvariable, nicht Normal-, sondern Multinomialverteilt ist, eignet sich die MSE-Verlustfunktion nicht für Klassifikationsprobleme. Damit die Ausgabeschicht, Eigenschaften einer Multinomialverteilung hat, wird dort häufig die Softmax-Funktion als Aktivierungsfunktion verwendet. Diese unterscheidet sich von den restlichen Aktivierungsfunktionen in so fern, als dass diese einen Vektor als Eingabe verwendet.

$$f_j^{(L)} = \sigma_{softmax}(\underline{u}^{(L)})_j = \frac{e^{u_j}}{\sum_{i=1}^{M^{(L)}} e^{u_i}} \tag{2.5.37}$$

Dadurch sind alle Werte der letzten Schicht im Intervall $[0, 1]$ und summieren sich zu Eins auf. Diese Werte werden als Pseudowahrscheinlichkeiten und als ein Maß für die Konfidenz des Modells interpretiert. Dies ist jedoch problematisch, da die Softmax-Funktion mithilfe von Exponentialfunktionen berechnet wird, welche sehr stark auf Differenzen der einzelnen Erregungen reagieren. Eine Loss-Funktion welche oftmals in Kombination mit der Softmax-Aktivierungsfunktion genutzt wird, ist die *Negative-Log-Likelihood*-Funktion (kurz NLL).

$$\mathcal{L}_{NLL}(\underline{\hat{y}}, \underline{y}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \overset{(y_j=1)}{=} -y_j \log(\hat{y}_j) \tag{2.5.38}$$

Sei $r(\rho)$ der Index, der Klasse zu der ${}^{\rho}\underline{x}$ zugehörig ist, also ${}^{\rho}y_{r(\rho)} = 1$ und ${}^{\rho}y_i = 0$ für $i \neq r(\rho)$. Dann ist der Gesamtverlust mittels NLL folgender.

$$\mathfrak{L}_{\mathrm{NLL}}(\omega) = -\frac{1}{|\omega|} \sum_{\rho=1}^{|\omega|} \log({}^{\rho}\hat{y}_{r(\rho)}) \tag{2.5.39}$$

Dieser Ausdruck entspricht, wie der Name schon suggeriert, einer Maximum-Likelihood-Schätzung und dem parameterabhängigen Teil der KL-Divergenz aus 2.4.3 und passt die Ausgabe, ohne eine Normalverteilungsannahme, der Zielwertverteilung an.

2.5.6 *Faltungsschichten*

Einen besonderen Aufbau von neuronalen Netzen ermöglichen *Faltungsschichten*, welche im englischen *Convolutional Layer* genannt werden. Eine Faltung beschreibt eine mathematische Operation, bei der lokale Skalarprodukte beziehungsweise lokale gewichtete Summen gebildet werden. Sei $\underline{f}(x, y) : M_x \times M_y \to \mathbb{R}$ ein Muster, welches ein Graustufenbild kodiert und $g(x, y) : m_x \times m_y \to \mathbb{R}$ eine Gewichtsmaske mit $m_x = \{-m, \ldots, -1, 0, 1, \ldots, m\}$ und $m_y = \{-n, \ldots, -1, 0, 1, \ldots, n\}$. Dann ist die Faltung $h(x, y)$ wie folgt definiert[Nie13].

$$h(x, y) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} f(x - i, y - j) \cdot g(i, j) \tag{2.5.40}$$

Eine intuitive Vorstellung einer Faltung ist, dass die Gewichtsmaske über das Bild geschoben wird und lokal die mit den Gewichten der Gewichtsmaske gewichtete Summe der Bildpunkte berechnet wird, welches in Abbildung 2.5.5 visualisiert ist. Im Falle der Faltungsschichten wird die Gewichtsmaske *Kernel* genannt. Die Erregung $\underline{u}^{(l)}$ der Schicht l berechnet sich durch Faltung der Aktivierung $\underline{f}^{(l-1)}$ der vorherigen Schicht mit dem Kernel. Der Vorteil gegenüber einer Vollvernetzten Schicht ist die verminderte Anzahl an Parametern, da diese für von der Eingabe geteilt werden. Die Kernels beschränken sich nicht auf zwei Dimensionen, sonder können beliebig viele Dimensionen haben. Werden beispielsweise RGB-Bilder als Eingabe verwendet, welche Drei Farbkanäle besitzen, können beispielsweise dreidimensionale Kernels verwendet werden. Pro Schicht werden hierbei mehrere Kernel genutzt, dessen Ausgabe einen *Feature*-Kanal der Aktivierungsschicht darstellt. Sei ein neuronales Netz zur Klassifikation von RGB-Bildern gegeben, dessen erste Schicht 16 Kernels der Dimension 5x5 mit 3 Kanälen besitzt. So bestünde die Aktivierung $\underline{f}^{(1)}$ aus 16 Feature-Kanälen, sodass, falls die zweite Schicht ebenfalls eine Faltungsschichtwäre, dessen Kernels

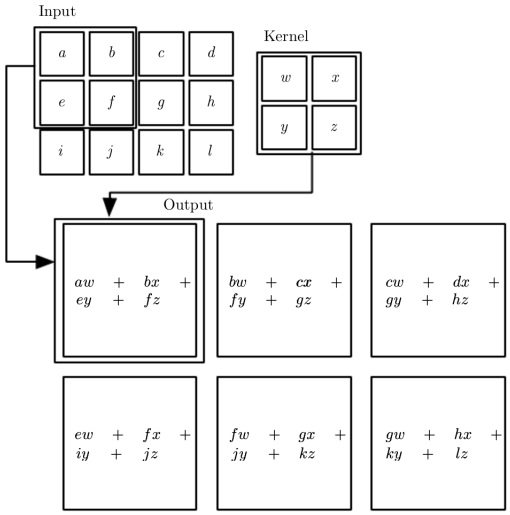


Abbildung 2.5.5: Visualisierung einer Faltung aus [GBC16]

ebenfalls 16 Kanäle hätte. Bei Faltungsschichten wird oftmals ein *Stride*-Parameter s angegeben. Dieser gibt an, um viele Einheiten, die Maske bei der Faltung pro Schritt bewegt werden soll, sodass die Dimension der Ausgabe kleiner wird. Die Berechnung im zweidimensionalen Fall ist folgende.

$$h_s(x,y) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} f(s \cdot x - i, s \cdot y - j) \cdot g(i,j) \tag{2.5.41}$$

Im folgenden wird ein Kernel mit $K(n \times m, c, s)$ bezeichnet wobei $n \times m$ die Grundflächendimension, c die Anzahl der Kanäle der Eingabe und s den Stride darstellt.

2.6 BAYES'SCHE NEURONALE NETZE

Werden klassische neuronale Netze zur Klassifikation, wie beispielsweise in Kapitel 2.5 vorgestellt, als probabilistische Modelle $P(\underline{y}|\ \underline{x}, \Theta_W)$ angesehen, so berechnet das neuronale Netz Wahrscheinlichkeiten für die möglichen Klassenzugehörigkeiten. Die Berechnung der Modellparameter Θ_W nach dem Maximum-Likelihood- oder Maximum-Aposteriori-Prinzip durch Gradientenabstieg ist eine Punktschätzung der Modellparameter. Daher umfasst das Modell keine Unsicherheiten bezüglich der Modellparameter und somit keine epistemische Unsicherheit. Bei *Bayes'schen neuronalen Netzen* wird statt einer Punktschätzung, $P(\Theta_W|\ \omega)$, die a-posteriori Wahrscheinlichkeit der Modellparameter Θ_W gegeben der Stichprobe ω berechnet. In Abbildung 2.6.1

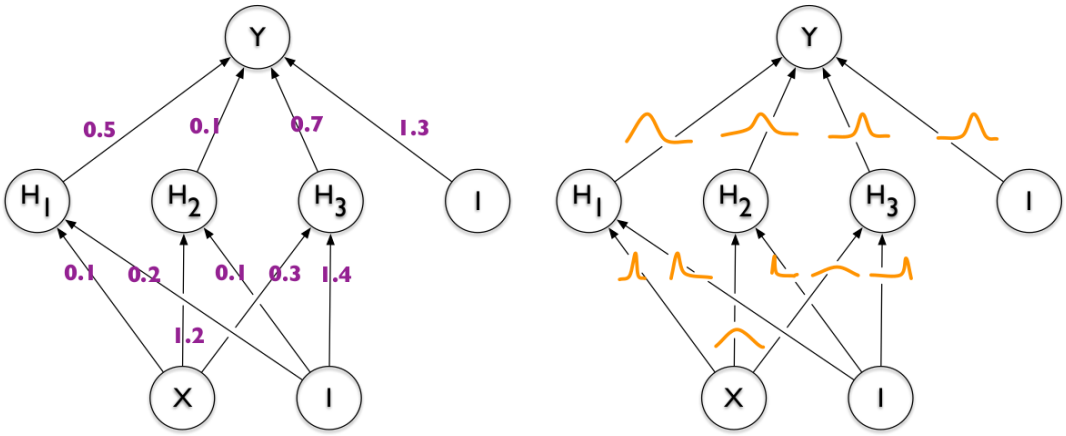


Abbildung 2.6.1: Links: Klassisches neuronales Netz. Rechts: Bayes'sches neuronales Netz. Grafik aus [BCKW15]

ist dieser Unterschied zum Verständnis verbildlicht. Die Verteilung $P(\Theta_W|\ \omega)$ wird durch den *Variational Posterior* $q(\Theta_W|\Theta_q)$ approximiert, welcher eine durch Θ_q parametrisierte Wahrscheinlichkeitsverteilung darstellt. Die Berechnung dieser Verteilung erfolgt mittels *Variational Learning* [BCKW15], in dem die Parameter Θ_q des Variational Posteriors $q(\Theta_W|\Theta_q)$ durch Minimierung der KL-Divergenz mit der a-posteriori Wahrscheinlichkeit der Modellparameter $P(\Theta_W|\ \omega)$ gefunden werden.

$$\Theta_q^* = \underset{\Theta_q}{\operatorname{argmin}}\ D_{KL}(\ q(\Theta_W|\Theta_q)\ \|\ P(\Theta_W|\ \omega)\ ) \tag{2.6.1}$$

$$= \underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q)\ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\Theta_W|\ \omega)}\right) d\Theta_W \tag{2.6.2}$$

Durch Umformen lässt sich die *Variational Free Energy*[BCKW15] (kurz *VFE*) Verlustfunktion ableiten.

$$
\begin{aligned}
&\underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\Theta_W|\ \omega)}\right) d\Theta_W \\
&= \underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln\left(\frac{q(\Theta_W|\Theta_q)P(\omega)}{P(\omega|\ \Theta_W)P(\Theta_W)}\right) d\Theta_W \\
&= \underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \left(\ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\omega|\ \Theta_W)P(\Theta_W)}\right) + \ln(P(\omega))\right) d\Theta_W \\
&= \underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\omega|\ \Theta_W)P(\Theta_W)}\right) d\Theta_W + \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln(P(\omega)) d\Theta_W \\
&= \underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\omega|\ \Theta_W)P(\Theta_W)}\right) d\Theta_W + \underbrace{\ln(P(\omega))}_{\text{Konstant}} \underbrace{\int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) d\Theta_W}_{=1} \\
&= \underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\omega|\ \Theta_W)P(\Theta_W)}\right) d\Theta_W \\
&= \underset{\Theta_q}{\operatorname{argmin}} \int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \left(\ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\Theta_W)}\right) - \ln(P(\omega|\ \Theta_W))\right) d\Theta_W \\
&= \underset{\Theta_q}{\operatorname{argmin}} \underbrace{\int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln\left(\frac{q(\Theta_W|\Theta_q)}{P(\Theta_W)}\right) d\Theta_W}_{= D_{KL}(\ q(\Theta_W|\Theta_q)\ \|\ P(\Theta_W)\ )} - \underbrace{\int_{-\infty}^{\infty} q(\Theta_W|\Theta_q) \ln(P(\omega|\ \Theta_W)) d\Theta_W}_{= \mathbb{E}_{q(\Theta_W|\Theta_q)}[\ln(P(\omega|\ \Theta_W))]}
\end{aligned}
$$

Somit ergibt sich die VFE-Verlustfunktion, welche es zu minimieren gilt.

$$
\mathfrak{L}_{VFE}(\omega) = \underbrace{D_{KL}(\ q(\Theta_W|\Theta_q)\ \|\ P(\Theta_W)\ )}_{\text{Unabhängig von der Stichprobe}} - \underbrace{\mathbb{E}_{q(\Theta_W|\Theta_q)}[\ln(P(\omega|\ \Theta_W))]}_{\text{Ahängig von der Stichprobe}} \tag{2.6.3}
$$

Diese setzt sich aus zwei Komponenten zusammen. Die erste Komponente ist unabhängig von der Stichprobe und stellt die KL-Divergenz zwischen dem Variational Posterior und der Verteilung $P(\Theta_W)$, dem Prior der Gewichte dar. Dieser Ausdruck wird *Complexitiy Cost* genannt und enthält die Information über die Komplexität der Gewichte des neuronalen Netzes, da dieser Maßgeblich vom Prior, welcher als Hyperparameter festgelegt werden muss, geprägt wird. Daher nimmt die Complexity Cost eine Regularisierungsrolle ein, um eine Überanpassung an die Daten zu vermeiden. Als Prior könnte Beispielsweise eine Mischverteilung bestehend aus mehreren Normalverteilungen gewählt werden. Es besteht die Möglichkeit, den Prior zu parametrisieren und diesen ebenfalls zu optimieren, welches dem Regularisierungszweck

gegensätzlich wäre. Dies führe zu schlechteren Ergebnissen als einen festen Prior zu benutzen [BCKW15]. Der Grund hierfür sei, dass, die im Vergleich zum Variational Posterior, wenigen Parameter des Priors sich zu Beginn der Optimierung in einem lokalen Minimum verfangen würden und somit sich der Variational Posterior schlechter von den initialen Werten entfernen würde. Die zweite Komponente der Verlustfunktion ist Abhängig von der Stichprobe und wird *Likelihood Cost* genannt, da es die Erwartete Loglikelihood der Stichprobe angibt. Somit stellt die Verlustfunktion insgesamt den Tradeoff zwischen der Simplizität der Gewichte des neuronalen Netzes und der Fähigkeit des Modells, der Komplexität der Stichprobe gerecht zu werden, dar. Zur Berechnung der VFE wird eine Monte Carlo Approximation getätigt. Dazu werden S mal Parameter $\Theta_{\mathbf{W}}^{(i)} \sim q(\Theta_{\mathbf{W}}|\Theta_q)$ generiert und der Verlust folgendermaßen berechnet.

