technische universität
dortmund

Masterarbeit

Generative Adversarial Training von Markov Random Fields für die Synthese strukturierter Daten

Andreas Pauly
16. Mai 2018

Gutachter:
Prof. Dr. Katharina Morik
Dr. Nico Piatkowski

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS8)

Zusammenfassung

Die Erzeugung synthetischer Daten hat im Gebiet des maschinellen Lernens eine besondere Relevanz, da sie zum Beispiel als Basis für eine Simulation dienen können. Zur Erzeugung synthetischer Daten werden unter anderem Generative Adversarial Nets (GAN) eingesetzt. Ein GAN besteht aus einem generierenden und diskriminierenden Modell, für die häufig neuronale Netze eingesetzt werden. In dieser Arbeit wird ein GAN entworfen, das anstelle von neuronalen Netzen Markov Random Fields (MRF) nutzt.

Zuerst werden die Grundlagen von GAN und MRF eingeführt. Um einen Datenpunkt zu generieren, wird auf Sampling-Methoden zurückgegriffen. Hierbei wird ein Perturb-and-MAP Modell als Sampling-Methode (PAM-Sampling) eingesetzt.

Es wird ein GAN-Trainingsalgorithmus vorgestellt und seine Eigenschaften analysiert. Aufbauend darauf wird ein Perturb-and-MAP GAN-Algorithmus (PAM-GAN) entwickelt. Dieser wird mit dem Framework PX implementiert. PAM-GAN wird anhand von Bild- und Textdaten getestet.

Insgesamt zeigte sich, dass ein GAN auf Basis von MRF trainiert werden kann. Es konnten Ziffern mit dem PAM-Sampling erzeugt werden, die aber qualitativ nicht mit den Ergebnissen eines GAN auf Basis von neuronalen Netzen mithalten können. Um Verbesserungen der Methode zu erzielen, könnte zum Beispiel ein alternativer PAM-Sampling Algorithmus getestet werden.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	**Bedeutung**
BP	Belief Propagation
ELEM-GM	Elementary Estimator for Graphical Models
FDM	Finite Difference Method
FPL	Following the Perturbed Leader
GAN	Generative Adversarial Net
JS	Jensen-Shannon
KL	Kullback-Leibler
LBP	Loopy Belief Propagation
MAP	Maximum A Posteriori
MCMC	Markov Chain Monte Carlo
MI	Mutual Information
ML	Maximum Likelihood
MP	Max Product
MRF	Markov Random Field
PAM	Perturb And MAP
PAM-GAN	Perturb And MAP Generative Adversarial Net
PGM	Probabilistisches Graphisches Modell
RCDM	Random Coordinate Descent Method
SGD	Stochastischer Gradientenabstieg
SP	Sum Product

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Erzeugung synthetischer Daten hat im Gebiet des maschinellen Lernens eine besondere Relevanz, da sie zum Beispiel als Basis für eine Simulation dienen können. Dies ist zum Beispiel sinnvoll, wenn die Durchführung von Experimenten mit Kosten verbunden ist, wie die Crashtests bei einem Auto, oder mit enormen Schäden an der Umwelt, wie bei Atomkraftwerken. Ein Teil des Modells wird durch naturwissenschaftliche Erkenntnisse der Physik und Chemie bestimmt, aber ein Teil wird durch die Statistik beschrieben.

Synthetische Daten werden auf der Grundlage echter Daten erzeugt. Dafür muss die Verteilung der echten Daten gelernt werden. Im maschinellen Lernen werden generative Modelle genutzt, um Verteilungen zu lernen. Ein generatives Modell lernt eine Verteilung, so dass der Entstehungsprozess der zu erzeugenden Daten gut abgebildet wird [5]. Hier können zum Beispiel auch Teile modelliert werden, die keine reale Entsprechung bei der Messung des Crashtests besitzen.

Eine aktuelle Methode zum Lernen eines generativen Modells sind Generative Adversarial Nets (GAN) [16]. Mit einem GAN kann eine beliebige Wahrscheinlichkeitsverteilung P_g gelernt werden, wenn nur ein Datensatz D vorliegt, welcher der unbekannten zu lernenden Verteilung P_{data} folgt. In einem GAN erzeugt der Generator g mittels Zufallseingaben synthetische Datenpunkte $\boldsymbol{x}'$, welche der Verteilung P_g folgen. Der Kritiker c testet, ob es sich um ein Datum $\boldsymbol{x}$ handelt, das P_{data} oder P_g folgt.

Auch wenn schon gute Ergebnisse zum Beispiel bei der Bilderzeugung erzielt wurden, befassen sich die neuen Publikationen mit den Problemen des Ansatzes. Zum Beispiel kann der Generator in eine Situation geraten, wo er sich nicht mehr verbessern kann. In diesem Fall sagt der Kritiker perfekt vorher, ob ein Datenpunkt aus P_g oder P_{data} stammt. In dieser Arbeit wird dieses Problem durch eine geschickte Wahl des Modells angegangen. Anstelle von den klassisch verwendeten neuronalen Netzen werden zwei probabilistische graphische Modelle verwendet. Die Datenpunkte werden approximativ mit Perturb-and-MAP (PAM) [31] generiert.

Die Arbeit ist wie folgt aufgebaut: Zuerst werden in Kapitel 2 die benötigten statistischen Grundlagen generativer Modelle und von MRF kurz vorgestellt. Darauf aufbauend werden in Kapitel 3 Sampling-Methoden erklärt, mit denen Datenpunkte aus der Verteilung, welche über das MRF definiert ist, gezogen werden. Eine Einführung in GAN, der theoretischen Eigenschaften und Methoden des Trainings, wird in Kapitel 4 gegeben. Kapitel 5 enthält die Implementierung in dem Framework PX. Die Entwicklung des PAM-GAN, der angewandten Approximationen sowie der zugehörige Lernalgorithmus wird daran anschließend gezeigt. Die Experimente erfolgen mit der Generierung von MNIST-Daten und eines Textdatensatzes in Kapitel 6. Den Abschluss bildet eine Zusammenfassung und mögliche weitere Arbeiten in Kapitel 7.

Kapitel 2

Probabilistische Graphische Modelle

Das GAN in dieser Arbeit wird mittels eines PGM realisiert, deshalb werden im Folgenden die benötigten Grundlagen vermittelt. Es werden die zwei Lernaufgaben des maschinellen Lernens betrachtet, welche in einem GAN angewandt werden:

1. Unüberwachtes Lernen: Finden eines generativen Modells, das die Struktur der Daten gut beschreibt.

2. Überwachtes Lernen: Klassifikation der erzeugten Daten mit der Fragestellung, ob die Daten der Verteilung des generativen Modells folgen oder nicht.

Dafür werden zuerst statistische Grundlagen, insbesondere statistische Modelle und deren Parameterschätzung, eingeführt. Aufbauend darauf werden PGM eingeführt und die benötigten Eigenschaften erläutert, insbesondere die Berechnung der Wahrscheinlichkeit der Variablen eines PGM, was ein Markov Random Field (MRF) als ein mögliches PGM ausmacht, sowie mögliche Ansätze zum Finden einer geeigneten Struktur des MRF.

2.1 Grundlagen

Sei $D = (\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n})^T$ eine Menge von n Datenpunkten, Trainingsdaten genannt. Jeder Datenpunkt $\boldsymbol{x}$ besteht aus m Messungen $x_1, ..., x_m$ und einem Label y. In dieser Arbeit wird die Annahme getroffen, dass jede Messung x_i nur Werte aus einem diskreten Raum $\mathcal{X}_i$ annehmen kann. Die Menge aller möglichen Messungen ergibt sich zu $\mathcal{X} = \bigotimes_{i=1}^{m} \mathcal{X}_i$. Die Messungen können Parameter aus der Produktion von Gütern sein, wie die Länge einer Schraube oder die Temperatur beim Schmelzen von Stahl. Das Label y gibt zum Beispiel an, ob das Produkt fehlerfrei ist.

Die in diesem Abschnitt vorgestellten Grundlagen der Statistik sind aus [18]. Die Wahrscheinlichkeitstheorie nach den Kolmogoroff-Axiomen bildet die statistische Grundlage in dieser Arbeit. Dabei wird ein Grundraum Ω betrachtet, der alle möglichen Messungen ent-

hält. Mit Ereignissen $\omega \subseteq \Omega$ werden zum Beispiel nur die Messungen betrachtet, wo ein Gut einen Fehler aufweist.

Kolmogoroff-Axiome

Definition 2.1.

- $P(\omega) \in \mathbb{R}^+ \; \forall \; \omega \subset \Omega$
- $P(\Omega) = 1$
- Seien $\omega_1, \omega_2, ..., \omega_n$ disjunkte Ereignisse, dann gilt $P(\omega_1 \cup \omega_2 \cup ... \cup \omega_n) = \sum_{i=1}^{n} P(\omega_i)$

Werden mehrere Messungen durchgeführt, können die daraus folgenden möglichen Ereignisse mittels einer Funktion $X : \Omega \to \mathbb{R}$ einem Wert zugeordnet werden. Wird mit $X(\omega)$ eine Wahrscheinlichkeit angegeben, ist X eine Zufallsvariable. Die Messungen in D sind Realisierungen von Zufallsvariablen, zu jeder der m möglichen Messungen in $\boldsymbol{x}$ existiert eine zugehörige Zufallsvariable X_i, gesamt gegeben durch den Zufallsvektor $\boldsymbol{X} = (X_1, X_2, ..., X_m)^T$. Die Realisierung des Zufallsvektors wird mit $\boldsymbol{X} = \boldsymbol{x}$ angegeben, was Folgendes angibt $(X_1 = x_1, ..., X_m = x_m)^T$.

Mittels der Wahrscheinlichkeitsrechnung kann die Realisierung von $X_u = x_u$ angegeben werden unter der Prämisse, dass auch $X_v = x_v$ gemessen wurde,

$$P(X_u = x_u \mid X_v = x_v) = \frac{P(X_u = x_u \cap X_v = x_v)}{P(X_v = x_v)}, \tag{2.1}$$

was die bedingte Wahrscheinlichkeit ist. Mit $P(X_u = x_u \cap X_v = x_v)$ wird die gemeinsame Wahrscheinlichkeit von zwei Zufallsvariablen bezeichnet.

Da mehrere Zufallsvariablen vorliegen, kann auch die gemeinsame Wahrscheinlichkeit angegeben werden, dass eine bestimmte Realisierung $\boldsymbol{X} = \boldsymbol{x}$ vorliegt:

$$P(X_1 = x_1, ..., X_m = x_m) \tag{2.2}$$

Liegt eine Reihe von Messungen von möglichen Ereignissen vor, können diese einer Wahrscheinlichkeitsverteilung, kurz Verteilung, folgen. Eine Wahrscheinlichkeitsverteilung ist eine Funktion, die einer Realisierung der Zufallsvariablen eine Wahrscheinlichkeit zuordnet $P(\boldsymbol{X} = \boldsymbol{x}) \to [0, 1]$. Die Realisierungen, denen eine Wahrscheinlichkeit > 0 zugeordnet wird, sind die Träger der Verteilung. Die Wahrscheinlichkeit einer Realisierung $X_i = x_i$ einer Zufallsvariablen, welche einer spezifischen Verteilung folgt, wird mittels einer diskreten Wahrscheinlichkeitsdichte $p(X = x)$ berechnet, die mit einer Verteilung assoziert ist. Diskret bedeutet, dass die Menge der möglichen Zustände höchstens abzählbar unendlich ist.

