Universität Konstanz
Fachbereich Mathematik und Statistik
&
Bundesamt für Sicherheit in der
Informationstechnik

Masterarbeit zum Thema:

Untersuchung & Entwicklung von Ansätzen zur Detektion von Poisoning-Angriffen

vorgelegt von

Lukas Schulth

unter der Betreuung von

Erstgutachter:
Herr Prof. Dr. Johannes Schropp
johannes.schropp@uni.kn

Herr Dr. Christian Berghoff
christian.berghoff@bsi.bund.de

Zweitgutachter:
Herr Prof. Dipl.-Ing. Markus Ullmann
markus.ullmann@bsi.bund.de

Herr Matthias Neu
matthias.neu@bsi.bund.de

1. Oktober 2021

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Inhaltsverzeichnis

Abkürzungsverzeichnis

AC	Activation-Clustering
AER	Angriffserfolgsrate
CNN	Convolutional Neural Network
ACC	Genauigkeit
GWD	Gromov-Wasserstein-Distanz
HC	Heatmap-Clustering
KI	Künstliche Intelligenz
LkPA	Label-konsistenter Poisoning-Angriff
LRP	Layer-wise Relevance Propagation
mAER	mittlere Angriffserfolgsrate
NN	Neuronales Netz
PA	Poisoning-Angriff
TNR	True Negative Rate
TPR	True Positive Rate

Kapitel 1

Einführung

Laut [SBG+21] befinden wir uns zurzeit in der dritten Hochphase der Künstlichen Intelligenz (KI). Diese ist auch in Deutschland angekommen. Es wird erwartet, dass mit Dienstleistungen und Produkten, die auf dem Einsatz von KI basieren, im Jahr 2025 Umsätze in Höhe von 488 Milliarden Euro generiert werden – damit würde ein Anteil von 13 Prozent am Bruttoinlandsprodukt erreicht werden.

Unternehmen, die bisher keine Methoden des Maschinellen Lernens als Teil eines KI-Systems verwenden, haben sich zumindest einmal die Frage gestellt, ob dadurch einzelne Arbeitsschritte erleichtert bzw. beschleunigt werden könnten und es damit zu einer Verbesserung der Produktivität kommen könnte.

Obwohl die Wahrnehmung verbreitet ist, dass Maschinelles Lernen als Teil von KI eine sehr neue Disziplin ist, reicht die zeitliche Entwicklung bis weit in das 20. Jahrhundert zurück. Das erste Neuronales Netz (NN) wurde 1943 von McCulloch & Pitts vorgestellt. Bis in die frühen 1960er Jahre war dies ein aktives Feld. Durch die Erfindung der Backpropagation erhielt das Maschinelle Lernen wieder große Aufmerksamkeit, da es nun möglich war, größere NNs zu konstruieren und trainieren, mit denen anschließend nicht-lineare Zusammenhänge in Daten gefunden werden konnten.
Aufgrund fehlender Rechenleistung und wenig verfügbarer gelabelter Daten verlor das Maschinelle Lernen wieder an Aufmerksamkeit. Die dritte Welle der KI, die bis heute anhält, geht auf drei bedeutende Entwicklungen um das Jahr 2010 zurück. Die Rechenkapazitäten vergößerten sich durch die Verwendung von Grafikkarten (GPUs) deutlich. Andererseits ermöglichte der Fokus auf Convolutional Neural Networks (CNNs) die sehr effiziente Analyse riesiger (Bild-)Datensätze. Als Durchbruch gilt hier die Vorestellung von

AlexNet [KSH12] im Rahmen der ImageNet-Challenge [DDS+09]. Als dritter Punkt ist die Verfügbarkeit riesiger gelabelter Datensätze über das Internet zu nennen.

Heute gibt es kaum noch einen Bereich, in dem Anwendungen auf Basis von KI keine Rolle spielen, sei es in der Produktion, Werbung, Kommunikation, Biometrie oder Automotive. Viele Unternehmen nutzen KI-Systeme, etwa um präzise Nachfrageprognosen anzustellen und das Kundenverhalten exakt vorherzusagen. Auf diese Weise lassen sich beispielsweise Logistikprozesse regional anpassen. Auch im Gesundheitswesen bedient man sich spezifischer KI-Tätigkeiten wie dem Anfertigen von Prognosen auf Basis von strukturierten Daten. Hier betrifft das etwa die Bildverarbeitung: So werden Röntgenbilder als Input in ein KI-System gegeben, der Output wird unterstützend für eine Diagnose verwendet. Das Erfassen von Bildinhalten ist auch beim autonomen Fahren entscheidend, wo Verkehrszeichen, Bäume, Fußgänger und Radfahrer fehlerfrei erkannt werden müssen. In solch sensiblen Anwendungsfeldern wie der medizinischen Diagnostik oder in sicherheitskritischen Bereichen müssen KI-Systeme absolut zuverlässige Problemlösungsstrategien liefern [FG19].

Dabei ist die Erklärbarkeit von Entscheidungen, die durch KI getroffen werden, in sicherheitskritischen Anwendungsbranchen eine Voraussetzung für die Akzeptanz bei den Nutzern, für Zulassungs- und Zertifizierungsverfahren oder das Einhalten der durch die Datenschutzgrundverordnung (DSGVO) geforderten Transparenzpflichten [DTK21].
Ein NN als KI-System kann als die Verkettung von linearen Funktionen und nicht-linearen Aktivierungsfunktionen verstanden werden. Zum Lebenszyklus eines KI-Systems gehören die Datenerhebung und Datenaufbereitung, der Trainingsprozess, das Testen und die Inferenz, bei der das parametrisierte Netzwerk in der Anwendung genutzt wird. Häufig sind NNs klassischen Verfahren in ihrer Präzision deutlich überlegen, gleichzeitig fehlt es aber an der Interpretierbarkeit und damit an der Nachvollziehbarkeit getroffener Entscheidungen.
Bei einem Entscheidungsbaum kann beispielsweise genau erklärt werden, aufgrund welcher Schwellenwerte welche Entscheidung zustande kommt. Im Falle eines NN liegen Millionen von Parametern vor, die genau verstanden und interpretiert werden müssen.

Die Verwendung von immer größeren NNs mit Millionen von Parametern führte zu einem Durchbruch im Bereich der Bilderkennung (Klassifikation, Tagging, Segmentierung, Detektion, ...), welcher sich anschließend auf andere Bereiche wie beispielsweise Spracherkennung, Spiele oder Video-Analyse

ausweitete.

Es ist bekannt, dass die Präzision der NNs mit größeren Architekturen und größeren Datensätzen deutlich steigt. Um Rechenkapazitäten zu sparen, werden die Daten oft nicht selbst erhoben und gelabelt, sondern bereits verfügbare Datensätze übernommen. Zum Training der NNs werden oft Cloud-Computing-Plattformen verwendet, anstatt das NN lokal zu trainieren oder es kommen vortrainierte NNs zum Einsatz, bei denen nur noch die letzten Schichten mit dem eigenen Datensatz trainiert werden müssen. All diese Schritte zur Einsparung von Ressourcen bedeuten gleichzeitig einen Verlust an Kontrolle, der von einem Angreifer ausgenutzt werden kann.

Eine Möglichkeit sind sogenannte Adversariale Angriffe, die das vollständig trainierte NN angreifen. Im Fall eines Klassifikationsproblems werden Datenpunkte so gestört, dass sie fälschlicherweise einer anderen Klasse zugeordnet werden.
Poisoning-Angriffen (PAs) finden im Gegensatz zu Adversarialen Angriffen vor bzw. während des Trainings statt. Dabei werden korrumpierte Daten in den Trainingsdatensatz eingeschleust, um die Vorhersagequalität eines NN zu verringern. Das Ziel des Angreifers ist es, im Fall eines Klassifikationsproblems beispielsweise, die Präzision einer einzelnen Klasse zu verringern, während sich die Präzision auf allen anderen Klassen gar nicht oder nur leicht verändert.
Ein Spezialfall sind die sogenannten Backdoor-Poisoning-Angriffe, bei denen der Angreifer eine Art Hintertür im Datensatz implementiert, die während der Inferenz ausgenutzt werden kann. Damit erfolgt eine aus Sicht des Angreifers gewollte falsche Klassifikation nur dann, wenn dem NN eine Eingabe präsentiert wird, die einen solchen Auslöser enthält.
Diese Angriffe können in ihrer Detektion noch zusätzlich erschwert werden, indem das entsprechende Label nicht abgeändert wird. In diesem Fall sprechen wir von Label-konsistenten Poisoning-Angriffen.

Es ist noch immer schwierig, die konzeptionell leicht umzusetzenden PAs erfolgreich zu detektieren. Die Detektion von PAs erweist sich grundsätzlich als schwierig, vor allem deshalb, weil sich das NN nur in der Präsenz eines Auslösers anders verhält.

Die in dieser Arbeit untersuchte Idee für ein verbessertes Verfahren zur Detektion von Poisoning-Angriffen liefert [LWB+19]. Dort werden Datensätze auf sogenannte Clever Hans-Artefakte (CH-Artefakte) untersucht. Ein CH-Artefakt bezeichnet ein Merkmal einer Netzeingabe einer bestimmten Klasse,

die zur selben, richtigen Entscheidung des NN führt, der Grund dafür jedoch abweichend ist. D.h. für eine Eingabe mit einem CH-Artefakt besitzt das NN eine andere Entscheidungsstrategie. Eine Möglichkeit, solche unterschiedlichen Strategien zu quantifizieren bietet die Methode Layer-wise Relevance Propagation (LRP), die einzelnen Bereichen einer Eingabe bestimmte Relevanzen bezüglich einer getroffenen Entscheidung zuordnet. Die LRP gehört zu den lokalen Verfahren der Erklärbaren Künstlichen Intelligenz, die Erklärungen für einzelne Netzentscheidungen liefern können. Basierend darauf wollen wir ein Clustering-Verfahren anwenden, um PAs zu detektieren. Anschließend vergleichen wir dieses Vorgehen mit dem sogenannten Activation-Clustering (AC), mit dem ebenfalls PAs detektiert werden können.

Die vorliegende Arbeit ist wie folgt strukturiert: In Kapitel 2 stellen wir die theoretischen Grundlagen vor, die für Angriffe und die Detektion von PAs notwendig sind. In Abschnitt 2.2 geben wir eine kurze Einführung in NNs und stellen den klassischen Trainingsprozess vor. Abschnitt 2.3 führt in die unterschiedlichen Möglichkeiten eines PA auf NN ein. Abschnitt 2.5 gibt eine kurze Übersicht über den Bereich der Erklärbaren Künstlichen Intelligenz, wobei ein Beispiel eines Verfahrens, die sogenannte Layer-wise Relevanz Propagation ausführlich in Unterabschnitt 2.5.3 vorgestellt wird. Den Kern dieser Arbeit bildet Unterunterabschnitt 2.4.4.3, wo wir zu Beginn die grundlegenden Bestandteile des Algorithmus zur Detektion von PAs auf NNs erklären. In Kapitel 3 stellen wir das verwendete Netz, den Datensatz, die ausgeführten Angriffe und Verteidigungen vor und vergleichen die einzelnen Verfahren miteinander. In Kapitel 4 geben wir eine Zusammenfassung und blicken auf mögliche weitere Untersuchungen.

Kapitel 2

Theoretische Grundlagen

2.1 Theorie des Optimalen Transports

In diesem Kapitel betrachten wir einige Konzepte aus dem Bereich des Optimalen Transports, um einen Distanzbegriff zu entwickeln, der im Unterschied zur pixelweisen euklidischen Distanz besser für das kMeans-Clustering (vgl. Unterabschnitt 2.4.2) geeignet sein könnte. Ausgehend von Gaspard Monge's ursprünglicher Transportproblem-Formulierung betrachten wir die durch Kantorovich vorgestellte Relaxierung dieses Problems. Die Lösung des Optimierungsproblems wird für die Definition der p-Wasserstein-Distanz verwendet. Damit wird ein Distanzbegriff zwischen einzelnen Punkten auf einen Distanzbegriff zwischen Wahrscheinlichkeitsmaßen übertragen. Für eine Verallgemeinerung auf verschiedene Grundräume wird die Gromov-Wasserstein-Distanz (GWD) definiert. Damit können wir zwei Heatmaps als sogenannte metrische Maßräume auffassen und eine Distanz zwischen diesen berechnen. Abschließend definieren wir sogenannte Gromov-Wasserstein-Baryzentren, die im Rahmen des kMeans-Clustering als Mittelwerte fungieren.
Wir beschäftigen uns im Folgenden ausschließlich mit diskreten Wahrscheinlichkeitsverteilungen. Grundlegend dafür sind die folgenden Definitionen.

2.1.1 Grundbegriffe der Maßtheorie

In diesem Abschnitt stellen wir ganz kurz die aus der Maß- und Wahrscheinlichkeitstheorie wichtigen Begriffe vor, die wir im Folgenden für die Konstruktion der Wasserstein- und Gromov-Wasserstein-Distanz benötigen.

Definition 2.1.1.1. *[BBB+01] Sei X eine beliebige Menge. Eine Funktion $d : X \times X \to \mathbb{R} \cup \{\infty\}$ heißt Metrik auf X, falls die folgenden Bedingungen für alle $x, y, z \in X$ gelten:*

(1) Positive Definitheit: $d(x,y) > 0$ *für* $x \neq y$ *und* $d(x,y) = 0$ *für* $x = y$,

(2) Symmetrie: $d(x,y) = d(y,x)$,

(3) Dreiecksungleichung: $d(x,z) \leq d(x,y) + d(y,z)$.

Ein *metrischer Raum* ist eine Menge, versehen mit einer Metrik. Formal gesehen ist ein metrischer Raum ein Paar (X,d), wobei d eine Metrik auf X ist. Elemente von X heißen Punkte und $d(x,y)$ bezeichnet die Distanz zwischen x und y.

Definition 2.1.1.2. *[COT19] Als Histogramm (oder: Zufallsvektor) bezeichnen wir ein Element* $\boldsymbol{a} \in \Sigma_n$*, das zum folgenden Wahrscheinlichkeits-Simplex gehört:*

$$\Sigma_n := \{\boldsymbol{a} \in \mathbb{R}_+^n | \sum_{i=1}^{n} \boldsymbol{a_i} = 1\}$$

Definition 2.1.1.3. *[COT19] Ein diskretes Maß mit Gewichtung* $\boldsymbol{a}$ *und Punkten* $x_1, ..., x_n \in X$ *wird geschrieben als*

$$\alpha = \sum_{i=1}^{n} \boldsymbol{a}_i \delta_{x_i}, \tag{2.1.1}$$

wobei δ_x *das Dirac-Maß an Position* x *ist. Gilt* $\boldsymbol{a} \in \Sigma_n$ *und* $\boldsymbol{a}_i > 0$ *f.a.* i*, so sprechen wir von einem diskreten Wahrscheinlichkeitsmaß.*

Definition 2.1.1.4. *[Gro07]*
Ein metrischer Maßraum ist ein Tripel (X, d_X, μ_X) *mit*

- (X, d_X) *ist ein kompakter metrischer Raum*
- μ_X *ist ein Borel-Maß auf* X *mit vollem Support, d.h.*

$$supp[\mu_X] = X. \tag{2.1.2}$$

Definition 2.1.1.5 (Isometrie)**.** *Seien zwei metrische Räume* (X, d_X) *und* (Y, d_Y) *gegeben. Wir nennen die Abbildung* $\phi : X \to Y$ *Isometrie, falls* ϕ *surjektiv ist und* $d_X(x,x') = d_Y(\phi(x), \phi(x'))$ *f.a.* $x, x' \in X$ *gilt.*
Falls eine solche Abbildung ϕ *existiert, nennen wir* X *und* Y *isometrisch.*

Definition 2.1.1.6 (Bildmaß)**.** *Seien* X, Y *zwei Maßräume,* $T : X \to Y$ *eine messbare Abbildung und* μ *ein Maß auf* X*. Dann ist das Bildmaß (oder: pushforwad-Maß)* $T^{\#}\mu$ *ein Maß auf* Y*, definiert durch*

$$T^{\#}\mu(A) = \mu(T^{-1}(A)) \tag{2.1.3}$$

für alle $A \subset Y$.

Definition 2.1.1.7 (Isomorphie)**.** *Zwei metrische Maßräume (X, d_X, μ_X) und (Y, d_Y, μ_Y) heißen isomorph, falls eine Maß-erhaltende Abbildung $f : X \to Y$ existiert, die $f_{\#}\mu_X = \mu_Y$ erfüllt.*

Mit $\mathcal{G}_{\mathcal{W}}$ bezeichnen wir die Menge aller metrischen Maßräume.

Beispiel 2.1.1.8. *Wir betrachten die beiden metrischen Maßräume*

$$\left(\{a,b\}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \{\frac{1}{2}, \frac{1}{2}\}\right) \text{ und } \left(\{a',b'\}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \{\frac{1}{4}, \frac{3}{4}\}\right). \tag{2.1.4}$$

Dann sind diese Räume isometrisch, jedoch nicht isomorph, siehe Abbildung 2.1.

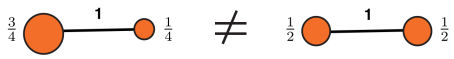

Abbildung 2.1: Die beiden metrischen Maßräume sind isometrisch, aber nicht isomorph.

Quelle: [COT19]

2.1.2 Optimaler Transport nach Kantorovich

In diesem Abschnitt wollen wir einen Ähnlichkeitsbegriff für Maße einführen. Der Bereich des Optimalen Transports beschäftigt sich mit der Frage, wie mehrere Objekte, die sich in einer bestimmten Verteilung an einem Startpunkt befinden, auf optimale Weise zu einem Zielpunkt transportiert werden können. Dabei wird die Verteilung der Objekte als Wahrscheinlichkeitsverteilung interpretiert.

2.1.2.1 Optimaler Transport (Formulierung von Monge)

Dieses Problem wurde zuerst von Gaspard Monge im Jahr 1781 formuliert. In dieser Formulierung wird eine Abbildung gesucht, die jedem Punkt in der Ausgangsverteilung einen Punkt in der Zielverteilung zuordnet. Dabei wird die gesamte Masse eines Punktes in der Startverteilung auf einen Punkt in der Zielverteilung verschoben. Eine solche Abbildung ist in Abbildung 2.2 dargestellt.
Für den Fall zweier diskreter Maße

$$\alpha = \sum_{i=1}^{n} \boldsymbol{a}_i \delta_{x_i} \text{ und } \beta = \sum_{j=1}^{m} \boldsymbol{b}_j \delta_{y_j} \tag{2.1.5}$$

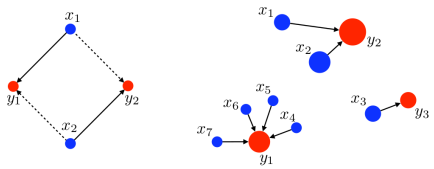

Abbildung 2.2: Links: Fehlende Eindeutigkeit in der Zuordnung. Die beiden Punkte x_1 und x_2 können sowohl den Punkten y_1 bzw. y_2 zugeordnet werden, um eine zulässige Abbildung zu erhalten. **Rechts:** Die Monge-Abbildung assoziiert das blaue Maße $\boldsymbol{\alpha}$ mit dem roten Maß $\boldsymbol{\beta}$. Dabei ist die Masse in den jeweiligen Punkten über den Flächeninhalt der Kreise dargestellt.

Quelle: [COT19]

können wir das Monge-Problem wie folgt formulieren:
Gesucht ist eine Abbildung $T : \{x_1, ..., x_n\} \to \{y_1, ..., y_m\}$, die die Bedingung

$$\forall j \in \{1, ..., m\} : \quad \boldsymbol{b}_j = \sum_{i:T(x_i)=y_i} \boldsymbol{a}_i \tag{2.1.6}$$

erfüllt, die wir in Kurzform als $T_{\#}\alpha = \beta$ schreiben. Aufgrund der Positivität von $\boldsymbol{b}$ ist die Abbildung notwendigerweise surjektiv. Als weitere Forderung gilt, dass die Abbildung T minimal bezüglich einer Transportkostenfunktion $c(x, y)$ für $(x, y) \in \mathbb{R} \times \mathbb{R}$ sein soll:

$$T = \min_S \Big\{ \sum_i c(x_i, S(x_i)) : S_{\#}\alpha = \beta \Big\}. \tag{2.1.7}$$

Neben der fehlenden Eindeutigkeit einer solchen Abbildung ist auch die Existenz nicht immer gegeben. Eine Lösung dafür liefert die Formulierung von Kantorovich im folgenen Abschnitt.

2.1.2.2 Optimaler Transport nach Kantorovich

Das Problem des Optimalen Transports nach Kantorovich gehört zu den typischen Optimal Transport Problemen. Es stellt eine Relaxierung der Formulierung von Gaspard Monge dar, bei der nun auch das Aufteilen von Masse (mass splitting) zulässig ist, d.h. die Masse an einem Startpunkt kann auf mehrere Zielpunkte abgebildet werden.

In Kantorovichs Formulierung ist eine Kopplung (oder: Transportabbildung/ Transportplan) $\boldsymbol{P} = (\boldsymbol{P}_{i,j})_{i,j} \in \mathbb{R}^{n \times m}$ gesucht, die die Kosten, die bei der Verschiebung eines diskreten Maßes $\boldsymbol{a}$ auf ein anderes diskretes Maß $\boldsymbol{b}$ bezüglich der Kosten $\boldsymbol{C} = (\boldsymbol{C}_{i,j})_{i,j} \in \mathbb{R}^{n \times m}$ entstehen, minimiert. Die Einträge

$\boldsymbol{P}_{i,j}$ geben an, wie viel der Masse von x_i nach y_j transportiert werden soll. Die Einträge $\boldsymbol{C}_{i,j}$ beschreiben die Kosten, um eine Einheit von x_i nach y_j zu transportieren.

Damit $\boldsymbol{P}$ eine Transportabbildung ist, muss $\boldsymbol{P} \in \Pi(a, b) = \{\boldsymbol{P} \geq \boldsymbol{0}, \boldsymbol{P}\boldsymbol{1}_m = \boldsymbol{a}, \boldsymbol{P}^\top \boldsymbol{1}_n = \boldsymbol{b}\}$ gelten. Dabei bedeutet die erste Forderung, dass nur positive Anteile verschoben werden können. Die beiden letzten Bedingungen fordern, dass die gesamte verschobene Masse der Start- bzw. Zielverteilung entspricht und somit keine Masse entsteht oder verloren geht.
Ein großer Vorteil des Optimalen Transports ist, dass neben dem Vergleich von diskreten mit diskreten und stetigen mit stetigen Verteilungen auch der Vergleich von diskreten und stetigen Verteilungen miteinander möglich ist. Transportpläne für diese drei verschiedenen Problemstellungen sind in Abbildung 2.3 dargestellt.

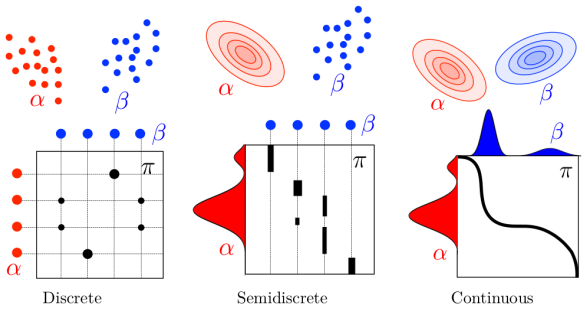

Abbildung 2.3: Links: Kopplung zwischen zwei diskreten Wahrscheinlichkeitsverteilungen. **Mitte:** Kopplung zwischen einer kontinuierlichen Wahrscheinlichkeitsverteilung $\boldsymbol{\alpha}$ und einer diskreten Wahrscheinlichkeitsverteilung $\boldsymbol{\beta}$. **Rechts:** Kopplung im kontinuierlichen Fall.

Quelle: [COT19]

Gesucht ist nun auch hier der optimale Transportplan, der die entstehenden Gesamtkosten minimiert. Kantorovichs Optimal Transport Problem lässt sich nun schreiben als

$$L_C(\boldsymbol{a}, \boldsymbol{b}) := \min_{P \in \Pi(\boldsymbol{a}, \boldsymbol{b})} \langle \boldsymbol{C}, \boldsymbol{P} \rangle := \min_{P \in \Pi(\boldsymbol{a}, \boldsymbol{b})} \sum_{i,j} \boldsymbol{C}_{i,j} \boldsymbol{P}_{i,j} \tag{2.1.8}$$

Dies ist ein lineares Problem, dessen Lösung nicht notwendigerweise eindeutig ist. Kantorovich gilt als Erfinder der linearen Optimierung [Tho18].

2.1.2.3 Metrische Eigenschaften

Für den Fall, dass die Grundkosten eine Metrik darstellen, ist auch die optimale Lösung des Optimal Transport Problems wieder eine Metrik [CA14] und definiert die Wasserstein-Distanz wie folgt:

Definition 2.1.2.1 (p-Wasserstein-Distanz auf Σ_n)**.** *Sei $n = m$ und für $p \geq 1$ gelte $\boldsymbol{C} = \boldsymbol{D}^p = (\boldsymbol{D}_{i,j}^p)_{i,j} \in \mathbb{R}^{n \times n}$, wobei $\boldsymbol{D} \in \mathbb{R}_+^{n \times n}$ eine Metrik ist. Dann definiert*

$$W_p(\boldsymbol{a}, \boldsymbol{b}) := L_{\boldsymbol{D}^p}(\boldsymbol{a}, \boldsymbol{b})^{1/p} \tag{2.1.9}$$

die p-Wasserstein-Distanz auf Σ_n

Lemma 2.1.2.2. *[COT19, Vil03] W_p ist eine Metrik*

Beweis. Nach Voraussetzung besitzt $\boldsymbol{C} = \boldsymbol{D}^p$ eine Nulldiagonale. Somit gilt $W_p(\boldsymbol{a}, \boldsymbol{a}) = 0$.. Die zugehörige Transportmatrix ist $\boldsymbol{P}^\star = diag(\boldsymbol{a})$. Aufgrund der Positivität aller Nicht-Diagonalelemente von $\boldsymbol{D}^p$ gilt $W(\boldsymbol{a}, \boldsymbol{b}) > 0$ für $\boldsymbol{a} \neq \boldsymbol{b}$, da in disem Fall jede zulässige Kopplung und damit insbesondere die optimale Kopplung ein Nicht-Diagonalelemnt ungleich 0 besitzt. Die Symmetrie von $W_p(\boldsymbol{a}, \boldsymbol{b})$ gilt wegen der Symmetrie von $\boldsymbol{D}^p$.