$$\mathfrak{L}_{VFE}(\omega) \approx \sum_{i=1}^{S} \ln(q(\Theta_{\mathbf{W}}^{(i)}|\Theta_q)) - \ln(P(\Theta_{\mathbf{W}}^{(i)})) - \ln(P(\omega|\ \Theta_{\mathbf{W}}^{(i)})) \tag{2.6.4}$$

Um diese Verlustfunktion mittels Gradientenabstig durch Backpropagation minimieren zu können, muss der Gradient $\nabla_{\Theta_q}\mathfrak{L}_{VFE}(\omega)$ gebildet werden, welcher von den stochastischen Gewichten $\Theta_{\mathbf{W}}^{(i)}$ abhängt, welche von Θ_q abhängen. Da eine stochastische Größe nicht abgeleitet werden kann, wird dazu eine Reparametrisierung angewandt. Dabei werden die Gewichtsparameter $\Theta_{\mathbf{W}}^{(i)} \sim q(\Theta_{\mathbf{W}}|\Theta_q)$ durch eine deterministische Transformation $t(\Theta_q, \underline{\epsilon}^{(i)})$ bestimmt, wobei $\underline{\epsilon}^{(i)} \sim n(\underline{\epsilon})$ eine nichtparametrische Wahrscheinlichkeitsverteilung darstellt. Dies hat das Ziel das Sampling aus der parametrisierten Verteilung $q(\Theta_{\mathbf{W}}^{(i)}|\Theta_q)$ auf ein Sampling aus einer nichtparametrisierten Verteilung $n(\underline{\epsilon})$ zurück zu führen, sodass der stochastische Teil konstant bezüglich des Gradienten ist und dieser gebildet werden kann. Wird beispielsweise eine diagonale Normalverteilung als Variational Posterior genutzt, also $q(\Theta_{\mathbf{W}}|\Theta_q) = \mathcal{N}(\Theta_{\mathbf{W}}|\underline{\mu}_q, \underline{\sigma}_q^2)$ gewählt, kann das Sampling von $q(\Theta_{\mathbf{W}}|\Theta_q)$ auf ein Sampling von $n(\underline{\epsilon}) = \mathcal{N}(\underline{\epsilon}|\underline{0}, \mathbf{I})$ zurückgeführt werden, wobei $\underline{0}$ der Nullvektor und $\mathbf{I}$ die Einheitsmatrix ist. Die Parameter Θ_q des Variational Posterios sind dann mit $\Theta_q = (\underline{\mu}_q, \underline{\sigma}_q)$ gegeben und die deterministische Transformation ist $t(\Theta_q, \epsilon) = \underline{\mu}_q + \underline{\sigma}_q \circ \underline{\epsilon}$, wobei $\circ$ die Komponentenweise Multiplikation darstellt. Somit werden die Gewichtsparameter, nachdem ein $\underline{\epsilon}^{(i)} \sim n(\underline{\epsilon})$ gesampled wurde, gemäß $\Theta_{\mathbf{W}}^{(i)} = t(\Theta_q, \underline{\epsilon}^{(i)})$ berechnet. Damit sind die Gewichtsparameter bezüglich der Verteilungsparameter $\underline{\mu}_q$ und $\underline{\sigma}_q$ ableitbar und eine Optimierung mittels Gradientenabstieg durch Backpropagation möglich. Dieses Verfahren wird daher *Bayes by Backprop* genannt [BCKW15].

Die Vorhersage des bayes'schen neuronalen Netzes kann über folgenden Erwartungswert berechnet werden.

$$P(\underline{y}|\underline{x}) = \mathbb{E}_{q(\Theta_W|\Theta_q)}[\, P(\underline{y}|\underline{x}, \Theta_W)\,] \tag{2.6.5}$$

Dieser Term entspricht der Ausgabe eines Ensembles aus unendlich vielen neuronalen Netzen und ist in der Praxis nicht berechenbar. Daher wird dieser durch die Monte Carlo Methode folgendermaßen approximiert.

$$P(\underline{y}|\underline{x}) \approx \frac{1}{S}\sum_{i=1}^{S} P(\underline{y}|\underline{x}, \Theta_W^{(i)}), \quad \text{wobei } \Theta_W^{(i)} \sim q(\Theta_W|\Theta_q) \tag{2.6.6}$$

2.6.1 *Unsicherheiten des bayes'schen neuronalen Netzes berechnen*

Einen Vorteil den bayes'sche neuronale Netze mit sich bringen, ist dass sie die Möglichkeit bieten die epistemische Unsicherheit zu berechnen. Dazu kann die Dekomposition der Gesamtunsicherheit des Modells nach [DHLDVU18] genutzt werden. Die Gesamtunsicherheit ϵ_g eines bayes'schen neuronalen Netzes mit Softmax-Ausgabelayer wird hierbei durch die Entropie (2.2.2) der Ausgabe dargestellt.

$$\epsilon_g = H(P(y|\underline{x})) = -\sum_{i=1}^{M^{(L)}} P(y_i|\underline{x}) \log(P(y_i|\underline{x})) \tag{2.6.7}$$

Die aleatorische Unsicherheit wird durch folgenden Ausdruck dargstellt.

$$\epsilon_a = \mathbb{E}_{q(\Theta_W|\Theta_q)}[\, H(P(y|\underline{x}, \Theta_W))\,] \tag{2.6.8}$$

Analog zu 2.6.6 wird ϵ_a durch die Monte Carlo Methode approximiert.

$$\epsilon_a \approx \frac{1}{S}\sum_{i=1}^{S} H(P(y|\underline{x}, \Theta_W^{(i)})), \quad \text{wobei } \Theta_W^{(i)} \sim (\Theta_W|\Theta_q) \tag{2.6.9}$$

Unter der Annahme aus 2.2.1, dass sich die Gesamtunsicherheit additiv aus aleatorischer und epistemischer Unsicherheit zusammensetzt, kann die epistemische unsicherheit wie folgt berechnet werden.

$$\epsilon_e = \epsilon_g - \epsilon_a \tag{2.6.10}$$

Unter der Annahme, dass Modelle mit $\Theta_W \sim q(\Theta_W|\Theta_q)$ eine ähnliche Vorhersage treffen, wenn $\underline{x} \sim P(\Omega)$ gilt und unterschiedliche Vorhersagen treffen, wenn $\underline{x} \nsim P(\Omega)$, ist zu erwarten, dass die epistemische Unsicherheit ϵ_e im letzteren Fall einen höheren Wert hat als im ersteren Fall. Somit kann ϵ_e zur erkennung von Out-of-Distribution Eingaben eingesetzt werden.

2.6.2 *Vor- und Nachteile bayes'scher neuronaler Netze*

Durch die intrinsische Regularisierung der Modellparameter und der Ensembleartigen Vorhersage ist zu erwarten, dass bayes'sche neuronale Netze eine höhere Leistung erzielen als ihre klassischen Gegenstücke. Des Weiteren ist es durch die stochastische Basis möglich epistemische und aleatorische Unsicherheiten zu berechnen. Der Preis hierfür ist jedoch erhöhter Rechenaufwand. Dieser besteht zum einen daraus, mehrfach Parameter aus einer Verteilungsdichte zu generieren und zum anderen aus der Bildung des Erwartungswertes, welcher gemäß 2.6.6 mittels S Samples approximiert wird. So müssen für die Inferenz Vorhersagen von S neuronalen Netzen berechnet werden, sodass die Inferenzzeit, abzüglich des Samplings, S mal so hoch ist wie die eines klassischen neuronalen Netzes. Das generieren von Zahlen, welcher einer bestimmten Verteilungsdichte folgen, ist nicht trivial und je nach gewählter Verteilungsdichte mit erheblichem Rechenaufwand verbunden. So beträgt der Rechenaufwand beim erzeugen eines d-dimensionalen Datenpunktes welches einer beliebigen, fest gewählten d-dimensionalen Normalverteilung folgt, $\mathcal{O}(d^3)$ *Floating Point Operations* kurz (*FLOPs*), wenn der *Cholesky Sampler* genutzt wird [VDC21]. Die aktuelle High-End Consumer Grafikkarte *Nvidia RTX 3090* hat eine theoretische Leistung von $35.58 \cdot 10^{12}$ FLOP pro Sekunde [RTX]. So würde das Sampling von einer Millionen Parametern aus dem Variational Posterior, stark vereinfacht gerechnet, ca. 8 Stunden dauern, wenn dazu der Cholesky Sampler genutzt würde. Wird eine diagonale Normalverteilung genutzt kann dieser Aufwand jedoch auf $\mathcal{O}(d)$ reduziert werden [VDC21]. In Betracht der großen Parameteranzahl bei Neuronalen Netzen, kann daher die Wahl des Variational Posteriors, aus dem mehrfach gesampled werden muss, erheblichen Einfluss auf die Inferenzzeiten nehmen. Ein weiterer Nachteil ist die Beschränkung des Suchraums der Gewichtsparameter durch den Variational Posterior, welcher eine parametrisierbare Wahrscheinlichkeitsverteilung sein muss. Daher kann es sein, dass das Netz eine schlechte Leistung aufweist, wenn die gewählte Wahrscheinlichkeitsverteilung, die wahre Verteilung nicht im benötigten Maße abbilden kann. Das heißt die Wahl des Variational posteriors ist generell auf die parametrisierbaren Verteilungen beschränkt und für Modelle mit sehr vielen Parametern, auf Verteilungen aus denen effizient gesamplet werden kann.

3

VERWANDTE ARBEITEN

In diesem Kapitel werden verwandte Arbeiten vorgestellt. Anzufangen wäre mit [HG16], welches eine Baseline für die Erkennung von Out-of-Distribution eingaben vorschlägt. In dieser wird der Softmax-Score der vorhergesagten Klasse als Indikator für eine Out-of-Distribution Eingabe verwendet. Dabei wird dort angenommen, dass der Softmax-Score niedriger ist, wenn das Modell unsicher ist. Daher wird dort mittels Threshold und dem größten Softmax-Score der Ausgabe des Modells die Out-of-Distribution-Eingaben-Klassifikation vorgenommen. Daher wird in dieser Arbeit dieses Verfahren als Referenz genommen und die Leistung der hybriden Modelle im Vergleich zur vorgeschlagenen Baseline bewertet. In [Cha21] werden ebenfalls hybride Modelle untersucht, wobei dort nur Modelle untersucht werden, bei denen die letzte Schicht bayes'sch ist. Dort wird außerdem eine Variation *Functional Bayesian Inference* verwendet, wobei dort die Unsicherheit in den Netzen durch probabilistische Aktivierungsfunktionen in den Schichten modelliert wird. Die Ergebnisse des Papers sind, dass hybride Netze, wie sie in dieser Arbeit genutzt werden, die Unsicherheiten besser abbilden als klassische Modelle und als die vorgestellte Variation, jedoch werden keine hybriden Modelle untereinander verglichen. Die Arbeit [HDG21] stellt in Frage, ob sich bayes'sche neuronale Netze grundsätzlich zur Erkennung von Out-of-Distribution Eingaben eignen. Denn dabei wird angenommen, dass die aus dem Variational Posterior gesampleten Modelle unterschiedliche Vorhersagen treffen, wenn die Eingabe Out-of-Distribution ist. Dabei wird zum Schluss gekommen, dass die Leistung des bayes'schen Modells maßgeblich vom ausgewählten Parameter-Prior abhängt. Eine weitere Arbeit, welche sich mit der Abbildung von Unsicherheiten neuronaler Netze auseinandersetzt ist [GPSW17]. Dort wird die mittels *Temperature Scaling* die Softmax-Ausgabe des Netzes skaliert, wodurch diese nach Kalibrierung nicht mehr zur Überkonfidenz neigen. Die Kalibrierung erfolgt dabei über einen aus der Validierungsmenge erlernten Parameter T mit der die Erregung der letzten Schicht des Netzes normiert wird. Dadurch werden die Exponenten bei der Softmax-Berechnung kleiner, sodass kleinere Unterschiede in der Erregung des Netzes besser in der Softmax-Ausgabe sichtbar werden.

4 METHODIK

In den bisherigen Abschnitten wurden die Grundlagen zum Verständnis bayes'scher neuronaler Netze erläutert und Bewertungskriterien präsentiert. Im folgenden wird ein hybrides Modell vorgestellt, welches experimentell untersucht werden soll. Ziel des hybriden Ansatzes ist es den Rechenaufwand bayes'scher neuronaler Netze zu reduzieren und dabei dessen Vorteile beizubehalten. Dazu wird die Klassifikationsleistung untersucht und insbesondere die Erkennung von Out-of-Distribution Eingaben auf Grundlage der Modellunsicherheit betrachtet. Dazu werden die in Kapitel 4.3 vorgestellten Bewertungsmaße, insbesondere das AUROC und AUPR Maß, genutzt. In diesem Kapitel wird daher das hybride Modell vorgestellt, der Aufbau der Experimente beschrieben und die Bewertungskriterien erläutert.