2.2 Statistische Modelle

In dieser Arbeit soll ein Modell gelernt werden, das eine unbekannte Verteilung P_g lernt, um Datenpunkte x' zu generieren, die dieser Verteilung folgen. Von der gesuchten Verteilung ist nur die Stichprobe D gegeben, die dieser Verteilung folgt.

2.2.1 Parameterbestimmung

Sei $P_{\boldsymbol{\theta}}$ die Familie an möglichen parametrischen Verteilungen, wobei die einzelnen Mitglieder der Familie durch $\boldsymbol{\theta}$, den Parametervektor, eindeutig identifiziert werden. Durch $P_{\boldsymbol{\theta}}$ ist ein Modell der Verteilung von D beschrieben, deren Parameter unbekannt sind. Sei als Beispiel die univariate Normalverteilung gegeben, deren Parametervektor ist gegeben durch [18]:

$$\boldsymbol{\theta} = \left(\mu, \sigma^2\right)^T \tag{2.3}$$

Dabei ist μ der Erwartungswert der Verteilung und σ^2 deren Varianz. Die Dichte der Verteilung ist gegeben durch

$$p\left(x \mid \boldsymbol{\theta}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(x-\mu\right)^2}{2\sigma^2}\right) \ . \tag{2.4}$$

Die Parameter können mit der Maximum-Liklihood-Methode (ML-Methode) geschätzt werden [19]. Für die ML-Methode müssen die Datenpunkte $D = \left(x_1, ..., x_n\right)^T$ alle derselben Verteilung folgen und unabhängig voneinander sein. Die ML-Methode berechnet die Parameter der Verteilung, welche für D die maximale Wahrscheinlichkeit ergeben. Mit unabhängig verteilten Datenpunkten ergibt sich die Likelihood zu

$$\mathcal{L}\left(D \mid \boldsymbol{\theta}\right) = \frac{1}{n} \prod_{i=1}^{n} p\left(x_i \mid \boldsymbol{\theta}\right) \ . \tag{2.5}$$

Dies wird durch die Verwendung des Logarithmus vereinfacht, der eine monotone Funktion ist und nicht das Maximum verändert, zusätzlich wird durch den Logarithmus die Berechnung numerisch stabiler. Mittels der Maximierung des Parameters $\boldsymbol{\theta}$ ergibt sich die Maximum-Likelihood-Verteilung:

$$\max_{\boldsymbol{\theta}} \mathcal{LL}\left(D \mid \boldsymbol{\theta}\right) = \max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log\left(p\left(x_i \mid \boldsymbol{\theta}\right)\right) \tag{2.6}$$

Zur Berechnung der Parameter werden diese einzeln optimiert, für die Normalverteilung sind dies μ und σ^2. Zur Optimierung wird (2.4) einmal nach jedem Parameter μ, σ^2 abgeleitet und 0 gesetzt. Für die Normalverteilung ergeben sich die folgenden Schätzer der Parameter:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$
$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2$$

Das $\hat{\mu}$ ergibt als Schätzung den empirischen Mittelwert und die geschätzte Varianz ergibt sich als Abweichung vom Mittelwert. In vielen Fällen kann die $\mathcal{LL}$ nur mit einem iterativen Verfahren, wie dem Gradientenabstieg berechnet werden, siehe Kapitel 4.2. Wenn allerdings die Parameter $\boldsymbol{\theta}$ berechnet und gleichzeitig Datenpunkte x' generiert werden müssen, funktioniert die ML-Methode nicht. Hierbei kann der EM-Algorithmus [19] eingesetzt werden, der in einem ersten Schritt die fehlenden Datenpunkte generiert und anschließend die ML-Methode ausführt.

In [35] wurde gezeigt, dass das Lernen von Parametern einer Verteilung mit der ML-Methode nicht zu einen generativen Modell führen muss, das gute Datenpunkte erzeugt. Das Modell könnten eine gute Likelihood haben und trotzdem verrauschte Datenpunkte erzeugen. Als Lösung muss für eine Aufgabe ein geeignetes Maß genutzt werden. Ein anderes Maß ist die Kullback-Leibler-Divergenz (KL-Divergenz) [25], diese ist für zwei Verteilungen P und Q wie folgt definiert:

$$KL\left(P \mid\mid Q\right) = \int_{-\infty}^{\infty} p\left(x\right) \log \frac{p\left(x\right)}{q\left(x\right)} dx \tag{2.7}$$

Sie gibt die erwartete Differenz zwischen zwei Verteilungen an; wenn diese klein wird, sind sich die beiden Verteilungen ähnlich. Zu beachten ist außerdem, dass das Resultat der KL-Divergenz nicht identisch ist, wenn die Eingaben vertauscht sind $KL\left(P \mid\mid Q\right) \neq KL\left(P \mid\mid Q\right)$, weshalb es keine Distanz ist. Es ist zu beachten, das die Minimierung der KL-Divergenz dem Ergebnis der gemittelten ML-Methode entspricht, was auch eher genutzt wird, denn oft ist Q nicht gegeben.

2.2.2 Klassifikation

Mit Wahrscheinlichkeiten können auch Klassifikationen bestimmt werden [19]. Für eine neue Messung $\boldsymbol{X} = \boldsymbol{x}_i$ ist die zugehörige Klasse $y_i \in \mathcal{Y}$, zum Beispiel ob ein Gut defekt ist oder nicht, zu bestimmen. Dafür kann eine Wahrscheinlichkeit für y_i berechnet werden, dies ergibt aber keinen eindeutigen Wert. Der Bayes-Klassifikator ist ein Ansatz mit Wahrscheinlichkeiten zu klassifizieren und wählt die Klasse mit maximaler Wahrscheinlichkeit:

$$\hat{y}_i = f\left(x\right) = arg\max_{y} P\left(x_i \mid y\right) \tag{2.8}$$

Die Funktion $f : \mathcal{X} \to \mathcal{Y}$ ordnet einem Datenpunkt x_i die Klasse $\hat{y}_i$ zu. Zur Berechnung der Klassifikation betrachte nochmals (2.1). Durch Umstellen der Formel ergibt sich:

$$P(x_i \cap y_i) = P(x_i \mid y_i) P(y_i) \tag{2.9}$$

Alternativ hätte auch $P(y_i \cap x_i)$ betrachtet werden können, welches die gleiche Wahrscheinlichkeit hat. Setzt man die beiden Terme gleich und stellt diese um, ergibt sich die Bayes-Regel

$$P(x_i \mid y_i) = \frac{P(y_i \mid x_i) P(x_i)}{P(y_i)} \ . \tag{2.10}$$

Dieser Ausdruck wird für die Klassifikation durch die Wahl von y_i maximiert. $P(y_i \mid x_i)$ beschreibt die Wahrscheinlichkeit, mit der x_i auftritt, wenn y bekannt ist, und ist häufig einfach zu berechnen. Der zweite Ausdruck $P(x_i)$ ist die a-priori-Wahrscheinlichkeit und gibt die Auftrittswahrscheinlichkeit von x an, ohne weitere Informationen zu betrachten. Zum Beispiel kann dies die relative Häufigkeit von x_i in D sein. Der Nenner normiert dies, damit es sich um eine Wahrscheinlichkeit handelt. Ist allerdings die Klassifikation das Ziel, kann der Nenner ignoriert werden, denn er ist für jedes mögliche x gleich und führt nicht zu einer anderen Klassifikation. Mit der Bayes Regel ergibt sich der Bayes-Klassifikator zu

$$f(x) = arg \max_y P(y \mid x) P(x) \ . \tag{2.11}$$

2.3 PGM

Ein probabilistische Modell ist gegeben durch $\boldsymbol{X}$, deren gemeinsame Verteilung ist durch (2.2) gegeben. Alle m Zufallsvariablen aus $\boldsymbol{X}$ sind untereinander abhängig, aber es gibt bedingte Unabhängigkeiten zwischen Teilmengen der Zufallsvariablen. Die Zufallsvariable X_i ist unabhängig von X_j gegeben X_u, wenn Folgendes gilt [23]:

$$P(X_i \perp X_j \mid X_u), \text{ wenn } P(X_i \mid X_j \cap X_u) = P(X_i \mid X_u) \tag{2.12}$$

$$\text{oder wenn } P(X_j \cap X_u) = 0 \tag{2.13}$$

Dies erlaubt eine Vereinfachung der Berechnung der gemeinsamen Verteilung, wenn die bedingten Unabhängigkeiten einfach dargestellt werden können.

Hier kommen die PGM zum Einsatz, die ein probabilistisches Modell mit einem Graphen kombinieren. Die grundlegenden Begriffe der Graphtheorie stammen aus [9], in dieser Arbeit werden ausschließlich ungerichtete Graphen betrachtet. Sei ein Graph $G = (V, E)$ gegeben, dabei ist $V = \{v_1, ..., v_m\}$ die Menge der Knoten und $E \subset V \times V$ die Menge der Kanten. Für einen Knoten v_i sind alle Knoten $v_j \in V$, mit $i \neq j$ adjazent, zwischen denen eine Kante $e_{ij} \in E$ vorliegt. Die Nachbarschaft für einen Knoten $Nb(v_i)$ ist gegeben durch die adjazenten Knoten. Eine Clique ist die Teilmenge von Knoten $V_C \subseteq V$, bei denen

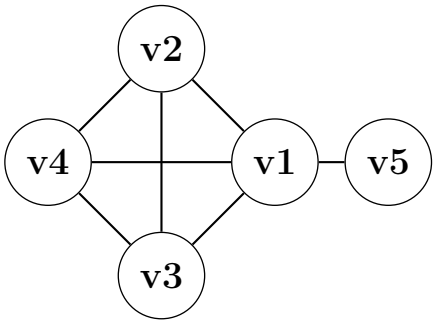


Abbildung 2.1: Beispielgraph, um die Grapheigenschaften zu zeigen.

für jedes Paar von Knoten $\forall v_i, v_j \in V_C$ eine Kante $e_{ij} \in E$ vorliegt. Anders gesagt, jeder Knoten $v_i \in V_C$ ist mit allen anderen Knoten $v_j \in V_C$, $i \neq j$ benachbart. Die Menge aller Cliquen eines Graphen G sei $\mathcal{C}$.

In Abbildung 2.1 ist ein ungerichteter Graph gegeben, um die grundlegenden Begriffe am Beispiel zu zeigen. Der Knoten v_3 ist adjazent zu den Knoten v_1, v_2 und v_4, die gemeinsam auch eine Clique bilden, die in dem gegebenen Graphen maximal ist, denn es gibt keine Clique mit mehr Knoten in dem Graphen. v_5 kann nicht Mitglied der Clique sein, denn es ist kein Nachbar von v_2, v_3, v_4, sondern nur von v_1, mit dem es eine paarweise Clique bildet. Es ist möglich für einen Knoten Mitglied verschiedener Cliquen zu sein.