Für den Nachweis der Dreiecksungleichung im Fall beliebiger Maße nutzt Villani [Vil03] das sogenannte Gluing Lemma. Im diskreten Fall ist die Konstruktion dieser geklebten Kopplung etwas einfacher. Seien $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \Sigma_n$. Seien $\boldsymbol{P}$ und $\boldsymbol{Q}$ zwei optimale Lösungen des Transportproblems zwischen $\boldsymbol{a}$ und $\boldsymbol{b}$ bzw. $\boldsymbol{b}$ und $\boldsymbol{c}$. Wir definieren $\tilde{\boldsymbol{b}}$, wobei $\tilde{\boldsymbol{b}}_j = \boldsymbol{b}_j$, falls $\boldsymbol{b}_j > 0$, und $\tilde{\boldsymbol{b}}_j = 1$ sonst gilt. Damit können wir

$$\boldsymbol{S} := \boldsymbol{P} diag(1/\tilde{\boldsymbol{b}}) \boldsymbol{Q} \in \mathbb{R}_+^{n \times n} \tag{2.1.10}$$

schreiben. Es gilt $\boldsymbol{S} \in \Pi(\boldsymbol{a}, \boldsymbol{c})$ wegen

$$\boldsymbol{S}\mathbf{1}_n = \boldsymbol{P} diag(1/\tilde{\boldsymbol{b}}) \boldsymbol{Q}\mathbf{1}_n = \boldsymbol{P}(\boldsymbol{b}/\tilde{\boldsymbol{b}}) = \boldsymbol{P}\mathbf{1}_{supp[\boldsymbol{b}]} = \boldsymbol{a}, \tag{2.1.11}$$

wobei $\mathbf{1}_{supp[\boldsymbol{b}]}$ den Vektor der Größe n bezeichnet, der Einsen an den Stellen mit Indizes j besitzt, für die auch $\boldsymbol{b}_j > 0$ gilt, und sonst aus Nullen besteht. Außerdem wurde $\boldsymbol{P}\mathbf{1}_{supp[\boldsymbol{b}]} = \boldsymbol{P}\mathbf{1}_n = \boldsymbol{a}$ benutzt, denn es gilt notwendigerweise $\boldsymbol{P}_{i,j} = 0$ für diejenigen j mit $\boldsymbol{b}_j = 0$. Analog folgt $\boldsymbol{S}^\top \mathbf{1}_n = \boldsymbol{c}$.
Damit erhalten wir

$$W_p(\boldsymbol{a},\boldsymbol{c}) = \left(\min_{\boldsymbol{P}\in\boldsymbol{\Pi}(\boldsymbol{a},\boldsymbol{c})} \langle \boldsymbol{P}, \boldsymbol{D}^p\rangle\right)^{1/p} \leq \langle \boldsymbol{S}, \boldsymbol{D}^p\rangle^{1/p} \tag{2.1.12}$$

$$= \left(\sum_{ik} \boldsymbol{D}_{ik}^p \sum_j \frac{\boldsymbol{P}_{ij}\boldsymbol{Q}_{jk}}{\tilde{\boldsymbol{b}}_j}\right)^{1/p} \leq \left(\sum_{ijk} (\boldsymbol{D}_{ij}+\boldsymbol{D}_{jk})^p \frac{\boldsymbol{P}_{ij}\boldsymbol{Q}_{jk}}{\tilde{\boldsymbol{b}}_j}\right)^{1/p} \tag{2.1.13}$$

$$\leq \left(\sum_{ijk} \boldsymbol{D}_{ij}^p \frac{\boldsymbol{P}_{ij}\boldsymbol{Q}_{jk}}{\tilde{\boldsymbol{b}}_{\boldsymbol{j}}}\right)^{1/p} + \left(\sum_{ijk} \boldsymbol{D}_{jk}^p \frac{\boldsymbol{P}_{ij}\boldsymbol{Q}_{jk}}{\tilde{\boldsymbol{b}}_j}\right)^{1/p}. \tag{2.1.14}$$

Dabei haben wir in der ersten Ungleichung benutzt, dass $\boldsymbol{S}$ keine optimale Lösung ist. Für die zweite Ungleichung wurde die Dreiecksungleichung für $\boldsymbol{D}$ verwendet und die dritte Ungleichung folgt aus der Minkowski-Ungleichung. Nach Umstellen der beiden letzten Terme erhalten wir damit

$$\begin{aligned} W_p(\boldsymbol{a},\boldsymbol{c}) &\leq \left(\sum_{ij} \boldsymbol{D}_{ij}^p \boldsymbol{P}_{ij} \sum_k \frac{\boldsymbol{Q}_{jk}}{\tilde{\boldsymbol{b}}_{\boldsymbol{j}}}\right)^{1/p} + \left(\sum_{jk} \boldsymbol{D}_{jk}^p \boldsymbol{Q}_{jk} \sum_i \frac{\boldsymbol{P}_{ij}}{\tilde{\boldsymbol{b}}_j}\right)^{1/p} \\ &= \left(\sum_{ij} \boldsymbol{D}_{ij}^p \boldsymbol{P}_{ij}\right)^{1/p} + \left(\sum_{jk} \boldsymbol{D}_{jk}^p \boldsymbol{Q}_{jk}\right)^{1/p} \\ &= W_p(\boldsymbol{a},\boldsymbol{b}) + W_p(\boldsymbol{b},\boldsymbol{c}) \end{aligned}$$

und damit die Behauptung. □

Bemerkung 2.1.2.3. *Für $p = 1$ ist die p-Wasserstein-Distanz auch als Earth Movers's Distance [RTG00] bekannt.*

2.1.2.4 Komplexitätsanalyse

Im Fall $n = m$ ist das Kantorovich-Problem und damit die Berechnung der Wasserstein-Distanz ein lineares Optimierungsproblem mit $\mathcal{O}(n)$ linearen Bedingungen. Für die Lösung kann beispielsweise der lineare Lee-Sinford-Algorithmus zur Berechnung einer Lösung in $\tilde{\mathcal{O}}(n^{2.5})$ [LS14] verwendet werden, der den vorherigen Standard von $\mathcal{O}(n^{2.5})$ [Ren88] verbessert.

2.1.2.5 Wasserstein-Baryzentren

Für den Fall zweier Wahrscheinlichkeitsmaße ist die Interpolation zwischen diesen als McCann's Interpolation [McC97] bekannt. Eine Verallgemeinerung

auf endlich viele Maße liefert [AC11], wodurch sich eine Art Mittelwertbegriff für Wahrscheinlichkeitsmaße ergibt. Das Problem lässt sich im diskreten Fall wie folgt formulieren:

Seien die S Histogramme $(b_s)_{s=1}^S$ mit $b_s \in \Sigma_{n_s}$ und die Gewichte $\lambda \in \Sigma_S$ gegeben. Für $n_s = n$ und $C_s = C = D^p$ erhalten wir das p-Wasserstein-Baryzentrum mit den baryzentrischen Koordinaten λ_s als Lösung des Minimierungsproblems

$$\min_{a \in \Sigma_n} \sum_{s=1}^{S} \lambda_s W_p^p(a, b_s). \tag{2.1.15}$$

Existenz und Eindeutigkeit werden in [AC11] diskutiert. Zwei Algorithmen zur Berechnung der Baryzentren werden in [COO15] vorgestellt.

2.1.3 Entropisch regularisierte Wasserstein-Distanz

In diesem Abschnitt betrachten wir die regularisierte Version der Wasserstein-Distanz mithilfe von Entropie. Diese von Marco Cuturi vorgestellte Regularisierung [Cut13] verhalf dem Gebiet des Computational Optimal Transport zu sehr großem Interesse. Durch die Regularisierung mit Entropie entsteht eine Kullback-Leibler-Distanz, für die Standard-Verfahren wie beispielsweise der Sinkhorn-Algorithmus verwendet werden können, sodass sich die Berechnungen vergleichsweise effizient durchführen lassen.

Die Entropie einer Matrix ist wie folgt definiert:

Definition 2.1.3.1 (Entropie)**.** *Für $T \in \mathbb{R}_+^{n \times n}$ definieren wir die Entropie als*

$$H(T) := -\sum_{i,j=1}^{N} T_{i,j}(log(T_{i,j}) - 1). \tag{2.1.16}$$

Lemma 2.1.3.2. *Das Minimierungsproblem*

$$L_C^{\varepsilon}(\boldsymbol{a}, \boldsymbol{b}) := \min_{P \in \Pi(\boldsymbol{a}, \boldsymbol{b})} \langle \boldsymbol{P}, \boldsymbol{C} \rangle - \varepsilon \boldsymbol{H}(\boldsymbol{P}) \tag{2.1.17}$$

ist ein ε-konvexes Problem und besitzt deshalb eine eindeutige optimale Lösung.

Beweis. Die Entropie H ist wegen der Hesse-Matrix $\partial^2 H(P) = -diag(1/P_{i,j})$ und $P_{i,j} \leq 1$ stark konkav. Durch die Multiplikation mit ε und anschließender Subtraktion wird aus (2.1.17) ein ε-konvexes Problem. □

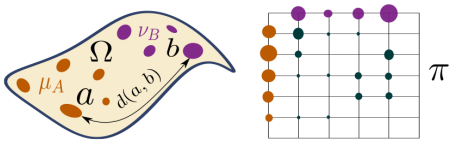

Abbildung 2.4: Links: Visualisierung der Wasserstein-Distanz zwischen den Verteilungen μ_A und ν_B auf dem metrischen Raum (Ω, d). **Rechts:** Möglicher Transportplan π.

Quelle: [Via19]

Bemerkung 2.1.3.3. *Für größere ε wird mehr Entropie gefordert*

Proposition 2.1.3.4 (Konvergenz in ε [COT19])**.** *Die eindeutige Lösung P_ε von* (2.1.17) *konvergiert gegen die optimale Lösung mit maximaler Entropie innerhalb der Menge aller optimalen Lösungen des Kantorovich-Problems, d.h.*

$$\boldsymbol{P}_\varepsilon \xrightarrow{\varepsilon\to 0} \arg\min_P\{-\boldsymbol{H}(\boldsymbol{P}) : \boldsymbol{P} \in \Pi(\boldsymbol{a},\boldsymbol{b}), \langle \boldsymbol{P}, \boldsymbol{C}\rangle = L_{\boldsymbol{C}}(\boldsymbol{a},\boldsymbol{b})\}, \tag{2.1.18}$$

$$L^\varepsilon_{\boldsymbol{C}}(\boldsymbol{a},\boldsymbol{b}) \xrightarrow{\varepsilon\to 0} L_{\boldsymbol{C}}(\boldsymbol{a},\boldsymbol{b}) \tag{2.1.19}$$

Zudem gilt

$$\boldsymbol{P}_\varepsilon \xrightarrow{\varepsilon\to\infty} \boldsymbol{a}\otimes\boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^\top = (\boldsymbol{a}_i\boldsymbol{b}_j)_{i,j}. \tag{2.1.20}$$

Beweis. Sei eine Folge $(\varepsilon_l)_l$ mit $\varepsilon_l \to 0$ und $\varepsilon_l > 0$ gegeben. Sei $\boldsymbol{P}_l$ die Lösung von (2.1.17) für $\varepsilon = \varepsilon_l$. Da $\Pi(\boldsymbol{a},\boldsymbol{b})$ beschränkt ist, existiert eine Teilfolge ε_k mit $\boldsymbol{P}_k \to \boldsymbol{P}^*$. Aufgrund der Abgeschlossenheit von $\Pi(\boldsymbol{a},\boldsymbol{b})$ folgt $\boldsymbol{P}^* \in \Pi(\boldsymbol{a},\boldsymbol{b})$.
Sei $\boldsymbol{P} \in \Pi(\boldsymbol{a},\boldsymbol{b})$ beliebig mit $\langle \boldsymbol{C}, \boldsymbol{P}\rangle = L_C(\boldsymbol{a},\boldsymbol{b})$. Aufgrund der Optimalität von $\boldsymbol{P}$ und $\boldsymbol{P}_l$ bezüglich $L_{\boldsymbol{C}}(\boldsymbol{a},\boldsymbol{b})$ bzw. $L^{\varepsilon_l}_{\boldsymbol{C}}(\boldsymbol{a},\boldsymbol{b})$ gilt

$$0 \le \langle \boldsymbol{C}, \boldsymbol{P}_l\rangle - \langle \boldsymbol{C}, \boldsymbol{P}\rangle \le \varepsilon_l(\boldsymbol{H}(\boldsymbol{P}_l) - \boldsymbol{H}(\boldsymbol{P})). \tag{2.1.21}$$

Aufgrund der Stetigkeit von $\boldsymbol{H}$ folgt für $l \to +\infty$ in (2.1.21) $\langle \boldsymbol{C}, \boldsymbol{P}^*\rangle = \langle \boldsymbol{C}, \boldsymbol{P}\rangle$, womit $\boldsymbol{P}^*$ ein zulässiger Punkt ist. Division durch ε_l in (2.1.21) und Übergang zum Grenzwert liefert $\boldsymbol{H}(\boldsymbol{P}) \le \boldsymbol{H}(\boldsymbol{P}^*)$. Folglich ist $\boldsymbol{P}^*$ eine Lösung von (2.1.18).
Da die Lösung $\boldsymbol{P}^*_0$ aufgrund der strikten Konvexität von $-\boldsymbol{H}$ eindeutig ist, gilt $\boldsymbol{P}^* = \boldsymbol{P}^*_0$ und die gesamte Folge konvergiert.

□

Bemerkung 2.1.3.5. (2.1.18) *zeigt, dass die Lösung für kleine ε gegen den Transportplan mit maximaler Entropie konvergiert. Im Gegensatz dazu bedeutet* (2.1.20)*, dass die Lösung für große Regularisierungsparameter gegen die Kopplung mit maximaler Entropie zwischen zwei gegebenen Randverteilungen $\boldsymbol{a}$ und $\boldsymbol{b}$ konvergiert.*

Bemerkung 2.1.3.6. *Für größere Werte von ε ergibt sich eine beschleunigte Berechnung der optimalen Kopplung.*

Interpretation als KL-Problem: Das Minimierungsproblem (2.1.17) kann außerdem als Spezialfall eines Kullback-Leibler-Minimierungsproblems betrachtet werden, bei dem eine Kopplungsmatrix $\boldsymbol{P}_\varepsilon$ gesucht ist, die im Sinne einer Kullback-Leibler-Divergenz möglichst nahe an einem Kernel K liegt. Wir definieren die Kullback-Leibler-Divergenz zwischen zwei Kopplungen [COT19] als

$$\boldsymbol{KL}(\boldsymbol{P}|\boldsymbol{K}) := \sum_{i,j} \boldsymbol{P}_{i,j} \log\left(\frac{\boldsymbol{P}_{i,j}}{\boldsymbol{K}_{i,j}}\right) - \boldsymbol{P}_{i,j} + \boldsymbol{K}_{i,j}. \tag{2.1.22}$$

Damit ist die eindeutige Lösung $\boldsymbol{P}_\varepsilon$ von Gleichung 2.1.17 eine Projektion des zur Kostenmatrix $\boldsymbol{C}$ gehörigen Gibbs-Kernels $\boldsymbol{K}_{i,j} := e^{-\frac{\boldsymbol{C}_{i,j}}{\varepsilon}}$ auf $\Pi(\boldsymbol{a}, \boldsymbol{b})$. Mit der obigen Definition erhalten wir

$$\boldsymbol{P}_\varepsilon = Proj^{\boldsymbol{KL}}_{\boldsymbol{\Pi}(\boldsymbol{a},\boldsymbol{b})}(\boldsymbol{K}) := \underset{P \in \boldsymbol{\Pi}(\boldsymbol{a},\boldsymbol{b})}{\arg\min}\, \boldsymbol{KL}(\boldsymbol{P}|\boldsymbol{K}). \tag{2.1.23}$$

2.1.3.1 Algorithmus von Sinkhorn und Variationen

In diesem Abschnitt sehen wir, dass die Lösung des regularisierten Problems eine besondere Form besitzt, die auf den Algorithmus von Sinkhorn führt.

Proposition 2.1.3.7. *Die Lösung des regularisierten Problems* (2.1.17) *besitzt die Form*

$$\forall (i,j) \in \{1,...,n\} \times \{1,...,m\} : \boldsymbol{P}_{i,j} = \boldsymbol{u}_i \boldsymbol{K}_{i,j} \boldsymbol{v}_j \tag{2.1.24}$$

für die beiden (unbekannten) Variablen $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^n_+ \times \mathbb{R}^m_+$.

Beweis. Wir führen für jede der beiden Nebenbedingungen die dualen Variablen $\boldsymbol{f} \in \mathbb{R}^n$ und $\boldsymbol{g} \in \mathbb{R}^m$ ein. Für die Lagrange-Funktion zu (2.1.17) erhalten wir damit:

$$\mathcal{L}(\boldsymbol{P}, \boldsymbol{f}, \boldsymbol{g}) = \langle \boldsymbol{P}, \boldsymbol{C} \rangle - \varepsilon \boldsymbol{H}(\boldsymbol{P}) - \langle \boldsymbol{f}, \boldsymbol{P}\mathbf{1}_m - \boldsymbol{a} \rangle - \langle \boldsymbol{g}, \boldsymbol{P}^\top \mathbf{1}_n - \boldsymbol{a} \rangle. \tag{2.1.25}$$

Mit der Optimalitätsbedingung erster Ordnung ergibt sich

$$\frac{\partial \mathcal{L}(\boldsymbol{P}, \boldsymbol{f}, \boldsymbol{g})}{\partial \boldsymbol{P}_{i,j}} = \boldsymbol{C}_{i,j} + \varepsilon \log(\boldsymbol{P}_{i,j}) - \boldsymbol{f}_i - \boldsymbol{g}_j = 0, \tag{2.1.26}$$

womit wir für eine optimale Kopplung $\boldsymbol{P}$ für das regularisierte Problem den Ausdruck $\boldsymbol{P}_{i,j} = e^{\boldsymbol{f}_i/\varepsilon} e^{-\boldsymbol{C}_{i,j}/\varepsilon} e^{\boldsymbol{g}_j/\varepsilon}$ erhalten, der in die gewünschte Form umgeschrieben werden kann. □

Die Faktorisierung der Lösung in (2.1.24) können wir in der folgenden Matrix-Form schreiben: $\boldsymbol{P} = diag(\boldsymbol{u})\boldsymbol{K}diag(\boldsymbol{v})$. Die beiden Variablen $(\boldsymbol{u}, \boldsymbol{v})$ müssen deshalb die folgenden nichtlinearen Gleichungen erfüllen, die aufgrund der geforderten Massenerhaltungsbedingung in $\Pi(\boldsymbol{a}, \boldsymbol{b})$ gelten:

$$diag(\boldsymbol{u})\boldsymbol{K}diag(\boldsymbol{v})\mathbf{1}_m = \boldsymbol{a} \quad \text{und} \quad diag(\boldsymbol{v})\boldsymbol{K}^\top diag(\boldsymbol{u})\mathbf{1}_n = \boldsymbol{b}. \tag{2.1.27}$$

Aufgrund der Beziehung $diag(\boldsymbol{v})\mathbf{1}_m = \boldsymbol{v}$ und selbiger Beziehung für $\boldsymbol{u}$ erhalten wir die folgende Vereinfachung

$$\boldsymbol{u} \odot (\boldsymbol{K}\boldsymbol{v}) = \boldsymbol{a} \quad \text{und} \quad \boldsymbol{v} \odot (\boldsymbol{K}^\top \boldsymbol{u}) = \boldsymbol{b}, \tag{2.1.28}$$

wobei $\odot$ für die komponentenweise Multiplikation zweier Vektoren steht. Dieses Problem ist als *Matrix Scaling Problem* [NR99] bekannt.

Eine Möglichkeit zur Lösung dieses Problems ist ein iteratives Verfahren, bei dem zunächst $\boldsymbol{u}$ so modifiziert wird, dass die linke Seite in (2.1.28) erfüllt ist, und anschließend die Modifikation von $\boldsymbol{v}$ vorgenommen wird, sodass die rechte Seite in Gleichung 2.1.28 gilt. Mit diesen beiden Modifikationen erhalten wir den Algorithmus von Sinkhorn, der aus den beiden folgenden Updates besteht:

$$\boldsymbol{u}^{(l+1)} := \frac{\boldsymbol{a}}{\boldsymbol{K}\boldsymbol{v}^{(l)}} \text{ und } \boldsymbol{v}^{(l+1)} := \frac{\boldsymbol{b}}{\boldsymbol{K}^\top \boldsymbol{u}^{(l+1)}}, \tag{2.1.29}$$

wobei zu Beginn mit einem beliebigen positiven Vektor, beispielsweise $\boldsymbol{v}^{(0)} = \mathbf{1}_m$ initialisiert wird und l den aktuellen Iterationsschritt bezeichnet. Die obige Division muss ebenfalls elementweise verstanden werden.
Die Konvergenz dieses Verfahrens wurde zuerst von Sinkhorn [Sin64] gezeigt und trägt deshalb den entsprechenden Namen.

Bemerkung 2.1.3.8. *Die Lösung des regularisierten Problems lässt sich nach der L-ten Sinkhorn-Iteration mit dem Transportplan wie folgt angeben:* $P_L = diag(u_L)Kdiag(v_L)$. *Die Gesamtkosten sind gegeben durch* $\langle P_L, C_{XY} \rangle = u_L^\top (K \odot C_{XY}) v_L$.

Für die globale Konvergenzanalayse der Sinkhorn-Algorithmus lässt sich alternativ zum Beweis in [Sin64] ein Zusammenhang mit der Hilbertschen Projektionsmetrik benutzen, der erstmals in [FL89] vorgestellt wurde. Diese ist wie folgt definiert.

Definition 2.1.3.9 (Projektive Hilbert-Metrik)**.** *Seien $u, u' \in \mathbb{R}^n_{+,\star}$. Dann ist die projektive Hilbert-Metrik $d_{\mathcal{H}}$ definiert durch*

$$d_{\mathcal{H}}(u, u') := \log \max_{i,j} \frac{u_i u'_j}{u_j u'_i}. \tag{2.1.30}$$

Bemerkung 2.1.3.10. *Die Hilbert-Metrik $d_{\mathcal{H}}$ definiert eine Metrik auf dem projektiven Kegel $\mathbb{R}^n_{+,\star}/\sim$, wobei $u \sim u'$ bedeutet, dass gilt: $\exists r > 0 : u = ru'$, d.h. die Vektoren u und u' sind bis auf Skalierung identisch.*

Das folgende Theorem zeigt, dass eine positive Matrix eine Kontraktion auf dem Kegel positiver Vektoren bezüglich der Hilbert-Metrik ist.

Theorem 2.1.3.11. *Sei $K \in \mathbb{R}^{n\times m}_{+,\star}$. Dann gilt für $(v, v') \in (\mathbb{R}^m_{+,\star})^2$*

$$d_{\mathcal{H}}(Kv, Kv') \leq \lambda(K) d_{\mathcal{H}}(v, v'), \text{ mit } \begin{cases} \lambda(K) := \frac{\sqrt{\nu(K)}-1}{\sqrt{\nu(K)}+1} \\ \nu(K) := \max_{i,j,k,l} \frac{K_{i,k}K_{j,l}}{K_{j,k}K_{i,l}}. \end{cases} \tag{2.1.31}$$

Mit dem folgenden Resultat erhalten wir die Konvergenz des Sinkhorn-Verfahrens:

Theorem 2.1.3.12. *[COT19] Es gilt $(u^{(l)}, v^{(l)}) \to (u^\star, v^\star)$ für $(u^{(l)}, v^{(l)})$ in* (2.1.29) *und*

$$d_{\mathcal{H}}(u^{(l)}, u^\star) = \mathcal{O}(\lambda(K)^{2l}), \qquad d_{\mathcal{H}}(v^{(l)}, v^\star) = \mathcal{O}(\lambda(K)^{2l}) \tag{2.1.32}$$

Es gelten außerdem die beiden folgenden Abschätzungen

$$d_{\mathcal{H}}(u^{(l)}, u^\star) \leq \frac{d_{\mathcal{H}}(P^{(l)}\mathbf{1}_m, a)}{1-\lambda(K)^2}, \tag{2.1.33}$$

$$d_{\mathcal{H}}(v^{(l)}, v^\star) \leq \frac{d_{\mathcal{H}}(P^{(l)\top}\mathbf{1}_n, b)}{1-\lambda(K)^2}, \tag{2.1.34}$$

wobei

$$P^{(l)} := diag(u^{(l)}) K diag(v^{(l)}) \tag{2.1.35}$$

gilt. Desweiteren gilt die Abschätzung

$$||\log(P^{(l)}) - \log(P^\star)||_\infty \leq d_{\mathcal{H}}(u^{(l)}, u^\star) + d_{\mathcal{H}}(v^{(l)}, v^\star), \tag{2.1.36}$$

wobei $P^\star$ die eindeutige Lösung von Gleichung 2.1.17 ist.