4.1 RECHENAUFWAND DER OPERATIONEN

Da im folgenden der Rechenaufwand verschiedener Modelle verglichen wird, wird im Folgenden die Notation definiert. Hierbei ist zu erwähnen, dass die real berechneten FLOPs je nach Hardware und Implementierung der Operationen variieren kann. Daher stellt die folgende Darstellung eine Abschätzung dar, welche die Annahme hat, dass entsprechend der Formeln ohne Optimierung der Berechnungsklauseln gerechnet wird. Mit

$$F(\cdot)$$

wird der Rechenaufwand in FLOPs einer Operation angegeben. So ist beispielsweise

$$F(1.1 + 2.2 \cdot 3.3) = 2$$

Gebe $\text{sample}(p(\underline{x}))$ die Operation an, mit der ein Element $\underline{x} \sim p(\underline{x})$ generiert wird. So würde

$$F(\text{sample}(p(\underline{x})))$$

die Anzahl an FLOPs angeben, die nötig sind um $\underline{x}$ aus der Verteilung $p(\underline{x})$ zu samplen. Der Aufwand zur Berechnung eines neuronalen Netzes wird nun schichtweise betrachtet. Wird die Erregung der des j-ten Neurons der l-ten Schicht $u_j^{(l)}$ eines vollvernetzten neuronalen Netzes gemäß Kapitel 2.5 als eine Funktion $u_j^{(l)}(\underline{w}_j^{(l)}, \underline{f}^{(l-1)})$ betrachtet,

wobei die Eingaben bereits berechnet sind, so ergibt sich für die Berechnung der FLOPs dieser Schicht folgendes.

$$F(u_j^{(l)}) = F\left(\underbrace{\sum_{i=0}^{M^{(l-1)}} \underbrace{w_{ij}^{(l)} f_i^{(l-1)}}_{\text{Eine Mutliplik. pro Summand}}}_{M^{(l-1)}\text{ Additionen}}\right) = \underbrace{M^{(l-1)}+1}_{\text{Additionen}} + \underbrace{M^{(l-1)}}_{\text{Multiplikationen}} \tag{4.1.1}$$

$$= 2 \cdot M^{(l-1)} + 1 \tag{4.1.2}$$

Die Schicht l besteht aus $M^{(l)}$ Neuronen, so dass für den Rechenaufwand der gesamten Schicht folgendes gilt.

$$F(\underline{u}^{(l)}) = M^{(l)} \cdot (2 \cdot M^{(l-1)} + 1) \tag{4.1.3}$$

Zur Berechnung der Aktivierung einer Schicht $\underline{f}^{(l)}$ muss zunächst die Erregung $\underline{u}^{(l)}$ berechnet werden und darauf hin komponentenweise die Aktivierungsfunktion der l-ten Schicht $\sigma^{(l)}(\cdot)$ angewandt werden. So ergibt sich folgendes.

$$F(\underline{f}^{(l)}) = F(\underline{u}^{(l)}) + M^{(l)} \cdot F(\sigma^{(l)}(\cdot)) \tag{4.1.4}$$

$$= M^{(l)} \cdot (2 \cdot M^{(l-1)} + F(\sigma^{(l)}(\cdot)) + 1) \tag{4.1.5}$$

Wird nun die Vorhersage $\underline{\hat{y}}$ eines L-schichtigen neuronalen Netzes berechnet hat dies folgenden Rechenaufwand, wobei $NN(M^{(0)}, M^{(1)}, \ldots, M^{(L)})$ das neuronale Netz bezeichnet.

$$F\left(NN(M^{(0)}, M^{(1)}, \ldots, M^{(L)})\right) = \sum_{l=1}^{L} F(\underline{f}^{(l)}) = \sum_{l=1}^{L} M^{(l)} \cdot (2 \cdot M^{(l-1)} + F(\sigma^{(l)}(\cdot)) + 1) \tag{4.1.6}$$

Der Rechenaufwand der Aktivierungsfunktionen wird im folgenden generell mit $F(\sigma_x(\cdot)) = 4$ abgeschätzt, da sie sehr stark von der Implementierung abhängt und nur einen kleinen Teil des Rechenaufwands ausmacht[1]. Lediglich der Rechenaufwand von $\sigma_{\text{softmax}}(\cdot)$ wird mit $F(\sigma_{\text{softmax}}(\underline{x})) = \dim(\underline{x})$ abgeschätzt, wobei $\dim(\underline{x})$ die Dimension

1 Den Großteil der Berechnung macht $M^{(l)} \cdot 2 \cdot M^{(l-1)} \in \mathcal{O}(M^{(l)} \cdot M^{(l-1)})$ aus, da $F(\sigma^{(l)}(\cdot))$ nur linearen Rechenaufwand von $\mathcal{O}(M^{(l)})$ hat

des Vektors $\underline{x}$ darstellt. Wird beispielsweise eine zwei-schichtige Architektur betrachtet, welche die Sigmoid-Aktivierung in der Eingabeschicht und die Softmax-Funktion in der Ausgabe nutzt, wobei der Eingabevektor 784-dimensional ist, die versteckte Schicht aus 256 Neuronen und die Ausgabeschicht aus 10 Neuronen besteht so ergibt dies folgenden Rechenaufwand, wobei $\mathrm{NN}(784, 256, 10)$ das Modell bezeichnet.

$$\begin{aligned} \mathrm{F}(\mathrm{NN}(784, 256, 10)) &= 256 \cdot (2 \cdot 784 + \mathrm{F}(\sigma_{sigmoid}(\cdot)) + 1) \\ &\quad + 10 \cdot (2 \cdot 256 + \mathrm{F}(\sigma_{softmax}(\cdot)) + 1) \\ &= 407918 \text{ FLOPs} \end{aligned}$$

Der Rechenaufwand einer Faltung mit dem Kernel $\mathrm{K}(m \times n, c, s)$, wobei $m \times m$ die Grundflächendimension, c die Anzahl der Kanäle der Eingabe $f(x, y, c)$ und s den Stride darstellt, lässt sich anhand der Dimension seiner Ausgabe berechnen, welche von der Eingabedimension und dem Stride abhängt. Im folgenden wird die Rechnung vereinfacht und Padding nicht betrachtet, da es nur marginalen Einfluss auf die Rechnung hat. Seien M_x und M_y die Größen der Grundfläche der Eingabe und g_{ijk} die Gewichte des Kernels. Dann ist die Dimension der Ausgabe der Faltung mit $\mathrm{K}(m \times m, c, s)$ durch $\frac{M_x}{s}$ und $\frac{M_y}{s}$ gegeben. Für jeden Punkt der Ausgabe muss der Ausdruck

$$\sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \sum_{k=0}^{c-1} f(x - i, y - j, k) \cdot g_{ijk}$$

berechnet werden, welcher aus $m \cdot m \cdot c$ Produkten und $m \cdot m \cdot c - 1$ Summen besteht. Damit ergibt sich für den Rechenaufwand folgendes.

$$\mathrm{F}(\mathrm{K}(m \times n, c, s)) = \frac{M_x}{s} \cdot \frac{M_y}{s} \cdot (2 \cdot m^2 \cdot c - 1) \approx 2 \cdot \frac{M_x}{s} \cdot \frac{M_y}{s} \cdot m^2 \cdot c \tag{4.1.7}$$

Für die Berechnungen in den Experimenten wird die Approximation genutzt.

4.2 HYBRIDE BAYES'SCHE NEURONALE NETZE

Das in Kapitel 2.6 vorgestellte bayes'sche Modell hat den Nachteil, dass die Inferenz rechenaufwändiger ist als die Inferenz bei klassischen Netzen. Die in Abschnitt 2.6.2 beschriebenen Gründe hierfür sind zweierlei: zum einen bringt das Sampling der Gewichtsparameter Rechanaufwand mit sich und zum anderen die Erwartungswertbildung.

4.2.1 *Rechenaufwand bayes'scher neuronaler Netze*

Das bayes'sche Gegenstück $\text{BNN}(M^{(0)}, M^{(1)}, \ldots, M^{(L)})$ eines neuronalen Netzes $\text{NN}(M^{(0)}, M^{(1)}, \ldots, M^{(L)})$ hat folgenden Rechenaufwand, wobei der Erwartungswert mit S Samples gemäß Formel 2.6.6 approximiert wird.

$$F\left(\text{BNN}(M^{(0)}, \ldots, M^{(L)})\right) = \underbrace{S \cdot F\left(\text{NN}(M^{(0)}, \ldots, M^{(L)})\right)}_{\text{Berechnung des Erwartungswertes}} + \underbrace{S \cdot F(\text{sample}(q(\Theta_W|\Theta_q)))}_{\text{Sampling}} \tag{4.2.1}$$

Als Beispiel wird der Rechenaufwand des bayes'sche Gegenstück $\text{BNN}(784, 256, 10)$ des zweischichtigen neuronalen Netzes $\text{NN}(784, 256, 10)$ aus vorherigem Kapitel betrachtet. Unter der Annahme, dass der Variational Posterior $q(\Theta_W|\Theta_q)$ eine diagonale Normalverteilung ist, kann, wie in Kapitel 2.6.2 beschrieben, mit $\mathcal{O}(|\Theta_W|)$ Rechenaufwand gesampled werden, wobei $|\Theta_W|$ die Anzahl der Gewichtsparameter angibt. Für das $\text{BNN}(784, 256, 10)$ Netz gilt[2] $|\Theta_W| = 784 \cdot 256 + 256 \cdot 10 = 203264$. Daher wird $F(\text{sample}(q(\Theta_W|\Theta_q)))$ für diagonale Normalverteilungen mit dieser Zahl abgeschätzt. Somit ergibt sich für den Rechenaufwand folgendes.

$$F\left(\text{BNN}(784, 256, 10)\right) = S \cdot 409454 + S \cdot 203264 = S \cdot 612718 \tag{4.2.2}$$

Wird nun $S = 10$ gewählt, so hat die Inferenz des bayes'schen neuronalen Netzes 15 mal mehr Rechenaufwand als ihr nicht-bayes'sches Gegenstück.

4.2.2 *Das hybride Modell*

Um den Rechenaufwand zu reduzieren ist es maßgeblich die Anzahl der gesampleten Parameter zu reduzieren und die Berechnung der Abschätzung des Erwartungswerts zu vereinfachen. Ersteres ist möglich in dem die einzelnen Gewichtsparameter $\Theta_w = \{w_1, w_2, \ldots, w_{|\Theta_w|}\}$ in zwei Teilmengen $\Theta_{w_b}, \Theta_{w_k}$ partitioniert werden. Die erste Teilmenge Θ_{w_k}, deren Elemente im folgenden *klassische Gewichte* genannt werden, beinhaltet hierbei die Gewichte, welche klassisch per Punktschätzung bestimmt werden. Die zweite Teilmenge Θ_{w_b}, dessen Elemente im Folgenden *bayes'sche Gewichte* genannt werden, beinhaltet die Gewichte, deren Verteilung $q(\Theta_{w_b}|\Theta_q))$ erlernt werden soll. Dabei gibt es potentiell $2^{|\Theta_W|}$ Möglichkeiten die Gewichte aufzuteilen. Hybride Netze, bei denen in jeder Schicht klassische und bayes'sche Gewichte gemischt sind,

2 Hierbei werden die Bias bei der Zählung zur vereinfachung ausgelassen

werden *elementweise hybride bayes'sche neuronale Netze* genannt. Bei dieser Art von hybriden Netzen wird $F(\mathrm{BNN}(\dots))$ nur durch die Reduktion des FLOPs im Sampling-Teil minimiert. Der Rechenaufwand des Erwartungswert-Teils kann jedoch durch geschicktes wählen von Θ_{w_b} und Θ_{w_k} ebenfalls reduziert werden. Dazu werden die Gewichte schichtweise den Partitionen zugeordnet. Seien die Gewichte schichtweise Gruppiert, das heißt also $\Theta_{\mathbf{W}} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(L)}\}$, wobei die Gewichtsmatrizen entsprechend 2.5.9 definiert sind. Dann ist ein Modell *schichtweise hybrid bayes'sch*, wenn die Partitionen $\Theta_{\mathbf{W}_b}$ und $\Theta_{\mathbf{W}_k}$ ausschließlich ganze Gewichtsmatrizen $\mathbf{W}^{(l)}$ beinhalten. So gibt es zu einem L-schichtigen neuronalen Netz 2^L mögliche schichtweise hybride bayes'sche neuronale Netze. Da $L << |\Theta_w|$ und somit $2^L << 2^{|\Theta_w|}$ gibt es viel weniger schichtweise hybride als elementweise hybride, sodass der Suchraum für ein optimales Modell kleiner ist. Nichts desto trotz wird der Suchraum für ein optimales schichtweise hybrides Modell exponentiell groß. Existiert ein l_b, sodass $\forall l \geqslant l_b . \mathbf{W}^{(l)} \in \Theta_{\mathbf{W}_b}$ gilt, wird das Modell *getrennt hybrides bayes'sches* neuronales Netz und l_b Trennindex genannt. Bei getrennt hybriden bayes'sche neuronale Netzen be-