Um einen Graphen G mit einer Verteilung P zu assoziieren, wird jede Zufallsvariable X_i mit einem Knoten v_i für $i = 1, ..., m$ assoziiert. Eine Kante e_{ij} bildet eine direkte Abhängigkeit zwischen X_i und X_j ab, womit der Graph G die Abhängigkeitsstruktur zwischen den Knoten angibt. Die bedingte Unabhängigkeit eines Graphen ist durch die Markov-Eigenschaft gegeben [23]:

Lokale Markov-Eigenschaft

Definition 2.2. Sei ein Knoten $v_i \in V$ eines Graphen G mit seiner Nachbarschaft $Nb(v_i)$ gegeben. Dann ist Knoten v_i unabhängig von allen anderen Knoten in V, die nicht zu seiner Nachbarschaft $Nb(v_i)$ gehören.

Für das Beispiel in Abbildung 2.1 heißt das, der Knoten v_5 ist unabhängig von v_2, v_3 und v_4 gegeben v_1, seinem einzigen Nachbarknoten.

Mittels des Graphen und der lokalen Markov-Eigenschaft kann die gemeinsame Verteilung faktorisiert werden. Ein Faktor wird dabei über eine Teilmenge von Knoten berechnet, den Cliquen des Graphen [37].

$$\psi_C : \left(\bigotimes_{s \in C} \mathcal{X}_s \right) \to \mathbb{R}^+ \tag{2.14}$$

Ein Faktor (2.14) ergibt ein Gewicht θ_i für eine mögliche Realisierung der Knoten einer Clique. Je größer θ_i ist, desto wahrscheinlicher ist die zugeordnete Realisierung der Knoten

der Clique. In einer Clique ist jeder Knoten mit jedem anderen Knoten benachbart, und daher ergibt sich θ_i über die Clique. Für den Beispielgraphen ergeben sich zwei Faktoren $\psi_{\{v_1,v_2,v_3,v_4\}}$ und $\psi_{\{v_1,v_5\}}$.

Diese Faktorisierung teilt G auf und berechnet die Wahrscheinlichkeiten mit den lokalen Werten der Cliquen. Zur Vereinfachung der Notation werden in den Formeln die Zufallsvariablen $\boldsymbol{X}$ weiterhin mit X_i notiert. Das Produkt der Faktoren der Cliquen entspricht einer unnormalisierten Wahrscheinlichkeitsdichte:

$$p\left(\boldsymbol{X}=\boldsymbol{x}\right) \propto \prod_{C\in\mathcal{C}} \psi_C\left(x_C\right) \tag{2.15}$$

Zur Normalisierung von (2.15) wird die Partition Function A genutzt:

$$A = \sum_{\boldsymbol{x}\in\mathcal{X}} \prod_{C\in\mathcal{C}} \psi_C\left(x_C\right) \tag{2.16}$$

Dieser Wert ergibt sich als Summe über die Produkte aller möglichen Realisierungen der Zufallsvariablen und die Berechnung enthält exponentiell viele Terme. Die gemeinsame Verteilung der Zufallsvariablen eines MRF ist gegeben durch die Dichtefunktion der Gibbs-Verteilung [37]:

$$p\left(\boldsymbol{X}=x\right) = \frac{1}{A}\prod_{c\in\mathcal{C}} \psi_c\left(x_c\right) \tag{2.17}$$

$$A = \sum_{x\in\mathcal{X}} \prod_{c\in\mathcal{C}} \psi_c\left(x_c\right) \tag{2.18}$$

Für einen ungerichteten Graphen, dessen Faktorisierung wie geschildert erfolgt, spricht man von einem Markov Random Field (MRF). Für ein MRF muss gelten $P\left(\boldsymbol{X}=\boldsymbol{x}\right) > 0$, für jede mögliche Realisierung der Zufallsvariablen. Die Wahrscheinlichkeit eines einzelnen Knotens hängt nur von seinen Nachbarn ab $P\left(X_i = x_i \mid Nb\left(X_i\right)\right)$ und von keinem weiteren Knoten [14].

Markov Random Field

Definition 2.3. Sei $G=(V,E)$ ein Graph und $\mathcal{C}\subseteq V$ die Menge der Cliquen des Graphen. Dann handelt es sich bei dem Vektor von Zufallsvariablen $\boldsymbol{X}$ um ein MRF genau dann, wenn $P\left(\boldsymbol{X}=\boldsymbol{x}\right)$ der Gibbs-Verteilung folgt, die durch $\mathcal{C}$ definiert ist.

Ein Beweis hierzu das es sich bei dem Graphen um ein MRF handelt ist in [17] veröffentlicht. Da das MRF der Gibbs-Verteilung folgt, ist ein MRF durch die Realisierungen der Cliquen des zugehörigen Graphen G gegeben.

Im weiteren Verlauf der Arbeit wird ein MRF als Exponentialfamilie [37] angegeben. Eine Exponentialfamilie besitzt eine Dichte der Form

$$p_{\boldsymbol{\theta}}\left(\boldsymbol{X}=\boldsymbol{x}\right) = h\left(\boldsymbol{X}=\boldsymbol{x}\right)\exp\left(\boldsymbol{\theta}^T\phi\left(\boldsymbol{X}=\boldsymbol{x}\right) - \log A\left(\boldsymbol{\theta}\right)\right)\ . \tag{2.19}$$

Für den weiteren Verlauf wird $h(\boldsymbol{X} = \boldsymbol{x}) = 1$ gesetzt, $A(\boldsymbol{\theta})$ ist weiterhin die Partition Function. $\phi(\boldsymbol{X} = \boldsymbol{x})$ ist die suffiziente Statistik und $\boldsymbol{\theta}$ ist der kanonische Parameter.

$\phi(\boldsymbol{X} = \boldsymbol{x})$ kodiert für ein MRF die Realisierung der Cliquen. Der Raum der möglichen Realisierungen einer Clique ist durch $\bigotimes_{s \in C} \mathcal{X}_s$ gegeben. Sei für jedes $s_i \in \mathcal{X}_s$ eine Indikatorfunktion gegeben

$$\mathbb{1}_{s_i}(s) = \begin{cases} 1, & \text{wenn } s_i = s \\ 0, & \text{sonst} \end{cases}, \tag{2.20}$$

wodurch ein Graph als Vektor von Indikatorfunktionen kodiert wird. Diese Darstellung heißt *overcomplete* und hat zwar den Nachteil, dass die Dichte der Gibbs-Verteilung nicht mehr eindeutig durch $\boldsymbol{\theta}$ indexiert ist, aber algorithmisch bietet sie für die Inferenz und Parameterschätzung Vorteile [37]. Einer jeden Realisierung s_i einer Clique C ist ein Gewicht θ_i zugeordnet. So ergibt sich für eine Clique C mit der Teilmenge an Knoten V_c der Faktor zu $\psi_c\left(\boldsymbol{\theta}_c^T \phi(V_c)\right)$. Für $\boldsymbol{X} = \boldsymbol{x}$, für alle Knoten des Graphen, gibt $\phi(\boldsymbol{X} = \boldsymbol{x})$ die Kodierung aller Cliquen $\mathcal{C}$ des Graphen an und $\boldsymbol{\theta}$ die zugehörigen Gewichte.

Die Dichte für ein MRF ergibt sich damit zu

$$p_{\boldsymbol{\theta}}(\boldsymbol{X} = \boldsymbol{x}) = \exp\left(\langle \phi(\boldsymbol{X} = \boldsymbol{x},), \boldsymbol{\theta} \rangle - \log A(\boldsymbol{\theta})\right) . \tag{2.21}$$

Die Wahrscheinlichkeitsdichte in (2.21) ist die Verteilung mit maximaler Shannon-Entropie:

$$H(p) := -\int_{\mathcal{X}} \log(p(x))\, p(x)\, dx \tag{2.22}$$

Wenn die Shannon-Entropie für eine diskrete Verteilung maximal ist, dann herrscht für die einzelnen Realisierungen der Zufallsvariablen die maximale Unsicherheit beim Ziehen von Datenpunkten aus dieser Verteilung [11].

2.3.1 Inferenz in einem MRF

Mit einem MRF wird ein Modell beschrieben und die Abhängigkeiten zwischen den Zufallsvariablen modelliert. Seien für dieses Kapitel der Graph und die zugehörigen Gewichte gegeben. Mit einem gelernten Modell bestehend aus der Gibbs-Verteilung in Form der Exponentialfamilie und einem Graphen G können Berechnungen durchgeführt werden. Wenn beispielsweise ein Bild gegeben ist, in dem manche Teile nicht aufgenommen wurden, können die fehlenden Pixel mit einem MRF berechnet werden. In einem Text kann die Vorhersage gemacht werden, was das wahrscheinlichste nächste Wort ist, gegeben die vorherigen Wörter.

Im Allgemeinen ist die nötige Aufgabe hierfür die Randverteilung einer Teilmenge $V_o \subseteq V$ der Knoten zu berechnen. Sei $V_r = \{V \setminus V_o\}$ die Menge der Knoten aus V, die nicht in

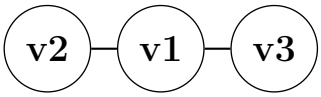


Abbildung 2.2: Beispielgraph für die Berechnung der Randverteilung

V_o sind. Die Knoten aus V_o liegen in einer Realisierung $\boldsymbol{X}_o = \boldsymbol{x}_o$ vor, die Randverteilung der Knoten ergibt sich als Summe über alle möglichen Realisierungen von $V_r = v_r$ gegeben durch $\boldsymbol{X}_r = \boldsymbol{x}_r$ [23]:

$$p\left(\boldsymbol{X}_o = \boldsymbol{x}_o\right) = \sum_{\boldsymbol{X}_r = \boldsymbol{x}_r} p\left(\boldsymbol{X}_o = \boldsymbol{x}_o, \boldsymbol{X}_r = \boldsymbol{x}_r\right) \tag{2.23}$$

In Abbildung 2.2 sei $\mathcal{X} = \{1, 2, 3\}$ für jeden Knoten gegeben. Die Zufallsvariable X_1 Des Knoten v_1 liege in der Realisierung 2 vor, die Randverteilung ergibt sich zu

$$p\left(X_1 = 2\right) = \sum_{X_2} \sum_{X_3} p\left(X_1 = 2, X_2 = x_2, X_3 = x_3\right) \ .$$

Wenn die Randverteilung verschiedener Knoten berechnet würde, müsste eine Reihenfolge festgelegt werden und dann würden die Randverteilungen nacheinander berechnet werden. Hierbei müssten dieselben Berechnungen mehrfach ausgeführt werden, wodurch die Komplexität exponentiell in der Dimension von $\mathcal{X}$, den möglichen Realisierungen der Knoten, wäre. Ein Ansatz, der die Anzahl an Berechnungen reduziert, ist die Belief Propagation (BP) [10]. Dieses Verfahren wird auch Sum-Product-Algorithmus (SP-Algorithmus) genannt. Das Verfahren existiert in zwei Formen, dem Sum-Product-Algorithmus (SP-Algorithmus) und dem Max-Product-Algorithmus (MP-Algorithmus). Mit dem SP-Algorithmus wird die Randverteilung der Knoten berechnet. Im Gegensatz dazu berechnet der MP-Algorithmus die Realisierung der Knotenmenge V mit der maximalen Wahrscheinlichkeit, sprich von $\boldsymbol{X}$. Da beide Verfahren in der Arbeit eingesetzt wurden und sich die Berechnungen nur an einer Stelle unterscheiden, wird der SP-Algorithmus vorgestellt und an der entsprechenden Stelle der Berechnung auf den Unterschied zum MP-Algorithmus verwiesen. Sei ein MRF gegeben durch den Graphen G und die Gibbs-Verteilung p. Die BP wurde auf Graphen mit der Struktur eines Baumes definiert, was bedeutet, dass die Cliquen durch die Kantenmenge E des Graphen und der Knotenmenge V gegeben sind. So besteht $\phi\left(\boldsymbol{X} = \boldsymbol{x}\right)$ aus den Indikatorfunktionen für die möglichen Realisierungen der Knoten und Kanten, ebenso ist $\boldsymbol{\theta} = (\boldsymbol{\theta}_i, \boldsymbol{\theta}_{ij}) \ \forall v_i \in V, \forall e_{ij} \in E$. So ergibt sich die Dichte der gemeinsame Verteilung des MRF zu

$$p\left(\boldsymbol{X} = \boldsymbol{x}\right) = \frac{1}{A} \prod_{i=1}^{m} \psi_i\left(X_i\right) \prod_{i=1}^{m} \prod_{j=1}^{Nb(X_i)} \psi_{ij}\left(X_i, X_j\right) \tag{2.24}$$