Beweis. Für ein beliebiges Paar $(v, v') \in (\mathbb{R}^m_{+,\star})^2$ gilt

$$d_{\mathcal{H}}(v, v') = d_{\mathcal{H}}(v/v', \mathbf{1}_m) = d_{\mathcal{H}}(\mathbf{1}_m/v, \mathbf{1}_m/v').$$

Mit dieser Beziehung und Theorem 2.1.3.11 erhalten wir

$$\begin{aligned} d_{\mathcal{H}}(u^{(l+1)}, u^\star) &= d_{\mathcal{H}}\left(\frac{a}{Kv^{(l)}}, \frac{a}{Kv^\star}\right) \\ &= d_{\mathcal{H}}(Kv^{(l)}, Kv^\star) \\ &\leq \lambda(K) d_{\mathcal{H}}(v^{(l)}, v^\star). \end{aligned}$$

Dies zeigt (2.1.32). Mit Hilfe der Dreiecksungleichung erhalten wir

$$\begin{aligned} d_{\mathcal{H}}(u^{(l)}, u^\star) &\leq d_{\mathcal{H}}(u^{(l+1)}, u^{(l)}) + d_{\mathcal{H}}(u^{(l+1)}, u^\star) \\ &\leq d_{\mathcal{H}}\left(\frac{a}{Kv^{(l)}}, u^{(l)}\right) + \lambda(K)^2 d_{\mathcal{H}}(u^{(l)}, u^\star) \\ &= d_{\mathcal{H}}\left(a, u^{(l)} \odot (Kv^{(l)})\right) + \lambda(K)^2 d_{\mathcal{H}}(u^{(l)}, u^\star) \\ &= d_{\mathcal{H}}\left(a, P^{(l)}\mathbf{1}_m\right) + \lambda(K)^2 d_{\mathcal{H}}(u^{(l)}, u^\star) \end{aligned}$$

Nach Division durch $1 - \lambda(K)^2$ erhalten wir Gleichung 2.1.33. Die zweite Abschätzung (2.1.34) folgt analog. Die Gleichung 2.1.36 gilt nach Lemma 3 in [FL89]. □

2.1.3.2 Komplexitätsanalyse

Altschuler [AWR17] liefert eine Komplexitätsanalyse für die Sinkhorn-Iterationen. Im Fall $n = m$ sind für die Wahl $\varepsilon = \frac{4\log(n)}{\tau}$ $\mathcal{O}(||C||^3_\infty \log(n)\tau^{-3})$ Sinkhorn-Iterationen (inklusive eines Rundungsschrittes) notwendig, um eine zulässige Kopplung $\hat{\boldsymbol{P}} \in \Pi(\boldsymbol{a}, \boldsymbol{b})$ zu berechnen, die die Abschätzung $\langle \hat{\boldsymbol{P}}, \boldsymbol{C} \rangle \leq L_{\boldsymbol{C}}(\boldsymbol{a}, \boldsymbol{b}) + \tau$ erfüllt. Somit liefert das Sinkhorn-Verfahren eine τ-Approximation des nicht regularisierten Problems in $\mathcal{O}(n^2 \log(n)\tau^{-3})$ Operationen. Gleichzeitig wird dort eine Greedy-Variante der Sinkhorn-Iterationen namens Greenkhorn präsentiert, die eine τ-Approximation in $\mathcal{O}(n^2\tau^{-3})$ Operationen liefert. [DGK18] verbessert diese auf $\mathcal{O}(n^2\tau^{-2})$ Operationen.

2.1.4 Gromov-Wasserstein-Distanz

In den vorherigen Abschnitten hatten wir jeweils Wahrscheinlichkeitsmaße auf demselben metrischen Raum verglichen. In diesem Abschnitt wollen wir diesen Abstandsbegriff auf die Distanz zweier verschiedener metrischer Maßräume verallgemeinern. Dazu definieren wir die Gromov-Wasserstein-Distanz (GWD), die der Wasserstein-Distanz sehr ähnlich und eine Relaxierung der Hausdorff-Distanz ist. Dabei folgen wir [COT19], [Vay20], [VCF+20], [PCS16].

2.1.4.1 Gromov-Wasserstein-Distanz

In diesem Abschnitt wollen wir nun den Begriff der Wasserstein-Distanz auf einen Distanzbegriff für zwei verschiedene Grundräume metrischer Maßräume verallgemeinern.

Definition 2.1.4.1 (Tensor-Matrix-Multiplikation)**.** *Für einen Tensor* $\mathcal{L} = (\mathcal{L}_{i,j,k,l})_{i,j,k,l}$ *und eine Matrix* $(T_{i,j})_{i,j}$ *definieren wir die Tensor-Matrix-Multiplikation als*

$$\mathcal{L} \otimes T := \left(\sum_{k,l} \mathcal{L}_{i,j,k,l} T_{k,l} \right)_{i,j} . \tag{2.1.37}$$

Wir definieren zunächst die beiden folgenden Distanzen:

Definition 2.1.4.2 (Hausdorff-Metrik)**.** *Sei* $\mathcal{X}$ *ein metrischer Raum. Für* $X, Y \subset \mathcal{X}$ *kompakt und nichtleer definieren wir die Hausdorff-Metrik* $d_H(X, Y)$ *als*

$$d_H(X, Y) = \max\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)\}. \tag{2.1.38}$$

Anschaulich haben zwei kompakte Teilmengen einen geringen Hausdorff-Abstand, wenn es zu jedem Element der einen Menge ein Element der anderen Menge gibt, zu dem dieses einen geringen Abstand hat. In Abbildung 2.5 ist die Hausdorff-Distanz zweier Mengen X und Y dargestellt.

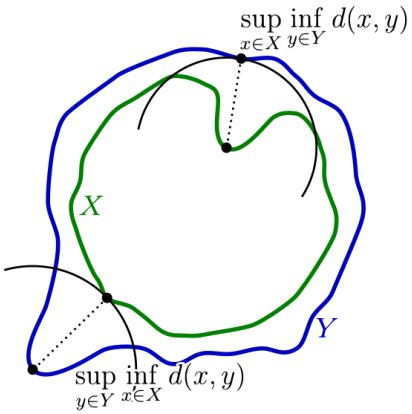

Abbildung 2.5: Visualisierung der Hausdorff-Distanz zwischen den Mengen X und Y in $\mathbb{R}^2$.

Quelle: [Com07]

Eine Teilmenge $R \subset X \times Y$ ist eine Korrespondenz zwischen den Mengen X und Y, falls $\pi_1(R) = X$ und $\pi_2(R) = Y$, wobei $\pi_1 : X \times Y \to X$ und

$\pi_2 : X \times Y \to Y$ die kanonischen Projektionen sind. Sei $\mathcal{R}(X, Y)$ die Menge aller Korrespondenzen zwischen X und Y.

Definition 2.1.4.3 (Gromov-Hausdorff-Metrik)**.** *Für die kompakten, metrischen Räume (X, d_X) und (Y, d_Y) ist die Gromov-Hausdorff-Metrik definiert als*

$$d_{\mathcal{GH}}(X, Y) = \frac{1}{2} \inf_R \sup_{(x,y),(x',y') \in R} |d_X(x, x') - d_Y(y, y')|, \tag{2.1.39}$$

wobei R über ganz $\mathcal{R}(X, Y)$ reicht.

Sei nun $p \geq 1$. Für zwei metrische Maßräume (X, d_X, μ_X) und (Y, d_Y, μ_Y) betrachten wir die Funktion

$$L(x, y, x', y') = |d_X(x, x') - d_Y(y, y')| \tag{2.1.40}$$

und definieren die GWD der Ordnung p [Mém11]:

$$d_{\mathcal{GW},p}(\mu_X, \mu_Y) := \left(\inf_{\pi \in \Pi(\mu_X, \mu_Y)} J_p(\pi) \right)^{1/p}, \tag{2.1.41}$$

wobei

$$J_p(\pi) := \int_{X \times Y \times X \times Y} L(x, y, x', y')^p d\pi(x, y) d\pi(x', y') \tag{2.1.42}$$

gilt und $\Pi(\mu_X, \mu_Y) = \{\pi \in P(X \times Y) | \forall (X_0, Y_0) \in X \times Y, \pi(X_0 \times Y) = \mu_X(X_0), \pi(X \times Y_0) = \mu_Y(Y_0)\}$ die Menge der Kopplungen der beiden Maßräume (X, μ_X) und (Y, μ_Y) ist [Vil03, VCF+20].

Bemerkung 2.1.4.4. *Mit dieser Definition erhalten wir nun eine relaxierte L^p-Version der Gromov-Hausdorff-Metrik* (2.1.39)*.*

Die GWD definiert eine Metrik auf dem Raum aller metrischen Räume $\mathcal{M}^{iso}$. Mit dieser Definition wir der Vergleich von Maßen über unterschiedlichen Grundräumen möglich und damit auch ein Vergleich von Objekten jeglicher Art.

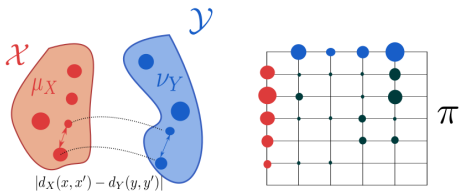

Abbildung 2.6: Links: Die beiden metrischen Maßräume $\mathcal{X}$ und $\mathcal{Y}$. **Rechts:** Zugehöriger Transportplan

Quelle: [VCF+20]

Wir zitieren einige Eigenschaften der GWD:

Theorem 2.1.4.5. *[Mém11, Thm. 5.1] Sei $p \in [1, \infty]$. Dann gilt:*

(a) $d_{\mathcal{GW},p}$ definiert eine Metrik auf der Menge aller Isomorphieklassen von metrischen Maßräumen.

(b) (Zwei Wahrscheinlichkeitsmaße auf demselben Raum) Sei (Z, d) ein kompakter metrischer Raum und α und β zwei verschiedene Borel-Maße auf Z. Sei $X = (Z, d, \alpha)$ und $Y = (Z, d, \beta)$. Dann gilt

$$d_{\mathcal{GW},p}(X, Y) \leq d^Z_{\mathcal{W},p}(\alpha, \beta). \tag{2.1.43}$$

Dieses Theorem zeigt nun, dass wir mit der GWD einen Distanzbegriff erhalten, der invariant gegenüber Rotationen, Translationen und Permutationen ist [Vay20].

Aufgrund genau dieser Invarianz vermuten wir, dass dieser Distanzbegriff deutlich besser als eine pixelweise L_2-Distanz für ein Clustering auf der Grundlage von Heatmaps ist, wie wir es in Unterunterabschnitt 2.4.4.3 benutzen werden. Diese Invarianz resultiert daraus, dass wir eine Metrik auf den Isomorphie-Klassen metrischer Maßräume betrachten.

Im Folgenden werden wir die numerischen Verfahren zur Bestimmung der GWD behandeln. Auch hier ermöglicht eine Entropie-regularisierte Version die einfache Berechnung einer Approximation wie im Fall der Wasserstein-Distanz.

Wir betrachten dazu wieder die diskreten Wahrscheinlichkeitsmaße $\mu = \sum_{i=1}^n a_i \delta_{x_i}$, $\nu = \sum_{j=1}^m b_j \delta_{y_j}$ auf den Räumen (X, d_X) bzw. (Y, d_Y). Mit

$\boldsymbol{C}_1$ und $\boldsymbol{C}_2$ bezeichnen wir die Distanzmatrizen der paarweisen Distanzen aller Punkte innerhalb des entsprechenden Raumes, d.h. es gilt

$$\forall(i,k) \in \{1,...,n\}^2 : C_1(i,k) = d_X(x_i, x_k) \tag{2.1.44}$$

und

$$\forall(j,l) \in \{1,...,m\}^2 : C_1(j,l) = d_Y(x_j, x_l). \tag{2.1.45}$$

Damit lautet das Gromov-Wasserstein-Problem im diskreten Fall:

$$GW_p^p(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{a}, \boldsymbol{b}) = \min_{P \in \Pi(\boldsymbol{a},\boldsymbol{b})} \sum_{i,j,k,l} |C_1(i,k) - C_2(j,l)|^p P_{i,j} P_{k,l} \tag{2.1.46}$$

$$= \min_{P \in \Pi(\boldsymbol{a},\boldsymbol{b})} \langle \boldsymbol{L}(\boldsymbol{C}_1, \boldsymbol{C}_2)^p \otimes P, P \rangle_{\mathcal{F}}, \tag{2.1.47}$$

wobei $\langle ., . \rangle_{\mathcal{F}}$ das Matrix-Produkt $\langle \boldsymbol{P}, \boldsymbol{Q} \rangle_{\mathcal{F}} = tr(\boldsymbol{Q}^\top \boldsymbol{P})$ bezeichnet und $\boldsymbol{L}(\boldsymbol{C}_1, \boldsymbol{C}_2) = (|C_1(i,k) - C_2(j,l)|)_{i,j,k,l}$ gilt. $\boldsymbol{L} \otimes P$ ist die Tensor-Matrix-Multiplikation nach (2.1.37).
Das Problem (2.1.47) ist ein nicht-konvexes quadratisches Optimierungsproblem, das im Allgemeinen NP-schwierig zu lösen und ebenfalls schwer zu approximieren ist.

Sei $\mathcal{L}(C_1, C_2) := (L(C_{1_{i,k}}, C_{2_{j,l}}))_{i,j,k,l}$. Dann erhalten wir mit der folgenden Propsition eine vereinfachte Darstellung für eine bestimmte Klasse von Verlustfunktionen.

Proposition 2.1.4.6. *[PCS16] Für die Verlustfunktion L gelte die Darstellung*

$$L(a,b) = f_1(a) + f_2(b) - h_1(a)h_2(b) \tag{2.1.48}$$

für die Funktionen f_1, f_2, h_1, h_2, dann gilt für $P \in \Pi(\boldsymbol{a}, \boldsymbol{b})$

$$\mathcal{L}(C_1, C_2) \otimes P = \boldsymbol{c}_{\boldsymbol{C}_1, \boldsymbol{C}_2} - h_1(C_1) P h_2(C_2)^\top, \tag{2.1.49}$$

wobei $\boldsymbol{c}_{\boldsymbol{C}_1,\boldsymbol{C}_2} := f_1(\boldsymbol{C}_1)\boldsymbol{a}\boldsymbol{1}_{N_2}^T + \boldsymbol{1}_{N_1}\boldsymbol{b}^T f_2(\boldsymbol{C}_2)^T$ unabhängig von P ist.

Für $p = 2$ und damit $\boldsymbol{L} = |\cdot|^2$ können wir (2.1.47) schreiben als:

$$\min_{P \in \Pi(\boldsymbol{a},\boldsymbol{b})} tr(\boldsymbol{c}_{\boldsymbol{C}_1,\boldsymbol{C}_2} P^\top) - 2tr(\boldsymbol{C}_1 P \boldsymbol{C}_2 P^\top). \tag{2.1.50}$$

In Standardform erhalten wir

$$\min_{\pi \in \Pi(\boldsymbol{a},\boldsymbol{b})} \boldsymbol{c}^\top \boldsymbol{x}(\boldsymbol{\pi}) + \frac{1}{2} \boldsymbol{x}(\boldsymbol{\pi})^\top \boldsymbol{Q} \boldsymbol{x}(\boldsymbol{\pi}), \tag{2.1.51}$$

wobei $\boldsymbol{x}(P) = vec(P), \boldsymbol{c} = vec(\boldsymbol{c}_{\boldsymbol{C}_1,\boldsymbol{C}_2})$ und $\boldsymbol{Q} = -4\boldsymbol{C}_1 \otimes_K \boldsymbol{C}_2$ gilt. $\otimes_K$ ist das Kronecker-Matrix-Produkt für $\boldsymbol{A} \in \mathbb{R}^{n\times m}, \boldsymbol{B} \in \mathbb{R}^{p\times q}, \boldsymbol{A} \otimes_K \boldsymbol{B} \in \mathbb{R}^{np\times mq}$ mit $\boldsymbol{A} \otimes_K \boldsymbol{B} = (A_{i,j}\boldsymbol{B})_{i,j}$.

(2.1.51) ist ein nicht konvexes quadratisches Optimierungsproblem, da die Hesse-Matrix $\boldsymbol{Q}$ im Allgemeinen nicht positiv semi-definit ist (die Eigenwerte sind die Produkte der Eigenwerte der Matrizen $\boldsymbol{C}_1$ und $\boldsymbol{C}_2$).

2.1.4.2 Entropisch regularisierte Gromov-Wasserstein-Distanz

Verwenden wir nun auch hier einen Regularisierungs-Term basierend auf der Entropie erhalten wir als regularisierte Version von (2.1.47) das folgende Optimierungsproblem:

$$GW_\varepsilon(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{a}, \boldsymbol{b}) = \min_{P\in\Pi(\boldsymbol{a},\boldsymbol{b})} \langle \boldsymbol{L}(\boldsymbol{C}_1, \boldsymbol{C}_2)^p \otimes P, P\rangle_{\mathcal{F}} - \varepsilon H(P) \tag{2.1.52}$$

$$= \min_{P\in\Pi(\boldsymbol{a},\boldsymbol{b})} \boldsymbol{\mathcal{E}}_{\boldsymbol{C}_1,\boldsymbol{C}_2}(P) - \varepsilon H(P) \tag{2.1.53}$$

für

$$\mathcal{E}_{\boldsymbol{C}_1,\boldsymbol{C}_2}(P) = \langle \boldsymbol{L}(\boldsymbol{C}_1, \boldsymbol{C}_2)^p \otimes P, P\rangle \tag{2.1.54}$$

Dieses nicht-konvexe Optimierungsproblem können wir mithilfe eines projektiven Gradienten-Verfahrens lösen, bei dem sowohl für den Gradienten-Schritt als auch die Projektion die KL-Divergenz verwendet wird [PCS16].

Die Iterationen sind gegeben durch

$$P \leftarrow Proj^{KL}_{\Pi(\boldsymbol{a},\boldsymbol{b})}\left(P \odot e^{-\tau(\nabla \boldsymbol{\mathcal{E}}_{\boldsymbol{C}_1,\boldsymbol{C}_2}(P) - \varepsilon\nabla H(P))}\right), \tag{2.1.55}$$

wobei die Schrittweite $\tau > 0$ und die KL-Projektion einer beliebigen Matrix K gegeben ist durch:

$$Proj^{KL}_{\Pi(\boldsymbol{a},\boldsymbol{b})}(K) := \operatorname*{arg\,min}_{P\in\Pi(\boldsymbol{a},\boldsymbol{b})} KL(P|K). \tag{2.1.56}$$

Es gilt außerdem

$$\nabla \boldsymbol{\mathcal{E}}_{C_1,C_2}(P) - \varepsilon\nabla H(P) = 2\boldsymbol{L}(\boldsymbol{C}_1, \boldsymbol{C}_2) \otimes P + \varepsilon \log(P). \tag{2.1.57}$$

Für den Fall $\tau = 1/\varepsilon$ wird aus der Iterationsvorschrift (2.1.55) die Lösung des entropisch regularisierten Transportproblems nach (2.1.17) mit den Grundkosten $2\boldsymbol{L}(\boldsymbol{C}_1, \boldsymbol{C}_2)^p \otimes P$ [PCS16].

Bemerkung 2.1.4.7 (Verbesserte Komplexität)**.** *Mit dem Resultat in Gleichung 2.1.49 können wir $\boldsymbol{L}(\boldsymbol{C}_1, \boldsymbol{C}_2) \otimes P$ effizient in der Größenordnung $\mathcal{O}(N_1^2 N_2 + N_2^2 N_1)$ mit ausschließlich Matrix/Matrix-Multiplikationen berechnen im Unterschied zur Komplexität von $\mathcal{O}(N_1^2 N_2^2)$ für die Implementierung von* (2.1.37)*.*

Bemerkung 2.1.4.8. *Die Iterationsvorschrift 2.1.55 definiert damit einen einfachen Algorithmus, der in jedem Update von P eine Sinkhorn-Projektion benötigt.*

Bemerkung 2.1.4.9. *Für gewöhnlich wählt man die Schrittweite τ im Gradientenverfahren nicht zu groß, um die Konvergenz des Verfahrens zu erhalten. Da die Schrittweite τ hier jedoch antiproportional zum Regularisierungs-Parameter ε ist, gibt es einen Trade-off zwischen der Konvergenz des Verfahrens und einer Unschärfe in der optimalen Lösung.*

2.1.4.3 Gromov-Wasserstein Baryzentren

In diesem Kapitel wollen wir uns mit dem „Mittelwert“ befassen, der elementar für das kMeans-Clustering (Unterabschnitt 2.4.2) ist. Auch hier soll der Mittelwert auf die abstrakte Ebene von gewichteten Distanzmatrizen angehoben werden. Dazu definieren wir im Folgenden sogenannte Wasserstein-Baryzentren und folgen [PCS16] für die Lösung des entstehenden Optimierungsproblems.

Definition 2.1.4.10 (Gromov-Wasserstein Baryzentrum)**.** *Seien die mit $(\lambda_s)_s$ gewichteten Distanzmatrizen $(C_s)_{s=1}^S$, mit $C_s \in \mathbb{R}^{N_s \times N_s}$, und die zugehörigen Histogramme $(\boldsymbol{a}_s)_s$ gegeben.*
Dann ist das Gromov-Wasserstein-Baryzentrum definiert durch

$$\min_{C \in \mathbb{R}^{N \times N}} \sum_s \lambda_s GW_\varepsilon(C, C_s, \boldsymbol{a}, \boldsymbol{a}_s). \tag{2.1.58}$$

Existenz- und Eindeutigkeitsresultate sind in [AC11] gegeben.

Bemerkung 2.1.4.11. *Wir gehen im Folgenden davon aus, dass sowohl die Histogramme $(\boldsymbol{a}_s)_s$ als auch das Histogramm $\boldsymbol{a}$ bekannt sind. Die Größe (N, N) des gesuchten Baryzentrums muss ebenfalls vorher festgelegt werden. Eine Erweiterung auf den Fall, dass auch p unbekannt sein sollte und damit als Optimierungsparameter aufgefasst wird, ist leicht möglich [PCS16].*

Wir können das Baryzentrum mithilfe von Transportplänen P umformulieren als

$$\min_{C, (P_s)_s} \sum_s \lambda_s (\mathcal{E}_{C, C_s}(P_s) - \varepsilon H(P_s)), \tag{2.1.59}$$

unter den Nebenbedingungen $\forall s : P_s \in \Pi(\boldsymbol{a}, \boldsymbol{a}_s) \subset \mathbb{R}_+^{N \times N_s}$. Für den Fall, dass L bezüglich der ersten Variable konvex ist, ist dieses Problem konvex bezüglich C. Bezüglich $(P_s)_s$ ist dieses Problem quadratisch aber nicht notwendigerweise positiv.
Das Problem nach (2.1.59) kann durch eine Block-Koordinaten-Relaxierung gelöst werden. Dabei wird iterativ abwechselnd bezüglich der Kopplungen $(P_s)_s$ und der Metrik C miminiert.

Minimierung bezüglich $(P_s)_s$. Anhand der Umformulierung (2.1.59) sehen wir, dass das Optimierungsproblem nach (2.1.58) bezüglich $(P_s)_s$ in S unabhängige GW_ε-Optimierungen

$$\forall s : \min_{P_s \in \Pi(\boldsymbol{a}, \boldsymbol{a}_s)} \mathcal{E}_{C,C_s}(P_s) - \varepsilon H(P_s) \tag{2.1.60}$$

zerfällt, die jeweils wie in Unterunterabschnitt 2.1.4.2 angegeben gelöst werden können.

Minimierung bezüglich C. Sei (P_s) gegeben. Dann lautet die Minimierung bezüglich C:

$$\min_C \sum_s \lambda_s \langle L(C, C_s) \otimes P, P \rangle. \tag{2.1.61}$$

Mit der folgenden Proposition erhalten wir für eine Klasse von Verlustfunktionen L eine Lösung in geschlossener Form.

Proposition 2.1.4.12. *[PCS16] Sei L eine Fehlerfunktion, die die Bedingung* (2.1.48) *erfüllt. Sei f_1'/h_1' invertierbar.*
Dann lässt sich die Lösung zu (2.1.61) *schreiben als:*

$$C = \left(\frac{f_1'}{h_1'}\right)^{-1} \left(\frac{\sum_s \lambda_s P_s^\top h_2(C_s) P_s}{\boldsymbol{a}\boldsymbol{a}^\top}\right), \tag{2.1.62}$$

wobei die Normalisierung $\sum_s \lambda_s = 1$ gilt.