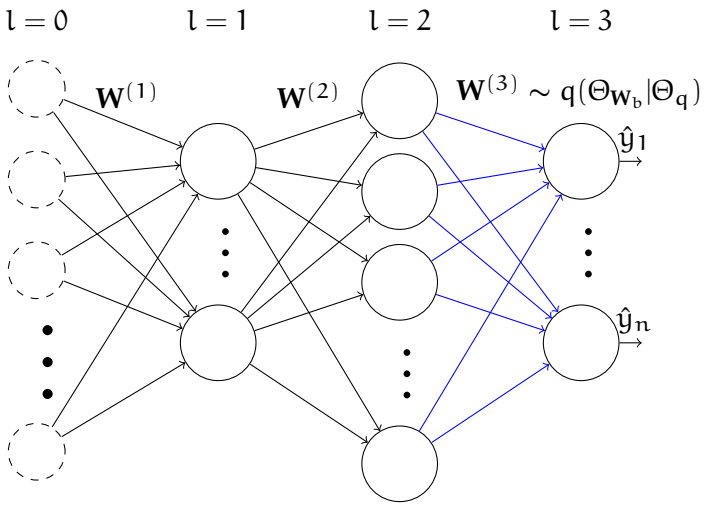


Abbildung 4.2.1: 3-Schichtiges getrennt hybrides neuronales Netzwerk mit $l_b = 3$. Klassische Gewichte sind in Schwarz dargestellt, bayes'sche in Blau.

steht das Modell bis zu der Gewichtsmatrize $\mathbf{W}^{(l_b-1)}$ aus klassischen Gewichten und $\mathbf{W}^{(l_b)}$ aus bayes'schen Gewichten. Bezeichne $\mathrm{GBNN}_{l_b}(M^{(0)}, \dots, M^{(L)})$ das getrennt hybride neuronale Netz mit Trennindex l_b. So lässt sich dieses Modell in die zwei Teilmodelle $\mathrm{NN}(M^{(0)}, \dots, M^{(l_b-1)})$ und $\mathrm{BNN}(M^{(l_b-1)}, \dots, M^{(L)})$ aufteilen, welches in Abbildung 4.2.1 zu sehen ist. Bei der Inferenz können nun FLOPs gespart werden

indem der Teil mit den klassischen Gewichten nur einmal und der bayes'sche Teil S mal berechnet wird. Insgesamt ergibt sich für den Rechenaufwand folgendes.

$$\begin{aligned} F(GBNN_{l_b}(M^{(0)}, \dots, M^{(L)})) &= F(NN(M^{(0)}, \dots, M^{(l_b-1)})) \\ &\quad + F(BNN(M^{(l_b-1)}, \dots, M^{(L)})) \\ &= F(NN(M^{(0)}, \dots, M^{(l_b-1)})) \\ &\quad + S \cdot F(NN(M^{(l_b-1)}, \dots, M^{(L)})) \\ &\quad + S \cdot F(\text{sample}(q(\Theta_{W_b}|\Theta_q))) \end{aligned} \tag{4.2.3}$$

Damit wird das getrennt-hybride bayes'sche neuronale Netz effizienter, je näher der Trennindex l_b der letzten Schicht ist. Sei $BNN(784, 100, 256, 10)$ ein 3-schichtiges bayes'sches neuronales Netz mit Sigmoid-Aktivierungsfunktion, der Softmax-Aktivierung in der Ausgabeschicht und einer diagonalen Normalverteilung als Variational Posterior. Dann ist der benötigte Rechenaufwand bei der Inferenz mit $S = 10$ Samples

$$F(BNN(784, 100, 256, 10)) = 3215700 \text{ FLOPs}.$$

Das entsprechende getrennt hybride Modell $GBNN_3(784, 100, 256, 10)$ hätte hingegen einen Rechenaufwand von

$$F(GBNN_3(784, 100, 256, 10)) = 287680 \text{ FLOPs}.$$

Das völlig klassische Modell $NN(784, 100, 256, 10)$ hat einen Rechenaufwand von

$$F(NN(784, 100, 256, 10)) = 215010 \text{ FLOPs}.$$

Zusammengefasst bedeutet dies, dass das $GBNN_3(784, 100, 256, 10)$ Modell nur 8.9% des Rechenaufwands des völlig bayes'schen Modells benötigt und etwa 1.34 mal mehr Rechenaufwand erfordert als das entsprechende klassische Modell.
Wie anhand der obigen Darstellung zu sehen ist, kann der Rechenaufwand deutlich gesenkt werden. Welchen Einfluss diese Reduktion auf die Leistungen des Modells hat wird in den folgenden Kapiteln experimentell untersucht.

4.2.3 *Bitvektordarstellung*

Im Folgenden wird eine Bitvektordarstellung für schichtweise hybride bayes'sche neuronale Netze mit L Schichten vorgestellt. Dabei geben die Bits eines Bitvektors der Länge L an, ob die Gewichtsmatrix der entsprechenden Schicht des hybriden

Netzes bayes'sch ist oder nicht. Sei $z \in \mathbb{N}$ die Darstellung des Bitvektors der Länge L als natürliche Zahl in Binärdarstellung mit korrespondierendem hybriden Modell mit Parametervektor $\Theta_{\mathbf{W}} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \ldots, \mathbf{W}^{(L)}\}$. Dann ist die Zugehörigkeit der Gewichtsmatrix $\mathbf{W}^{(l)}$ zu den Partitionen $\Theta_{\mathbf{W}_b}$,$\Theta_{\mathbf{W}_k}$ wie folgt durch z bestimmt.

$$\begin{aligned} &\mathbf{W}^{(l)} \in \Theta_{\mathbf{W}_b} \Leftrightarrow z \bmod 2^{(L-l)} = 1 \\ &\text{und} \\ &\mathbf{W}^{(l)} \in \Theta_{\mathbf{W}_k} \Leftrightarrow z \bmod 2^{(L-l)} = 0 \end{aligned} \tag{4.2.4}$$

Da die Darstellungen der Modelle meistens die Eingabeschicht auf der linken Seite und die Ausgabeschicht auf der rechten Seite haben, kodiert das höchstwertige Bit 2^{L-1} die Zugehörigkeit der Gewichtsmatrix $\mathbf{W}^{(1)}$ zu den Partitionen $\Theta_{\mathbf{W}_b}$, $\Theta_{\mathbf{W}_k}$ und das niedrigstwertige Bit 2^0 die Zugehörigkeit der Gewichtsmatrix $\mathbf{W}^{(L)}$. Somit können mit einem Bitvektor die Zugehörigkeiten der Gewichte des schichtweise hybrid bayes'schen neuronalen Netzes eindeutig beschrieben werden. Daher werden schichtweise hybride Modelle im Folgenden mit $\mathrm{BNN}_b(M^{(0)}, \ldots, M^{(L)})$ bezeichnet. Ein vollständig bayes'sches neuronales Netz mit drei Schichten hätte die Bezeichnung $\mathrm{BNN}_{111}(\ldots)$ und ein entsprechendes klassisches neuronales Netz die Bezeichnung $\mathrm{BNN}_{000}(\ldots)$. Die Bezeichnung für das in Abbildung 4.2.1 zu sehende Modell wäre dementsprechend $\mathrm{BNN}_{001}(\ldots)$.

4.3 BEWERTUNG VON MODELLEN

In diesem Kapitel werden Methoden vorgestellt, anhand derer die Leistung von Klassifikatoren quantitativ bewertet werden kann [SL09]. Sei die annotierte Stichprobe $\omega = \{({}^{\rho}\underline{x}, {}^{\rho}y) | \rho = 1, \ldots, N\}$ aus dem betrachteten Problemkreis gegeben, wobei ${}^{\rho}y$ die Klassenzugehörigkeit angibt. Mit ${}^{\rho}\hat{y}$ sei die Vorhersage des Modells bei Eingabe von ${}^{\rho}\underline{x}$ bezeichnet.

4.3.1 *Bewertung binärer Klassifikatoren*

Zunächst wird der Zweiklassen-Fall ${}^{\rho}y \in \{\text{Positiv}, \text{Negativ}\}$ betrachtet, wobei häufig eine Klasse als *positive* und die andere als *negative* Klasse deklariert wird. Hierbei können Evaluationsmetriken aus der Konfusionsmatrix abgeleitet werden, welche wie folgt aufgebaut ist.

	$\hat{y}$ Positiv	$\hat{y}$ Negativ	$\sum$
y Positiv	TP	FN	TP + FN
y Negativ	FP	TN	FP + TN
$\sum$	TP + FP	FN + TN	$\lvert\omega\rvert$

Abbildung 4.3.1: Konfusionsmatrix

Hierbei gibt TP die Anzahl der positiven Stichprobenelemente an, welche das Modell auch als positiv klassifiziert hat. Diese Stichprobenelemente werden *True Positives* genannt. Die *False Negatives* werden mit FN bezeichnet und geben die positiven Stichprobenelemente an, welche vom Modell fälschlicherweise zur negativen Klasse zugeordnet wurden. Die *False Positives* FP und *True Negatives* TN sind analog dazu definiert. Aus diesen Elementen lassen sich nun Metriken definieren, welche die Leistung eines binären Klassifikators bemessen.

$$\text{Accuracy} = \frac{TP + TN}{|\omega|} \tag{4.3.1}$$

Die *Accuracy* gibt das Verhältnis korrekt klassifizierter Stichprobenelemente zur gesamten Stichprobengröße an. Somit kann eingesehen werden, wie viel Prozent der Vorhersagen des Klassifikators korrekt sind.

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{TP}{|\{{}^{\rho}\underline{x} \mid {}^{\rho}\hat{y} = \text{Positiv}\}|} \tag{4.3.2}$$

Die *Precision* gibt das Verhältnis korrekt positiv klassifizierter Stichprobenelemente zu allen als positiv klassifizierten Stichprobenelementen an. Mit ihr kann eingesehen werden, wie verlässlich eine positive Klassifikation ist.

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{TN}{|\{\rho\underline{x} \mid \rho y = \text{Negativ}\}|} \tag{4.3.3}$$

Die *Specificity* ist das negative Analogon zur Precision. Sie gibt das Verhältnis korrekt negativ klassifizierter Stichprobenelemente zu allen negativen Stichprobenelementen an. Mit ihr kann eingesehen werden, wie verlässlich eine negative Klassifikation ist.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{|\{\rho\underline{x} \mid \rho y = \text{Positiv}\}|} \tag{4.3.4}$$

Der *Recall* gibt das Verhältnis korrekt klassifizierter positiver Stichprobenelemente zu allen positiven Stichprobenelementen an. Mit Hilfe des Recalls kann eingesehen werden, wie hoch der Anteil erkannter positiver Stichprobenelemente ist.

$$\text{FPR} = \frac{FP}{FP + TN} = \frac{FP}{|\{\rho\underline{x} \mid \rho y = \text{Negativ}\}|} \tag{4.3.5}$$

Die *False-Positive-Rate* gibt das Verhältnis inkorrekt positiv klassifizierter Stichprobenelemente zu allen negativen Stichprobenelementen an. Damit kann abgeschätzt werden, wie viele der negativen Stichprobenelemente anteilmäßig fälschlicherweise als positiv klassifiziert werden.
Diese Maße können jedoch irreführend sein, wenn die Anteile an positiven und negativen Stichprobenelementen ungleich sind. Sei $k_p(\underline{x})$ ein Klassifikator, welcher für alle $\underline{x}$ die am häufigsten vertretene Klasse vorhersagt, also $\forall\underline{x} \in \Omega.\ k_{max}(\underline{x}) = \underset{i}{\text{argmax}}|\Omega_i|$. Sind beispielsweise 99% der Stichprobenelemente positiv und 1% negativ, würde $k_{max}(\underline{x})$ eine Accuracy von 0.99 erreichen. Die Precision läge bei 0.99 und der Recall wäre bei 1. Diese Werte sind alle sehr nah am optimalen Wert 1. Lediglich die Specificity liegt bei 0, da kein Element als negativ klassifiziert wird. Somit ist es wichtig die Klassenverteilung bei der Interpretation der Werte zu beachten und mehrere Maße zu Rate zu ziehen.

4.3.2 *Thresholdabhängige binäre Klassifikatoren*

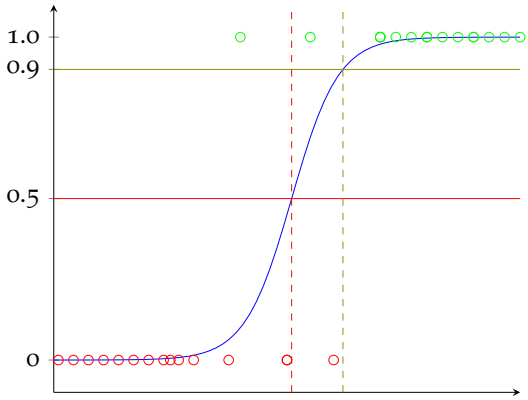
Klassifikatoren, bei denen erst ein Wert berechnet und dann anhand eines Thresholds τ eine Klassenzuordnung getätigt wird, können verschiedene Werte in den Gütemaßen erzielen. Zu Veranschaulichungszwecken wird nun ein einfaches logistisches

Regressionsmodell[HTF09] $l(\underline{x})$ für binäre Klassifikation betrachtet. Die Funktion $l(\underline{x})$ soll dabei die a-posteriori Wahrscheinlichkeit der positiven Klasse erlernen.

$$l(\underline{x}) = p(^{\rho}y = \text{Positiv} \mid {}^{\rho}\underline{x}) = \frac{\exp(\beta_0 + \underline{\beta}^T \underline{x})}{1 + \exp(\beta_0 + \underline{\beta}^T \underline{x})} \tag{4.3.6}$$

Der Klassifikator $k_l(\underline{x})$ ist dann wie folgt definiert.

$$k_l(\underline{x}) = \begin{cases} \text{Positiv, wenn } p(^{\rho}y = \text{Positiv} \mid {}^{\rho}\underline{x}) \geqslant \tau \\ \text{Negativ, wenn } p(^{\rho}y = \text{Positiv} \mid {}^{\rho}\underline{x}) < \tau \end{cases} \tag{4.3.7}$$



$\tau = 0.5$	$\hat{y}$ Positiv	$\hat{y}$ Negativ	$\sum$
y Positiv	11	1	12
y Negativ	1	13	14
$\sum$	12	14	26

$\tau = 0.9$	$\hat{y}$ Positiv	$\hat{y}$ Negativ	$\sum$
y Positiv	10	2	12
y Negativ	0	14	14
$\sum$	10	16	26

Abbildung 4.3.2: Logistische Regression mit verschiedenen Thresholds und zugehörigen Konfusionsmatrizen. Die positiven Stichprobenelemente sind durch grüne und die negativen rote Kreise abgebildet. Die logistische Kurve $l(\underline{x})$ ist durch die blaue kurve dargestellt.