Um die Randverteilung einer Zufallsvariable zu berechnen, muss die Summe über die restlichen gebildet werden. Hierbei ist zu beachten, dass die Randverteilung der Zufallsvariablen X_i eines Knotens v_i nur durch seine direkten Nachbarn $Nb\left(v_i\right)$ beeinflusst wird. In

Abbildung 2.2 wird der Knoten v_1 nur durch die Nachbarn v_2 und v_3 beeinflusst. Die BP berechnet für einen Knoten zwei Werte, einmal den Glauben (Belief) über den eigenen Zustand und daraus folgend den Belief über die möglichen Realisierungen der benachbarten Knoten. Dieser wird über Nachrichten $m_{ij}(v_j)$ zwischen den Knoten ausgetauscht. Um eine Nachricht an einen Nachbarn v_j zu senden, muss der Knoten v_i von allen anderen Nachbarn $\{Nb(v_i) \setminus v_j\}$ eine Nachricht m_{ik} $\forall v_k \in Nb(v_i)$ erhalten haben. Im SP-Algorithmus werden die Nachrichten mit der Formel

$$m_{ij}^{new}(v_j) = \sum_{X_i} \psi_{ij}(X_i, X_j)\, \psi_i(X_i) \prod_{k \in Nb(v_i) \setminus v_j} m_{ki}^{old}(v_i) \tag{2.25}$$

berechnet, aus der sich die Randverteilung der Knoten ausrechnen lässt. Für den MP-Algorithmus lautet die Formel für eine Nachricht:

$$m_{ij}^{new}(v_j) = \max_{X_i} \psi_{ij}(X_i, X_j)\, \psi_i(X_i) \prod_{k \in Nb(v_i) \setminus v_j} m_{ki}^{old}(x_i) \tag{2.26}$$

Der MP-Algorithmus berechnet die maximale Wahrscheinlichkeit der gemeinsamen Verteilung des MRF, aber nicht direkt die zugehörigen Realisierungen der Knoten oder deren Randverteilung. Die Idee zur Bestimmung der Realisierungen ist ein simples Backtracking. Dafür werden die Nachrichten zur Berechnung der maximalen Wahrscheinlichkeit vom ersten Knoten v_1 bis zum letzten Knoten v_m gesandt, eine Nachricht geht von Knoten v_{n-1} zu Knoten v_n und beruht auf allen anderen Nachbarn $\{Nb(v_{n-1}) \setminus v_n\}$. Dabei gibt es in der Zufallsvariable X_{n-1} des Knotens v_{n-1} eine oder mehrere Realisierungen, die zu einer maximalen Wahrscheinlichkeit in der Zufallsvariable X_n des Knotens v_n führen, diese werden für das Backtracking gespeichert. Bei der Zufallsvariable X_m von Knoten v_m ist die Realisierung der maximalen Wahrscheinlichkeit bekannt, dann wird Knoten v_{m-1} betrachtet und eine der möglichen Realisierungen ausgewählt. Diese Prozedur wird fortgesetzt, bis man bei der Zufallsvariable X_1 Knoten v_1 ist und somit die Realisierung der Knotenmenge für die maximalen Wahrscheinlichkeit ausgewählt wurde [5]. So ist die Realisierung $\boldsymbol{X} = \boldsymbol{x}$ des MRF gegeben, mit der maximalen Wahrscheinlichkeit welche als MAP des MRF verwendet wird.

Die Belief eines Knotens kann mit den eingehenden Nachrichten berechnet werden und entspricht für einen Graphen mit einer Baumstruktur der Randverteilung

$$b_i(v_i) \propto \psi_i(X_i) \prod_{k \in Nb(v_i)} m_{ki}(v_i)\ . \tag{2.27}$$

Für einen Graphen, der einen Zyklus enthält, gibt es Pfade im Graphen $v_i, ..., v_j$, für die $v_i = v_j$ gilt. In dem Fall ergibt die BP nur eine Approximation für die Randverteilungen und die MAP, da ein Knoten jetzt mehrfach dieselbe Nachricht bekommen kann. Die BP wird trotzdem einfach angewandt und es wird von der Loopy Belief Propagation

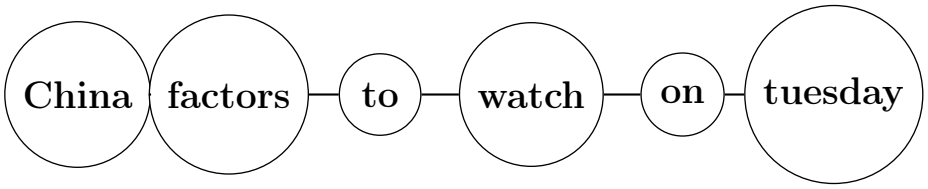


Abbildung 2.3: Graph einer Überschrift von Reuters [24], wobei jeder Knoten in einer Realisierung vorliegt, die durch das entsprechende Wort gegeben ist.

(LBP) gesprochen. Zum Start verschickt jeder Knoten v_i eine Nachricht $m_{ij} = 1$ an seine Nachbarschaft $Nb(v_i)$, damit jeder Knoten des MRF in der Lage ist eine Nachricht zu verschicken. Die LBP liefert eine gute Approximation der Randverteilung [29] welche für viele Anwendungen ausreichend ist.

2.3.2 Lernen eines Modells des MRF

Ein MRF besteht aus der durch den Graphen G gegebenen Abhängigkeitsstruktur, welche in $\phi(\boldsymbol{X} = \boldsymbol{x})$ kodiert wird, und dem Gewichtsvektoren $\boldsymbol{\theta}$ der jeweiligen Realisierung der Cliquen. Zur Modellbildung gehören zwei Aufgaben, für die in beiden Fällen derselbe Trainingsdatensatz D vorliegt:

1. Finden einer geeigneten Struktur
2. Berechnen des zugehörigen Gewichtsvektors

Bestimmung eines geeigneten Graphen

Das Resultat soll ein generatives Modell sein, wobei der Graph die Abhängigkeitsstruktur zwischen den Zufallsvariablen abbildet. Die Abhängigkeitsstruktur eines generativen Modells kann auch Knoten enthalten, die keiner realen messbaren Größe zugeordnet sind [5]. Die Struktur wird dazu genutzt Zusammenhänge zu modellieren.

Je nach verwendetem Datensatz sieht die Abhängigkeitsstruktur anders aus. In dieser Arbeit werden zwei einfache Abhängigkeitsstrukturen genutzt, eine Kette und ein Gitter. Die Struktur des Graphen in Abbildung 2.3 bildet Sätze ab, die aus sechs Strings bestehen, deren Abhängigkeiten sich aus dem vorherigen und dem nachfolgenden Wort ergeben, was eine Kette von Knoten ergibt.

Um ein Bild als MRF abzubilden, kann jeder Pixel als Zufallsvariable betrachtet werden. Ein Pixel hängt so mit seinen direkten horizontal und vertikal benachbarten Pixeln zusammen (Vierer-Nachbarschaft). Für die Rekonstruktion von Bildern wurden schon von [14] Gitter-Graphen eingesetzt, was deshalb in dieser Arbeit als generatives Modell genutzt wurde.

Für die Betrachtung von Bildern wird auch noch zusätzlich ein regularisierter Ansatz getestet [38]. Dabei wird ein vollständiger Graph betrachtet und bestimmte Kanten, die ei-

ne Bedingung nicht erfüllen, werden gestrichen, was einen dünn besetzten Graphen ergibt. Dies ist besonders für das Lernen von Verteilungen wichtig, wo die Anzahl an Zufallsvariablen höher ist als die Größe des Trainingsdatensatzes: $m >> n$. Dieses Verfahren heißt ELEM-GM-Schätzer und optimiert folgende Formel:

$$\begin{aligned} &\min_{\theta} || \boldsymbol{\theta} ||_{1,E} \\ &\text{s.t. } || \boldsymbol{\theta} - B^*_{trw}\left(\hat{\phi}\right) ||_{\infty,E} \end{aligned} \tag{2.28}$$

Wobei $\bar{\boldsymbol{\theta}} := B^*_{trw}$ eine Approximation der Entropie des MRF darstellt und $\hat{\phi}$ sind die Mittelwerte über die suffiziente Statistik ϕ. Die approximierte Entropie B^*_{trw} ergibt sich durch die Approximation des Graphen des MRF als Baum [36]. Die einzelnen Elemente von $\bar{\boldsymbol{\theta}}$ sind gegeben durch folgende Formeln [38]:

$$\bar{\theta}_{s;j} = \log \hat{\phi}_{s;j} \tag{2.29}$$

$$\bar{\theta}_{st;jk} = \rho_{st} \log \frac{\hat{\phi}_{st;jk}}{\hat{\phi}_{s;j}\hat{\phi}_{t;k}} \tag{2.30}$$

Wobei $\hat{\phi}_{s;j}$ für die Zufallsvariable X_s eines Knoten v_s den Mittelwert über D angibt, dass X_s in Realisierung j vorlag. Analog ergibt sich für eine Kante e_{st} mit den Knoten v_s, v_t $\hat{\phi}_{st;jk}$ der Mittelwert, das die zugeordneten Zufallsvariblen X_s, X_t in Realisierung $X_s = j$ und $X_t = k$ vorlagen in D. ρ_{st} wird bei der Approximation des MRF für die Kanten berechnet.

In der zweiten Formel in (2.28) ist $|| \cdot ||_{\infty,E}$ die Maximumsnorm über die Kanten und gibt das Maximum der Subtraktion zurück, wofür es eine geschlossene Form gibt mit der Soft-Thresholding-Funktion:

$$S_\lambda\left(\bar{\boldsymbol{\theta}}\right)_i = \text{ sign}\left(\bar{\theta}_i\right) \max\left(\mid \bar{\theta}_i \mid -\lambda, 0\right) \tag{2.31}$$

In (2.31) werden $\bar{\theta}_i$, die kleiner als λ sind, 0 gesetzt. Dies wird sowohl für die Knoten als auch für die Kantengewichte berechnet. Hierbei wird für jede Kante das größte zugeordnete Gewicht θ_i betrachtet. Sollte dies kleiner als λ sein, wird die Kante nicht in den Graphen aufgenommen, wodurch der Graph dünn besetzt ist. Dieses Verfahren ist einfacher als die klassischen Ansätze und bedarf keiner aufwendigen Optimierungsverfahren.