Beweis. Nach (2.1.49) können wir das zu minimierende Funktional schreiben als

$$\sum_s \lambda_s \langle f_1(C)\boldsymbol{a}\mathbf{1}^\top + \mathbf{1}\boldsymbol{a}_s^\top f_2(C_s) - h_1(C) P_s h_2(C_s)^\top, P_s \rangle. \tag{2.1.63}$$

Die Optimalitätsbedingung erster Ordnung lautet folglich

$$f_1'(C) \odot \boldsymbol{a}\boldsymbol{a}^\top = h_1'(C) \odot \sum_s \lambda_s P_s h_2(C_s) P_s^\top. \tag{2.1.64}$$

Durch Umstellen der Gleichung erhalten wir die angegebene Form. □

Für den Fall $L = L_2$ wird aus dem Update nach (2.1.62) die folgende Vorschrift:

$$C \leftarrow \frac{1}{\boldsymbol{a}\boldsymbol{a}^\top} \sum_s \lambda_s P_s^\top C_s P_s. \tag{2.1.65}$$

Für den Fall $L = KL$ ergibt sich die folgende Verfahrensvorschrift:

$$C \leftarrow \left(\frac{1}{\boldsymbol{a}\boldsymbol{a}^\top} \sum_s \lambda_s P_s^\top log(C_s) P_s \right). \tag{2.1.66}$$

Algorithm 1 Berechnung der GW_ε-Baryzentren

Input: $(C_s, p_s), p$.
Output: C.
Initialize C.
repeat
  // Minimize over $(P_s)_s$
  for $s = 1$ **to** S **do**
    Initialize T_s.
    repeat
      // Compute $c_s = \mathcal{L}(C, C_s) \otimes P_s$ using (2.1.49)
      $c_s \leftarrow f_1(C) + f_2(C_s)^\top - h_1(C) P_s h_2(C_s)^\top$
      // Sinkhorn iterations to compute $L_{C_s}(p, q)$
      Initialize $a \leftarrow \mathbf{1}$, set $K \leftarrow e^{-c_s/\varepsilon}$.
      repeat
        $a \leftarrow \frac{p}{Kb}, b \leftarrow \frac{q}{K^\top a}$.
      until convergence
      $P_s \leftarrow diag(a) K diag(b)$.
    until convergence
  end for
  // Minimize over C
  $C \leftarrow \left(\frac{f_1'}{h_1'}\right)^{-1} \left(\frac{\sum_s \lambda_s P_s^\top h_2(C_s) P_s}{pp^\top}\right)$
until convergence

Psuedocode. In Algorithmus 1 ist die Berechnung der Baryzentren in Pseudocode angegeben. Dabei wird abwechselnd über P_s und C minimiert. Für die Minimierung über P_s wird ein projektives Gradienten-Verfahren verwendet, bei dem der Sinkhorn-Algorithmus 2 für die Berechnung der Projektion benutzt wird.

Algorithm 2 Sinkhorn-Algorithmus

Input: $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{C}, \varepsilon > 0$
Output: Optimal Coupling P^*.
Initialize $\boldsymbol{u}^{(0)}, \boldsymbol{v}^{(0)} = \mathbf{1}, \boldsymbol{K} = exp(-\frac{\boldsymbol{C}}{\varepsilon})$.
repeat
$\boldsymbol{u}^{(i)} = \boldsymbol{a} \oslash \boldsymbol{K}^\top \boldsymbol{v}^{(i-1)}$
$\boldsymbol{v}^{(i)} = \boldsymbol{b} \oslash \boldsymbol{K}\boldsymbol{u}^{(i-1)}$
until convergence

Wir haben in diesem Kapitel gesehen, wie mithilfe des Wasserstein-Baryzentrums eine Art Mittelwert für Wahrscheinlichkeitsverteilungen definiert werden kann.

Wir verfügen mit der diskreten p-GWD nun also über einen Distanzbegriff zwischen zwei verschiedenen Heatmaps (vgl. Unterabschnitt 2.5.3), die den zentralen Bestandteil des Detektionsverfahrens bilden.

2.2 Neuronale Netze

Neuronale Netze (NNs) gehören zur Kategorie des überwachten Lernens, mit denen eine Klassifikations- oder Regressionsaufgabe gelöst werden kann. Mit f_θ bezeichnen wir ein solches NN, das über die Parameter $\theta = (w, b)$ gegeben ist. Ausgehend von einem Datensatz $\mathcal{D} = \{(x_i, y_i) | i = 1, ..., N\}$, bestehend aus bekannten Ein-/Ausgabe-Paaren (x, y), wird das Netz trainiert, sodass es auf unbekannte Daten angewendet werden kann.
Wir betrachten im Folgenden einen Klassifikator, der durch ein NN gegeben ist. Dabei soll eine Eingabe x einer von K verschiedenen Klassen zugeordnet werden. Ein Netz können wir als eine Funktion $f_\theta : \mathbb{R}^d \to \mathbb{R}^K$ betrachten. Dabei beschreibt d die Dimension der Eingabe-Daten und K steht für die Anzahl der Klassen. Einer neuen Netzeingabe x ordnen wir die Klasse

$$\arg\max_{k=1,...,K} f(x)_k \tag{2.2.1}$$

zu.
Durch die Softmax-Funktion $\sigma : \mathbb{R}^K \to [0, 1]^K$, gegeben durch

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}} \text{ für } i = 1, ..., K \text{ und } x = (x_1, ..., x_K) \in \mathbb{R}^K, \tag{2.2.2}$$

lassen sich die K Ausgaben als Wahrscheinlichkeiten interpretieren, wodurch

sich zu einer Netzausgabe zusätzlich angeben lässt, wie sicher sich das NN bei einer bestimmten Entscheidung ist.
Für das Training unterteilen wir den Datensatz in einen Trainings-, Validierungs- und Test-Datensatz. Die beiden ersten werden während des Trainings verwendet, um die optimalen Netz-Parameter zu bestimmen. Dies geschieht in der Regel mithilfe eine stochastischen Gradienten-Verfahrens. Dabei werden die Netz-Parameter so optimiert, dass die Summe über die Fehler aller Datenpaare $(x, y) \in \mathcal{D}$ bezüglich einer Verlustfunktion $\mathcal{L}$ minimal ist, d.h. das folgende Problem wird gelöst:

$$\arg\min_{\theta} \sum_{i=1}^{N} \mathcal{L}(x_i, y_i, \theta). \tag{2.2.3}$$

Für das Gradienten-Verfahren werden die Gradienten des Netzes für verschiedene Eingaben benötigt. Die Bestimmung der Gradienten wird als Backpropagation bezeichnet.

Der während des Trainings nicht verwendete Test-Datensatz dient dazu, die Vorhersagequalität des Netzes auf einem für das Netz unbekannten Datensatz auszuwerten und damit eine Aussage über die Qualität des Netzes im Realbetrieb bei nie vorher gesehenen Daten treffen zu können.
Um den aufwendigen Trainingsprozess zu beschleunigen, kann das Training abgebrochen werden, sobald sich auf dem Validierungsdatensatz für eine vorher bestimmte Anzahl von Trainingsepochen keine Verbesserung des Fehlers ergeben hat. Dieses Abbruchkriterium wird als *early stopping* bezeichnet und die Epochenanzahl nach dem zuletzt besten Fehler wird mit dem Parameter *patience* angegeben.

NNs lassen sich schematisch als die Verknüpfung von Neuronen in verschiedenen Netzschichten darstellen. Im Folgenden beschreiben wir den typischen Aufbau von Netz, die vor allem im Bereich der Bildklassifikation verwendet werden.

NNs bestehen aus einer Eingabe- und einer Ausgabe-Schicht, zwischen denen sich wiederum mehrere Zwischenschichten befinden.
Bei Feed-Forward-Netzen wird die Information im Netz ausschließlich von links nach rechts, d.h. von der Eingabe- zur Ausgabe-Schicht, durch das Netz transportiert. Dabei können keine Netzschichten übersprungen oder mehrmals durchlaufen werden.
Wir bezeichnen mit l die $l-$te Schicht eines NN. Die Werte der Neuronen innerhalb einer Schicht l werden mit $x_i^{(l)}$, die der darauffolgenden Schicht

$l+1$ mit $x_j^{(l+1)}$ bezeichnet.

Um die Neuronen-Werte $x_i^{(l)}$ einer Schicht l an die Schicht $l+1$ weiterzugeben, werden diese mit den Kantengewichten $w_{ij}^{(l,l+1)}$ multipliziert und ein Bias-Term b addiert, bevor eine nicht-lineare Aktivierungsfunktion g angewendet wird. Für den Neuronen-Wert x_j erhalten wir:

$$x_j^{(l+1)} = g(\sum_i x_i^{(l)} w_{ij}^{(l,l+1)} + b_j^{(l+1)}). \tag{2.2.4}$$

Bei sogenannten ReLU-Netzen wird beispielsweise $g(z) = \max\{0, z\}$ als nicht-lineare Aktivierungsfunktion verwendet, sodass auch f_θ eine nicht-lineare Funktion ist. Diese Nichtlinearität ermöglicht die überragende Leistung von NNs.

Als lokale bzw. globale Vor-Aktivierungen werden die Werte $z_{ij} = x_i^{(l)} \cdot w_{ij}^{(l,l+1)}$ bzw. $z_j = \sum_i z_{ij} + b_j$ bezeichnet.

Eine schematische Darstellung eines kleinen NN ist in Abbildung 2.7 dargestellt. Dieses Netzen besitzt eine Eingabe-, eine Ausgabe- und eine Zwischenschicht.

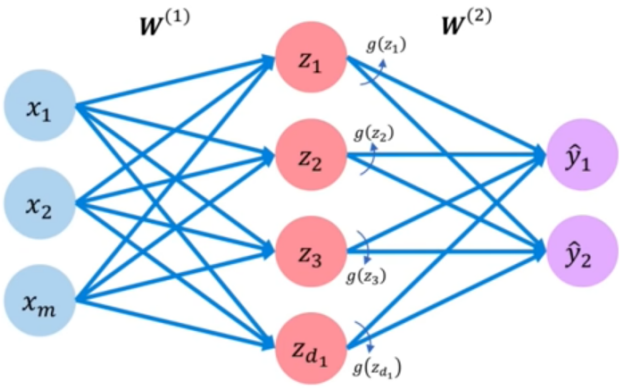

Abbildung 2.7: Beispiel eines kleinen Neuronalen Netzes

Quelle: [Ami21]

Der Parameter $\theta = (w, b)$ setzt sich zusammen aus den Kantengewichten w_{ij} zwischen den Neuronen zweier benachbarter Schichten l und $l+1$, die als Matrizen $W^{(l)}$ geschrieben werden können und den Biases $b^{(l)}$ pro Netz-Schicht. Es gilt also $w = [W^{(1)}, ...], b = [b^{(1)}, ...]$.

Unter einer Netz-Architektur verstehen wir ein NN, bei dem lediglich die Struktur festgelegt ist. Wir sprechen von einem Modell, wenn die einzelnen Parameter durch den Trainingsprozess optimiert und damit bestimmt wurden.

2.2.1 Convolutional Neural Networks

Bei Convolutional Neural Networks (CNNs) werden sogenannte Convolutional Layers als Netzschichten verwendet. Die Idee besteht darin, dass in einem Bild beispielsweise nahe beieinander gelegene Merkmale im Bild wichtiger sind, als weit auseinander liegende.

Der klassische Aufbau von CNNs besteht aus mehreren Convolutional Layers, zwischen denen sich sogenannte Pooling-Schichten befinden. Diese sich abwechselnden Schichten werden mit einer klassischen vollständig verknüpften Netzschicht abgeschlossen.

Convolutional Layer:
Bei den Convolutional Layers wird ein Filter über die Eingabe bewegt, siehe Abbildung 2.8, woraus eine sogenannte Feature-Map entsteht. Die Gewichte der Filter werden während des Trainings gelernt. Die Größe sowie die Schrittweite (stride), mit der ein Filter in der Raumrichtung verschoben wird, werden vor Beginn des Trainings festgelegt.

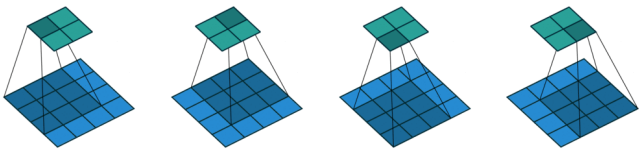

Abbildung 2.8: Funktionsweise einer Convolutional layer mit Filtergröße 2×2 und Schrittweite $stride = 1$

Quelle: [Mat21]

Pooling Layer:
Auf diesen Feature-Maps werden anschließend sogenannte Pooling-Operationen durchgeführt. Dabei wird beispielsweise ein Average-, Sum- oder Max-Pooling verwendet, um die Dimension einer Feature-Map zu verringern. Wird zum Beispiel für das Max-Pooling ein Fenster der Größe 2×2 ausgewählt, so wird eine Feature-Map in der Breite und Höhe halbiert, indem das Maximum über diese vier Felder ausgewählt wird.

Abbildung 2.9: Inception-Module

Quelle: [SLJ+15]

Ein weiterer Vorteil von CNNs neben den sehr guten Ergebnissen im Bereich der Bild-Klassifikation besteht darin, dass durch die reduzierte Anzahl an Verbindungen zwischen Neuronen zweier benachbarter Netzschichten die Anzahl der Parameter im Vergleich zu NNs, die nur aus vollständig verknüpften Schichten bestehen, deutlich reduziert werden kann.

2.2.2 Inception-Netze

Bei Inception-Netzen werden nun mehrere Convolution-Operationen nicht nur nacheinander, sondern nebeneinander ausgeführt. Die Ausgaben werden anschließend wieder zusammengefasst (Merging). Diese Art von Schicht bildet das sogenannte Inception-Modul, das für die besondere Struktur der Inception-Netze verantwortlich ist. Dieses wird wieder abwechselnd mit Average- und Max-Pooling-Schichten ausgeführt. Die ursprünglichen Versionen des Inception-Moduls sind in Abbildung 2.9 abgebildet.
In der zweiten Version der Inception-Netze werden 1×1-Convolutions zur Dimensionsreduktion benutzt, um den Rechenaufwand weiter zu senken. Durch das sehr tiefe NN entstand auch hier das Problem, dass die Gradienten bei der Backpropagation verschwinden. Um dies zu verhindern, wurden zwei zusätzliche Hilfs-Klassifikatoren an verschiedenen Stellen im NN implementiert, die während des Trainings einen Fehler für die Backpropagation beisteuern. Nach dem abgeschlossenen Training werden diese Hilfsklassifikatoren nicht mehr benutzt. Das vollständige NN ist in Abbildung 2.10 dargestellt.

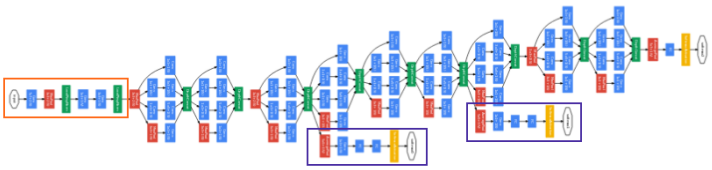

Abbildung 2.10: Inception v1

Quelle: [SLJ+15]

2.3 Poisoning-Angriffe

Während NNs für schwierige Aufgaben vollautomatisch hochpräzise Entscheidungen treffen, sind diese leider gleichzeitig durch einen Angreifer sehr leicht manipulierbar. Bei Poisoning-Angriffen manipuliert ein Angreifer den Datensatz, der von einem Opfer für das Training eines NN verwendet wird. Ein Angreifer möchte das Training so beeinflussen, dass die Vorhersagequalität des NN im Realbetrieb allgemein verringert ist oder nur bestimmte Eingaben falsch klassifiziert werden. Eine Spezialform der letzteren Angriffsmöglichkeit sind sogenannte Backdoor-Poisoning-Angriffe, die eine Hintertür im NN implementieren, die im Realbetrieb ausgenutzt werden kann. Dabei werden dann nur diejenigen Bilder einer bestimmten Klasse einer anderen bestimmten Klasse falsch zugeordnet, die einen Auslöser besitzen, der vom NN erkannt wird. Im Unterschied zu Adversarialen Angriffen, die nach dem Training konstruiert und benutzt werden, finden die PAs bereits während des Trainings statt, um nach dem Training ausgenutzt zu werden. Nach einem erfolgreichen PA arbeitet das Opfer folglich mit einem manipulierten/korrumpierten NN.

Kenntnisse des Angreifers: Wir gehen davon aus, dass der Angreifer volle Kenntnis über die Netzwerkarchitektur und Zugriff auf den Datensatz besitzt.

Ziel des Angreifers: Der Angreifer verfolgt das Ziel, während des Trainings eine Hintertür im NN zu implementieren, die über den Auslöser im Realbetrieb genutzt werden kann. Außerdem möchte der Angreifer möglichst wenig Veränderungen am Datensatz durchführen, d.h. nur wenige Datenpunkte sollen manipuliert werden, während der Angriff noch immer erfolgreich funktioniert.
Ebenfalls wichtig für den Angreifer ist es, dass das NN in Abwesenheit eines

Auslösers im Realbetrieb möglichst ähnlich gut wie dasselbe Modell, trainiert auf nicht korrumpierten Daten, funktioniert. Andernfalls würde sich bereits während des Trainings ein Hinweis darauf ergeben, dass der Datensatz manipuliert ist.

2.3.1 Standard-Poisoning-Angriffe

Bei sogenannten ungerichteten Poisoning-Angriffen (PAs), soll durch die Manipulation des Datensatzes die Vorhersagequalität des NN abnehmen.

Bei gerichteten PAs bestimmt der Angreifer eine Start- und Zielklasse und manipuliert den Datensatz so, dass Elemente x_{source} der Klasse y_{source} fälschlicherweise der Klasse y_{target} zugeordnet werden.
Für die spezielleren Backdoor-PAs soll eine Hintertür im NN f_θ implementiert werden, die später so ausgenutzt werden kann, dass die oben beschriebene falsche Zuordnung nur in Anwesenheit des Auslösers geschieht. Wir können dies wie folgt formulieren: Sei $f_{\theta,p}$ das manipulierte NN und $x_{source,p}$ ein Element der Klasse y_{source}, welches im Unterschied zu x_{source} derselben Klasse einen Auslöser enthält. Es soll dann gelten:

$$f_{\theta,p}(x_{source,p}) = y_{target}$$
$$f_{\theta,p}(x_{source}) = y_{source}$$

Die erste Aussage verlangt, dass ein Element mit einem Auslöser vom manipulierten NN der Zielklasse zugeordnet wird. Die zweite Aussage fordert, dass die Anwendung des manipulierten NN auf den nicht-korrumpierten Datenpunkten keine andere Entscheidung als die Anwendung des nicht manipulierten NN zur Folge hat. D.h. in Abwesenheit des Auslösers funktioniert das NN wie gewünscht. Dies ist einer der Gründe, weshalb es so schwierig ist, PAs zu detektieren.
Um einen solchen Backdoor-Poisoning-Angriff zu implementieren, werden zwei Klassen als Start- und Zielklasse ausgewählt und bei einem bestimmten Anteil der Bilder in der Startklasse wird der Auslöser eingefügt und das Label auf die Zielklasse abgeändert.

Ein Beispiel eines manipulierten Bildes ist in Abbildung 2.11 abgebildet. Dabei wird ein Auslöser in Form eines Stickers auf dem Verkehrsschild eingefügt und das Label auf die Klasse 80 km/h abgeändert.

In diesem Fall von PAs spielt es keine Rolle, ob der Angreifer die Netzarchitektur kennt. Wichtig ist nur, dass dieser Zugriff auf den Datensatz besitzt.

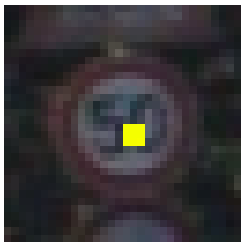

Abbildung 2.11: Korrumpierte Datenpunkte aus der Start-Klasse '50 km/h' mit gelb-grünem Sticker und dem Label '80 km/h' der Zielklasse.

Wir befassen uns im Folgenden ausschließlich mit Backdoor-Poisoning-Angriffen und sprechen deshalb nur von Standard-Angriffen.

2.3.2 Label-konsistente Poisoning-Angriffe

Bei den vorherigen Standard-Angriffen war es der Fall, dass das Label und das entsprechende Bild nicht mehr zusammenpassen. Ein händisches Durchsuchen des Datensatzes (wenn auch sehr aufwendig) könnte damit ebenfalls zur Detektion eines Angriffs führen.
Eine deutlich schwieriger zu detektierende Art von PAs sind sogenannte Label-konsistente Poisoning-Angriffe (LkPAs), bei denen genau diese Schwachstelle eliminiert ist, d.h. Label und Bild passen wieder zueinander, während der Angriff noch immer erfolgreich funktioniert.
Auch hier wird wieder ein Auslöser im Bild eingefügt. Auf die Abänderung des Labels wird nun jedoch verzichtet.
Es ist das Ziel, ein Bild vor dem Anbringen des Auslösers so zu modifizieren, dass es für das menschliche Auge noch immer zur entsprechenden Klasse gehört, für das NN aber so schwierig zu klassifizieren ist, dass es sich auf den Auslöser anstatt auf das ursprüngliche Bild verlässt. Im Anschluss wird wieder ein Auslöser eingefügt.
In [TTM19] werden zwei Verfahren vorgestellt, die einzelne Bilder so stören können, dass die Klassifikation erschwert wird. Das erste Verfahren besteht aus einer Einbettung in einen niedrig-dimensionalen Raum, das auch bei Autoencodern [BKG20] verwendet wird.
Beim zweiten Verfahren wird ein sogenannter Projektiver Gradienten-Abstieg-Angriff durchgeführt. Dabei wird ein Adversarialer Angriff in leicht abgewandelter Form genutzt, um eine Netzeingabe x zu stören, sodass diese vom NN falsch klassifiziert wird, für das menschliche Auge jedoch noch immer zur

selben Klasse gehört. Im Unterschied zu Adversarialen Angriffen, bei denen die falschen Klassifikationen nach dem Training eines NN stattfinden, sollen die falschen Klassifikationen in diesem Fall bereits während des Trainings des angegriffenen NN stattfinden. Diese Störungen lassen sich auch von einem Modell oder sogar einer Architektur auf andere übertragen [SZS+13,PMG16]. Zu beachten ist hier, dass für die Manipulation der Datenpunkte ein NN benötigt wird. Im Idealfall kennt der Angreifer die Netzarchitektur des Opfers und kann damit dasselbe NN für die Implementierung des Angriffs verwenden. Durch die leichte Übertragbarkeit von einem NN auf ein anderes, lässt sich dieser Angriff jedoch auch ohne die Kenntnis der vom Opfer ausgewählten Architektur implementieren.
Für ein auf nicht manipulierten Daten trainiertes NN f_θ mit Verlustfunktion $\mathcal{L}$ und ein Eingabe-Paar (x, y), wird die modifizierte Version von x konstruiert als

$$x_{adv} = \underset{||x'-x||_p \leq \varepsilon}{\arg\max}\, \mathcal{L}(x', y, \theta), \tag{2.3.1}$$

für $p > 1$ und $\varepsilon > 0$.

Dieses Optimierungsproblem wird mit einem projektiven Gradienten-Verfahren [MMS+17] wie folgt gelöst:
Für jeden Datenpunkt x beschreibt $\mathcal{S} \subset \mathbb{R}^d$ die Menge aller zulässigen Störungen. Das Optimierungsproblem wird mit der Iteration

$$x^{t+1} = \Pi_{x+\mathcal{S}}(x^t + \alpha sgn(\nabla_x \mathcal{L}(x, y, \theta))) \tag{2.3.2}$$

für eine Schrittweite $\alpha > 0$ und eine Anzahl an Iterationen n gelöst. Dabei bezeichnet $\Pi_{x+\mathcal{S}}(\cdot)$ den Projektionsschritt auf die Menge aller zulässigen Abweichungen von x.
Im Unterschied zu [TTM19] ändern wir nur Bilder im Datensatz ab und fügen nicht zusätzlich zum Original x auch x_{adv} hinzu. Damit ändert sich die Anzahl an Datenpunkten durch den Angriff nicht.

Für den Auslöser können wir denselben wie im Standard-Fall oder einen schwarz-weißen Sticker der Größe 3×3 benutzen, der in Lesereihenfolge die Farben ssw-sws-wsw (s:schwarz, w:weiß) besitzt. Damit haben wir einen Angriff implementiert, bei dem Label und Bild wieder zusammenpassen. In einem weiteren Schritt ist es möglich, die Sichtbarkeit des Auslösers zu verringern. Dabei werden nun nicht die Pixel-Werte $v_i \in \{0, 255\}^3$ direkt ersetzt, sondern eine Amplitude wird auf allen drei Farbkanälen addiert bzw. subtrahiert. Auf den vorher schwarzen Feldern wird $amp \in \{16, 32, 64, 128, 256\}$ addiert und auf den vorher weißen Feldern subtrahiert. Für $amp = 256$ ergibt sich der Fall, bei dem die Pixel-Werte durch schwarze bzw. weiße Pixel

getauscht werden. Im entsprechenden Paper [TTM19] wird auch ein Auslöser eingefügt, bei dem sich jeweils ein solcher Sticker in jeder Bildecke befindet.

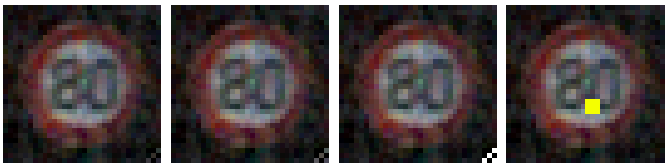

Abbildung 2.12: Korrumpierte Datenpunkte mit Amplitudensticker unten rechts für $amp = 32, 64, 255$ und dem Standard-Auslöser (rechts).

2.3.3 Bewertung von Poisoning-Angriffen

Um verschiedene PAs miteinander vergleichen zu können, benötigen wir ein Maß dafür, wie erfolgreich ein bestimmter Angriff funktioniert.
Für den Fall der Standard-Angriffe wird der gewählte Auslöser bei allen Bildern der Startklasse im Testdatensatz eingefügt und der Anteil an Bildern bestimmt, die aus der Startklasse stammen und in Anwesenheit des Auslösers der Zielklasse zugeordnet werden.
Dieser Anteil definiert die sogenannte Angriffserfolgsrate (AER). Für die LkPAs werden im Test-Datensatz alle Bilder mit dem entsprechenden Auslöser versehen und es kann eine AER pro Klasse berechnet werden. Es ist zu beachten, dass für Angriffe, die mit einer reduzierten Amplitudenstärke durchgeführt werden, die Bilder im Test-Datensatz dennoch mit Auslösern mit voller Amplitudenstärke versehen werden. Wir bewerten die Qualität eines LkPA über die mittlere Angriffserfolgsrate (mAER) über alle Klassen außer der Zielklasse.