In Abbildung 4.3.2 ist Beispielhaft ein solches eindimensionales Modell visualisiert. Dort wird ersichtlich, dass sich die Werte in der Konfusionsmatrix für verschiedene τ unterscheiden, sodass sich unterschiedliche Klassifikationsleistungen ergeben.

Werte für $\tau = 0.5$:	Werte für $\tau = 0.9$:
Accuracy $= 0.92$	Accuracy $= 0.92$
Precision $= 0.91$	Precision $= 1.00$
Recall $= 0.91$	Recall $= 0.83$
FPR $= 0.07$	FPR $= 0.00$

Die Klassifikationsmaße können nun dabei helfen, den Klassifikator $k_l(\underline{x})$ für verschiedene τ zu vergleichen und ein, für das Anwendungsszenario geeignetes τ, zu wählen.

Mit den bisher vorgestellten Maßen wird es jedoch problematisch, wenn zwei unterschiedliche, thresholdabhängige Klassifikatoren miteinander verglichen werden sollen. Hierfür ist ein Maß nötig, welches unabhängig vom gewählten Threshold auskunft über die Klassifikationsleistung geben kann. Sei $m(\underline{x})$ ein Modell, welches für $\underline{x} \in \Omega$ einen Wert, im folgenden *Score* genannt, berechnet und $k_m(\underline{x}|\tau)$ der thresholdabhängige binäre Klassifikator, welcher wie folgt definiert ist.

$$k_m(\underline{x}|\tau) = \begin{cases} \text{positiv, wenn } m(\underline{x}) \geqslant \tau \\ \text{negativ, wenn } m(\underline{x}) < \tau \end{cases} \tag{4.3.8}$$

Dann lässt sich die *Receiver Operating Characteristic* (kurz *ROC*) [HM82] Kurve erstellen, in dem für alle möglichen Thresholds τ das Tupel $(\text{FPR}, \text{Recall})$ bestimmmt wird und diese in einen Koordinatensystem eingetragen werden. Dadurch entsteht eine Kurve mit dem Definitionsbereich $[0, 1]$ und Wertebereich $[0, 1]$. In Abbildung 4.3.3 sind

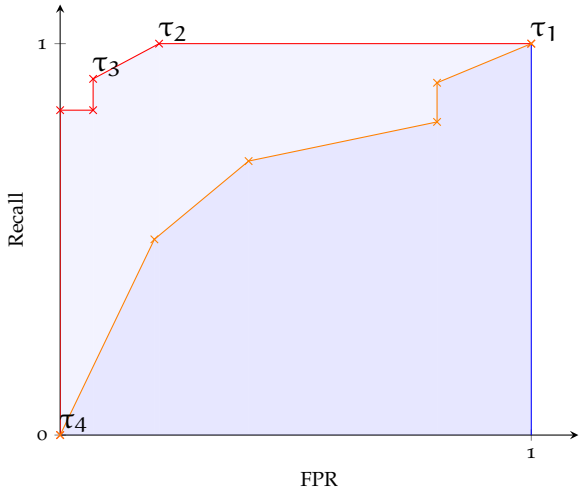


Abbildung 4.3.3: Visualisierung von ROC-Kurven zweier Modelle. Die rote Kurve entspricht der ROC des $k_1(\underline{x})$ Klassifikators

zwei ROC-Kurven zweier Klassifikatoren dargestellt. Der Punkt τ_1 entspricht hierbei dem Paar $(1, 1)$, wobei der Threshold so hoch ist, dass alle Eingaben der Positiven Klasse zugeordnet werden, dementsprechend sind der Recall und die FPR gleich Eins. Analog dazu ist beim Punkt τ_4 mit den Werten $(0, 0)$ der Threshold so hoch, sodass alle Eingaben der negativen Klasse zugeordnet werden. Der Punkt τ_2 mit den Werten $(0.21, 1)$ verbessert die Klassifikationsleistung, da die FPR gesenkt wird während der

Recall konstant bleibt. Der Punkt τ_3 stellt einen Trade-off dar, da hierbei gegenüber τ_2 die FPR gesenkt wird, aber auf der anderen Seite auch der Recall sinkt. Insgesamt lässt sich anhand der ROC-Kurve einsehen, welche (FPR, Recall)-Paare ein Klassifikator $k_m(\underline{x})$ erreichen kann. Wird die Kurve in Orange in Abbildung 4.3.3 betrachtet, kann festgestellt werden, dass für jeden Punkt auf der Kurve, ein entsprechender Punkt auf der roten Kurve liegt, der bessere Werte hat. Somit kann beim Vergleich von Klassifikatoren gesagt werden, dass der Klassifikator besser ist, dessen Kurve über der Kurve des anderen Klassifikators liegt. Daher wird als Gütemaß die Fläche unter der ROC-Kurve verwendet. Dieser Wert wird *Area under the Receiver Operating Characteristic Curve* (kurz *AuROC*)[HM82] genannt und bewegt sich im Wertebereich $[0, 1]$. Sie beschreibt die zusammengefasste Performanz eines thresholdabhängigen, binären Klassifikators mit einem skalaren Wert und kann als die Wahrscheinlichkeit interpretiert werden, dass das Modell für eine positive Eingabe einen höheren Score ausgibt, als für eine negative Eingabe. Als Referenzwert gilt der AuROC Wert von 0.5, der von einem Klassifikator erreicht wird, der für die Klassifikation einen zufälligen Wert nutzt [HG16]. Die entsprechende ROC-Kurve, wäre in dem Falle eine Gerade, welche durch die Punkte $(0, 0)$ und $(1, 1)$ geht. Alle Werte über und unter 0.5 sind besser als der Zufalls-Klassifikator. Ein idealer Klassifikator erreicht einen AuROC von Eins beziehungsweise Null, je nach dem, welche Klasse als die positive Klasse definiert ist.
Ein Problem, welches das AuROC-Maß mit sich bringt, ist der Einfluss der Klassenverteilungen. Ist eine Klasse in der Stichprobe häufiger Vertreten, kann das Ergebnis verzerrt werden. Dies liegt daran, dass der Recall und die FPR, wie in 4.3.4 und 4.3.5 zu sehen ist, durch eine Division mit der Anzahl an positiven beziehungsweise negativen Stichprobenelementen berechnet werden.
Daher wird in [HG16] vorgeschlagen das *Area under the Precision Recall*-Maß (kurz *AuPR*) [SR15] als zusätzliche Evaluationsmetrik für binäre thresholdabhängige Klassifikatoren zu nutzen. Diese wird wie das AuROC-Maß berechnet, wobei die Kurve mittels (Recall, Precision)-Tupeln gebildet wird. Den Referenzwert, also die Baseline, für dieses Maß stellt hierbei das Verhältnis der Anzahl positiver Stichprobenelemente zum Stichprobenumfang $\frac{|\{^{\rho}\underline{x} \mid {}^{\rho}\underline{y}=\text{Positiv}\}|}{|\omega|}$ dar. Dieser AuPR-Wert würde durch den Zufalls-Klassifikator erreicht werden. Die Precision eines Klassifikators wird entsprechend 4.3.2 durch eine Division mit $|\{^{\rho}\underline{x} \mid {}^{\rho}\underline{\hat{y}} = \text{Positiv}\}|$ berechnet und ist somit robust gegenüber asymmetrischen Klassenverteilungen. Dadurch, dass das AuPR-Maß auf Grundlage der Precision berechnet wird, übernimmt es diese Eigenschaft [SR15].

5 EXPERIMENTE

Das Ziel der Experimente, ist es, die Leistungen der hybriden Modelle, welche in Kapitel 4.2.2 vorgestellt wurden, zu untersuchen, diese untereinander und mit den nicht-hybriden bayes'schen/klassischen Modellen zu vergleichen. Der Vergleich hybrider Netze untereinander soll Aufschluss darüber geben, in wie weit sich die Auswahl und die Position der bayes'schen Gewichte auf die Leistungen auswirkt. Des weiteren soll untersucht werden, in wie weit sich die Anzahl der Samples S zur Erwartungswertbildung auf die untersuchten Leistungen auswirken. Die Leistungen der Modelle sollen anhand der in Kapitel 4.3 präsentierten Methoden quantitativ gemessen werden. Die reine Klassifikationsleistung wird anhand der Accuracy und die Out-of-Distribution Erkennungsleistung anhand von AUROC/AUPR bewertet. Dabei gibt es zwei Experimente.
Im ersten Experiment wird das zwei-schichtige neuronale Netz aus [HG16] mit dem MNIST [MNI] Datensatz verwendet, um einen Vergleich mit den dort erzielten Ergebnissen zu bieten. Dabei werden verschiedene Datensätze als Quelle für Out-of-Distribution Eingaben verwendet. Abschließend werden die Ergebnisse der Verschiedenen Modelle verglichen und diese den Ergebnissen aus [HG16] gegenüber gestellt. Im zweiten Experiment wird ein komplexeres neuronales Netz mit *ResNet18*-Architektur [HZRS16] mit dem *CIFAR10* [Kri09] Datensatz verwendet. Hierbei werden zum Training der Modelle einige Klassen ausgeschlossen, welche als Out-of-Distribution Eingaben dienen sollen. Abschließend werden die hybriden Modelle bewertet und es wird untersucht, in wie weit sich die Anzahl der Samples S auf die Leistungen der Modelle auswirkt. Des weiteren wird untersucht in wie weit die Klassifikationsleistung der Modelle Auswirkungen auf das Erkennen von Out-of-Distribution Eingaben haben.

5.1 EINFACHES NEURONALES NETZ

Im ersten Experiment wird ein zwei-schichtiges neuronales Netz $\mathrm{NN}(784, 256, 10)$ mit der GELU-Aktivierungsfunktion [HG20] in der versteckten Schicht und der Softmax-Aktivierung in der Ausgabeschicht genutzt, welche dem Aufbau des Modells in [HG16] entspricht. Da es sich um ein kleines Modell handelt, werden alle möglichen korrespondierenden schichtweise hybriden bayes'schen neuronalen Netze $\mathrm{BNN}_{00}(784, 256, 10)$,

$BNN_{01}(784, 256, 10)$, $BNN_{10}(784, 256, 10)$ und $BNN_{11}(784, 256, 10)$ betrachtet, wobei die Erwartungswertbildung mit $S = 10$ Samples approximiert wird. Als Variational Posterior wird eine diagonale Normalverteilung gewählt. Als Prior $P(\Theta_W)$ wird gemäß der Empfehlung in [BCKW15] ein *Spike-and-slab-Prior* genutzt, welcher eine Mischverteilung bestehend aus zwei Normalverteilungen ist. Dieser ist wie folgt definiert.

$$P(\Theta_W) = \prod_{w_{ij} \in \Theta_W} 0.75 \cdot \mathcal{N}(w_{ij}|\mu = 0, \sigma^2 = 0.1^2) + 0.25 \cdot \mathcal{N}(w_{ij}|\mu = 0, \sigma^2 = 0.0005^2) \tag{5.1.1}$$

Das erlernen der Parameter erfolgt mittels ADAM-Optimierer[ADA] mit einer Learningrate von 0.0001 über 30 Epochen[1]. Die Modelle werden mit dem MNIST-Datensatz trainiert. Dieser ist eine frei verfügbare Sammlung von handschriftlichen Ziffern. In Abbildung A.0.2 sind exemplarisch einige Stichprobenelemente visualisiert. Hierbei besteht der Datensatz aus 70000 Graustufenbildern mit einer Auflösung von 28×28 Pixeln, wobei 60000 Exemplare zum Training der Modelle vorgesehen sind und 10000 zum Validieren. Dabei sind die Ziffern auf den Bildern zentriert und größennormalisiert, sodass die Vorverarbeitung erleichtert wird. Die Häufigkeiten der Ziffern sind

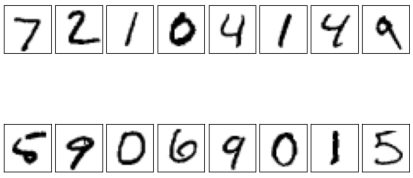


Abbildung 5.1.1: Ein Ausschnitt aus dem MNIST-Datensatz

nicht vollständig gleichverteilt und die Ziffern treten im Datensatz mit den folgenden Häufigkeiten auf.

Ziffer	0	1	2	3	4	5	6	7	8	9
Häufigkeit	6903	7877	6990	7141	6824	6313	6876	7293	6825	6958

Abbildung 5.1.2: Häufigkeiten der einzelnen Ziffern im MNIST-Datensatz

Zur Untersuchung der Out-of-Distribution Erkennung, wurden zwei Datensätze, deren

1 Wobei in den 30 Epochen, das Modell mit der höchsten Accuracy auf den Validierungsdaten zur Evaluation ausgewählt wird.