Berechnung der Parameter

Das Problem wird hier nur kurz betrachtet, da es in dieser Arbeit mittels des Generative Adverserial Training gelöst wird. Wenn der Parametervektor $\boldsymbol{\theta}$ der unbekannten Verteilung, aus der D stammt, berechnet werden soll, kann dies mit der ML-Methode (2.6) geschehen, in welche die Dichte der Exponentialfamilie (2.21) eingesetzt wird:

$$\mathcal{L}\left(D \mid \boldsymbol{\theta}\right) = \prod_{i=1}^{n} \exp\left(\left\langle \phi\left(\boldsymbol{X} = \boldsymbol{x}\right), \boldsymbol{\theta} \right\rangle - \log A\left(\boldsymbol{\theta}\right)\right) \tag{2.32}$$

Die Likelihood (2.32) kann mit dem Logarithmus vereinfacht werden und muss dann maximiert werden, gemäß der ML-Methode. Dabei müsste diese Likelihood mit einer Gradientenmethode optimiert werden. Für den Gradienten wird die Ableitung in jeder Dimension von (2.32) gebildet, was die stärkste Steigung der Funktion angibt. Dann werden die Parameter iterativ verbessert, siehe Kapitel 4.2.

Kapitel 3

Sampling

Das MRF ist gegeben durch die Gibbs-Verteilung $P_{\boldsymbol{\theta}}(x)$, welche durch die Realisierung der Cliquen vorliegt. Um einen Datenpunkt aus der Verteilung zu ziehen, muss (2.16) berechnet werden, was aber nicht effizient und exakt möglich ist. Eine Gruppe von Algorithmen ist durch die Markov Chain Monte Carlo (MCMC) Verfahren gegeben [1], die eine einfachere Verteilung $Q(x)$ nutzen, die um einen Faktor ϵ von der originalen Verteilung abweicht. Im Folgenden werden die Grundlagen vom MCMC vorgestellt und daran anschließend die Realisierung des Perturb-and-MAP (PAM) Samplers.

3.1 Grundlagen Markov Chain Monte Carlo

Die Grundlagen zu MCMC stammen aus [1]. In diesem Abschnitt wird das Ziehen von Datenpunkten x aus diskreten Mengenräumen $\mathcal{X}$ betrachtet.

Das Monte-Carlo-Prinzip ist anwendbar mit einem Datensatz D, wo jedes Datum x einer Verteilung P folgt. Wenn D ausreichend groß ist, kann mittels des Monte-Carlo-Prinzips die Zielverteilung approximiert werden, oder deren Momente. Ein Beispiel ist der Erwartungswert, der bei einem ausreichend großen D durch den Mittelwert approximiert werden kann.

$$I_n(f) = \frac{1}{n}\sum_{i=1}^{n} f(X = x_i) \rightarrow_{n\rightarrow\infty} I(f) = \int_{\mathcal{X}} f(X = x)\, p(X = x)\, dx \tag{3.1}$$

$f : \mathcal{X} \rightarrow \mathbb{R}$ gibt einen Wert für eine Realisierung an und p ist die Dichtefunktion von P.

Das Ziel von MCMC-Methoden ist es Datenpunkte aus einer Verteilung P_θ zu ziehen, zum Beispiel der Verteilung eines MRF. Die Datenpunkte werden iterativ erzeugt. Es handelt sich bei der Prozedur um eine Markov-Kette, wenn der aktuelle Datenpunkt x_i nur auf dem vorherigen Datenpunkt x_{i-1} beruht:

$$P(X = x_i \mid X = x_{i-1}, ..., X = x_1) = T(X = x_i \mid X = x_{i-1}) \tag{3.2}$$

T ist eine stochastische Transitionsmatrix und beschreibt die Änderungen der Wahrscheinlichkeiten der möglichen Realisierungen der Zufallsvariablen der Verteilung P. Wenn T für jede Änderung der Realisierung eine positive Wahrscheinlichkeit enthält und sich nicht immer die Wahrscheinlichkeit derselben Realisierung ändert, erhält man als Ergebnis eine invariante Verteilung, die der gesuchten Verteilung $P_{\boldsymbol{\theta}}$ entspricht.

Gerade für PGM ist der Gibbs-Sampler eine bedeutende MCMC-Methode. Sei $\boldsymbol{X} = (X_1, ..., X_m)^T$ ein Vektor von Zufallsvariablen und die bedingten Wahrscheinlichkeiten

$$P_{\boldsymbol{\theta}}\left(X_i \mid X_1, ..., X_{i-1}, X_{i+1}, ..., X_m\right) \tag{3.3}$$

gegeben. Das Verfahren zieht dann M^{Gibbs} mal $\forall X_i \in \boldsymbol{X}$ eine mögliche Realisierung, in Iteration k gegeben durch

$$X_i^k \sim P_\theta\left(X_i \mid X_1^k, ..., X_{i-1}^k, X_{i+1}^{k-1}, X_m^{k-1}\right). \tag{3.4}$$

Das heißt, dass die Realisierung einer Zufallsvariable X_i in Iteration k nur von den bisher aktualisierten Zufallsvariablen abhängt, und die Zufallsvariablen $X_{i+1}, ..., X_m$ noch den Wert von Iteration $k-1$ enthalten. Für ein MRF sei zu bedenken, dass die bedingte Wahrscheinlichkeit des Knoten v_i, welcher mit X_i assoziiert wurde, gegeben ist durch $P_\theta\left(X_i \mid Nb\left(X_i\right)\right)$ und daher einfach berechnet werden kann. Zu Beginn des Gibbs-Samplers wird die Realisierung jeder Zufallsvariablen zufällig ausgewählt und dann folgt eine Burn-In-Phase, in der K Iterationen (3.4) für jede Zufallsvariable aufgerufen werden, damit die Verteilung des Gibbs-Samplers der gesuchten Verteilung P_θ entspricht [13]. Die in der Burn-In-Phase erzeugten Datenpunkte werden verworfen. Dann können Datenpunkte aus der Verteilung gezogen werden. Damit diese allerdings unabhängig verteilt sind, müsste nach jedem Aufruf erneut eine Burn-In-Phase durchgeführt werden.

Insgesamt ergibt sich für den Gibbs-Sampler eine Kette an Aufrufen des Samplers, um einen Datenpunkt x, der der Verteilung P_θ folgt, zu erhalten. Ist der Gradient dieses Verfahrens benötigt, müsste jede Iteration bei der Erzeugung abgeleitet werden, um das Resultat zu erhalten, was mit dem Gibbs-Sampler nicht praktisch umgesetzt werden kann. Folgend wird eine Methode zum Sampling von Datenpunkten vorgestellt, die eine Art One-Shot-MCMC darstellt und effizient zu berechnen ist.

3.2 Grundlagen Perturb and Map

Das Ziel von Perturb-and-MAP (PAM) [31] ist es Datenpunkte zu generieren, die der Gibbs-Verteilung (2.21) folgen oder nur gering von dieser abweichen. Im Folgenden werden die Grundlagen des PAM-Verfahrens vorgestellt, die aus der Online-Entscheidungstheorie stammen. Dort geht es darum, Algorithmen zu entwickeln, die zu einem Zeitpunkt $t = 1$, ..., *TIMESTEPS* eine der möglichen Entscheidung d_i^t treffen, die mit Kosten L_i^t verbunden sind für $i = 1, ..., n$. Das Ziel dieser Algorithmen ist es, die Entscheidung zu einem Zeitpunkt

t so zu treffen, dass die Kosten am Ende minimal sind. Dabei liegen nur Informationen zu den Kosten der Entscheidungen von den Zeitpunkten $0, ..., t-1$ vor, aber zukünftige Kosten sind nicht bekannt [22].

Seien $L_i^t \in [0,1]\,, \forall i = 1, ..., n, \forall t \in TIMESTEPS$ die Kosten einer Entscheidung d_i^t zu einem Zeitpunkt t, sodass die Kosten als Wahrscheinlichkeiten interpretierbar sind. Dann sei das Ziel einen Algorithmus zu entwickeln, der zu einem Zeitpunkt t die Realisierung des MRF mit den geringsten Kosten wählt. Die Anzahl der möglichen Entscheidungen ist gegeben durch die Größe des Zustandsraums des MRF, $\mathcal{X}^m$, welcher alle möglichen Realisierungen aller Zufallsvariablen $\boldsymbol{X}$ des MRF umfasst. Die Realisierung mit den minimalen Kosten kann durch einen entsprechend parametrisierten Aufruf des MP-Algorithmus mit anschließendem Backtracking erfolgen.

3.2.1 Following the Perturbed Leader

Sei für jede mögliche Realisierung des MRF ein Experte gegeben, der eine Entscheidung d_i^t trifft, die mit Kosten L_i^t verbunden ist. Eine Entscheidung ist eine mögliche Realisierung des MRF $i = 1, ..., \mid \mathcal{X}^m \mid$. In [22] wurde zur Lösung des Entscheidungsproblems der Algorithmus "Following the Perturbed Leader"(FPL) eingeführt (vergleiche Algorithmus 1).

for $t = 1$ **to** $TIMESTEPS$ **do**
  Ziehe für jede mögliche Entscheidung d_i^t, $i \in \{1, 2, ..., n\}$
  $z\left(d_i^t\right) \sim$ Exponentialverteilung mit Dichte $p_t\left(d\right) = \lambda \exp\left(-\lambda d\right)$;
  Treffe die Entscheidung d_i, für die $L^{<t}\left(d_i\right) - p^t\left(d_i\right)$ minimal ist
  mit $L^{<t}\left(d_i\right) =$ bisherige Kosten von Entscheidung d_i;
end

Algorithmus 1 : FPL-Algorithmus.

Anstelle die gegebenen Kosten L_i^t zum Zeitpunkt t berücksichtigen, werden diese mit einer Zufallszahl $z\left(d_i\right)$ verzerrt, die einer Exponentialverteilung folgt, welche durch die Dichtefunktion $p^t\left(d_i\right)$ gegeben ist. Die bisherigen Kosten einer Entscheidung $L_i^t\left(d_i\right)$ werden berücksichtigt und durch die Verzerrung mit p_t kommt der Zufall herein. Dies ergibt einen einfach zu implementierenden Algorithmus, dessen erwartete Kosten nur gering von den optimalen Kosten abweichen [22].

Eine Verbesserung der Approximation von FPL wurde in [26] eingeführt, dafür wird der Verlust L_i^t zu einem Zeitpunkt t, der immer noch verzerrt wird, durch eine Zufallsvariable C_i ersetzt. Diese Zufallsvariable ist exponentialverteilt mit dem Parameter λ und für den Verlust durch eine Entscheidung d_i^t ergibt sich $C_i \sim \exp\left(-\lambda L_i^{<t}\right)$. Wenn C_i exponential-

verteilt sind und $I = arg\min_i C_i$ gewählt wurde, dann gilt $P(I = i) \sim \exp\left(-\lambda L_i^{<t}\right)$ [26]. Die Wahrscheinlichkeit ergibt sich zu

$$P(I = i) = \frac{\exp\left(-\lambda L_i^{<t}\right)}{\sum_j \exp\left(-\lambda L_j^{<t}\right)} . \tag{3.5}$$

Sind die Zufallsvariablen $C_1, ..., C_n$ unabhängig identisch verteilt, dann ergibt sich die Wahrscheinlichkeit des Minimums für bestimmte Verteilungen zu

$$P(argmin(C_1, ..., C_n) = i) \sim \exp(-L_i) . \tag{3.6}$$

Damit dies erfüllt ist, muss eine entsprechende Verteilung gewählt werden, weshalb im PAM-Sampling die Gumbel-Verteilung gewählt wurde. Dies bildet die Grundlage für das PAM-Sampling, was folgend vorgestellt wird.