2.4 Detektionsverfahren

In diesem Kapitel beschäftigen wir uns mit gängigen Methoden zur Detektion von PAs und geben am Ende einen kurzen Ausblick auf die Idee für einen neuen Ansatz. Wir wollen beide Arten von PAs erfolgreich detektieren.

2.4.1 Detektion eines Poisoning-Angriffs

Es muss zunächst diejenige Klasse bestimmt werden, für die der Verdacht besteht, dass über diese ein PA durchgeführt wurde.

Anschließend kommt ein Clustering-Algorithmus zum Einsatz, der die Daten im Idealfall in in zwei Cluster aus sauberen Datenpunkten und korrumpierten Datenpunkten aufteilt.

Zur Bestimmung, ob eine Klasse korrumpierte Daten enthält, kann das Ergebnis des Clusterings mit den folgenden Methoden untersucht werden:

Vergleich der relativen Größe: Eine Möglichkeit, korrumpierte Datenpunkte zu erkennen, ist der Vergleich der relativen Größen der beiden Cluster. Laut [CCB+18] liegt die relative Größe bei nicht korrumpierten Klassen bei ca. 50 Prozent. Bei korrumpierten Daten und einem erfolgreichen Clustering würde die relative Größe dann dem prozentualen Anteil an korrumpierten Datenpunkten entsprechen.

Silhouettenkoeffizient: Eine weitere Möglichkeit besteht darin, die Qualität des Clusterings mit Hilfe des Silhouette-Koeffizienten zu beschreiben. Dieser gibt an, wie gut ein Clustering zu den gegebenen Datenpunkten mit den entsprechenden Labeln passt und ist wie folgt definiert: Sei das Ergebnis eines Clustering-Algorithmus mit verschiedenen Clustern gegeben. Zu einer Beobachtung x im Cluster A wird die Silhouette

$$s(x) = \frac{d(B,x) - d(A,x)}{\max\{d(A,x), d(B,x)\}} \tag{2.4.1}$$

definiert, wobei $d(A,x) = \frac{1}{n_A - 1} \sum_{a \in A, a \neq x} d(a,x)$ dem mittleren Abstand einer Beobachtung innerhalb dieses Clusters zu allen anderen Beobachtungen dieser Klasse entspricht. Dabei steht n_A für die Anzahl der Beobachtungen in Cluster A. $d(B,x) = \min_{C \neq A} d(C,x)$ beschreibt die Distanz von x zum nächstgelegenen Cluster B. Der Silhouetten-Koeffizient SC ist nun definiert als

$$SC = \max_k \tilde{s}(k), \tag{2.4.2}$$

wobei $\tilde{s}(k)$ der Mittelwert der Silhoutten aller Datenpunkte im gesamten Datensatz unter Verwendung von k Clustern ist. Damit ist der Silhouttenkoeffizient ein Maß dafür, wie gut ein Clustering für eine vorher fixierte Clusteranzahl k zum Datensatz passt.

Exklusives Retraining: Beim exklusiven Retraining wird das NN von Grund auf neu trainiert. Das oder die verdächtigen Cluster werden beim erneuten Training nicht verwendet. Mithilfe des neu trainierten NN werden dann anschließend die vorenthaltenen, verdächtigen Cluster klassifiziert. Falls das

Cluster Aktivierungen von Datenpunkten enthält, die zum Label des Datenpunktes gehören, erwarten wir, dass die Vorhersage des NN mit dem Label übereinstimmen. Gehören die Aktivierungen eines Datenpunktes im verdächtigen Cluster jedoch zu einer anderen Klasse als die durch das Label angedeutete Klasse, so sollte das NN den Datenpunkt einer anderen Klasse zuordnen. Um nun zu entscheiden, ob ein verdächtiges Cluster korrumpiert oder nicht korrumpiert ist, wird wie folgt vorgegangen: Sei l die Anzahl an Vorhersagen, die zum Label des Datenpunktes passen. Sei p die größte Anzahl an Vorhersagen, die für eine weitere Klasse C sprechen, wobei C nicht die Klasse mit den Labeln des zu untersuchenden Clusters ist. Der Quotient $\frac{l}{p}$ gibt dann an, ob das Cluster korrumpiert ist oder nicht: Es wird ein Schwellenwert $T > 0$ gesetzt. Gilt $\frac{l}{p} < T$, wurden mehr Datenpunkte einer anderen Klasse zugeordnet und das Cluster wird als korrumpiert deklariert. Umgekehrt wird das verdächtige Cluster im Fall von $\frac{l}{p} > T$ als nicht korrumpiert eingestuft.

2.4.2 kMeans-Clustering

Das kMeans-Clustering gehört zu den unüberwachten Clustering-Verfahren. Bei diesen Verfahren existieren keine Datenpaare, die für eine Art Training verwendet werden können. In der ursprünglichen Formulierung von kMeans sind n Datenpunkte $x_1, ..., x_n \in \mathbb{R}^d$ und $k \in \mathbb{Z}_{>0}$ gegeben. Die Datenpunkte sollen so auf die einzelnen Cluster verteilt werden, dass die Distanzen innerhalb eines Clusters zum Cluster-Zentrum minimal sind. Für das kMeans-Clustering müssen wir im Vorfeld eine Clusteranzahl k fixieren und die Mittelpunkte der k Cluster initialisieren.

In Lloyd's Formulierung [Llo82] wird mit einer gleich-verteilten zufälligen Initialisierung von k Cluster-Zentren begonnen. Jeder Punkt im Datensatz wird anschließend dem nächstgelegenen Cluster-Zentrum zugeordnet. Für jedes Cluster wird anschließend ein neues Zentrum als Mittelwert berechnet.
Diese beiden Schritte aus Zuordnung und Mittelpunktberechnung werden so lange wiederholt, bis sich die Cluster-Zentren nicht mehr ändern oder eine maximale Iterationszahl erreicht ist.
Eine Variation des kMeans-Algorithmus, der sowohl die Laufzeit als auch die Genauigkeit verbessert, ist der sogenannte kMeans++-Algorithmus [AV06]. Dabei wird die Initialisierung in abgewandelter Form durchgeführt. Nach gleichverteilter, zufälliger Bestimmung des ersten Cluster-Zentrums werden die übrigen $k - 1$ Zentren wie folgt gewählt: Die Wahl erfolgt proportional zur maximalen quadratischen Distanz zu allen vorher bestimmten Cluster-Zentren. Damit sind die Zentren so über den Datensatz verteilt, dass sie maximal weit auseinander liegen.

Anschließend werden die im kMeans-Algorithmus üblichen Schritte aus Distanzberechnung und Mittelwert-Berechnung wiederholt.

Das kMeans++-Clustering ist nochmals in Algorithmus 3 dargestellt.

Algorithm 3 kMeans-Algorithmus

Input: Data points $\mathcal{X} = \{x_1, ..., x_n\}$, Number of Clusters k, distance function d, where $D(x)$ is the minimal distance $d(x, c)$, where c are the already choosen cluster centers.

Output: k Cluster centers $\mathcal{C} = \{c_1, ..., c_k\}$.

Arbitrarily choose k initial centers $\mathcal{C} = c_1, ..., c_k$

repeat

For each $i \in \{1, ..., k\}$, set the cluster C_i to be the set of points in $\mathcal{X}$ that are closer to c_i than they are to c_j for all $j \neq i$.

For each $i \in \{1, ..., k\}$, set c_i to be the centere of mass of all points in $C_i : c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$.

until convergence

Die Initialisierung für den kMeans++-Algorithmus ist in Algorithmus 4 notiert.

Algorithm 4 kMeans++-Initalisierung

Input: Data points $\mathcal{X} = \{x_1, ..., x_n\}$, Number of Clusters k.

Output: k Clusters.

Choose an initial center c_1 uniformly at random from $\mathcal{X}$.

repeat

Choose the next center c_i, selecting $c_i = x' \in \mathcal{X}$ with probability $\frac{D(x')^2}{\sum_{x \in \mathcal{X}} D(x)^2}$.

until we have choosen a total of k centers

Es bleibt also die Bestimmung des Wertes k. Dafür benutzen wir die Spektralzerlegung, die wir im nächsten Abschnitt vorstellen.

2.4.3 Spektrales Clustering/Spektrale Zerlegung

Wir gehen davon aus, dass uns ein metrischer Raum (X, d) gegeben ist, wobei $X = \{x_i, ..., x_n\}$ gilt. In [VL07] wird der metrische Raum als Graph $G = (V, E)$ interpretiert. Damit wird aus dem Clustering-Problem ein äquivalentes Partitionierungsproblem in k Partitionen:

Finde eine Partitionierung des Graphen G, sodass die Kantengewichte innerhalb eines Clusters minimal sind.

Um aus dem metrischen Raum (X, d) einen Graphen zu erhalten, müssen zwischen den Knoten des Graphen Kanten definiert werden. Dafür existieren die folgenden drei Möglichkeiten:

- ε-Nachbarschaftsgraph: Hier wird eine Kante zwischen zwei Knoten u und v gesetzt, falls $d(u, v) \leq \varepsilon$ für $\varepsilon > 0$ gilt.
- kNN-Graph: Hier wird eine Kante zwischen zwei Knoten u und v gesetzt, falls v unter den k vielen Knoten ist, die am nächsten an u liegen.
- Vollständiger Graph: Hier wird eine Kante zwischen allen Knoten im Graphen gesetzt.

Analog zu [LWB+19] entscheiden wir uns für einen kNN-Graphen mit $k = 10$. Die Adjazenzmatrix $A = (a_{ij})_{ij}$ eines Graphen G ist definiert durch

$$a_{ij} = \begin{cases} 1, & \text{falls } u \text{ und } v \text{ verbunden sind,} \\ 0, & \text{sonst.} \end{cases} \tag{2.4.3}$$

Ausgehend davon lässt sich die zu einem Graphen G gehörige Laplace-Matrix definieren:

Definition 2.4.3.1 (Laplace-Matrix)**.** *Sei $G = (V, E)$ ein Graph mit Adjazenzmatrix A. Sei $D = (d_{ij})_{ij}$ eine Diagonalmatrix mit $d_i = \sum_{j=1}^{n} a_{ij}$. Sei $L = D - A$ die nicht normalisierte Laplace-Matrix. Dann lässt sich die symmetrische und normalisierte Laplace-Matrix L_{sym} definieren als*

$$L_{sym} = D^{-1/2} L D^{-1/2}. \tag{2.4.4}$$

Wir zitieren das folgende Resultat:

Theorem 2.4.3.2 (Anzahl an zusammenhängenden Komponenten und Spektrum von L_{sym} [VL07])**.** *Sei G ein ungerichteter Graph mit nicht-negativen Gewichten. Dann gibt die Vielfachheit der Eigenwerte 0 von L_{sym} die Anzahl an zusammenhängenden Komponenten $A_1, ..., A_k$ an.*

Damit können wir also die optimale Clusteranzahl k durch eine spektrale Zerlegung bestimmen und anschließend das Clustering durchführen.

2.4.4 Clustering-Verfahren

Alle in dieser Arbeit vorgestellten Detektionsverfahren basieren auf der Idee, die Daten in einen korrumpierten und einen unkorrumpierten Anteil aufzuteilen. Dafür wird jeweils ein Clusteringverfahren verwendet, das unterschiedliche Repräsentationen der Daten benutzt.

Bemerkung 2.4.4.1. *Um von einer verdächtigen Klasse in einem Datensatz sprechen zu können, muss auch diese erst einmal identifiziert werden. In [AMN+19] Kapitel '2.3. Fisher Discriminant Analysis for Clever Hans identification' wird ein Verfahren vorgestellt, mit dem verdächtige Klassen identifiziert werden können. Für diese würde man anschließend das Heatmap-Clustering durchführen. Wir gehen im Folgenden davon aus, dass wir die verdächtige Klasse bereits identifiziert haben.*

Im Folgenden stellen wir drei solcher Clustering-Verfahren vor.

2.4.4.1 Clustering auf den Rohdaten

Der einfachste Ansatz, um korrumpierte Datenpunkte zu erkennen, ist ein kMeans-Clustering, das direkt (bzw. nach einer Dimensionsreduktion) auf den Eingabedaten einer Klasse durchgeführt wird.

2.4.4.2 Activation-Clustering

Das Activation-Clustering (AC) als Verteidigungsstrategie basiert auf der Annahme, dass bestimmte Schichten innerhalb des NN die Entscheidung, dass ein Bild mit einem Auslöser falsch klassifiziert wird, sehr gut codieren. Für die Detektion der Hintertüren im Datensatz sollen nun genau diese Aktivierungen als Grundlage für das kMeans-Clustering herangezogen werden. Das AC wird erstmalig in [CCB+18] vorgestellt und nutzt aufgrund experimenteller Untersuchungen stets die Aktivierungen der vorletzten Netzschicht. Eine Kombination von Aktivierungen mehrerer Schichten wäre ebenfalls denkbar.

2.4.4.3 Heatmap-Clustering

In diesem Abschnitt stellen wir das Heatmap-Clustering (HC) vor, mit dem PAs detektiert werden können. Dieses von uns neu entwickelte Verfahren stellt den Kern dieser Arbeit dar und funktioniert wie im Folgenden dargestellt.

Grundsätzliches Vorgehen

Die Idee besteht darin, ein kMeans-Clustering auf einer Klasse durchzuführen, die in dem Sinne für verdächtig gehalten wird, dass innerhalb dieser Klasse Datenpunkte eingefügt wurden, die nicht zum ursprünglichen Datensatz gehören und eine Hintertür im NN implementieren sollen.

Wichtig für das Clustering sind hierbei die Repräsentation der Daten, auf denen geclustert wird, sowie die Metriken, die eine Distanz und damit auch einen Mittelwertbegriff für mehrere Datenpunkte definieren.
Als Repräsentation benutzen wir die Heatmaps, die mit der LRP, wie in Unterabschnitt 2.5.3 beschrieben, erzeugt werden.

Dieses Verfahren besteht also darin, eine Spektrale Relevanz Analyse [LWB+19] durchzuführen, die aus einem Spektralen Clustering auf Heatmaps durchgeführt wird, die mit der LRP berechnet wurden.

Wir können die einzelnen Schritte des Verfahrens wie folgt zusammenfassen:

- Berechnung der Heatmaps mithilfe der LRP (vgl. Unterabschnitt 2.5.3)
- Berechnung einer Distanzmatrix
- Spektrale Zerlegung (vgl. Unterabschnitt 2.4.2)
  (Bestimmung der verschiedenen Cluster innerhalb einer Klasse)
- kMeans-Clustering zur Bestimmung der korrumpierten Datenpunkte (vgl. Unterabschnitt 2.4.2)

Verwendete Distanzen und Approximationen

Bisher sind wir nicht näher auf die Distanz zwischen zwei verschiedenen Heatmaps eingegangen. Um die Struktur innerhalb einer Klasse zu analysieren, benötigen wir jedoch eine Metrik.
Die naheliegendste Möglichkeit von einer Distanz zwischen zwei Heatmaps zu sprechen, ist vermutlich eine L^p-Distanz, bei der die Pixelwerte die einzelnen Komponenten darstellen.
Eine andere Idee ist die Verwendung eines Distanzbegriffs, der im Unterschied zur L^p-Distanz auch rotations- und translationsinvariant ist. Die Vermutung ist hier, dass wir dadurch einen Distanzbegriff erhalten, der eher mit der menschlichen Wahrnehmung übereinstimmt und sich besser für das Clustering eignet. Dafür verwenden wir die in Abschnitt 2.1 vorgestellte GWD und berechnen eine Approximation wie in Unterunterabschnitt 2.1.4.2 angegeben.

2.4.5 Bewertung von Detektionsverfahren

Die Bewertung eines Detektionsverfahrens entspricht der Bewertung des zugrundeliegenden Clustering-Verfahrens. Für die Implementierung der PAs wie in Abschnitt 2.3 beschrieben, ergeben sich innerhalb der verdächtigen Klassen genau zwei Cluster. Wir führen deshalb ein kMeans-Clustering für $k = 2$ Cluster durch. Das kMeans-Clustering weist die einzelnen Datenpunkte der unkorrumpierten oder korrumpierten Klasse zu. Wir bezeichnen den korrumpierten Fall als positives Ergebnis und den unkorrumpierten Fall als negatives Ergebnis.

Für ein gegebenes Clustering und unter Kenntnis der Grundwahrheiten bezüglich der Manipulation, können wir die folgenden Größen definieren:

TP (true positives) steht für die Anzahl an positiven Zuordnungen bei positiver Grundwahrheit, d.h. ein korrumpiertes Bild wird richtigerweise als korrumpiert deklariert.

TN (true negatives) beschreibt die Anzahl an negativen Zuordnungen bei negativer Grundwahrheit.

Für die falschen Zuordnungen zählen wir mit ***FP*** (false positives) die Anzahl der positiven Zuordnungen, obwohl die Grundwahrheit negativ ist.

Umgekehrt steht ***FN*** (false negatives) für die Anzahl an negativen Zuordnungen bei positiver Grundwahrheit.

Damit lassen sich für genau zwei gesuchte Cluster die folgenden Größen definieren, um die Qualität des Clusterings zu beschreiben:
Die Genauigkeit (accuracy) der Vorhersage ist definiert durch den Quotienten

$$ACC = \frac{TP + TN}{TP + FN + FP + TN}. \tag{2.4.5}$$

Um zu beschreiben, wie gut korrumpierte Datenpunkte als korrumpiert erkannt werden, verwenden wir die True Positive Rate

$$TPR = \frac{TP}{TP + FN}. \tag{2.4.6}$$

Analog beschreibt die True Negative Rate

$$TNR = \frac{TN}{TN + FP}, \tag{2.4.7}$$

wie gut das Clustering nicht korrumpierte Datenpunkte als nicht korrumpiert erkennt.

2.5 Erklärbare Künstliche Intelligenz

Unter dem Begriff Erklärbare Künstliche Intelligenz werden Methoden zusammengefasst, die einzelne Modelle oder Entscheidungen eines Modells erklären.

Bei unkritischen Anwendungen wie beispielsweise Produkt- und Filmempfehlungen oder maschineller Übersetzung spielt die Frage, wie das KI-System zu einer Entscheidung bzw. Ausgabe gekommen ist, keine entscheidende Rolle. Anders sieht es jedoch bei Anwendungen in sicherheitskritischen Bereichen aus. Hier kann es sein, dass die Erklärung für das Zustandekommen einer Entscheidung genauso wichtig wie die Entscheidung selbst [BBM+16] ist. Zudem sollte die verwendeten Modelle auch nicht diskriminierend sein und durch die Datenschutzgrundverordnung (DSGVO) besitzt der Nutzer sogar ein Recht auf Erklärbarkeit.

Es wird zwischen sogenannten White-Box- und Black-Box-Modellen unterschieden. Bei White-Box-Modellen lassen sich die algorithmischen Zusammenhänge sehr leicht nachvollziehen und die getroffenen Entscheidungen sind daher selbsterklärend. Dazu gehören beispielsweise Lineare Modelle, Regelsysteme oder Entscheidungsbäume.

Bei Black-Box-Modellen wie NNs ist es aufgrund ihrer Verflechtung und Vielschichtigkeit in der Regel nicht mehr möglich, die innere Funktionsweise des Modells nachzuvollziehen. Zumindest für die Erklärung von Einzelentscheidungen (lokale Erklärbarkeit) können dann jedoch zusätzliche Erklärungswerkzeuge eingesetzt werden, um nachträglich die Nachvollziehbarkeit zu erhöhen.

Lipton [Lip16] führt eine grundsätzliche Begriffserklärung ein. Dabei wird zwischen den Begriffen Transparenz (oder Interpretierbarkeit) und Erklärbarkeit wie folgt unterschieden [DTK21].

Transparenz:
Transparenz wird als eine Modelleigenschaft verstanden. Ist die Transparenz eines Modells gegeben, so ist es unter der Annahme nachvollziehbarer Eingangsgrößen selbsterklärend. Die Eigenschaft der Transparenz wird in [Lip16]

auf drei verschiedenen Ebenen definiert. Dabei wird zwischen Simulierbarkeit (Ebene des gesamten Modells), Unterteilbarkeit (Ebene einzelner Modellkomponenten, z.B. Parameter) und Algorithmischer Transparenz unterschieden. Es wird von einer hierarchischen Abhängigkeit ausgegangen, sodass die Simulierbarkeit eines Systems dessen Unterteilbarkeit und dessen algorithmische Transparenz impliziert.
Ein System heißt simulierbar, wenn auch eine Person die Entscheidungen des zugrundeliegenden Algorithmus in angemessener Zeit nachvollziehen kann, indem sie alle notwendigen Berechnungsschritte selbst ausführt [Lip16].
In einem unterteilbaren System können die einzelnen Komponenten (Eingangsdaten, Parameter, Modellebenen, Berechnungen etc.) mit einer intuitiven Beschreibung versehen werden, sodass ihre Funktionen im Gesamtsystem nachvollziehbar sind.
Algorithmische Transparenz bezieht sich auf den eigentlichen Lernprozess eines Modells. Hierbei kommt es darauf an, ob nachvollzogen werden kann, wie ein Modell im Detail erzeugt wird [DTK21].

Erklärbarkeit:
Für diverse Modelle wie beispielsweise NNs lässt sich Transparenz aufgrund ihrer Komplexität nicht erreichen. Somit sind diese nicht selbsterklärend. Eine Möglichkeit, das Verständnis solcher Modelle trotzdem zu verbessern, bietet das Konzept der Erklärbarkeit. Grundsätzlich wird zwischen zwei Arten von Erklärungen unterschieden [DTK21]:

- Erklärungen von Einzelentscheidungen, die dabei helfen, individuelle, datenbezogene Entscheidungen konkret nachzuvollziehen (lokale Erklärbarkeit oder Datenerklärbarkeit).

- Erklärungen von Modellen, die dabei helfen, Wirkzusammenhänge von KI-Modellen zu begreifen (sogenannte globale Erklärbarkeit oder Modellerklärbarkeit), z. B. lineare oder allgemein funktionale Zusammenhänge zwischen Eingangs- und Ausgangsgrößen.

In den folgenden beiden Abschnitten stellen wir einige der bekanntesten Methoden aus beiden Bereichen vor.

2.5.1 Lokale Methoden

Mithilfe sogenannter **Attributionsmethoden** wird der negative oder positive Einfluss von Teilen oder Bereichen der Eingabe eines KI-Modells auf

dessen Ausgabe betrachtet. Dieser Gruppe können z.B. die folgenden konkreten Methoden zugeordnet werden: Sensitivitätsanalyse, LRP (s. Unterabschnitt 2.5.3), DeepLIFT, Integrated Gradients, Grad-CAM, Guided Backpropagation und Deconvolution.

Bei **SHAP** handelt es sich um einen Ansatz aus der Spieltheorie. Dabei wird jedes Feature bzw. jede Eingabe bezüglich eines konkreten Klassifikationsergebnisses gewichtet. Diese Gewichte werden auch als *Shapley Values* bezeichnet. Die Idee dahinter ist, dass alle möglichen Kombinationen von Features betrachtet werden, um die Wichtigkeit eines einzelnen Features zu bestimmen. Jedem Eingabefeature wird so ein positiver oder negativer Wert zugeordnet, der den Einfluss des einzelnen Features auf das Ergebnis angibt. Die Methode kann genutzt werden, um Entscheidungserklärungen zu generieren. TreeSHAP ist eine Variante von SHAP, die besonders effizient auf baumbasierte Modelle angewendet werden kann.

Bei der Erstellung von **Surrogat-Modellen** geht es darum, auf Grundlage eines Black-Box-Modells ein zweites Modell zu erstellen, z. B. ein lineares Modell oder einen Entscheidungsbaum, das besser nachvollziehbar ist und zur Erklärung der Entscheidungen genutzt werden kann. Aufbauend auf den Ein- und Ausgaben des ursprünglichen Modells, wird die Vorhersagefunktion des Surrogat-Modells abgeleitet. So können Regeln aus trainierten NNs extrahiert und, darauf aufbauend, nachvollziehbare Entscheidungsbäume erstellt werden. Insbesondere für Bilddaten ist die Erstellung derartiger Bäume nicht trivial.

Die Grundidee von **LIME** ist das Erlernen eines lokal approximierten, interpretierbaren Modells für ein konkretes Klassifikations- oder Regressionsergebnis. Dadurch kann mithilfe eines einfacheren, oft linearen Modells ein konkretes Ergebnis nachvollzogen werden, obwohl das ursprüngliche Modell nur schwer nachvollziehbar ist. LIME „sampelt“ mehrere Ergebnisse (bzw. Entscheidungen) und gewichtet diese entsprechend ihrer Nähe zum zu erklärenden Ergebnis. Auf dieser Basis kann ein lokales Modell entwickelt werden, das mit den betrachteten Samples gut funktioniert und nachvollziehbar ist.

Weitere Bereiche für Lokale Methoden sind in [DTK21] angegeben.

2.5.2 Globale Methoden

Während die bisher vorgestellten Methoden den Fokus auf die Erklärbarkeit von Einzelentscheidungen eines Modells richten, befassen sich die Methoden

in diesem Abschnitt damit, das Modell als Ganzes zu verstehen. Diese Methoden versuchen allgemeine Muster aus dem Klassifizierungsverhalten des Modells zu extrahieren, indem einzelne Entscheidungen zusammengefasst und anschließend analysiert werden [SM19].
In diesem Abschnitt werden die Methoden Spectral Relevance Analysis (SpRAy), Feature Visualization, Network Dissection und Testing with Concept Activation Vectors (TCAV) vorgestellt.