Inhalte den Daten aus MNIST ähneln und zwei Datensätze, deren Inhalte den Daten aus MNIST nicht ähneln, ausgewählt. Die MNIST-ähnlichen Daten sind der EMNIST-Datensatz[EMN] und der KMNIST-Datensatz [KMN]. Der EMNIST-Datensatz ist eine Erweiterung des MNIST-Datensatzes um handgeschriebene Buchstaben. In den Experimenten werden daher nur die Buchstaben aus dem Datensatz als Out-of-Distribution Eingaben verwendet. Der KMNIST-Datensatz besteht aus kalligraphisch geschriebenen Zeichen aus dem japanischen und bietet, wie der MNIST-Datensatz, Zehn Klassen. Die MNIST-Unähnlichen Datensätze sind Faschion-MNIST[FMN] und CIFAR10. Fashion-MNIST ist ein an MNIST angelehnter Datensatz der Firma Modefirma Zalando, welche zentrierte und größennormalisierte Graustufenbilder von Kleidungsstücken beinhaltet. Der CIFAR10 Datensatz wird in Abschnitt 5.2 dargestellt. Bis auf den CIFAR10 Datensatz entspricht das Datenformat der Datensätze dem Datenformat von MNIST. Das heißt alle Daten sind 28×28 Pixel große Graustufenbilder. Die Bilder des CIFAR10 Datensatzes werden daher zu Graustufenbildern transformiert und es wird ein 28×28 Pixel großer zufälliger Ausschnitt verwendet. Des Weiteren wird als letzte Out-of-Distribution Eingabemenge ein Datensatz aus zufällig generierten Bilder verwendet, dessen Pixel zufällig aus einer Gleichverteilung gesampled wurden. Dieser Datensatz wird *RandomUniform* genannt
Das Erkennen von Out-of-Distribution Eingaben erfolgt beim klassischen Modell $\mathrm{NN}(784, 256, 10) = \mathrm{BNN}_{00}(784, 256, 10)$ mittels Softmax-Scores der vorhergesagten Klassen [HG16]. Dabei werden die Out-of-Distribution Eingaben als positive Klasse und die In-Distribution Eingaben als negative Klasse definiert. Liegt der Softmax-Score unter einem bestimmten Threshold, so wird die Eingabe als positiv Klassifiziert. Analog dazu wird bei dem vollständig bayes'schen Modell und den hybriden Modellen statt dem Softmax-Score, die epistemische Unsicherheit verwendet, um Out-of-Distribution Eingaben zu klassifizieren. Bei Betrachtung der Klassifikations-

Modell	BNN_{00}	BNN_{01}	BNN_{10}	BNN_{11}
Accuracy	97.740%	98.310%	97.900%	97.670%

Abbildung 5.1.3: Klassifikationsleistung der Modelle auf den Validierungsdaten

leistungen in Abbildung 5.1.3 fällt auf, dass das Modell BNN_{01} die besten Ergebnisse erzielt. Am schlechtesten fällt die Leistung des vollbayes'schen Modells aus und die hybriden Modelle schneiden am besten ab. Jedoch ist zu erwähnen, dass die Modelle sehr nah beieinander liegen und keine signifikanten Unterschiede vorliegen. Bei der Erkennungsleistung von Out-of-Distribution Eingaben ist ein signifikanter Unterschied

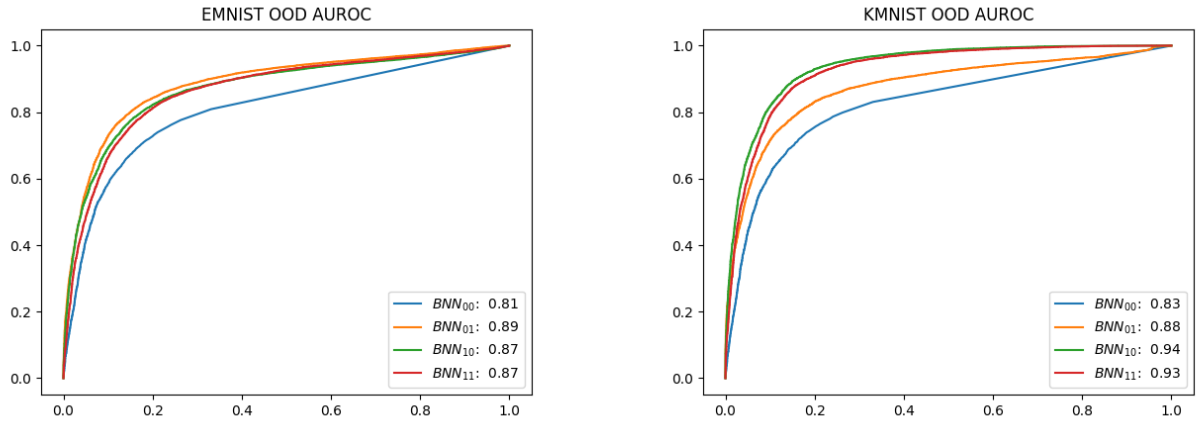


Abbildung 5.1.4: ROC-Kurven der Out-of-Distribution Erkennung. Die Legende gibt das AUROC-Maß unter den Kurven an. X-Achse entspricht der FPR und die Y-Achse dem Recall.

in Abbildung 5.1.4 zu erkennen[2]. Hierbei schlagen die hybriden Modelle deutlich das klassische Modell und sind teilweise besser als das vollbayes'sche Modell.

AUROC/AUPR	BNN_{00}	BNN_{01}	BNN_{10}	BNN_{11}
EMNIST	0.81 / 0.88	**0.89** / **0.94**	0.87 / 0.93	0.87 / 0.93
KMNIST	0.83 / 0.81	0.88 / 0.89	**0.94** / **0.93**	0.93 / 0.91
FashionMNIST	0.79 / 0.80	0.72 / 0.78	**0.92** / **0.90**	0.82 / 0.80
CIFAR10	0.77 / 0.77	0.83 / 0.87	**0.96** / **0.96**	0.92 / 0.91
RandomUniform	0.86 / 0.84	0.98 / 0.98	**0.99** / **0.99**	0.91 / 0.88
Durchschnitt	0.81 / 0.82	0.86 / 0.89	0.94 / 0.94	0.89 / 0.89

Abbildung 5.1.5: Out-of-Distribution Erkennungsleistung gemessen mit AUROC/AUPR

Wird die Tabelle in Abbildung 5.1.5 betrachtet, ist zu sehen, dass die bayes'schen Modelle insgesamt eine höhere Out-of-Distribution Erkennungsleistung aufweisen als das klassische Modell. Hierbei sei zu erwähnen, dass bei der Bildung der AUROC- und AUPR-Werte auf balancierte Eingabemengen geachtet wurde, sodass die Baseline

2 Die Plots zu den restlichen Datensätzen und die Plots zu AUPR-Darstellung könn en im Anhang A eingesehen werden

für das AUPR-Maß 0.5 ist. Das Modell BNN_{10}, mit den bayes'schen Gewichten in der ersten Schicht, erzielt sogar eine signifikant bessere Leistung als das Modell BNN_{01} mit bayes'schen Gewichten in der letzten Schicht. Die Begründung hierfür könnte jedoch, statt in der Position der bayes'schen Schichten, viel mehr darin liegen, dass $\mathbf{W}^{(1)} \in \mathbb{R}^{768\times256}$ gegenüber $\mathbf{W}^{(2)} \in \mathbb{R}^{256\times10}$ mehr Parameter besitzt und daher mehr Gewichte bayes'sch sind. Dieser Begründung widerspricht jedoch der Tatsache, dass BNN_{10} auch durchschnittlich besser ist als das vollbayes'sche Modell BNN_{11}, welches mehr bayes'sche Parameter hat. Insgesamt lässt sich der Rechenaufwand mit

Modell	BNN_{00}	BNN_{01}	BNN_{10}	BNN_{11}
FLOPs	407 918	457 540	4 279 884	4 282 444
Rel. BNN_{00}	1	1.12	10.49	10.50

Abbildung 5.1.6: Rechenaufwand in FLOPs. Die Zeile *Rel.* BNN_{00} gibt den Mehraufwand relativ zu BNN_{00} an.

dem besten Modell BNN_{10} nur minimal gegenüber BNN_{11} reduzieren. Denn beide benötigen ca. 10.5 mal mehr Rechenaufwand als BNN_{00}, welches Tabelle 5.1.6 zu entnehmen ist. BNN_{01} hingegen benötigt nur 1.12 mal mehr Rechenaufwand gegenüber dem klassischen Modell, hat jedoch hingegen eine überlegene Out-of-Distribution Erkennungsleistung bei vergleichbarer Klassifikationsleistung.

5.2 RESNET-18

Im zweiten Experiment wird das ResNet-18 Modell[HZRS16] untersucht, welches ein tiefes Netz, bestehend aus vielen Faltungsschichten und *Skip-Connections*, ist. Skip-Connections sind Verbindungen zwischen Zwei nicht aufeinander folgenden Schichten in Richtung Ausgabe. Hierbei besteht das ResNet18 aus den Schichten *conv1*, *conv2_x*, *conv3_x*, *conv4_x*, *conv5_x*, *1000-d fc*. Dabei ist conv1 eine Faltungsschicht mit 64 Kerneln der Dimension 7x7 mit Stride gleich zwei. Die Schicht conv2_x besteht dabei aus einer 3×3 max-pool Schicht mit Stride gleich Zwei und Vier aufeinander Folgenden Faltungen mit jeweils 64 Kernel der Größe 64. Die letzte Schicht ist eine vollvernetzte Schicht $\text{NN}(1000, K)$, wobei K die Anzahl der Klassen ist. Der Aufbau der restlichen Schichten kann der Tabelle in Abbildung 5.2.1 aus der Spalte *18-Layer* entnommen werden. In den Experimenten wird daher für die hybriden Modelle die Bitvektornotation ResNet_b benutzt werden, wobei b kodiert, welche der Schichten *conv1*, *conv2_x*, *conv3_x*, *conv4_x*, *conv5_x* und *1000-d fc* bayes'sch ist, wobei das niedrigstwertige Bit *1000-d fc* kodiert und das hörstwertigste Bit *conv1*. Als Variational Posterior wird

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Abbildung 5.2.1: Struktur der ResNet Modelle. Tabelle entnommen aus [HZRS16]

wieder eine diagonale Normalverteilung gewählt. Der genutzte Prior entspricht dem Prior aus dem ersten Experiment. Das erlernen der Parameter erfolgt mittels ADAM-Optimierer[ADA] mit einer Learningrate von 0.0001 über 40 Epochen[3]. Die Modelle werden mit dem CIFAR10-Datensatz trainiert. Der frei zugängliche CIFAR10-Datensatz

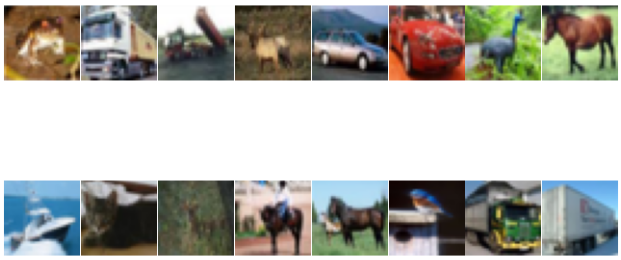


Abbildung 5.2.2: Ein Ausschnitt aus dem CIFAR10-Datensatz

besteht aus 70000 farbigen Bildern mit der Auflösung 32×32 Pixeln, von denen ein Ausschnitt in Abbildung 5.2.2 zu sehen ist. Dabei sind Zehn Klassen gegeben: Flugzeug, PKW, Vogel, Katze, Hirsch, Hund, Frosch, Pferd, Schiff und LKW. Bei der Auswahl der Bilder für den Datensatz wurde zudem darauf geachtet, die Klassen möglichst disjunkt zu halten. Das heißt, dass beispielsweise in der Klasse der Autos und LKW,

3 Wobei in den 40 Epochen, das Modell mit der höchsten Accuracy auf den Validierungsdaten zur Evaluation ausgewählt wird.

keine Bilder von Fahrzeugen enthalten sind, welche in beide Kategorien fallen könnten, wie zum Beispiel ein Transporter. Zu jeder Klasse sind 7000 Bilder gegeben, wobei 6000 zum Erlernen der Modellparameter genutzt werden und 1000 zum validieren. Damit sind die Klassen gleichverteilt. Beim trainieren der Netze wurden zufällige 32×32 Pixel große Ausschnitte aus den 4 Pixel Zero-gepaddeten Ursprungsbildern verwendet, welche mit einer Wahrscheinlichkeit von 0.5 horizontal gespiegelt wurden, um eine bessere Generalisierung zu erzielen.