3.3 PAM Sampling

In [31] wurde die Idee des FPL, insbesondere die Berechnung der Wahrscheinlichkeit, die in (3.6) gegeben ist, auf die Problemstellung des Samplings angewandt. Aufbauend auf der Exponentialfamilie (2.21), ergibt sich, dass die Wahrscheinlichkeitsdichte für eine Realisierung proportional zu

$$p_\theta(\boldsymbol{X} = \boldsymbol{x}) \propto \langle \phi(\boldsymbol{x}), \boldsymbol{\theta} \rangle \tag{3.7}$$

ist, was der unnormalisierten Wahrscheinlichkeit entspricht. Verzerrt wird $\boldsymbol{\theta}$ mit anschließender Minimierung der Wahrscheinlichkeit des MRF, sodass sich das Modell für das Sampling ergibt zu:

$$x(\boldsymbol{z}) = arg \min_{\boldsymbol{x'} \in \mathcal{X}^m} \langle \phi(\boldsymbol{x'}), \boldsymbol{\theta} + \boldsymbol{z} \rangle \tag{3.8}$$

$x(\boldsymbol{z})$ ist der erzeugte Datenpunkt und $\boldsymbol{z}$ ist der Zufallsvektor der Verzerrung, welcher der Gumbel-Verteilung folgt. Mit $\phi(\boldsymbol{x'})$ ist weiterhin die suffiziente Statistik des MRF gemeint. Die Wahrscheinlichkeitsdichte der Gumbel-Verteilung ist gegeben durch

$$\mathcal{G}(x, \mu) = \exp((x - \mu) - \exp(x - \mu)) \tag{3.9}$$

mit μ als Modus der Verteilung, also dem Punkt mit der maximalen Wahrscheinlichkeitsmasse. Die Dichte der Gumbel-Verteilung für $\mu = 0$ wird in Abbildung 3.1 gezeigt.

Die Theorie des PAM-Samplings baut darauf auf, dass nach dem Verzerren das Minimum berechnet wird, wofür die Dichte der Gumbel-Verteilung wie in (3.9) parametrisiert wird. Ist das Maximum der Wahrscheinlichkeit gesucht, müssen nur die Parameter geändert werden zu $\mathcal{G}(-x + \mu, \mu)$, da die Wahrscheinlichkeit in der Gumbel-Verteilung für das Maximum das gespiegelte Minimum ist.

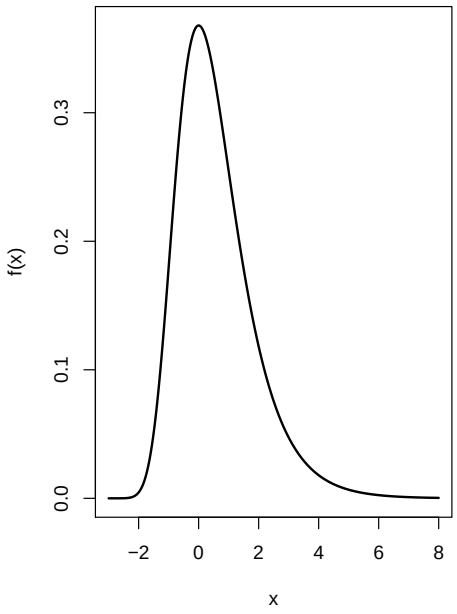


Abbildung 3.1: Wahrscheinlichkeitsdichte der Gumbel-Verteilung für einen Modus von 0 und eine Skalierung von 1.

Wenn $\boldsymbol{Z} \sim \mathcal{G}(x, \mu)$ gilt, ist das Lemma aus [26] gegeben durch (3.6) erfüllt. Dann ist die Wahrscheinlichkeit, dass mit Gewicht $\boldsymbol{\theta_i} + \boldsymbol{z_i}$ das Minimum bestimmt wird:

$$P\left(argmin\left(\theta_1 + z_1, ..., \theta_{|\boldsymbol{\theta}^g|} + z_{|\boldsymbol{\theta}^g|}\right) = i\right) = \frac{\exp\left(-\theta_i\right)}{\sum_{j=1}^{n} \exp\left(-\theta_j\right)} \tag{3.10}$$

Mit Hilfe dieser Wahrscheinlichkeit ergibt sich, wenn der komplette Gewichtsvektor $\boldsymbol{\theta}$ des MRF mit einem Vektor $\boldsymbol{Z}$ verzerrt wird, wo jedes Element $z_i \in \boldsymbol{Z}$ unabhängig identisch verteilt ist, dann entspricht die Wahrscheinlichkeitsdichte des verzerrten Modells der Dichte des unverzerrten MRF (Proposition 3, Beweis in Zusatzmaterialien [31])

$$p_{PAM}\left(\phi\left(x\right), \boldsymbol{\theta}\right) = p_{GIBBS}\left(\phi\left(\boldsymbol{x}\right), \boldsymbol{\theta}\right) \tag{3.11}$$

Allerdings ist nicht komplett $\boldsymbol{\theta}$ zu verzerren, da in diesem Fall Fehler auftreten können. Durch die suffiziente Statistik $\phi\left(\boldsymbol{x}\right)$ ist die Struktur des Graphen gegeben und eine Kante beschreibt die Abhängigkeit zwischen zwei Knoten. Bei vollständiger Verzerrung kann diese Eigenschaft verloren gehen und Abhängigkeiten können sich über mehr als zwei Knoten ergeben. Des Weiteren ist durch die Exponentialfamilie die Verteilung gegeben, welche die maximale Entropie hat, aber wenn $\boldsymbol{\theta}$ komplett verzerrt wird, hat die resultierende Verteilung eine geringere Entropie. Außerdem müssen so für jedes Gewicht Werte aus der Gumbel-Verteilung gezogen werden, was in $\mid \boldsymbol{\theta}^g \mid$ gezogenen Gewichten resultiert, und als Folge wäre dieser Ansatz nicht performant.

Als Lösung der Probleme wurde in [31] gezeigt, dass es genügt, nur eine Teilmenge der Gewichte zu verzerren. Das resultierende Modell ist immer noch eine gute Approximation der echten Gibbs-Verteilung. Vom Gewichtsvektor sind die Elemente $\boldsymbol{\theta}_{PAM}$ auszuwählen, die im Sampling verzerrt werden. Im Artikel von [31] wurde ein perfektes Matching angewandt, was auf Bildern angewandt werden kann, die auch das Einsatzgebiet im Artikel waren. Ein perfektes Matching ist ein Algorithmus, der die Kanten des MRF so auswählt, dass von jedem Knoten eine Kante zu einem anderen Knoten ausgeht [9], aber keine Kante darf sich einen Knoten mit einer anderen Kante teilen. Zum Beispiel ist dies nicht der Fall bei einem dünn besetzten Graphen, hier könnte ein Knoten viele ausgehende Kanten haben und ein perfektes Matching wäre nicht möglich.In Kapitel 5.2 werden zwei implementierte Ansätze zur Auswahl der zu verzerrenden Gewichte vorgestellt. Das Modell, welches sich durch die Verzerrung der Gewichte $\boldsymbol{\theta}_{PAM}$ ergibt, ordnet jeder möglichen Realisierung des Gibbs-Modells eine positive Wahrscheinlichkeit zu, weil der Träger der Gumbel-Verteilung über komplett $\mathbb{R}$ geht. Das bedeutet, dass keine mögliche Realisierung der Zufallsvariablen ausgeschlossen wird und jeder mögliche Datenpunkt auch durch das PAM-Modell realisiert werden kann (Proposition 4 im Zusatzmaterial zu [31]).

Kapitel 4

Generative Adversarial Nets

Die Disziplin, für die ein GAN am häufigsten eingesetzt wird, ist die Erzeugung von realistischen Bildern. Ein Bild kann als multivariate Verteilung P_{data} angegeben werden, wo jeder Pixel eine Zufallsvariable darstellt, die Werte aus einem diskreten Raum $\mathcal{X}$ annehmen kann, den Farbwerten. Soll ein Modell P_g von P_{data} gelernt werden, kann dies mittels der ML-Methode geschehen, nur sind die resultierenden Bilder nicht unbedingt von guter Qualität [35].

Eine Möglichkeit die Verteilung zu lernen, so dass man gute Bilder erhält, sind GAN [16]. Dabei seien D die Trainingsdaten, welche der Verteilung P_{data} folgen. Das Ziel des GAN ist es ein Modell P_g zu lernen, den Generator, der Datenpunkte erzeugt, deren Verteilung nicht von P_{data} zu unterscheiden ist. Im gegebenen Beispiel würden Bilder erzeugt werden, die sich nicht von denen aus D unterscheiden.

In Abbildung 4.1 ist der Aufbau eines GAN dargestellt. Der vorliegende Kritiker ist eine parametrisierte Funktion f, die klassifiziert, ob ein Datenpunkt $\boldsymbol{x}$ der Verteilung P_g folgt, so dass der Generator ihn erzeugt hat, oder P_{data} folgt und somit aus D stammt. Während der Trainingsprozedur des GAN verbessern g und f iterativ ihre Parameter. Der Kritiker sieht in jeder Prozedur zuerst Datenpunkte aus D und generierte Datenpunkte des Generators und aktualisiert damit seine Parameter um die beiden besser zu klassifizieren. Mit den verbesserten Parametern des Kritikers wird der Generator aktualisiert, so dass er Datenpunkte $\boldsymbol{x}$ erzeugt, die es dem Kritiker schwerer machen, diese von Datenpunkten aus

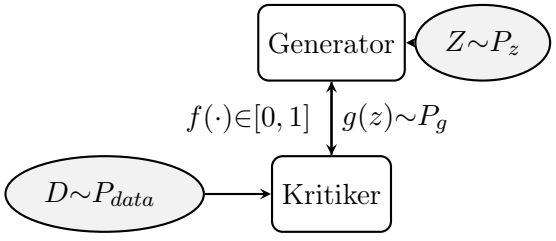


Abbildung 4.1: Skizze eines GAN.

D zu unterscheiden. Das Optimum der beiden Funktionen und damit des GAN liegt vor, wenn

$$P_g = P_{data} \tag{4.1}$$

gilt. Für den Kritiker sehen die Verteilungen P_g und P_{data} somit gleich aus und er kann sie nicht mehr unterscheiden. Sollte der Generator weiterhin seine Parameter ändern, könnte der Kritiker die Punkte wieder besser unterscheiden und das Ergebnis wäre schlechter, dasselbe gilt für eine Änderung der Parameter des Kritikers.

Um dies zu erreichen, werden die Parameter $\boldsymbol{\theta}^c$ vom Kritiker und $\boldsymbol{\theta}^g$ vom Generator gemeinsam gelernt. Wenn die Wahrscheinlichkeitsdichten des Generators und der Trainingsdaten gleich sind ($p_g = p_{data}$), werden die zugehörigen Verteilungen als gleich betrachtet.