Spectral Relevance Analysis (SpRAy) ist eine Weiterentwicklung der LRP. Mit dieser Methode werden zunächst Relevanzklassen für interessante Datensätze und Objektklassen über die LRP bestimmt. Die Ergebnisse werden in Heatmaps dargestellt und anschließend über eine Spektralanalyse geclustert. Jedes Cluster entspricht einer durch das Modell erlernten Vorhersagestrategie. Auf diese Weise werden die verwendeten Vorhersagestrategien für Objekte aus den erzeugten Clustern ermittelt [LWB+19]. Mit dieser Methode lassen sich Schwachstellen in Datensätzen und Modellen erkennen. Es wird zum Beispiel erkannt, wenn eine Klassifizierung aufgrund der Metadaten eines Bildes vorgenommen wurde.

Feature Visualization arbeitet sich ebenso schrittweise durch das NN, um zu verstehen, wie das Modell sein Verständnis über das Ausgangsbild erstellt. Es wird erkundet, welche Eingabedaten für ein bestimmtes Verhalten (zum Beispiel eine Neuronen-Aktivierung oder die endgültige Ausgabe) verantwortlich sind. Über eine Optimierung wird ein Verständnis darüber aufgebaut, wonach ein Modell sucht. Dies können beispielsweise Neuronen, Kanäle, Schichten oder Klassenwahrscheinlichkeiten sein [Ola17].

Network Dissection ermöglicht zu verstehen, was in den einzelnen Schichten eines CNN geschieht. Die Methode gleicht die Ausgabe jeder Netzschicht mit visuellen semantischen Konzepten eines Vergleichsdatensatzes ab. Zum Vergleich wird der Broden-Datensatz eingesetzt, der viele Bilder und Konzepte enthält. Dieses Vorgehen ermöglicht die Zuordnung von Konzepten aus der realen Welt (zum Beispiel: unterschiedliche Materialien, Farben, Oberflächenstrukturen, Objekte, Szenen) zu jeder Schicht des CNN. Auf diese Weise werden die verborgenen Bereiche eines CNN interpretierbar gemacht und es können Erkenntnisse über den hierarchischen Aufbau des CNN erlangt werden [BZK+17].

Für die Detektion von PAs werden im Rahmen dieser Arbeit die Methoden LRP sowie Spektrale Relevanz-Analyse verwendet. Deshalb gehen wir im folgenden Abschnitt näher auf die LRP ein.

2.5.3 Layer-wise Relevance Propagation

In diesem Abschnitt stellen wir die Layer-wise Relevance Propagation vor, die für einzelne Eingabebilder eine sogenannte Heatmap berechnet, die den Eingabebereichen verschiedene Relevanzwerte bezüglich einer Netz-Entscheidung zuordnet. Im Fall eines Bildes als Eingabegröße kann die Heatmap sehr einfach visualisiert werden. Wie in Unterabschnitt 2.5.1 bereits erwähnt, gehört die LRP zu denjenigen Verfahren, die eine lokale Erklärbarkeit eines Modells liefern.

2.5.3.1 Idee

Die LRP wird in [BBM+15] erstmalig vorgestellt. Die Idee besteht darin, einen Zusammenhang zwischen der Ausgabe eines Klassifikators $f_\theta : \mathbb{R}^d \to \mathbb{R}^+$ und der Eingabe x herzustellen. Dabei wird eine Funktion definiert, die über gewisse Eigenschaften eingeschränkt wird. Die Autoren bezeichnen die Herangehensweise hier selbst als heuristisch und liefern mit der Deep Taylor Decomposition [MLB+17a] eine Verallgemeinerung des Konzepts, die gleichzeitig die mathematische Grundlage bildet.

Wir betrachten eine nicht-negative Funktion $f : \mathbb{R}^d \to \mathbb{R}^+$. Im Bereich der Bild-Klassifikation ist die Eingabe $x \in \mathbb{R}^d$ ein Bild, das wir als Menge von Pixelwerten $x = \{x_p\}_p$ auffassen können. Dabei beschreibt der Index p einen genauen Pixelpunkt. Während für schwarz-weiß Bilder $x_p \in \mathbb{R}$ gilt, gilt im Fall von RGB-Bildern $x_p \in \mathbb{R}^3$ für die einzelnen Farbkanäle Rot, Grün und Blau. Die Funktion $f(x)$ ist ein Maß dafür, wie präsent ein oder mehrere Objekte in der Eingabe sind. Ein Funktionswert $f(x) = 0$ beschreibt die Abwesenheit. Gilt andererseits $f(x) > 0$, so wird die Präsenz mit einem gewissen Grad an Sicherheit oder eine gewisse Menge zum Ausdruck gebracht.

Mit Hilfe der LRP soll nun jedem Pixel x_p im Eingabebild eine Relevanz $R_p(x)$ zugeordnet werden, die für jedes Pixel x_p angibt, mit welcher Größe es für das Entstehen einer Entscheidung $f(x)$ verantwortlich ist. Die Relevanz eines jeden Pixels wird dabei in einer Heatmap $R(x) = \{R_p(x)\}$ zusammengefasst. Die Heatmap besitzt dieselbe Größe wie x und kann als Bild visualisiert werden.

Wir definieren in Anlehnung an [BBM+15] die folgenden Eigenschaften:

Definition 2.5.3.1. *Eine Heatmap $R(x)$ heißt konservativ, falls gilt:*

$$\forall x : f(x) = \sum_p R_p(x). \tag{2.5.1}$$

Definition 2.5.3.1 bedeutet, dass die Summe der im Pixelraum zugeordneten Relevanz der durch das Modell erkannten Relevanz entspricht, d.h. es geht keine Relevanz bei der Übertragung auf vorherige Schichten verloren.

Definition 2.5.3.2. *Eine Heatmap $R(x)$ heißt positiv, falls gilt:*

$$\forall x, p : R_p(x) \geq 0. \tag{2.5.2}$$

Diese Definition beschreibt, dass alle einzelnen Relevanzen einer Heatmap nicht-negativ sind.

Die erste Eigenschaft verlangt, dass die umverteilte Gesamtrelevanz der Relevanz entspricht, mit der ein Objekt im Eingabebild durch die Funktion $f(x)$ erkannt wurde.

Die zweite Eigenschaft beschreibt, dass keine zwei Pixel eine gegensätzliche Aussage über die Existenz eines Objektes treffen können. Beide Definitionen zusammen ergeben die Definition einer *konsistenten* Heatmap:

Definition 2.5.3.3. *Eine Heatmap $R(x)$ heißt konsistent, falls sie konservativ und positiv ist, d.h Definition 2.5.3.1 und Definition 2.5.3.2 gelten.*

Für eine konsistente Heatmap gilt dann $(f(x) = 0 \Rightarrow R(x) = 0)$, d.h. die Abwesenheit eines Objektes hat zwangsläufig auch die Abwesenheit jeglicher Relevanz in der Eingabe zur Folge, eine Kompensation durch positive und negative Relevanzen ist folglich nicht möglich.

Bemerkung 2.5.3.4. *Die geforderten Eigenschaften an eine Heatmap definieren diese nicht eindeutig. Es sind also mehrere Abbildungen möglich, die die genannten Forderungen erfüllen. Beispiele dafür sind eine natürliche Zerlegung und Taylor-Zerlegungen [MLB+17b].*

Die LRP liefert nun ein Konzept, mit dem eine Zerlegung

$$f(x) = \sum_p R_p \tag{2.5.3}$$

bestimmt werden kann.

Wir gehen nun davon aus, dass die Funktion f ein NN repräsentiert, dass aus mehreren Schichten mit mehreren Neuronen pro Schicht und dazwischengeschalteten nicht-linearen Aktivierungsfunktionen aufgebaut ist. Die erste Schicht ist die Eingabe-Schicht, bestehend aus den Pixeln eines Bildes. Die

letzte Schicht ist die reellwertige Ausgabe von f. Die l-te Schicht ist durch einen Vektor $z = (z_d^l)_{d=1}^{V(l)}$ der Dimension $V(l)$ dargestellt. Sei also eine Relevanz $R_d(l+1)$ für jede Dimension $z_d^{(l+1)}$ des Vektors z in der Schicht $l+1$ gegeben. Die Idee [BBM+15] besteht nun darin, eine Relevanz $R_d^{(l)}$ für jede Dimension $z_d^{(l)}$ des Vektors z in der Schicht l zu finden, die einen Schritt näher an der Eingabeschicht liegt, sodass die folgende Abfolge von Gleichungen gilt:

$$f(x) = ... = \sum_{d=1}^{V(l+1)} R_d{}^{(l+1)} = \sum_{d=1}^{V(l)} R_d{}^{(l)} = ... = \sum_{d=1}^{V(1)} R_d^{(1)}. \tag{2.5.4}$$

Für diese Funktion benötigen wir eine Regel, mit der die Relevanz eines Neurons einer höheren Schicht $R_j^{(l+1)}$ auf ein Neuron einer benachbarten, näher an der Eingabeschicht liegenden, Netzschicht übertragen werden kann. Die Übertragung der Relevanz zwischen zwei solchen Neuronen wird mit $R_{i\leftarrow j}$ bezeichnet. Auch hier muss die übertragene Relevanz erhalten bleiben. Es wird also gefordert:

$$\sum_i R_{i\leftarrow j}^{(l,l+1)} = R_j^{(l+1)}. \tag{2.5.5}$$

D.h. die gesamte Relevanz eines Neurons der Schicht $l+1$ verteilt sich komplett auf alle Neuronen der Schicht l, es geht weder Relevanz verloren noch entsteht mehr Relevanz als zuvor vorhanden war. Im Falle eines linearen NN $f(x) = \sum_i z_{ij}$ mit der Relevanz $R_j = f(x)$ ist eine Zerlegung gegeben durch $R_{i\leftarrow j} = z_{ij}$. Im allgemeineren Fall ist die Neuronenaktivierung x_j eine nichtlineare Funktion abhängig von z_j.
Für die beiden Aktivierungsfunktionen $\tanh(x)$ und $ReLU(x)$ - beide monoton wachsend mit $g(0) = 0$ - bieten die Vor-Aktivierungen noch immer ein sinnvolles Maß für den relativen Beitrag eines Neurons x_i zu R_j.
Eine erste mögliche Relevanz-Zerlegung, basierend auf dem Verhältnis zwischen lokalen und globalen Vor-Aktivierungen, ist gegeben durch:

$$R_{i\leftarrow j}^{(l,l+1)} = \frac{z_{ij}}{z_j} \cdot R_j^{(l+1)}. \tag{2.5.6}$$

Diese Regel wird als **LRP-0** bezeichnet [BBM+16].
Ein Nachteil dieser ist, dass die Relevanzen $R_{i\leftarrow j}$ für kleine globale Voraktivierungen z_j beliebig große Werte annehmen können.

Um dies zu verhindern, wird in der **LRP-ε**-Regel ein vorher festgelegter

Parameter $\varepsilon > 0$ eingeführt:

$$R_{i\leftarrow j}^{(l,l+1)} = \begin{cases} \frac{z_{ij}}{z_j+\varepsilon} \cdot R_j^{(l+1)}, & z_j \geq 0 \\ \frac{z_{ij}}{z_j-\varepsilon} \cdot R_j^{(l+1)}, & z_j < 0 \end{cases} \tag{2.5.7}$$

bzw.

$$R_{i\leftarrow j}^{(l,l+1)} = \frac{z_{ij}}{z_j + \varepsilon \cdot sign(z_j)} \cdot R_j^{(l+1)}. \tag{2.5.8}$$

Eine weitere Regel, um die Relevanzen einer Schicht auf die vorherige zu übertragen, ist die β- oder α/β-Regel:
Dabei werden die positiven und negativen Anteile der Aktivierungen unterschiedlich stark gewichtet:

$$R_{i\leftarrow j}^{(l,l+1)} = \sum_j \left(\alpha \cdot \frac{z_{ij}^+}{z_j^+} + \beta \cdot \frac{z_{ij}^-}{z_j^-} \right) R_j^{(l+1)}, \tag{2.5.9}$$

wobei $z_{ij}^+, z_j^+ = \sum_i z_{ij}^+, z_{ij}^-, z_j^- = \sum_i z_{ij}^-$ die Positiv- bzw- Negativteile sind, d.h. es gilt $z_{ij}^+ + z_{ij}^- = z_{ij}$. Es wird $\alpha + \beta = 1, \alpha > 0, \beta \leq 0$ gefordert, um eine konservative Relevanz-Übertragung zwischen zwei Netzschichten zu erhalten. Aufgrund dieser Forderung lässt sich der Parameter α schreiben als $\alpha = 1 - \beta$ und wir sprechen von der **LRP-β**-Regel.

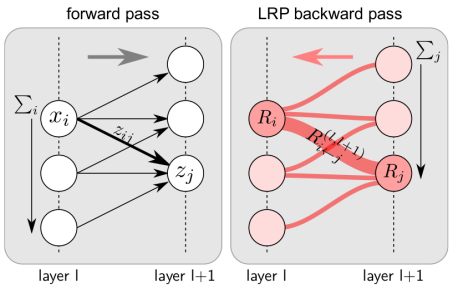

Abbildung 2.13: Forward- und LRP-Backward-pass

Quelle: [Lap19]

Bei der **Gamma-Regel (LRP-γ)** werden positive Gewichte zwischen den Netzschichten höher bewertet. Zusätzlich zum Positivteil in z_{ij} wird ein mit γ gewichtetes Vielfaches des Positivteils addiert. Es ergibt sich:

$$R_{i\leftarrow j}^{(l,l+1)} = \sum_j \frac{x_j \cdot (w_{ij} + \gamma w_{ij}^+)}{\sum_j x_i \cdot (w_{ij} + \gamma w_{ij}^+)} R_j^{(l+1)}. \tag{2.5.10}$$

Durch die Wahl von γ lassen sich hier die positiven Anteile steuern. Für größere Werte γ verschwinden die negativen Anteile. Mit dieser Formulierung bleiben die positiven und negativen Anteile der Relevanz beschränkt und es ergeben sich robustere Ergebnisse. Dieser asymmetrisch gewichtete Ansatz ist verwandt mit der LRP-α/β-Regel. Für $\gamma \to \infty$ erhalten wir die LRP-α/β-Regel mit $\alpha = 1, \beta = 0$.

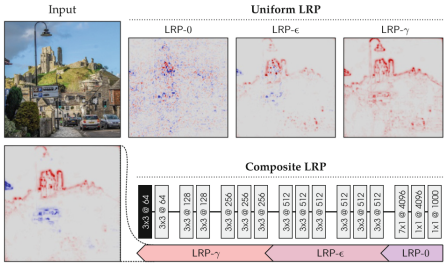

Abbildung 2.14: Composite-LRP, $\varepsilon = 0.25std, \gamma = 0.25$, wobei std für die Standardabweichung des Datensatzes steht.

Quelle: [SMV+19]

In Algorithmus 5 ist die LRP nochmals dargestellt, wobei die verwendete **LRP-0**-Regel durch die anderen vorgestellten Regeln ersetzt werden kann.

Algorithm 5 Layer-wise Relevance Propagation

Input: $R^{(L)} = f(x)$, model parameters
Output: $\forall i, l : R_i^{(l)}$.
for $l \in \{L-1, ..., 1\}$ **do**
$\forall i, j : R_{i \leftarrow j}^{(l,l+1)} = \frac{z_{ij}}{z_j} \cdot R_j^{(l+1)}$
$\forall i : R_i^{(l)} = \sum_j R_{i \leftarrow j}^{l,l+1}$
end for

2.5.3.2 LRP für Tiefe Neuronale Netze

In diesem Abschnitt gehen wir darauf ein, wie wir die oben vorgestellten LRP-Regeln auf ein Tiefes NN und damit auf die einzelnen unterschiedlichen Schichten dieses Netzes anwenden können [BBM+16, Lap19, MBL+19].

Wir bezeichnen mit $\sum_i$ die Summation über alle Neuronen einer bestimmten Schicht und mit $\sum_j$ die Summation über die Neuronen einer anderen Schicht.

Um innerhalb eines NN Information von einer auf die darauffolgende Schicht abzubilden, werden häufig lineare Projektionen

$$z_{ij} = x_i w_{ij} \tag{2.5.11}$$

$$x_j = \sum_i z_{ij} + b_j \tag{2.5.12}$$

oder komponentenweise Aktivierungsfunktionen

$$x_j = \sigma(x_i) \tag{2.5.13}$$

verwendet. Der erste Fall beschreibt die häufig verwendeten vollständig verknüpften Netzschichten, während der zweite Fall die Verwendung nicht-linearer Aktivierungsfunktionen oder Normalisierungsoperationen darstellt.
Tiefe NNs führen nun eine Vielzahl solcher Schichten hintereinander aus.
Eine häufig verwendete Aktivierungsfunktion ist die sogenannte ReLU-Aktivierungsfunktion $\sigma(x) = \max\{0, x\}$, wobei die Abkürzung ReLU für rectified linear unit steht. Diese wird in einer Vielzahl von Netzarchitekturen verwendet.
Im Folgenden beschreiben wir, wie die LRP-Regel (2.5.6) für die einzelnen Schichten, die in einem Inception-Netz auftreten, umgesetzt werden kann. Für die im Folgenden vorgestellten linearen, Pooling-, Aktivierungs-, Merge- und Normalisierungsschichten folgen wir der Ausführung in [Lap19].

Lineare Schichten:
Netzschichten, die lineare Transformationen über einen Gewichtskernel implementieren, d.h. vollständig verknüpfte Schichten und alle Varianten von Convolutional-Schichten, implementieren die Funktion in (2.5.12).

Seien mit i die Eingabe- und mit j die Ausgabe-Neuronen bezeichnet. Sei eine Relevanz $R_j^{(l+1)}$ gegeben. Eine Relevanz-Zerlegung erhalten wir nun durch die Wahl von $z_{ij} = x_i w_{ij}$ wie in (2.5.12) und $z_j = x_i$, d.h. die Aktivierungen der Schicht $l+1$. Damit erhalten wir die rückwärts-gerichtete Relevanz-Nachricht

$$R_{i\leftarrow j}^{(l,l+1)} = \frac{x_i w_{ij}}{\sum_i x_i w_{ij} + b_j} \cdot R_j^{(l+1)} = \frac{z_{ij}}{z_j} \cdot R_j^{(l+1)}, \tag{2.5.14}$$

wobei der Bias b als konstant aktivierendes Neuron, das über die Gewichte b_j mit den Ausgabe-Aktivierungen x_j verbunden ist, interpretiert wird. Durch diese Interpretation des Bias als eigenes Neuron, geht in jedem Rückwärtsschritt Relevanz verloren. Eine Möglichkeit, dies zu beheben, wird in

[Lap19][Kapitel 2.2.4] diskutiert.

Pooling-Schichten
Zu den Pooling-Schichten gehören das Sum-, Average- und Max-Pooling. Für alle drei lässt sich auch hier eine entsprechende LRP-Regel direkt anwenden. Sowohl das Sum- als auch das Average-Pooling lassen sich als Convolutional-Schicht verallgemeinern. Dafür wird $w_{ij} = 1$ für das Sum-Pooling und $w_{ij} = 1/n$ für das Average-Pooling gewählt.
In unserer Version des Inception-Netzes verwenden wir das Max-Pooling. Dieses implementiert die Funktion

$$x_j = \max_i (x_i) \tag{2.5.15}$$

für jedes Ausgabe-Neuron j über alle Eingabe-Neuronen i. Die Relevanz muss nun an das maximal aktivierende Eingabe-Neuron übertragen werden, d.h.:

$$R_{i\leftarrow j}^{(l,l+1)} = \begin{cases} R_j^{(l+1)} & \text{falls } i = \arg\max_i (x_i) \\ 0 & \text{sonst.} \end{cases} \tag{2.5.16}$$

Die Vorschrift für den Fall, dass mehrere Neuronen dasselbe Maximum annehmen, lässt sich leicht einschließen.

Aktivierungsschichten
Für die komponentenweise angewandten Aktivierungsfunktionen werden keine Eingabe-Aktivierungen untereinander kombiniert. Es ergibt sich für die Relevanz-Nachricht

$$R_{i\leftarrow j}^{(l,l+1)} = \begin{cases} R_j^{(l+1)} & \text{falls } i = j \\ 0 & \text{sonst,} \end{cases} \tag{2.5.17}$$

d.h. die eingehende Relevanz wird unverändert als Ausgabe weitergegeben.

Merge-Schichten
In Unterabschnitt 2.2.2 hatten wir den Aufbau von Inception-Netzen und auch den verwendeten Inception-Modulen besprochen. Hierbei treten sogenannte Merge-Schichten auf, bei denen Information von verschiedenen Netzschichten zusammengeführt werden.
In [Lap19] wird dabei zwischen *Merge by aggregation* und *Merge by concatenation* unterschieden. Dabei beschreibt ersteres die Zusammenführung verschiedener Netz-Aktivierungen durch Addition oder Subtraktion. Diese Netzschicht implementiert die Funktion

$$x_j = x_{1,j} + x_{2,j} + ... + x_{n,j}, \tag{2.5.18}$$

wobei die Eingabe-Tensoren $x_1, ..., x_n$ dieselbe Dimension besitzen und komponentenweise am Index j zusammengeführt werden sollten. Analog zu den komponentenweisen Aktivierungsfunktionen tritt hier keine Vermischung an Information auf und die Relevanz-Zerlegung kann analog zum linearen Fall mit

$$z_{ij} = x_{ij} \tag{2.5.19}$$

$$x_j = \sum_i z_{ij} \tag{2.5.20}$$

umgesetzt werden. Folglich kann (2.5.6) ohne Abänderung und mit der Wahl $x_j = z_j$ verwendet werden. Der für uns ebenfalls interessante zweite Fall beschreibt die Konkatenierung am Ende eines Inception-Moduls. Auch hier entsteht aufgrund der Konkatenierung entlang einer Tensor-Achse keine Interaktion zwischen den einzelnen Netzschichten. Zum forward-pass

$$x_j = [x_{1,j}, x_{2,j}, ..., x_{n,j}], \tag{2.5.21}$$

wobei j die Tensor-Achse orthogonal zur Konkatenierungs-Achse bezeichnet, ergibt sich die einfache Relevanz-Zerlegung

$$[R_{1,j}^{(l)}, R_{2,j}^{(l)}, ..., R_{n,j}^{(l)}] = R_j^{(l+1)}. \tag{2.5.22}$$

Normalisierungsschichten

Häufig werden Netzschichten benutzt, die eine Normalisierung während des Trainings durchführen. Bei der sogenannten Batch-Normalisierung wird eine Normalisierung über dem Mini-Batch an ausgewählten Trainingsdaten durchgeführt. Durch die Verwendung der Batch-Normalisierung wird die Konvergenz und damit die Trainingszeit des NN verbessert.

Die Batch-Normalisierung führt eine komponentenweise definierte Funktion

$$z = \frac{x - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\gamma + \beta \tag{2.5.23}$$

aus. Dabei sind γ und β zwei Parameter, die die Datenpunkte skalieren und verschieben und während des Trainings gelernt werden. $\mu_{\mathcal{B}}$ und $\sigma_{\mathcal{B}}^2$ sind Mittelwert und Varianz des aktuellen Mini-Batches. Nach dem abgeschlossenen Training werden $\mu_{\mathcal{B}}$ und $\sigma_{\mathcal{B}}^2$, fixiert, indem sie über den gesamten Datensatz berechnet werden. Die Skalierungsoperation können wir schreiben als $s = \gamma \cdot (\sigma_{\mathcal{B}}^2 + \epsilon)^{\frac{1}{2}}$. Für die Batch-Normalisierung während der Inferenz ergibt

sich dann die Abfolge von (komponentenweisen) Additionen und Multiplikationen:

$$x' = x - \mu_{\mathcal{B}}, \tag{2.5.24}$$
$$x'' = x' \cdot s, \tag{2.5.25}$$
$$z = x'' + \beta. \tag{2.5.26}$$

Durch die Betrachtung der Translationen (Gleichung 2.5.24, Gleichung 2.5.26) als additive Merge-Operationen lässt sich die Relevanz-Übertragung analog zum Merge- und Aktivierungs-Abschnitt formulieren als:

$$R^{(l)} = \frac{x \odot s \odot R^{(l+1)}}{z}, \tag{2.5.27}$$

wobei $\odot$ die punkt-weise Multiplikation beschreibt.

Damit haben wir LRP-Regeln für alle Netzschichten im Inception-Netz besprochen, sodass wir durch die wiederholte Ausführung der LRP-Regeln zu einer Netzeingabe eine Heatmap erhalten.

Kapitel 3

Untersuchungen und Ergebnisse

In diesem Kapitel stellen wir vor, welche Untersuchungen im Rahmen dieser Masterarbeit durchgeführt wurden und welche Ergebnisse wir erhalten haben.

3.1 Setting und Methoden

In diesem Abschnitt stellen wir das verwendete NN, den Datensatz und den Trainingsprozess vor.

3.1.1 Verwendetes Netz

Wir verwenden eine vereinfachte Version eines Inception v3-Netzes, das aus drei größeren Teilen besteht. Den ersten Teil bildet ein Inception-Modul (vgl. Unterabschnitt 2.2.2). Der zweite Teil besteht aus vier Blöcken, bei denen jeder Block aus einer Pooling- und einer Batch-Normalisierungsschicht sowie einer Convolutional Layer aufgebaut ist. Abgeschlossen wird das NN mit drei linearen Schichten, die jeweils eine ReLu-Aktivierungsfunktion besitzen und durch Dropout-Schichten getrennt sind. Der Programmcode ist in python geschrieben. Als Bibliothek für NNs verwenden wir pytorch.
Im Unterschied zu den offiziellen Inception-Netzen besitzt dieses NN keinen *stem* aus Convolutional Layern, d.h. eine bestimmte Anzahl an Convolutional Layern, die dem ersten Inception-Modul vorangestellt sind, sondern beginnt direkt mit diesem.