Bei der Untersuchung der Out-of-Distribution-Erkennungsleistung in den Experimenten wird nicht der gesamte Datensatz zum erlernen der Modellparameter genutzt. Es werden jeweils zwei Klassen ausgeschlossen, welche als Out-of-Distribution Eingaben dienen sollen. Um die Messungen der Leistungen aufgrund der fehlenden Klassen in den Trainingsdaten nicht zu verfälschen beziehungsweise von den fehlenden Klassen abhängig zu machen, werden zufällig 8 mal zwei Klassen ausgeschlossen und am Ende die durchschnittlichen Leistungen der Modelle verglichen. Es werden insgesamt 9 Architekturen untersucht. Diese sind ResNet_{000000}, ResNet_{010000}, ResNet_{000001}, ResNet_{001110}, ResNet_{000100}, ResNet_{010101}, ResNet_{001010}, ResNet_{000111} und ResNet_{111111}. Insgesamt wurden daher $8 \cdot 9 = 72$ Modelle betrachtet, mit je 8 Modellen pro Architektur. Bei der Betrachtung der Leistungen des Modells fällt auf, dass alle Modelle

Modell	Accuracy	AUROC OOD	AUPR OOD
ResNet_{000000}	$\mathbf{0.843 \pm 0.014}$	0.650 ± 0.016	0.282 ± 0.010
ResNet_{000001}	0.837 ± 0.013	0.701 ± 0.013	0.388 ± 0.022
ResNet_{000100}	0.842 ± 0.012	0.748 ± 0.021	0.516 ± 0.039
ResNet_{000111}	0.840 ± 0.011	0.726 ± 0.018	0.434 ± 0.037
ResNet_{001010}	0.835 ± 0.013	$\mathbf{0.761 \pm 0.016}$	0.520 ± 0.036
ResNet_{001110}	0.835 ± 0.012	0.756 ± 0.020	0.505 ± 0.040
ResNet_{010000}	0.839 ± 0.011	0.758 ± 0.023	$\mathbf{0.534 \pm 0.036}$
ResNet_{010101}	0.829 ± 0.016	0.736 ± 0.015	0.485 ± 0.022
ResNet_{111111}	0.818 ± 0.015	0.747 ± 0.017	0.470 ± 0.035

Abbildung 5.2.3: Leistungen der Modelle. Die Accuracy wurde auf den Validierungsdaten berechnet und AUROC/AUPR anhand der ausgeschlossenen Klassen. Die AUPR Baseline ist 0.2, da die OOD-Klassen unbalanciert sind

eine ähnliche durchschnittliche Klassifikationsleistung aufweisen, wobei das klassische Modell knapp die beste Klassifikationsleistung erzielt. Werden die AUROC-

und AUPR-Werte bei der Out-Of-Distribution-Eingaben-Klassifikation betrachtet, so fällt auf, dass alle Modelle die AUROC-Baseline von 0.5 und die AUPR-Baseline von 0.2 schlagen. Insbesondere weisen alle hybriden Modelle eine deutlich höhere Out-Of-Distribution-Eingaben-Erkennungsleistung auf, als das Klassische Modell ResNet_{000000}. Die beste Out-Of-Distribution-Eingaben-Erkennungsleistung weisen ResNet_{001010} und ResNet_{010000} auf. Diese haben sogar eine bessere Leistung als das vollbayes'sche Modell ResNet_{111111}. In Abbildung 5.2.4 ist die Leistung der hybriden Modelle in Abhängigkeit von der Anzahl an Samples, welche zur Erwartungswertbildung gemäß 2.6.6 genutzt werden. Dort ist zu erkennen, dass selbst für die Sampleanzahl $S = 1$ die Klassifikationsleistung nicht signifikant sinkt. So bleibt die Klassifikationsleitung bei den hybriden Modellen nahezu Konstant. Die Erkennungsleistung durch epistemische Unsicherheit ist bei den $S = 1$ auf dem Minimum der Messwerte und erhöht sich Sprunghaft für $S = 2$. So wächst der AUROC Wert nur sehr Langsam mit der Anzahl an Samples. Zwischen $S = 2$ und $S = 10$ beträgt die durchschnittliche Differenz des AUROC-Wertes nur 0.033 und die durchschnittliche Differenz des AUPR-Wertes 0.091. Da alle Modelle im Durchschnitt einen um mindestens 0.05 höheren AUROC-Wert und einen um mindestens 0.1 höheren AUPR-Wert gegenüber dem klassischen Modell haben, schlagen die hybriden Modelle sogar bei einer Samplinganzahl von $S = 2$ das klassische Modell. So bringen die hybriden Modelle bei maximal doppeltem Rechenaufwand deutlich bessere Out-of-Distribution Eingaben Erkennung. Bei der Wahl von ResNet_{000001} ist dieser Mehraufwand minimal, da nur ca. 10% der Gewichte bayes'sch sind und das Modell getrennt hybrid ist.

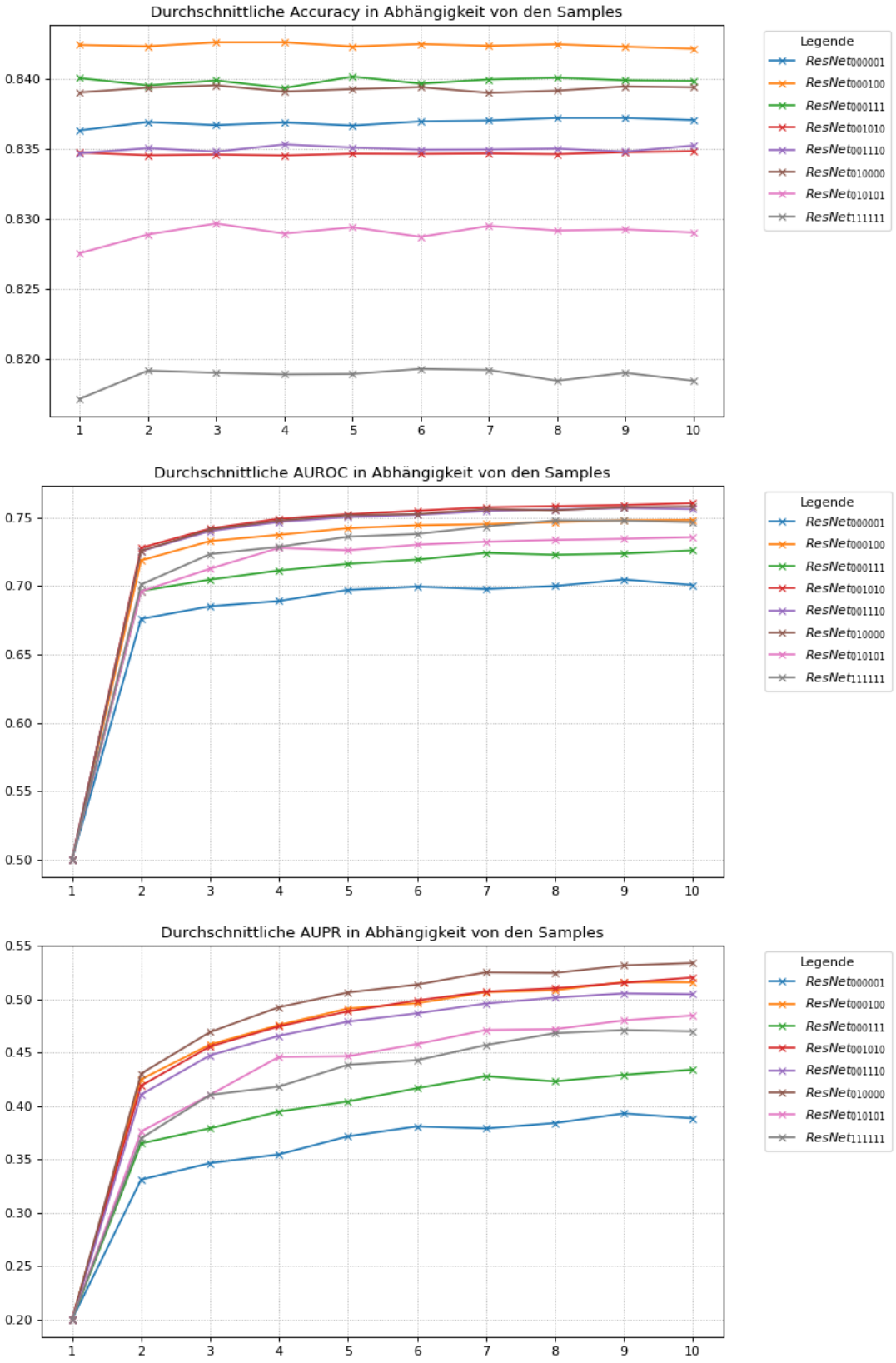


Abbildung 5.2.4: Die Durchschnittlichen Leistungen der Modelle in Abhängigkeit von der Anzahl an Samples, welche zur Erwartungswertbildung gemäß 2.6.6 genutzt werden.

6 FAZIT

In dieser Arbeit wurden die Rechenaufwändigen bayes'schen neuronalen Netze untersucht, deren Rechenaufwand zum einen aus dem Sampling der Parameter und zum anderen aus der Erwartungswertbildung bestand. Die Reduktion des Rechenaufwands wurde anhand der Reduktion der bayes'schen Gewichte sowie, der Positionierung dieser und der Reduktion des Samplings bei der Erwartungswertbildung vorgenommen. Dabei wurde experimentell Untersucht, in wie weit sich diese ändernden Faktoren auf die Klassifikationsleistungen auswirken. Hierbei wurde festgestellt, dass die hybriden Netze bei der Erkennung von Out-of-Distribution Eingaben, die von [HG16] vorgeschlagene Baseline schlagen und dabei teilweise bessere Leistungen erzielen als ihre vollbayes'schen Gegenstücke und dabei ihre Klassifikationsleistung beibehalten. Jedoch schwanken die Resultate unter den hybriden Netzen und es konnte kein Muster erkannt werden, welches angibt wie die bayes'schen Gewichte zu positionieren sind. Jedoch erzielen die hybriden Modelle in der Out-of-Distribution-Eingaben-Erkennung bessere Ergebnisse als die nicht-bayes'schen Modelle, selbst wenn nur eine Schicht bayes'sch ist. In Abschnitt 5.1 wurde gezeigt, dass mit 1,12 mal Mehraufwand, das Modell verbessert werden kann. In Abschnitt 5.2 wurde dabei auch gezeigt, dass die Erwartungswertbildung mit nur zwei Samples bei der Erwartungswertbildung brauchbare Ergebnisse liefert. Insgesamt lässt sich das Fazit ziehen, dass durch hybride Modelle und eine kleine Samplinganzahl bayes'sche neuronale Netze deutlich effizienter werden, ohne dabei signifikant an Leistung zu verlieren, sodass bayes'sche Modelle in der Praxis relevanter werden könnten. Dabei könnte in Zukunft untersucht werden, in wie weit sich die Auswahl des Variational Posteriors und des Priors auf die Leistungen der Modelle auswirken. Des weiteren könnten weitere komplexere Modelle untersucht werden.

A

PLOTS

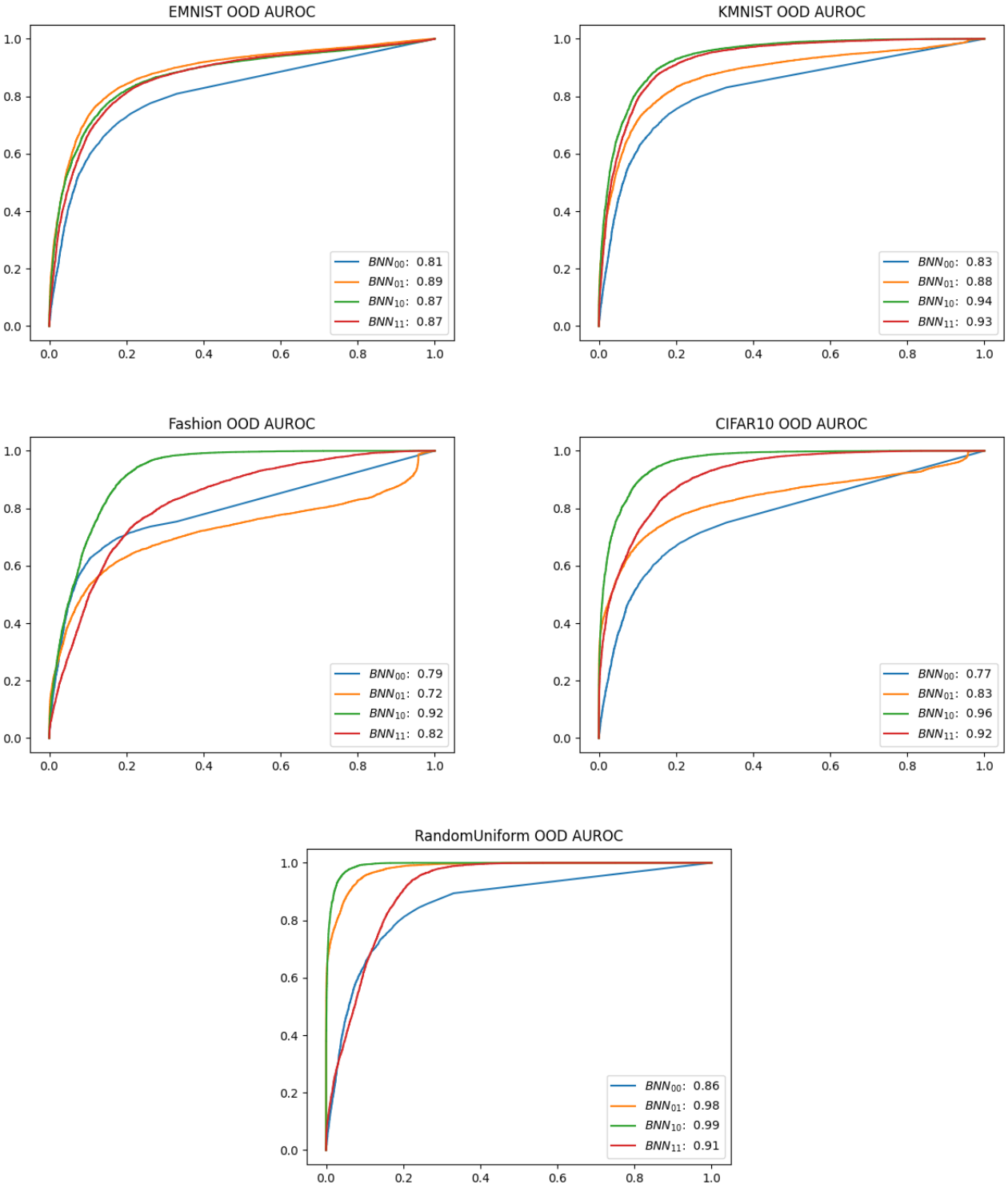


Abbildung A.0.1: AUROC Kurven der Modelle in der Out-of-Distribution Erkennung. Die Legende gibt die Fläche unter den Kurven an. X-Achse entspricht der FPR und die Y-Achse dem Recall.