Die Auswahl der Funktionen f und g ist durch nichts beschränkt, in der Literatur werden in den meisten Fällen neuronale Netze verwendet. Mit einem neuronalen Netz g können direkt keine zufälligen Datenpunkte aus der Verteilung P_g gezogen werden. Um dies zu erreichen, wird eine Zufallsvariable $Z \sim P_Z$ als Eingabe des Generators gezogen, die Wahl von P_Z muss vom Anwender getroffen werden. Die Ausgabe von $g_{\theta^g} : \mathcal{Z} \to \mathcal{X}$ mit der Eingabe z ergibt den generierten Datenpunkt. Aufgabe des Kritikers $f_{\theta^c} : \mathcal{X} \to [0, 1]$ ist es die Entscheidung zu treffen, ob

$$f(\boldsymbol{x}) = \begin{cases} 1 & , \boldsymbol{x} \sim P_{data} \\ 0 & , \boldsymbol{x} \sim P_g. \end{cases} \tag{4.2}$$

4.1 Training eines GAN

Die Informationen in diesem Abschnitt stammen aus [16]. Gegeben seien n Trainingsdaten D aus dem Raum $\mathcal{X}^m$, die der Verteilung P_{data} folgen. Die Zielfunktion des GAN ist

$$\min_g \max_f \mathbb{E}_D \left[\log f(x)\right] + \mathbb{E}_Z \left[\log\left(1 - f\left(g\left(z\right)\right)\right)\right] \ . \tag{4.3}$$

Im vorgestellten Ansatz wird ein stochastisches Gradientenverfahren (s.u.) angewandt, welches die Zielfunktion optimiert. Um die Parameter $\boldsymbol{\theta}^c$ und $\boldsymbol{\theta}^g$ des Kritikers und des Generators zu optimieren, werden die Ableitungen von (4.3) nach $\boldsymbol{\theta}^c$ und $\boldsymbol{\theta}^g$ bestimmt. Hierbei ist zu beachten, dass im weiteren Verlauf der Erwartungswert $\mathbb{E}$, welcher eine theoretische Größe ist, zur Berechnung durch den Mittelwert approximiert wird. Damit ergeben sich die Gradienten des Kritikers und des Generators zu:

$$\nabla_{\boldsymbol{\theta}^c} \frac{1}{m} \sum_{i=1}^{m} \left[\log f(x_i) + \log\left(1 - f\left(g\left(z_i\right)\right)\right)\right] \tag{4.4}$$

$$\nabla_{\boldsymbol{\theta}^g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - f\left(g\left(z_i\right)\right)\right) \tag{4.5}$$

$\nabla_{\boldsymbol{\theta}^c}$ und $\nabla_{\boldsymbol{\theta}^g}$ sind die Gradienten des Kritikers und des Generators. Zu beachten ist, dass das gesuchte Optimum des Kritikers f die perfekte Trennung zwischen den beiden Verteilungen ist, also das Maximum. Der Generator g sucht den Punkt, wo der Kritiker immer falsch liegt, das Minimum. In Algorithmus 2 ist die Trainingsprozedur zum Lernen der Gewichte $\boldsymbol{\theta}^c$ und $\boldsymbol{\theta}^g$ gegeben.

Data : Trainingsdaten D, Zufallsvariable Z, Anzahl Trainingsiterationen $ITER$, Anzahl der Iterationen des Kritikers $ITER_f$, Batchsize Kritiker m_f, Batchsize Generator m_g, Schrittweite Kritiker α, Schrittweite Generator β

Result : Parameter $\boldsymbol{\theta}^c$, $\boldsymbol{\theta}^g$

Initialisiere $\boldsymbol{\theta}^c$, $\boldsymbol{\theta}^g$;

for $i = 1, ..., ITER$ **do**

  for $j = 1, ..., ITER_f$ **do**

    D_{m_f} = Ziehe m_f Datenpunkte aus D;

    Z_{m_f} = Ziehe m_f Datenpunkte aus Z;

    $\Delta_{\boldsymbol{\theta}^c} = \nabla_{\boldsymbol{\theta}^c} \frac{1}{m_f} \sum_{i=1}^{m_f} \left[\log\left(x_i\right) + \log\left(1 - f\left(g\left(z_i\right)\right)\right)\right]$;

    $\boldsymbol{\theta}^c = \boldsymbol{\theta}^c + \alpha \Delta_{\boldsymbol{\theta}^c}$;

  end

  Z_{m_g} = Ziehe m_g Datenpunkte aus Z;

  $\Delta_{\boldsymbol{\theta}^g} = \nabla_{\boldsymbol{\theta}^g} \frac{1}{m_g} \sum_{i=1}^{m_g} \log\left[1 - f\left(g\left(z_i\right)\right)\right]$;

  $\boldsymbol{\theta}^g = \boldsymbol{\theta}^g - \beta \Delta_{\boldsymbol{\theta}^g}$;

end

Algorithmus 2 : GAN-Training.

In Algorithmus 2 ist mit Z eine Zufallsvariable gemeint, die einer Verteilung folgt, zum Beispiel $Z \sim \mathcal{U}(0, 1)$. Das Training läuft über eine vorher festgelegte Anzahl an Iterationen, da kein Abbruchkriterium im Sinne einer Konvergenz gegeben ist.

4.1.1 Analyse des Trainings

Während des Trainings wird die Distanz zwischen den Verteilungen iterativ verringert, hier berechnet durch die Jensen-Shannon-Divergenz (JS-Divergenz)

$$JSD\left(P_{data} \mid\mid P_g\right) = \frac{1}{2} KL\left(P_{data} \mid\mid M\right) + \frac{1}{2} KL\left(P_g \mid\mid M\right), \tag{4.6}$$

$$\text{wobei } M = \frac{1}{2}\left(P_{data} + P_g\right) \ . \tag{4.7}$$

$KL\left(P_{data} \,||\, P_g\right)$ ist die KL-Divergenz (2.7). Im Gegensatz zur KL-Divergenz ist die JS-Divergenz symmetrisch und immer endlich.

Der Algorithmus 2 optimiert den Kritiker f_g für einen vorgegebenen Generator g mit dem Optimum

$$f_g = \frac{p_{data}\left(x\right)}{p_{data}\left(x\right) + p_g\left(x\right)} \ . \tag{4.8}$$

Beweis

Der Beweis ist angelehnt an [34].

In Gleichung (4.3) wird das Maximum des Kritikers f bestimmt für einen fixen Generator g.

$$\max_f \mathbb{E}_D\left[\log f\left(x\right)\right] + \mathbb{E}_Z\left[\log\left(1 - f\left(g\left(z\right)\right)\right)\right] \tag{4.9}$$

Setzt man die Definition des Erwartungswerts ein, erhält man daraus

$$\int_x p_{data}\left(x\right) \cdot \log\left(f\left(x\right)\right) dx + \int_z p_z\left(z\right) \cdot \log\left[1 - f\left(g\left(z\right)\right)\right] dz \ .$$

Betrachtet man den Generator $g : \mathcal{Z} \to \mathcal{X}$, welcher die Verteilung P_g definiert, kann man den zweiten Summanden umschreiben als

$$\int_x p_g\left(x\right) \cdot \log\left[1 - f\left(x\right)\right] dx.$$

Dabei gibt p_g die Wahrscheinlichkeit an, dass der Generator mit seinem zufälligen Parameter z den Datenpunkt x erzeugt: $p_g\left(x\right) = \int_z p_z\left(z\right) \cdot g\left(z\right) dz = x$. Damit können die beiden Integrale zusammengefasst werden zu

$$\int_x p_{data}\left(x\right) \cdot \log\left(f\left(x\right)\right) + p_g\left(x\right) \cdot \log\left[1 - f\left(x\right)\right] dx. \tag{4.10}$$

Das Integral wird maximal, wenn der Integrand sein Maximum hat. Betrachte hierfür folgendes Ersatzproblem, wobei $a = p_{data}$, $b = p_g$ und $y = f\left(x\right)$ ist. Dabei gilt $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$ und $y \in (0, 1)$:

$$\begin{aligned} a \cdot \log\left(y\right) + b \cdot \left(\log\left(1 - y\right)\right) \\ \log\left(y^a \cdot \left(1 - y\right)^b\right) \end{aligned} \tag{4.11}$$

Hiervon wird das Maximum über die Ableitung bestimmt:

$$
\begin{aligned}
0 =& \frac{\partial}{\partial y} \log\left(y^a \cdot (1-y)^b\right) \\
=& \frac{1}{y^a \cdot (1-y)^b} \cdot \left[a y^{a-1} \cdot (1-y)^b + \left(y^a \cdot b\,(1-y)^{b-1} \cdot -1\right)\right] \\
f\prime(y) =& \frac{a}{y} - \frac{b}{(1-y)} = 0 \\
=& \frac{a\,(1-y) - yb}{y\,(1-y)} \\
=& \frac{a}{y\,(1-y)} - \frac{ay}{y\,(1-y)} - \frac{yb}{y\,(1-y)} \\
=& \frac{a}{y\,(1-y)} = \frac{y\,(a+b)}{y\,(1-y)} \\
=& \frac{a}{a+b}
\end{aligned}
$$

Betrachte vom Ersatzproblem die zweite Ableitung an der Stelle $\frac{a}{a+b}$. Ist diese kleiner als 0, liegt ein Maximum vor.

$$
\begin{aligned}
f\prime\,(y) =& \frac{a}{y} - \frac{b}{(1-y)^2} \\
f\prime\prime\,(y) =& \frac{0 \cdot y - a \cdot 1}{y^2} - \frac{0 \cdot (1-y) - b \cdot -1}{(1-y)^2} \\
f\prime\prime\left(\frac{a}{a+b}\right) =& -\frac{a}{\left(\frac{a}{a+b}\right)} - \frac{b}{\left(\frac{b}{a+b}\right)} < 0
\end{aligned}
$$

□

Mit dem perfekten Kritiker ergeben sich die minimalen Kosten von $-\log(4)$ genau dann, wenn $p_g = p_{data}$ gilt.

Beweis

Der Beweis ist angelehnt an [34].

Die minimalen Kosten beruhen auf dem optimalen Kritiker $f_g = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$, der sich für einen gegebenen Generator ergibt.

Setzt man f_g in (4.10) ein und setzt $p_g = p_{data}$, ergeben sich die Kosten von $-\log(4)$.