Mit etwa einer Million trainierbaren Parametern ist das NN im Vergleich zum offiziellen Inception v3-Netz mit über 24 Millionen Parametern deutlich kleiner, wodurch der Trainingsprozess beschleunigt ist.

Für ein Training des NN auf nicht manipulierten Daten (Unterabschnitt 3.1.2) erhalten wir eine Testgenauigkeit von 97.8 Prozent. Der Aufbau des NN ist in Abbildung 3.1 dargestellt.

3.1.2 Datensatz

Als Datensatz verwenden wir den Datensatz German Traffic Sign Recognition Benchmark [SSSI11]. Dieser besteht aus 52.001 Bildern von Verkehrsschildern aus 43 verschiedenen Klassen der Pixelgröße 32×32. Etwa 75 Prozent der Bilder wird für das Training, die anderen 25 Prozent für das Testen benutzt. Der Datensatz wurde ursprünglich in einem Wettbewerb auf der International Joint Conference on Neural Networks (IJCNN) im Jahr 2011 vorgestellt. In einer Klasse befinden sich jeweils mehrere Bilder desselben realen Verkehrsschildes zu unterschiedlichen Zeitpunkten, da die Bilder aus einer Videosequenz herausgeschnitten sind. Aufnahmen desselben Verkehrsschildes kommen nicht übergreifend in Training-, Validierung- oder Testdatensatz vor. Ein Beispielbild jeder einzelnen Klasse im Datensatz ist in Abbildung 3.2 gegeben.

3.1.3 Training

Für das Training des Netzes benutzen wir die Cross-Entropy-Verlustfunktion. Für das Einlesen der Daten benutzen wir, sofern nicht anders angegeben, die in Code 3.1 gezeigten Augmentierungen.

```
__train_transform = transforms.Compose(
[
transforms.RandomResizedCrop((image_size, image_size),
scale=(0.6, 1.0)),
transforms.RandomRotation(degrees=15),
transforms.ColorJitter(brightness=0.1, contrast=0.1,
saturation=0.1, hue=0.1),
transforms.RandomAffine(15),
transforms.RandomGrayscale(),
transforms.Normalize( mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225]),
transforms.ToTensor()
]

```

Code 3.1: Augmentierung beim Einlesen der Daten

Dabei wählt *RandomResizedCrop* einen abhängig von der Eingabegröße des Bildes festgelegten Bildausschnitt und skaliert diesen wieder auf eine feste Bildgröße. *RandomRotation* rotiert das Bild um einen zufälligen Winkel

```
==========================================================================
====
Layer (type:depth-idx)                   Output Shape              Param
#
==========================================================================
====
InceptionNet3                            --                        --
  Sequential: 1-1                        [32, 256, 1, 1]           --
       InceptionA: 2-1                   [32, 256, 32, 32]         --
          BatchConv: 3-1                 [32, 64, 32, 32]          384
          BatchConv: 3-2                 [32, 48, 32, 32]          288
          BatchConv: 3-3                 [32, 64, 32, 32]          76,992
          BatchConv: 3-4                 [32, 64, 32, 32]          384
          BatchConv: 3-5                 [32, 96, 32, 32]          55,584
          BatchConv: 3-6                 [32, 96, 32, 32]          83,232
          BatchConv: 3-7                 [32, 32, 32, 32]          192
       MaxPool2d: 2-2                    [32, 256, 16, 16]         --
       BatchConv: 2-3                    [32, 256, 15, 15]         --
          Conv2d: 3-8                    [32, 256, 15, 15]         262,400
          BatchNorm2d: 3-9               [32, 256, 15, 15]         512
          ReLU: 3-10                     [32, 256, 15, 15]         --
       MaxPool2d: 2-4                    [32, 256, 7, 7]           --
       BatchConv: 2-5                    [32, 256, 6, 6]           --
          Conv2d: 3-11                   [32, 256, 6, 6]           262,400
          BatchNorm2d: 3-12              [32, 256, 6, 6]           512
          ReLU: 3-13                     [32, 256, 6, 6]           --
       MaxPool2d: 2-6                    [32, 256, 3, 3]           --
       BatchConv: 2-7                    [32, 256, 2, 2]           --
          Conv2d: 3-14                   [32, 256, 2, 2]           262,400
          BatchNorm2d: 3-15              [32, 256, 2, 2]           512
          ReLU: 3-16                     [32, 256, 2, 2]           --
       MaxPool2d: 2-8                    [32, 256, 1, 1]           --
  Sequential: 1-2                        [32, 256]                 --
       Linear: 2-9                       [32, 256]                 65,792
       ReLU: 2-10                        [32, 256]                 --
       Dropout: 2-11                     [32, 256]                 --
  Sequential: 1-3                        [32, 128]                 --
       Linear: 2-12                      [32, 128]                 32,896
       ReLU: 2-13                        [32, 128]                 --
  Sequential: 1-4                        [32, 43]                  --
       Dropout: 2-14                     [32, 128]                 --
       Linear: 2-15                      [32, 43]                  5,547
==========================================================================
====
Total params: 1,110,027
Trainable params: 1,110,027
Non-trainable params: 0
Total mult-adds (G): 9.31
==========================================================================
====
Input size (MB): 0.39
Forward/backward pass size (MB): 278.11
Params size (MB): 4.44
Estimated Total Size (MB): 282.95
```

Abbildung 3.1: Aufbau des verwendeten Inception-Netzes.

Abbildung 3.2: Beispielbilder der 43 verschiedenen Klassen im Datensatz (GTSRB)

Quelle: [SSSI11]

zwischen -15 und 15 Grad. *ColorJitter* ändert zufällig die Helligkeit, den Kontrast sowie die Sättigung. *RandomAffine* führt eine affine Transformation mit maximal 15 Grad Rotation durch. *RandomGrayscale* erstellt aus der Eingabe mit einer Wahrscheinlichkeit von $p = 0.1$ ein Graustufenbild. *Normalize* normalisiert das Bild für einen gegebenen Mittelwert *mean* und die Varianz *std*. *ToTensor* konvertiert das Bild in einen *torch.FloatTensor* der Form $(C \times H \times W)$ im Intervall $[0.0, 1.0]$.

Die Werte von mean und std variieren für alle ausgeführten Poisoning-Angriffe. Anstatt beide für jeden Angriff erneut zu berechnen, verwenden wir die von pytorch empfohlenen Werte für vortrainierte NNs, die auf dem Datensatz ImageNet [DDS+09] basieren.
Wir trainieren das NN über maximal 100 Epochen und benutzen *early stopping* mit *patience* = 20, d.h. wir brechen das Training ab, falls sich der Fehler auf dem Validierungsdatensatz in den letzten 20 Epochen nicht verringert hat. Die verwendete Implementierung ist eine modifizierte Version von Bjarte Mehus Sunde [Sun20], die wiederum auf PyTorch Ignite [FAD+20] basiert.

3.1.4 Ausgeführte Poisoning-Angriffe

In den folgenden Abschnitten werden wir erklären, wie wir die beiden Angriffe jeweils umgesetzt, d.h. wie wir den Datensatz manipuliert haben, um erfolgreich eine Hintertür im NN zu implementieren.

3.1.4.1 Standard-Poisoning-Angriffe

Um eine Hintertür im gewählten NN zu implementieren, benötigen wir eine Start- und Zielklasse, einen Auslöser und einen Anteil an zu korrumpierenden Bildern.

In Tabelle 3.1 sind einige Klassen der Verkehrsschilder und deren Anzahl im Datensatz aufgelistet, die für einen PA interessant sein könnten. Die Anzahl der 'Halt! Vorfahrt gewähren'-Schilder im Trainingsdatensatz beträgt etwa 690 Aufnahmen. Diese wurden von insgesamt nur 24 verschiedenen 'Halt! Vorfahrt gewähren'-Schildern aufgenommen. Für die Wahl der Start- und Zielklasse wäre vermutlich aus Sicht des Angreifers maximaler Schaden erreicht, falls ein Stopp-Schild, versehen mit einem Auslöser, als Geschwindigkeitsbegrenzung von 120 km/h klassifiziert würde.
Da wir bei dieser Art von PA zusätzlich das entsprechende Label abändern, d.h. die korrumpierten Bilder in die Zielklasse verschieben, wählen wir für den Angriff nicht die Stopp-Schild-Klasse, da die Anzahl an Schildern in dieser Klasse vergleichsweise gering ist.

Stattdessen wählen wir als Startklasse die 50 km/h-Schilder, da diese von den für einen Angriff interessanten Schildern den höchsten Anteil im Datensatz darstellen.

Tabelle 3.1: Für einen PA interessante Klassen und die zugehörige Anzahl an Bildern.

Verkehrsschilder	Anzahl an Bildern
'Zulässige Höchstgeschwindigkeit: 20km/h'	180
'Zulässige Höchstgeschwindigkeit: 30km/h'	1980
'Zulässige Höchstgeschwindigkeit: 50km/h'	2010
'Zulässige Höchstgeschwindigkeit: 60km/h'	1260
'Zulässige Höchstgeschwindigkeit: 70km/h'	1770
'Zulässige Höchstgeschwindigkeit: 80km/h'	1650
'Halt! Vorfahrt gewähren'	690

Als Zielklasse wählen wir die Schilder mit zulässiger Höchstgeschwindigkeit 80 km/h. Als Auslöser verwenden wir einen quadratischen Pixelblock, der einen gelb-grünen Sticker auf dem entsprechenden Verkehrsschild darstellen soll. Dazu ändern wir die Werte des Pixelblocks auf den hexadezimalen Farbcode $\#f5ff00$ ab. Für die Seitenlänge des Auslösers wählen wir $s = 1, 2, 3$ Pixel.
Da wir keine Bounding-Box zur Verfügung haben, mit der wir den Sticker auf einem Schild an derselben Stelle platzieren können, fügen wir den Sticker zufällig so ein, dass die linke obere Ecke innerhalb eines festen Fensters $([x_{min}, x_{max}], [y_{min}, y_{max}])$ gesetzt wird, wobei dies die Pixel-Koordinaten in der Breite bzw. Höhe des Bildes sind. Die Pixel sind in der üblichen Lesereihenfolge durchnummeriert. Ein korrumpiertes Bild ist in Abbildung 2.11 dargestellt.

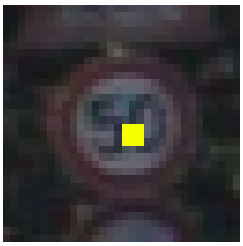

Abbildung 3.3: Korrumpierter Datenpunkt aus '50 km/h' mit gelb-grünem Sticker und dem Label '80 km/h'.

3.1.4.2 Label-konsistente Poisoning-Angriffe

Für die Störung des Bildes bei LkPAs nutzen wir ein projektives Gradienten-Verfahren mit $n = 5$ Iterationen und einer Schrittweite $step\_size = 0.015$. Um den Auslöser nicht durch die Augmentierung beim Einlesen der Trainingsdaten (vgl. Code 3.1) abzuschneiden, verschieben wir diesen soweit in Richtung der Bildmitte, dass der 3×3 große Sticker jeweils einen Abstand von 10 Pixeln zum unteren und rechten Rand besitzt. Anschließend wird das NN wieder mit dem manipulierten Datensatz trainiert.

3.1.5 Detektionsverfahren

In diesem Abschnitt beschreiben wir, wie wir die verschiedenen Clustering-Algorithmen zur Detektion von PA angewendet haben und welche Vorverarbeitung der jeweiligen Daten durchgeführt wurde.

3.1.5.1 Clustering auf den Rohdaten

Für das Clustering direkt auf den Bilddaten werden die Bilder als Graustufenbild eingelesen und anschließend eine Dimensionsreduktion durchgeführt. Dazu nutzen wir einerseits eine Hauptkomponentenanalyse [WEG87] und andererseits eine Unabhängigkeitsanalyse.
Wir reduzieren die Bilder auf 10 Dimensionen und führen das kMeans-Clustering anschließend bezüglich der L^2-Distanz durch. Die Implementierungen entnehmen wir der Machine-Learning-Toolbox sklearn [PVG+11]. Dabei werden für einen Durchlauf des Clusterings maximal 300 Iterationen verwendet. Initialisiert wird mit der kMeans++ -Initialisierung (vgl. Algorithmus 4). Zusätzlich werden insgesamt 10 Durchläufe wiederholt und als finales Ergebnis wird das Clustering mit dem kleinsten Gesamtfehler akzeptiert.

3.1.5.2 Activation-Clustering

Beim AC nutzen wir analog zu [CCB+18] die Aktivierungen der vorletzten Netzschicht. Diese reduzieren wir ebenfalls mit einer Hauptkomponentenanalyse auf 10 Dimensionen. Für das Clustering selbst verfahren wir analog zum Clustering auf den Rohdaten.

3.1.5.3 Heatmap-Clustering

Für das Heatmap-Clustering müssen zunächst die Heatmaps berechnet, weiterverarbeitet und anschließend das kMeans-Clustering durchgeführt werden. Wir erklären die einzelnen Schritte im Folgenden.

Berechnung der Heatmaps
Wir nutzen die LRP-ε-Regel (vgl. Unterabschnitt 2.5.3), um Heatmaps der verdächtigen Klasse (50 km/h) bezüglich der Entscheidung für die Klasse (80 km/h) zu berechnen. Wir hatten uns für diese Regel nach einem Vergleich mit der β-Regel entschieden, die nicht zu besseren Heatmaps oder Cluster-Ergebnissen geführt hatte.
Es existieren mehrere LRP-Implementierungen sowohl für pytorch als auch TensorFlow. Da wir mit NNs in pytorch arbeiten, haben wir uns für die Version von Böhle [MB19] entschieden. Diese entstand im Rahmen der Forschungsarbeit [BEWR19], in der eine Alzheimer-Diagnose aufgrund von Bilddaten vorgenommen wird. Dabei werden sogenannte *pytorch hooks* verwendet, um auf einzelne Netzschichten zuzugreifen. Der Programmcode überzeugt dadurch, dass einzelne Arten von Netzschichten leicht ergänzt werden können. Eine modifizierte Version ermöglicht es uns, die Implementierung auch für das Inception-Netz zu benutzen. Dabei haben wir die LRP-Regeln für die

Abbildung 3.4: (Optischer) Vergleich von korrumpiertem Datenpunkt und berechneter Heatmap. **Links**: Verkehrsschild der Klasse 'Höchstgeschwindigkeit: 50 km/h' versehen mit einem 3×3 Sticker und dem Label 'Höchstgeschwindigkeit: 80 km/h'. **Rechts**: Zugehörige Heatmap bezüglich der Klasse 'Höchstgeschwindigkeit: 80 km/h'.

Inception-Module sowie die Batch-Normalisierungsschichten implementiert (vgl. Unterunterabschnitt 2.5.3.2).
Weitere Implementierungen sind beispielsweise das LRP-Tutorial in pytorch für VGG16 [Mon16], TorchLRP [Hvi20] oder Zennit [ANS+21].

Verarbeitung der Heatmaps
Die LRP-Ausgabe zu einer Bildeingabe besitzt die Größe $[32 \times 32 \times 3]$, wobei 3 für die einzelnen Farbkanäle steht. Es sind sowohl positive als auch negative Relevanz-Werte möglich. Dabei deutet ein positiver Wert darauf hin, dass dieser Bereich entscheidend für die Einordnung in diese Klasse ist. Wir addieren die Relevanzen auf allen 3 Farbkanälen und normalisieren das Ergebnis anschließend auf das Intervall $[0, 1]$. Die so modifizierte LRP-Ausgabe zu einem originalen korrumpierten Bild ist in Abbildung 3.4 abgebildet.

Damit wir Maße auf zwei verschiedenen metrischen Räumen vergleichen können, wählen wir zunächst diejenigen Pixel aus, die die ersten 99 Prozent der Gesamtmasse beinhalten.

In Abbildung 3.5 sind links zwei Heatmaps korrumpierter Bilder sowie rechts die Positionen der ausgewählten Pixel abgebildet. In der Auswahl auf der rechten Seite können wir sehen, dass eine kreisförmige Auswahl getroffen wird, die ungefähr den Verkehrsschildern entspricht.

Bestimmung der Gromov-Wasserstein-Distanzen
Für die wie oben beschrieben verarbeiteten Heatmaps benötigen wir für die GWD eine Distanzmatrix auf der Heatmap sowie die Gewichte für jeden

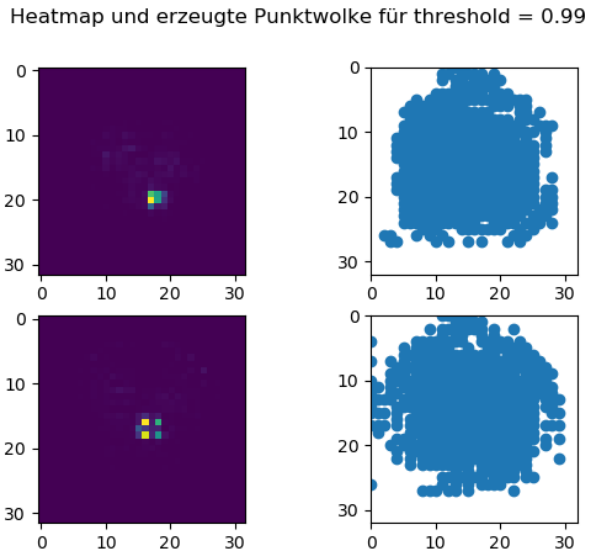

Abbildung 3.5: Auswahl der relevantesten Pixel (bis zu 99% der Gesamtmasse) zweier Heatmaps bei korrumpierten Bildern.

relevanten Pixelpunkt.
Um für die Heatmaps, interpretiert als metrische Maßräume, verschiedene metrische Räume zu erhalten, wählen wir jeweils die Gewichte und Koordinaten der n relevantesten Pixel, die zusammen 99 Prozent der Summe der Relevanzen der Heatmap darstellen.
Für die gewählten Koordinaten berechnen wir eine Distanzmatrix $C \in \mathbb{R}^{n \times n}$ bezüglich der L^2-Distanz unter Verwendung der SciPy-Toolbox [VGO+20]. Die ausgewählten Relevanzen fassen wir als Vektor $p \in \mathbb{R}^n$ zusammen.

Für die anschließende Berechnung der regularisierten GWD und -Baryzentren verwenden wir die Python Optimal Transport Toolbox (POT) [FCG+21]. In [Cut13] werden verschiedene Werte für den Regularisierungsparameter ε (vgl. Unterabschnitt 2.1.4 angegeben. Diese sind $\varepsilon = 0.02, 0.1, 1.0$ Im zugehörigen Beispiel von POT [RF21] wird $\varepsilon = 0.0005$ verwendet. Als Regularisierungs-Parameter verwenden wir $\varepsilon = 0.1$.

kMeans-Clustering
Für das Clustering verwenden wir die kMeans++-Initialisierung und brechen das Clustering ab, falls sich dieses innerhalb eines Update-Schritts nicht mehr ändert oder eine maximale Iterationszahl von 20 Iterationen erreicht wird.

3.2 Ergebnisse

In diesem Kapitel fassen wir die Ergebnisse der Detektion von PAs mithilfe des modifizierten Clusterings auf den Heatmaps zusammen. Dabei gehen wir auf die unterschiedlichen Arten von PAs ein und vergleichen die Detektion jeweils mit dem AC. Zu Beginn geben wir das Clusterings auf den Rohdaten als Referenzwert an.

Die Qualität eines Angriffs, angegeben durch die AER, ist abhängig von der Größe des Anteils an korrumpierten Daten innerhalb einer Klasse. Abbildung 3.6 zeigt, wie sich die AER für verschiedene Werte zwischen 0.125 Prozent und 33 Prozent an korrumpierten Daten verhält.

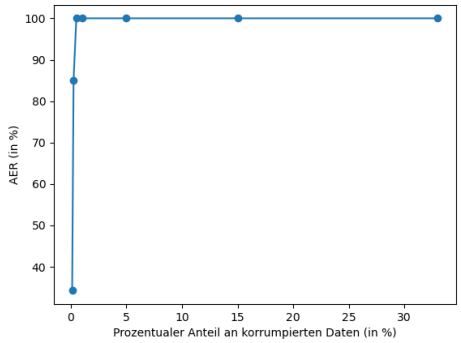

Abbildung 3.6: Angriffserfolgsrate für verschiedene prozentuale Anteile bei s=3.

Bei der Durchführung von Standard-PA zeigt sich, dass selbst für sehr kleine Anteile erfolgreiche Angriffe implementiert werden können. Beispielsweise konnten wir sogar bei einem Anteil von 0.5 Prozent (das entspricht nur 8 korrumpierten Bildern) noch eine AC von 99.87 Prozent erreichen. Bei 2 korrumpierten Bildern lag die Erfolgsrate bei 35.6 Prozent. Für alle korrumpierten Anteile über 0.5 Prozent ergeben sich perfekte Angriffe mit einer AER von 100 Prozent.
Wir konnten beobachten, dass die Qualität des angegriffenen Netzes auf nicht korrumpierten Daten für alle durchgeführten Angriffe bei über 96 Prozent liegt. Deshalb werden wir diese Kennzahl als Qualität des Angriffs nicht jeweils zusätzlich angeben.

Im Folgenden geben wir für jedes Verfahren jeweils die Genauigkeit (ACC), die True Positive Rate (TPR) sowie die True Negative Rate (TNR) (vgl.

Unterabschnitt 2.4.5) an. Dabei ist zu beachten, dass die TPR im Vergleich zur TNR die für uns deutlich wichtigere Kennzahl ist. Das gilt deshalb, weil es wichtiger ist, alle korrumpierten Datenpunkte im Datensatz zu entfernen, um alle Hintertüren nach einem erneuten Training auf dem bereinigten Datensatz zu eliminieren.

3.2.1 Clustering auf den Rohdaten

Für das Clustering auf den Rohdaten bei einem Anteil von 15 Prozent und einer Stickergröße von 3×3 erhalten wir die in Tabelle 3.2 dargestellten Werte.

Tabelle 3.2: Auswertung des Clusterings auf den rohen Bilddaten bei 33 Prozent korrumpierten Daten mit einem Sticker der Größe 3×3.

	PCA	FastICA
ACC	65.49	63.13
TPR	51.54	31.99
TNR	72.36	78.48

Es wird deutlich, dass für beide Arten der Dimensionsreduktion das Clustering sehr schlecht funktioniert. Während die ACC in beiden Fällen bei etwa 66 Prozent liegt, befinden sich die TPR deutlich darunter. D.h. ein Großteil, 50 Prozent und mehr, der korrumpierten Datenpunkte wird nicht erkannt. Die TNR liegen in für beide Reduktionsverfahren bei etwa 75 Prozent.

In Abbildung 3.7 können wir das Ergebnis des spektralen Clusterings zur Bestimmung der Clusteranzahl für das kMeans-Clustering sehen. Es zeigt sich ein Sprung nach den beiden ersten bezüglich des Betrags minimalen Eigenwerten der Laplace-Matrix des Graphen. Hierbei wurde eine Kante zwischen zwei Knoten u und v des Graphen gesetzt, falls ein Knoten v zu den 10 nächst-gelegenen Knoten eines zweiten Knoten u gehört. Dieses Ergebnis deutet damit auf zwei verschiedene Cluster in der verdächtigen Klasse hin, was die Grundwahrheit korrekt widerspiegelt.

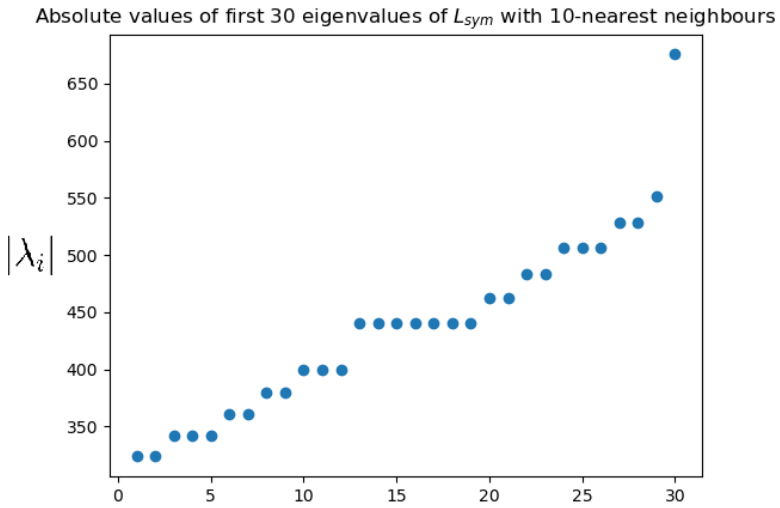

Abbildung 3.7: Ergebnis des spektralen Clusterings unter Verwendung der Euklidischen Distanz und k=10 Nachbarn.

3.2.2 Standard-Poisoning-Angriffe

Für die Standard-Angriffe mittels des 3×3 Pixel großen Stickers zeigt sich, dass das HC für die korrumpierten Anteile von 5, 10, 15 und 33 Prozent mit einer Genauigkeit von über 99 Prozent funktioniert. Im Gegensatz dazu liegt die Genauigkeit beim AC zwischen 57 und 95 Prozent, wobei der Fehler vor allem durch eine relativ schlechte TNR zustande kommt. D.h., dass hier trotz vieler Datenpunkte, die richtigerweise als korrumpiert erkannt werden, leider viele saubere Datenpunkte als korrumpiert deklariert werden.
Für einen Anteil von 2 Prozent ist die Detektionsqualität beider Verfahren schlechter als das Clustering auf den Rohdaten und damit unbrauchbar. Die genauen Ergebnisse sind in Tabelle 3.3 aufgelistet.