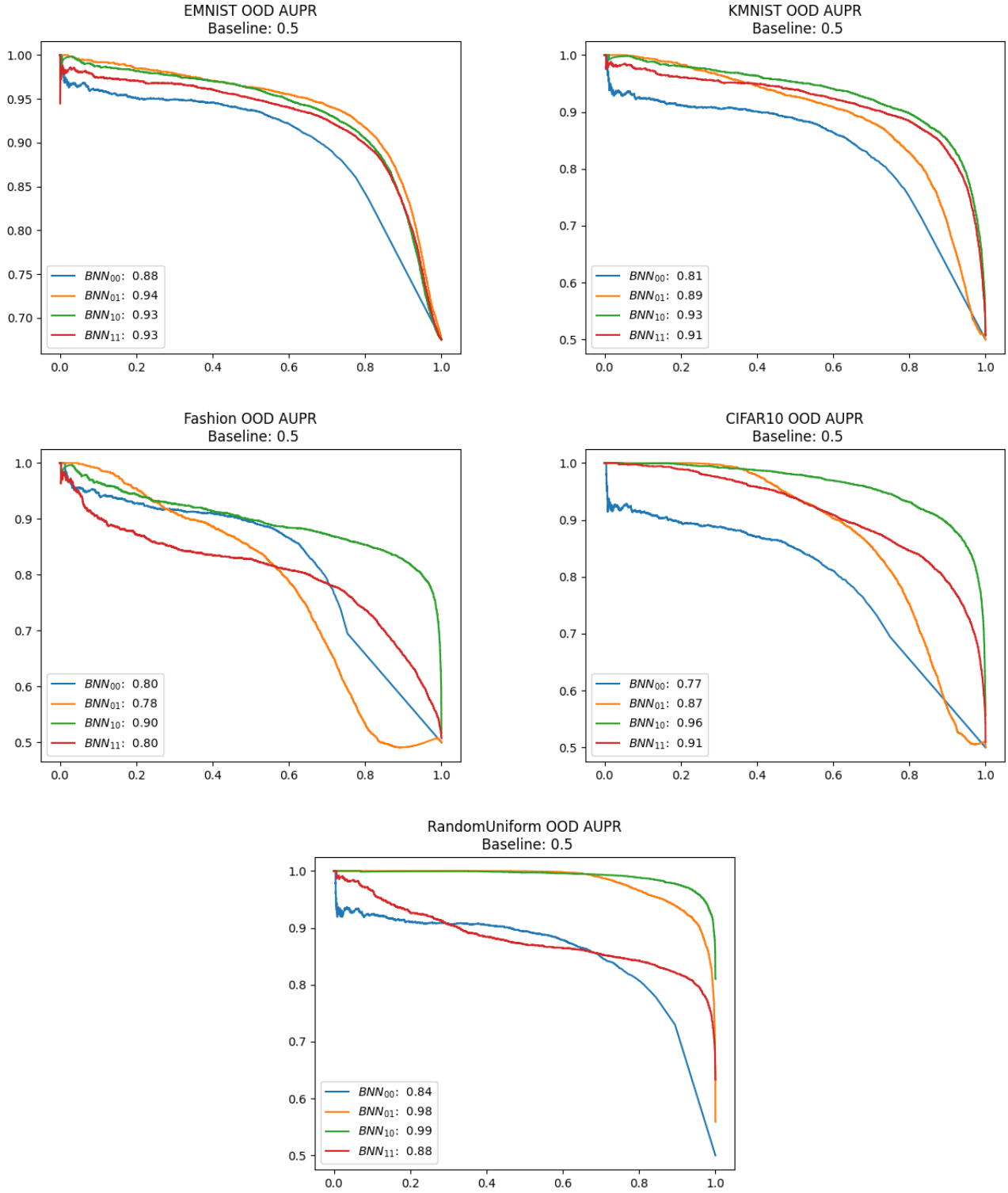


Abbildung A.0.2: AUPR Kurven der Modelle in der Out-of-Distribution Erkennung. Die Legende gibt die Fläche unter den Kurven an. X-Achse entspricht dem Recall und die Y-Achse der Precision.

LITERATURVERZEICHNIS

[ADA] *Pytorch ADAM*. (Zugriff: 30.09.2021). https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam

[Aka73] In: Akaike, Hirotogu: *Information Theory and an Extension of the Maximum Likelihood Principle*. New York, NY : Springer New York, 1973, S. 199–213

[BCKW15] Blundell, Charles ; Cornebise, Julien ; Kavukcuoglu, Koray ; Wierstra, Daan: Weight Uncertainty in Neural Networks. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, JMLR.org, 2015 (ICML'15), S. 1613–1622

[Bis06] Bishop, Christopher M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg : Springer-Verlag, 2006. – ISBN 0387310738

[BJ20] Beheshti, Nazanin ; Johnsson, Lennart: Squeeze U-Net: A Memory and Energy Efficient Image Segmentation Network. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, Juni 2020

[Bot91] Bottou, L.: Stochastic Gradient Learning in Neural Networks, 1991

[BSF94] Bengio, Y. ; Simard, P. ; Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. In: *IEEE Transactions on Neural Networks* 5 (1994), Nr. 2, S. 157–166. http://dx.doi.org/10.1109/72.279181. – DOI 10.1109/72.279181

[Cha21] Chang, Daniel T.: *Hybrid Bayesian Neural Networks with Functional Probabilistic Layers*. 2021

[CMM+18] Chui, Michael ; Manyika, James ; Miremadi, Mehdi ; Henke, Nicolaus ; Chung, Rita ; Nel, Pieter ; Malhotra, Sankalp: *Notes from the AI frontier: Insights From Hundrets of Use Cases*. April 2018 https://www.mckinsey.com/~/media/McKinsey/Featured%20Insights/Artificial%

20Intelligence/Notes%20from%20the%20AI%20frontier%20Applications%20and%20value%20of%20deep%20learning/Notes-from-the-AI-frontier-Insights-from-hundreds-of-use-cases-Discussion-pdf

[DHLDVU18] Depeweg, Stefan ; Hernández-Lobato, José M. ; Doshi-Velez, Finale ; Udluft, Steffen: *Decomposition of Uncertainty in Bayesian Deep Learning for Efficient and Risk-sensitive Learning*. 2018

[EMN] *EMNIST: an extension of MNIST to handwritten letters*. (Zugriff: 30.09.2021). https://www.westernsydney.edu.au/icns/reproducible_research/publication_support_materials/emnist

[Fle00] In: Fletcher, R.: *Structure of Methods*. John Wiley & Sons, Ltd, 2000. – ISBN 9781118723203

[FMN] *Fashion-MNIST*. (Zugriff: 30.09.2021). https://github.com/zalandoresearch/fashion-mnist

[GBC16] Goodfellow, Ian J. ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. Cambridge, MA, USA : MIT Press, 2016. – 109ff S. – http://www.deeplearningbook.org

[GPSW17] Guo, Chuan ; Pleiss, Geoff ; Sun, Yu ; Weinberger, Kilian Q.: *On Calibration of Modern Neural Networks*. 2017

[HDG21] Henning, Christian ; D'Angelo, Francesco ; Grewe, Benjamin F.: *Are Bayesian neural networks intrinsically good at out-of-distribution detection?* 2021

[Hei27] Heisenberg, W.: über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. In: *Zeitschrift für Physik* 43 (1927), März, Nr. 3-4, 172–198. http://dx.doi.org/10.1007/bf01397280. – DOI 10.1007/bf01397280

[HG16] Hendrycks, Dan ; Gimpel, Kevin: A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In: *CoRR* abs/1610.02136 (2016). http://arxiv.org/abs/1610.02136

[HG20] Hendrycks, Dan ; Gimpel, Kevin: *Gaussian Error Linear Units (GELUs)*. 2020

[HKH17] Hong, Seunghoon ; Kwak, Suha ; Han, Bohyung: Weakly Supervised Learning with Deep Convolutional Neural Networks for Semantic Segmentation: Understanding Semantic Layout of Images with Minimum Human Supervision. In: *IEEE Signal Processing Magazine* 34 (2017), nov, Nr. 6, S. 39–49

[HM82] Hanley, J A. ; McNeil, B J.: The meaning and use of the area under a receiver operating characteristic (ROC) curve. In: *Radiology* 143 (1982), April, Nr. 1, 29–36. http://dx.doi.org/10.1148/radiology.143.1.7063747. – DOI 10.1148/radiology.143.1.7063747

[HSW89] Hornik, Kurt ; Stinchcombe, Maxwell ; White, Halbert: Multilayer feedforward networks are universal approximators. In: *Neural Networks* 2 (1989), Januar, Nr. 5, 359–366. http://dx.doi.org/10.1016/0893-6080(89)90020-8. – DOI 10.1016/0893–6080(89)90020–8

[HTF09] Hastie, Trevor ; Tibshirani, Robert ; Friedman, Jerome: *The elements of statistical learning: data mining, inference and prediction*. 2. Springer, 2009 http://www-stat.stanford.edu/~tibs/ElemStatLearn/

[HZRS15] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, Dezember 2015

[HZRS16] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Juni 2016

[JK16] Jonas, Eric ; Kording, Konrad P.: Could a Neuroscientist Understand a Microprocessor? (2016), Mai. http://dx.doi.org/10.1101/055624. – DOI 10.1101/055624

[KG17] Kendall, Alex ; Gal, Yarin: What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In: *CoRR* abs/1703.04977 (2017). http://arxiv.org/abs/1703.04977

[KH91] Krogh, Anders ; Hertz, John A.: A Simple Weight Decay Can Improve Generalization. In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1991 (NIPS'91). – ISBN 1558602224, S. 950–957

[KMN] *Kuzushiji-MNIST*. (Zugriff: 30.09.2021). https://github.com/rois-codh/kmnist

[Kri09] Krizhevsky, Alex: Learning Multiple Layers of Features from Tiny Images. (2009). https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[KSW15] Kingma, Durk P. ; Salimans, Tim ; Welling, Max: Variational Dropout and the Local Reparameterization Trick. In: Cortes, C. (Hrsg.) ; Lawrence, N. (Hrsg.) ; Lee, D. (Hrsg.) ; Sugiyama, M. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 28, Curran Associates, Inc., 2015

[Kul59] Kullback, Solomon: *Information Theory and Statistics*. New York : Wiley, 1959

[MNI] *THE MNIST DATABASE*. (Zugriff: 30.09.2021). http://yann.lecun.com/exdb/mnist/

[Mus19] Musk, Elon: An integrated brain-machine interface platform with thousands of channels. (2019), jul

[Nie13] Niemann, H.: *Klassifikation von Mustern*. Springer Berlin Heidelberg, 2013. – ISBN 9783642475177

[PMB13] Pascanu, Razvan ; Mikolov, Tomas ; Bengio, Yoshua: On the difficulty of training recurrent neural networks. In: Dasgupta, Sanjoy (Hrsg.) ; McAllester, David (Hrsg.): *Proceedings of the 30th International Conference on Machine Learning* Bd. 28. Atlanta, Georgia, USA : PMLR, 17–19 Jun 2013 (Proceedings of Machine Learning Research 3), 1310–1318

[QK20] Qian, Xin ; Klabjan, Diego: *The Impact of the Mini-batch Size on the Variance of Gradients in Stochastic Gradient Descent*. 2020

[RFB15] Ronneberger, Olaf ; Fischer, Philipp ; Brox, Thomas: *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015

[RHW86] Rumelhart, David E. ; Hinton, Geoffrey E. ; Williams, Ronald J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), Oktober, Nr. 6088, 533–536. http://dx.doi.org/10.1038/323533a0. – DOI 10.1038/323533a0

[Ros57] ROSENBLATT, F.: The perceptron - A perceiving and recognizing automaton / Cornell Aeronautical Laboratory. Ithaca, New York, January 1957 (85-460-1). – Forschungsbericht

[RTX] *NVIDIA GeForce RTX 3090*. (Zugriff: 27.09.2021). https://www.techpowerup.com/gpu-specs/geforce-rtx-3090.c3622

[Rud17] RUDER, Sebastian: *An overview of gradient descent optimization algorithms*. 2017

[Sha48] SHANNON, C. E.: A mathematical theory of communication. In: *The Bell System Technical Journal* 27 (1948), Nr. 3, S. 379–423. http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x. – DOI 10.1002/j.1538–7305.1948.tb01338.x

[SL09] SOKOLOVA, Marina ; LAPALME, Guy: A systematic analysis of performance measures for classification tasks. In: *Information Processing & Management* 45 (2009), Juli, Nr. 4, 427–437. http://dx.doi.org/10.1016/j.ipm.2009.03.002. – DOI 10.1016/j.ipm.2009.03.002

[SR15] SAITO, Takaya ; REHMSMEIER, Marc: The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. In: *PLOS ONE* 10 (2015), März, Nr. 3, e0118432. http://dx.doi.org/10.1371/journal.pone.0118432. – DOI 10.1371/journal.pone.0118432

[Unf] *Driver killed in self-driving car accident for first time*. (Zugriff: 12.09.2021). https://www.pbs.org/newshour/nation/driver-killed-in-self-driving-car-accident-for-first-time

[VDC21] VONO, Maxime ; DOBIGEON, Nicolas ; CHAINAIS, Pierre: *High-dimensional Gaussian sampling: a review and a unifying approach based on a stochastic proximal point algorithm*. 2021

Eidesstattliche Versicherung
(Affidavit)

Özdemir, Berat — 157935

Name, Vorname
(Last name, first name)

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

Hybride bayessche neuronale Netze

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Ort, Datum
(Place, date)

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Ort, Datum
(Place, date)

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.