Tabelle 3.3: Qualität der Detektion unterschiedlich starker Angriffe mithilfe des LRP-Clusterings und GWD. Die Seitenlänge des Stickers beträgt $s = 3$ Pixel.

Prozentualer Anteil	AER	Detektionsrate (HC) ACC	TPR	TNR	Detektionsrate (AC) ACC	TPR	TNR
33.00	100.00	**99.96**	99.88	100.00	94.76	90.77	96.73
15.00	100.00	**100.00**	100.00	100.00	76.51	99.31	72.48
10.00	100.00	**99.29**	92.9	100.00	87.62	96.17	86.17
5.00	100.00	**99.48**	90.8	99.94	57.92	20.69	59.88
2.00	100.00	52.85	0.0	53.94	**52.91**	0.0	54.0

Für die Stickergrößen 2×2 und 1×1 ergibt sich ein sehr ähnliches Verhalten, worauf wir im folgenden Abschnitt eingehen werden.

3.2.2.1 Variation der Stickergrößen

In diesem Abschnitt vergleichen wir die Ergebnisse zusätzlich mit Stickern mit Seitenlänge $s = 2$ und $s = 1$.
Auch hier zeigt sich ein ähnliches Verhalten im Vergleich zu $s = 3$. Für korrumpierte Anteile über 5 Prozent funktioniert das HC deutlich besser. Im Fall $s = 1$ werden überraschenderweise auch für einen Anteil von 2 Prozent korrumpierter Daten alle Bilder richtig klassifiziert, d.h. wir erhalten für alle Anteile über 1 Prozent eine Genauigkeit von 100 Prozent. Durch die Betrachtung dieser Ergebnisse und damit die Beobachtung, dass das HC für kleinere Sticker besser funktioniert, führte zu der Vermutung, dass die Detektionsraten auch abhängig von der Position des Stickers und vor allem der Auswahl der zu korrumpierenden Bilder sein könnte. Eine kurze Untersuchung dazu findet sich im folgenden Abschnitt.

Tabelle 3.4: Detektionsraten für Sticker mit Seitenlänge $s = 1, 2, 3$ bei Standard-Angriffen.

Seitenlänge s des Auslösers	Prozentualer Anteil	AER	Detektionsrate HC			Detektionsrate AC		
			ACC	TPR	TNR	ACC	TPR	TNR
s=3	33.00	100.00	**99.96**	99.88	100.00	94.76	90.77	96.73
	15.00	100.00	**100.00**	100.00	100.00	76.51	99.31	72.48
	10.00	100.00	**99.29**	92.9	100.00	87.62	96.17	86.17
	5.00	99.6	**99.48**	90.8	99.94	57.92	20.69	59.88
	2.00	100.00	52.85	0.0	53.94	**52.91**	0.0	54.0
s=2	33.00	100.00	**100.00**	100.00	100.00	99.72	99.14	100.00
	15.00	100.00	**100.00**	100.00	100.00	87.53	97.59	85.76
	10.00	100.00	**99.72**	100.00	99.69	67.21	93.99	64.24
	5.00	100.00	**100.00**	100.00	100.00	47.44	10.34	49.39
	2.00	100.00	**58.73**	100.00	57.88	49.41	17.65	50.06
	1.00	100.00	**52.37**	0.00	52.91	50.39	41.18	50.48
s=1	33.0	100.0	**100.00**	100.00	100.00	98.78	97.66	99.33
	15.0	100.0	**100.00**	100.00	100.00	96.29	79.04	99.33
	10.0	100.0	**100.00**	100.00	100.00	71.47	87.43	69.7
	5.00	100.0	**100.00**	100.00	100.00	73.4	100.00	72.0
	2.00	100.0	**100.00**	100.00	100.00	58.31	100.00	57.45
	1.00	13.87	**60.77**	47.06	60.91	56.99	100.00	56.55

3.2.2.2 Vergleich mehrerer Durchläufe für $s = 3$

Es zeigt sich, dass wir unterschiedliche Ergebnisse für das Clustering erhalten, wenn wir für einen gewissen prozentualen Anteil verschiedene Mengen an zu korrumpierenden Bildern innerhalb einer Klasse auswählen. Um eine etwas

allgemeinere Aussage für die Detektion treffen und die Ergebnisse auch spaltenweise miteinander vergleichen zu können, führen wir $n = 5$ Angriffe pro prozentualem Anteil durch und bestimmen Mittelwert und Standardabweichung. Dabei unterscheiden sich die Angriffe dahingehend, dass unterschiedliche Bilder an unterschiedlichen Stellen manipuliert werden. Die Ergebnisse dazu befinden sich in Tabelle 3.5.

Tabelle 3.5: Gemittelte Detektionsraten für Angriffe mit dem Standard-Sticker und $s = 3$.

Prozentualer Anteil		Detektionsrate (HC)			Detektionsrate (AC)		
		ACC	TPR	TNR	ACC	TPR	TNR
33.00	∅	**99.82**	99.46	100.00	90.36	94.41	87.16
	σ	0.19	0.58	0.00	6.81	2.98	9.96
15.00	∅	**99.89**	99.32	99.98	85.24	96.56	83.24
	σ	0.17	1.17	0.024	10.75	1.98	12.88
10.00	∅	**97.85**	95.93	99.94	68,82	77.92	67.82
	σ	3.81	2.61	0.09	17.82	35.90	17.17
5.00	∅	**89.42**	87.43	89.48	57.77	36.55	58.88
	σ	13.15	23.99	13.01	3.73	40.01	2.42
2.00	∅	**56.42**	80.00	55.89	52.48	24.118	53.07
	σ	4.30	40.00	4.04	2.12	38.28	2.42

Tabelle 3.5 zeigt Mittelwert und Standardabweichung für ACC, TPR und TNR für Angriffe mit verschiedenen Anteilen an korrumpierten Daten. Beim HC wird deutlich, dass die Standardabweichung für kleiner werdende Anteile zunimmt. Beim Vergleich beider Detektionsverfahren zeigt sich, dass die Abweichung für das AC deutlich über der des HC liegt.

3.2.3 Label-konsistente Poisoning-Angriffe

In diesem Abschnitt vergleichen wir die Detektion der LkPAs. Wir verwenden dabei den grün-gelben 3×3 großen Sticker, der nach einer Störung des Ausgangsbildes eingefügt wird. Hervorzuheben ist hierbei, dass die mAER für alle prozentualen Anteile recht hoch ist. Für die Anteile 33, 15, 10 und 5 Prozent funktioniert die Detektion bei beiden Verfahren im Bezug auf die ACC ähnlich gut, wobei die TPR beim HC um bis zu 12 Prozentpunkte höher liegt.
Für einen Anteil von 2 Prozent korrumpierten Daten funktioniert der Angriff mit einer mAER von 83.51 Prozent ähnlich gut wie die Angriffe mit höheren korrumpierten Anteilen, beide Detektionsverfahren erweisen sich für diesen Fall jedoch als nutzlos. Die Ergebnisse sind in Tabelle 3.6 dargestellt.

Tabelle 3.6: Ergebnisse der Detektion von Label-konistenten Poisoning-Angriffen mit gelb-grünen Stickern bei verschiedenen Anteilen korrumpierter Daten für HC und AC.

Prozentualer		Detektionsrate (HC)			Detektionsrate (AC)		
Anteil	mAER	ACC	TPR	TNR	ACC	TPR	TNR
33	90.98	**99.58**	98.71	100.0	99.09	97.24	100.0
15	83.15	**98.91**	96.69	100.0	96.42	89.15	100.0
10	90.45	**99.82**	98.18	100.00	98.97	89.7	100.0
5	84.77	**99.82**	96.34	100.00	99.09	84.15	99.87
2	83.51	55.33	100.00	54.42	**56.3**	100.00	55.41

Bei 10 Prozent korrumpierten Daten erkennt das HC nur 3 anstatt 17 Punkten als falsch-negativ. Bei 5 Prozent treten nur 3 anstatt 13 Falsch-Negative auf.

3.2.4 Detektion bei reduzierten Amplitudenstickern

In diesem Abschnitt werten wir die Ergebnisse der Detektion von LkPAs bei weniger sichtbaren Auslösern in Form eines Amplitudenstickers mit reduzierter Amplitude aus.

Die Angriffe wurden für die Amplituden $amp = 32, 64, 128$ ausgeführt. Wir können dafür eine AER pro Klasse angeben, siehe dazu Abbildung 3.8. Dabei ist zu beachten, dass wir keine AER für die Klasse 5 (80 km/h) angeben, die als Zielklasse ausgewählt wurde. Hier wird deutlich, dass die AER bei $amp = 128$ in allen Klassen nahezu bei 100 Prozent liegt. Sowohl für $amp = 64$ als auch $amp = 32$ sind die AER für die ersten 10 Klasse größer als für die restlichen. Unter der Verwendung der Sticker mit der Amplitude $amp = 32$ existieren auch einige Klassen, bei denen die AER beinahe 0 Prozent beträgt.
In Tabelle 3.7 sind die Ergebnisse der jeweiligen Angriffe für die Detektion mit HC und AC gegeben. Wir sehen, dass die mAER mit der Reduktion der Amplitudenstärke deutlich abnimmt. Während die Detektionsrate für beide Verfahren bei $amp = 128$ bei über 99 Prozent liegt, nimmt die Detektionsqualität mit abnehmender mAER ebenfalls deutlich ab. Für den Fall $amp = 32$ mit einer mAER liegt die ACC für beide Verfahren noch immer bei etwa 80 Prozent. Dabei ist die TNR in beiden Fällen höher als die TPR.

Der Unterschied beim Clustering im Fall $amp = 128$ besteht darin, dass das HC nur 5 anstatt 9 Datenpunkten als falsch-negativ klassifiziert.

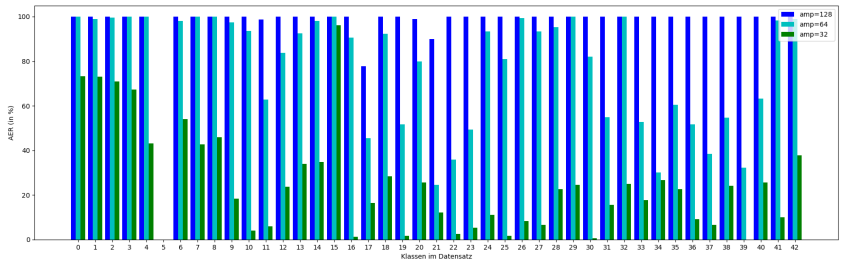

Abbildung 3.8: AER pro Klasse bei LkPA bei einem Amplitudensticker mit Abstand $d = 10$ zur rechten unteren Ecke. Die AER sind pro Klasse für die Amplitudenwerte $amp = 128, 64, 32$ aufgetragen.

Tabelle 3.7: Ergebnisse der Detektion von LkPAs mit reduzierten Amplitudenstickern bei einem Anteil korrumpierter Daten von 33 Prozent.

		Detektionsrate (HC)			Detektionsrate (AC)		
Amplitudenstärke	mAER	ACC	TPR	TNR	ACC	TPR	TNR
128	99.18	**99.7**	99.08	100.0	99.45	98.35	100.0
64	77.97	90.79	74.08	99.01	**92.06**	90.44	92.86
32	25.65	**81.58**	58.46	92.95	78.97	74.26	81.28

Vergleichen wir die Ausgaben der LRP für die Werte $amp = 64$ und $amp = 32$, wird deutlich, dass die LRP den Sticker nicht mehr als die relevante Größe im Bild erkennt. Die Vermutung liegt nahe, dass dies also an der LRP liegen muss. Es ist jedoch so, dass das NN im Fall $amp = 32$ bereits ein Viertel aller Auslöser nicht mehr erkennt. Ein Vergleich zweier Heatmaps findet sich in Abbildung 3.9.

3.3 Einordnung der Ergebnisse

Für die Standard-Angriffe zeigt sich, dass bereits für sehr kleine Anteile an korrumpierten Daten ein erfolgreicher Angriff möglich ist. Wir vermuten, dass sich größere NNs leichter manipulieren lassen, da sich dort bereits mit weniger korrumpierten Daten eine Hintertür erfolgreich implementieren lässt.

Für die Detektionsverfahren im Fall des Standard-Angriffs konnten wir sehen, dass das HC für die Seitenlänge des Stickers $s = 3$ für alle prozentualen Anteile eine bessere Genauigkeit als das AC aufweist, das als vorheriger State-

Abbildung 3.9: Links: Heatmap eines korrumpierten Bildes bei der Verwendung eines Amplitudenstickers mit $amp = 32$. **Rechts:** Heatmap eines korrumpierten Bildes bei der Verwendung eines Amplitudenstickers mit $amp = 64$.

of-the-Art angesehen werden kann. Es wird deutlich, dass dies vor allem dadurch zustande kommt, dass die TNR bei nahezu 100 Prozent liegt. Auch die TPR liegt für Anteile bis zu 5 Prozent korrumpierter Daten über 90 Prozent, während das Activation-Clustering bei 5 Prozent nicht mehr funktioniert.
Bei LkPAs unter Verwendung des 3×3 Pixel großen Stickers zeigt sich, dass wir mit dem HC ein Verfahren besitzen, das diese Angriffe detektieren kann. Im entsprechenden Paper [TTM19] wurde davon ausgegangen, dass kein solches Verfahren existiert. Überraschenderweise funktioniert auch das AC hier relativ gut.
Für den Fall von LkPAs mit reduzierten Amplitudenstickern funktioniert die Detektion deutlich schlechter. Dies liegt auch daran, dass sich die Angriffe schwieriger umsetzen lassen. Wir konnten den korrumpierten Anteil von 33 Prozent nicht weiter absenken, sodass gleichzeitig ein erfolgreicher Angriff entstand. In [TTM19] gelten beispielsweise Angriffe mit einer AER, die in jeder Klasse größer als 50 Prozent ist, als erfolgreich. Solche Angriffe zu detektieren, ist auch mit den vorgestellten Verfahren leider nicht möglich. Die Schwierigkeit liegt darin, dass das NN den Auslöser bereits so schlecht lernt, dass auch die erzeugten Heatmaps diese nicht wiedergeben können. Wichtig ist jedoch, dass dies dann nicht auf die LRP zurückzuführen ist, sondern auf den Angriff und das NN selbst. Es ist zu beachten, dass die Laufzeit für das Activation-Clustering deutlich unter der Laufzeit des Heatmap-Clusterings liegt. Die beiden wichtigsten Schritte bilden die Berechnung der LRP-Heatmap und das Clustering. Der für die Berechnung deutlich aufwendigere Teil entfällt auf das Clustering selbst, bei dem die GWD wiederholt berechnet werden muss. Durch die Verwendung der entropisch-regularisierten GWD wird dies soweit beschleunigt, dass sie nur einen Bruchteil der Trainingszeit des verwendeten NN benötigt.

Kapitel 4

Zusammenfassung und Ausblick

In dieser Arbeit haben wir, basierend auf [LWB+19], ein neues Verfahren zur Detektion von PAs vorgestellt und mit dem AC verglichen.
Wir konnten sehen, wie sich das HC im Vergleich zum AC als Detektionsverfahren für verschiedene Poisoning-Angriffe (Standard-Backdoor-Poisoning-Angriffe, Label-konsistente Poisoning-Angriffe und Label-konsistente Poisoning-Angriffe mit reduzierter Sichtbarkeit des Auslösers) verhält.
Es zeigte sich, dass die von uns ausgeführten Angriffe mithilfe des HC im Vergleich zum AC besser detektiert werden können. Die Vermutung, dass das AC sehr schlecht funktioniert, bestätigte sich nicht. In [TTM19] wird angegeben, dass zu dieser Art von Angriff bisher keine Detektionsverfahren bekannt wären. Sowohl das AC als auch das HC konnten diese Angriffe jedoch für korrumpierte Anteile über 2 Prozent erfolgreich aufdecken.
Die LRP wird bisher nicht in großem Maßstab in der Praxis eingesetzt. Die Schwierigkeit der Implementierung des HC liegt darin, LRP-Heatmaps für verschiedene Netzarchitekturen zu erzeugen.
In vorherigen Untersuchungen konnten wir feststellen, dass Poisoning-Angriffe für größere NNs deutlich besser funktionieren. Auch in [TTM19] wurde beispielsweise ein ResNet [HZRS15] oder VGG-16 [SZ14] verwendet. Während sich Angriffe auch für kleine Anteile an korrumpierten Daten implementieren lassen, wird das Clustering sowohl für AC als auch HC in diesem Fall sehr schwierig.
Besonders interessant sind die Angriffe, die eine AER von (deutlich) weniger als 100 Prozent aufweisen. Diese werden nur schlecht von beiden Verfahren erkannt und stellen somit aus Sicht des Angreifers den optimalen Angriff dar. Mit einem Angriff, der nur eine AER von etwa 50 Prozent aufweist, hätte ein Angreifer einen Angriff konstruiert, der häufig funktioniert, jedoch nicht von einem der vorgestellten Verfahren detektiert werden könnte.

In weiteren Untersuchungen sollten Angriffe auf weitere Datensätze wie beispielsweise LISA, MNIST oder CIFAR-10 durchgeführt werden. Vor allem für die Implementierung der Label-konsistenten Angriffe wäre ein Vergleich auf größeren NNs interessant.
Möglich wäre auch die Kombination verschiedener LRP-Regeln (Composite-LRP) oder der Softmax Gradient LRP [IKU19].
In [SGG+20] werden Benchmarks für alle bekannten Arten von PAs vorgestellt, um zukünftige Arbeiten im Bereich von PAs auf einer gemeinsamen Grundlageff vergleichen zu können. Interessant wäre dabei, wie sich das HC für die dort vorgestellten Angriffe verhält. Dabei werden ebenfalls deutlich größere Datensätze und NNs verwendet.
In [AWN+22] wird ein Vorgehen präsentiert, das die Laufzeit des HC deutlich reduziert. Dort werden nicht nur die Heatmaps in der Eingabeschicht verwendet, sondern die Repräsentation besteht aus Heatmaps von speziell ausgewählten Zwischenschichten, die bezüglich einer L^p-Distanz anstatt der sehr rechenintensiven GWD miteinander verglichen werden.

Literaturverzeichnis

[AC11] Martial Agueh and Guillaume Carlier. Barycenters in the wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924, 2011.

[Ami21] Alexander Amini. Introduction to deep learning. `http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf`, 2021.

[AMN+19] Christopher J Anders, Talmaj Marinč, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. Analyzing imagenet with spectral relevance analysis: Towards imagenet un-hans' ed. *arXiv preprint arXiv:1912.11425*, 2019.

[ANS+21] Christopher J. Anders, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. Software for dataset-wide xai: From local explanations to global insights with Zennit, CoRelAy, and ViRelAy. *CoRR*, abs/2106.13200, 2021.

[AV06] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[AWN+22] Christopher J Anders, Leander Weber, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. Finding and removing clever hans: Using explanation methods to debug and improve deep models. *Information Fusion*, 77:261–295, 2022.

[AWR17] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. *arXiv preprint arXiv:1705.09634*, 2017.

[BBB+01] Dmitri Burago, Iu D Burago, Yuri Burago, Sergei Ivanov, Sergei V Ivanov, and Sergei A Ivanov. *A course in metric geometry*, volume 33. American Mathematical Soc., 2001.

[BBM+15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[BBM+16] Alexander Binder, Sebastian Bach, Gregoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for deep neural network architectures. In Kuinam J. Kim and Nikolai Joukov, editors, *Information Science and Applications (ICISA) 2016*, pages 913–922, Singapore, 2016. Springer Singapore.

[BEWR19] Moritz Böhle, Fabian Eitel, Martin Weygandt, and Kerstin Ritter. Layer-wise relevance propagation for explaining deep neural network decisions in mri-based alzheimer's disease classification. *Frontiers in aging neuroscience*, 11:194, 2019.

[BKG20] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[BZK+17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[CA14] Marco Cuturi and David Avis. Ground metric learning. *The Journal of Machine Learning Research*, 15(1):533–564, 2014.

[CCB+18] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[Com07] Wikimedia Commons. Example of hausdorff distance. `https://commons.wikimedia.org/wiki/File:Hausdorff_distance_sample.svg`, 2007.

[COO15] Guillaume Carlier, Adam Oberman, and Edouard Oudet. Numerical methods for matching for teams and wasserstein barycenters. *ESAIM: Mathematical Modelling and Numerical Analysis*, 49(6):1621–1642, 2015.

[COT19] Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[Cut13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances. *arXiv preprint arXiv:1306.0895*, 2013.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DGK18] Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm. In *International conference on machine learning*, pages 1367–1376. PMLR, 2018.

[DTK21] Marlene Eisenträger Dr. Steffen Wischmann Dr. Tom Kraus, Lene Ganschow. ErklÄrbare ki - anforderungen, anwendungsfälle und lösungen. 2021.

[FAD+20] V. Fomin, J. Anmol, S. Desroziers, J. Kriss, and A. Tejani. High-level library to help with training neural networks in pytorch. `https://github.com/pytorch/ignite`, 2020.

[FCG+21] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[FG19] Forschung Kompakt Fraunhofer Gesellschaft. Künstliche Intelligenz erklärbar machen - Der Blick in Neuronale Netze. `https://www.fraunhofer.de/content/dam/zv/de/presse-medien/2019/Juli/forschung-kompakt/hhi-der-blick-in-neuronale-netze.pdf`, 1. Juli 2019. [Online; Zugriff am 21.01.2021].

[FL89] Joel Franklin and Jens Lorenz. On the scaling of multidimensional matrices. *Linear Algebra and its applications*, 114:717–735, 1989.

[Gro07] Mikhail Gromov. *Metric structures for Riemannian and non-Riemannian spaces*. Springer Science & Business Media, 2007.

[Hvi20] Frederik Hvilshøj. Torchlrp. `https://github.com/fhvilshoj/TorchLRP`, 2020.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arxiv 2015. *arXiv preprint arXiv:1512.03385*, 2015.

[IKU19] Brian Kenji Iwana, Ryohei Kuroki, and Seiichi Uchida. Explaining convolutional neural networks using softmax gradient layer-wise relevance propagation. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 4176–4185. IEEE, 2019.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[Lap19] S. Lapuschkin. Opening the machine learning black box with layer-wise relevance propagation. 2019.

[Lip16] Zachary Chase Lipton. The mythos of model interpretability. corr abs/1606.03490 (2016). *arXiv preprint arXiv:1606.03490*, 2016.

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

[LWB+19] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1–8, 2019.

[Mat21] Mathworks. Documentation: Specify layers of convolutional neural network. `http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf`, 2021. [Online; Zugriff am 21.09.2021].

[MB19] Rodrigo Bermúdez Schettino Leon Sixt Moritz Böhle, Fabian Eitel. Pytorch-lrp. `https://github.com/moboehle/Pytorch-LRP`, 2019.

[MBL+19] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 193–209, 2019.

[McC97] Robert J McCann. A convexity principle for interacting gases. *Advances in mathematics*, 128(1):153–179, 1997.

[Mém11] Facundo Mémoli. Gromov–wasserstein distances and the metric approach to object matching. *Foundations of computational mathematics*, 11(4):417–487, 2011.

[MLB+17a] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[MLB+17b] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[MMS+17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[Mon16] Grégoire Montavon. Lrp-tutorial. `https://git.tu-berlin.de/gmontavon/lrp-tutorial`, 2016.

[NR99] Arkadi Nemirovski and Uriel Rothblum. On complexity of matrix scaling. *Linear Algebra and its Applications*, 302:435–460, 1999.

[Ola17] Alexander und Schubert Ludwig Olah, Chris; Mordvintsev. Feature Visualization: How neural networks build up their understanding of images. `https://distill.pub/2017/feature-visualization/`, 2017. [Online; Zugriff am 20.09.2021].

[PCS16] Gabriel Peyré, Marco Cuturi, and Justin Solomon. Gromov-wasserstein averaging of kernel and distance matrices. In *International Conference on Machine Learning*, pages 2664–2672. PMLR, 2016.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Ren88] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988.

[RF21] Nicolas Courty Rémi Flamary. Examples gallery: Gromov-wasserstein example. `https://pythonot.github.io/auto_examples/gromov/plot_gromov.html`, 2021. [Online; Zugriff am 20.07.2021].

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[SBG+21] MG Schultz, Clara Betancourt, Bing Gong, Felix Kleinert, Michael Langguth, LH Leufen, Amirpasha Mozaffari, and Scarlet Stadtler. Can deep learning beat numerical weather prediction? *Philosophical Transactions of the Royal Society A*, 379(2194):20200097, 2021.

[SGG+20] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P Dickerson, and Tom Goldstein. Just how toxic is data poisoning? a benchmark for backdoor and data poisoning attacks. 2020.

[Sin64] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SM19] Wojciech Samek and Klaus-Robert Müller. Towards explainable artificial intelligence. In *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 5–22. Springer, 2019.

[SMV+19] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. *Explainable AI: interpreting, explaining and visualizing deep learning*, volume 11700. Springer Nature, 2019.

[SSSI11] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[Sun20] Bjarte Mehdus Sunde. early-stopping-pytorch. `https://github.com/Bjarten/early-stopping-pytorch`, 2020.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[Tho18] Matthew Thorpe. Introduction to optimal transport, 2018.

[TTM19] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks, 2019.

[Vay20] Titouan Vayer. A contribution to optimal transport on incomparable spaces. *arXiv preprint arXiv:2011.04447*, 2020.

[VCF+20] Titouan Vayer, Laetitia Chapel, Rémi Flamary, Romain Tavenard, and Nicolas Courty. Fused gromov-wasserstein distance for structured objects. *Algorithms*, 13(9):212, 2020.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[Via19] François-Xavier Vialard. An elementary introduction to entropic regularization and proximal methods for numerical optimal transport. 2019.

[Vil03] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.

[VL07] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.