Die minimierte Distanz zwischen den Verteilungen, hier die JS-Divergenz, wird aus (4.10) hergeleitet, die mit 0 erweitert wird

$$
\begin{aligned}
&(\log(2) - \log(2))\, p_{data} \text{ und} \\
&(\log(2) - \log(2))\, p_g
\end{aligned}
$$

Das Ergebnis sind die Kosten für den Generator g, der Kritiker ist schon gegeben mit f_g.

$$
\begin{aligned}
C(g) = & \int_x (\log(2) - \log(2)) \cdot p_{data}(x) + p_{data}(x) \cdot \log\left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right) \\
& + (\log(2) - \log(2)) \cdot p_g(x) + p_g(x) \cdot \log\left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right) dx \\
= & \int_x -\log(2)\left(p_{data}(x) + p_g(x)\right) dx \\
& + \int_x p_{data} \cdot \left(\log(2) + \log\left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right)\right) dx \\
& + \int_x p_g \cdot \left(\log(2) + \log\left(\frac{p_g(x)}{p_{data}(x) + p_g(x)}\right)\right) dx
\end{aligned}
$$

Im ersten Integral kann $-\log(2)$ aus dem Integral gezogen werden, wodurch die Summe von zwei Wahrscheinlichkeitsdichten übrig bleibt. Das Integral einer Dichte über den gesamten Raum ergibt 1, daher ergibt sich für das erste Integral 2, das dann wieder $-\log(4)$ ergibt. Beim zweiten und dritten Integral werden die Logarithmen zusammengezogen, dabei gilt

$$
\begin{aligned}
& \log\left(2 \cdot \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right) = \\
& \log\left(\frac{p_{data}(x)}{\frac{p_{data}(x) + p_g(x)}{2}}\right).
\end{aligned}
$$

Mit diesen Ergebnissen wird die obige Formel zu

$$
\begin{aligned}
-\log(4) & + \int_x p_{data}(x) \cdot \log\left(\frac{p_{data}(x)}{\frac{p_{data}(x) + p_g(x)}{2}}\right) dx \\
& + \int_x p_g(x) \cdot \log\left(\frac{p_g(x)}{\frac{p_{data}(x) + p_g(x)}{2}}\right) dx.
\end{aligned}
$$

Die Integrale entsprechen der KL-Divergenz (2.7) mit $q = \frac{p_{data}(x) + p_g(x)}{2}$ und p dem jeweiligen Zähler. Dies entspricht bis auf den Faktor 2 der JS-Divergenz (4.7), wodurch sich ergibt

$$
-\log(4) + 2 \cdot JSD\left(P_{data} \,||\, P_g\right). \tag{4.12}
$$

Die JS-Divergenz ist immer größer gleich 0 und nimmt nur den Wert 0 an, falls die beiden Verteilungen identisch sind. In diesem Fall betragen die Kosten $-\log(4)$. □

4.1.2 Probleme des Trainings

Eine Iteration des Trainings verbessert in einem ersten Schritt den Kritiker, gefolgt von der Verbesserung des Generators. Dabei zeigt sich schon, dass die Aktualisierung des Generators von dem Kritiker in der aktuellen Iteration k abhängt:

$$\nabla_{\theta_k^g} = \frac{1}{m_g} \sum_{i=1}^{m_g} \frac{\frac{\partial}{\partial \theta^g} g\left(z_i\right) \frac{\partial}{\partial g(z_i)} f\left(g\left(z_i\right)\right)}{1 - f\left(g\left(z_i\right)\right)} \tag{4.13}$$

m_g ist die Batchsize des Generators. Wenn einer der Terme im Zähler 0 ist, dann ändern sich die Parameter des Generators in Iteration k nicht mehr. In diesem Fall wird vom *mode collapse* oder *mode dropping* gesprochen [15]. Der Generator kann in diesem Fall seine Parameter nicht mehr ändern und erzeugt nur noch denselben Datenpunkt. Siehe zum Beispiel Abbildung 4.2, der Generator erzeugt nur noch ein Bild der 1 und wird sich nicht mehr verbessern.

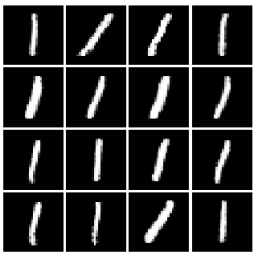


Abbildung 4.2: Mode collapse bei MNIST-Daten. Der Generator erzeugt nur noch dasselbe Beispiel und kann kein anderes Beispiel mehr finden [21].

In [2] wurde gezeigt, das es einen optimalen Kritiker f_{opt} gibt, der immer die korrekte Entscheidung trifft mit $P_{data}\left(f\left(x\right)=1\right)=1$ und $P_g\left(f\left(x\right)=0\right)=1$, und dessen Gradient 0 ist. Ein Spezialfall dafür ist folgend gegeben:

Theorem 1

Seien zwei Verteilungen P_{data} und P_g gegeben, deren Träger auf zwei abgeschlossenen disjunkten Teilmengen $\mathcal{M}$ und $\mathcal{P}$ liegen. Dann gibt es einen glatten optimalen Kritiker $f^*: \mathcal{X} \to [0,1]$, der immer die korrekte Entscheidung trifft und für den $\nabla_x f^*\left(x\right)=0$ für alle $x \in \mathcal{M} \cup \mathcal{P}$ gilt.

Die Beweisidee aus [2] ist wie folgt: Man betrachtet die Träger der Verteilungen P_g und P_{data}, welche in zwei disjunkten Teilmengen $\mathcal{P}$ und $\mathcal{M}$ liegen. Die Distanz $d\left(\mathcal{P},\mathcal{M}\right) =: \delta > 0$ zwischen den beiden Teilmengen ist immer positiv. Dann gibt es durch das Lemma von Urysohn eine Funktion $f^*\left(x\right)$, welche für ein x die zugehörige Menge $\mathcal{M}$ oder $\mathcal{P}$ perfekt vorhersagt. Seien weiterhin die Teilmengen $\hat{\mathcal{M}} = \{x : d\left(x, M\right) \leq \delta/3\}$ und $\hat{\mathcal{P}}$, welches sich analog ergibt, gegeben. Dann gibt es für ein $x \in \mathcal{M}$ einen Ball $B = B\left(x, \delta/3\right)$ auf dem f^* konstant ist und daher $\nabla_x f^* = 0$ ist.

Dieser Speziallfall wurde in [2] auch verallgemeinert. Für den allgemeinen Fall, wenn die Teilmengen nicht disjunkt sind, gilt die Behauptung mit einer hohen Wahrscheinlichkeit. In dem Artikel wurde auch ein Ansatz zur Lösung vorgestellt, der dem PAM ähnelt, dabei werden die Parameter mit $\epsilon \in \mathbb{R}$ verzerrt, wodurch die Träger der Verteilung über dem gesamten Raum liegen. Aber dies ergibt nicht mehr die korrekte Verteilung und ϵ muss in Abhängigkeit des Abstandes der Träger gewählt werden, was für das PAM-Sampling gelöst ist. Mit dem Einsatz des PAM-Samplings und eines MRF sollte der Fall des perfekten Kritikers, der einen Gradienten von 0 hat, nicht eintreten. Zusätzlich ist die Verteilung eine Approximation der Gibbs-Verteilung und das Ergebnis sollte der gesuchten Verteilung ähnlicher sein.

4.2 Gradienten-Methoden

Ein GAN lernt die Parameter der beiden verwendeten Methoden mittels Gradientenmethoden. In [6] wurde eine Möglichkeit des Gradientenabstieg beschrieben. Sei ein Trainingsdatensatz $D = ((\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_n, \boldsymbol{y}_n))$ gegeben. x_i sind Features und y_i ist ein Outcome, das auf den Features beruht. Weiterhin sei eine Familie von parametrisierbaren Funktionen $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ gegeben und $\boldsymbol{\theta}$ indexiert die einzelnen Mitglieder der Familie. Die Funktion $f_{\boldsymbol{\theta}}(x)$ sagt ein $\hat{y}$ vorher für eine neue Messung x mit möglichst wenig Fehlern. Der tatsächliche Fehler wird durch die folgende Verlustfunktion approximiert:

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} l\left(f_{\boldsymbol{\theta}}(x_i), y_i\right) \ . \tag{4.14}$$

(4.15)

Sei die Verlustfunktion gegeben durch $Q((x_i, y_i), \boldsymbol{\theta}) = l\left(f_{\boldsymbol{\theta}}(x), y\right)$.

Um jetzt das Mitglied der Familie $f_{\boldsymbol{\theta}}$ zu bestimmen, für das der Fehler minimal wird, wird der Gradientenabstieg verwendet. Der Gradient von Q ist die partielle Ableitung $\frac{\partial}{\partial \boldsymbol{\theta}} Q((x_i, y_i), \boldsymbol{\theta})$ nach jeder Dimension von $\boldsymbol{\theta}$ und gibt die stärkste Steigung von Q an.

Die Parameter $\boldsymbol{\theta}$ sollen so gewählt werden, dass der Verlust minimal wird. Daher wird der Gradientenabstieg eingesetzt und die Aktualisierung des Parametervektors $\boldsymbol{\theta}$ für Iteration k ergibt sich zu:

$$\boldsymbol{\theta}_{k+1}^{c} = \boldsymbol{\theta}_{k}^{c} - \alpha \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \boldsymbol{\theta}^{c}} Q((x_i, y_i), \boldsymbol{\theta}_k) \, . \tag{4.16}$$

Dabei ist α die Schrittweite des Verfahrens und $\frac{\partial}{\partial \boldsymbol{\theta}^{c}} Q$ der Gradient von Q, der die Steigung an der Stelle x angibt, welche als Suchrichtung verwendet wird. Wenn die Funktion minimiert werden soll, wird der negative Gradient genutzt wie in Formel (4.16). Dadurch werden die Parameter bestimmt, die zum kleinstmöglichen Wert für Q führen. Ist die Maximierung das Ziel, wird der positive Gradient verwendet, was die Form

$\boldsymbol{\theta}_k^c + \alpha \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \boldsymbol{\theta}^c} Q\left(\left(x_i, y_i\right), \boldsymbol{\theta}_k\right)$ ergeben und zum größtmöglichen Wert von Q führen würde. Für die Minimierung wird in jeder Iteration der Verlust von $Q\left(\left(x_i, y_i\right), \boldsymbol{\theta}_k\right)$ geringer und bildet so eine Folge von Werten, die zum minimalen Verlust konvergieren, der das Optimum der Folge darstellt [7]. Wenn das initiale Gewicht $\boldsymbol{\theta}_0$ nahe genug am Optimum ist, und α ausreichend klein gewählt ist, dann werden n Schritte benötigt bis zum Erreichen eines Gewichts, wo Q minimal ist.

Bei jeder Iteration des Gradientenabstiegs wird der komplette Trainingsdatensatz genutzt. Eine Laufzeitverbesserung wird durch den stochastischen Gradientenabstieg (SGD) erzielt, wobei nur eine Teilmenge der Daten D_k mit $n_k = \mid D_k \mid$ in einer Iteration genutzt wird [6]:

$$\theta_{k+1}^c = \boldsymbol{\theta}_k^c - \alpha \frac{1}{n_k} \sum_{i=1}^{n_k} \frac{\partial}{\partial \boldsymbol{\theta}^c} l\left(f_{\boldsymbol{\theta}_k^c}\left(x_i\right), y_i\right) \ . \tag{4.17}$$

Weil nur eine Teilmenge der Daten betrachtet wird, enthält (4.17) noch einen Teil Rauschen, der in (4.16) nicht vorkommt. Die Schrittweite α ist klein genug zu wählen, damit das Optimum gefunden wird.

4.3 Diskussion

Ein GAN lernt eine Verteilung P_g durch die Betrachtung von Datenpunkten, die der gesuchten Verteilung P_{data} folgen. Dabei wird ein Distanzmaß zwischen den Verteilungen minimiert, wie die JS-Divergenz. Als Kritiker und Generator kommen in den meisten Veröffentlichungen neuronale Netze zum Einsatz, wobei je nach Veröffentlichung die Zielfunktionen des Generators oder Kritikers variieren, um mittels einer Heuristik die Ergebnisse zu verbessern. Der Generator zieht die Datenpunkte aus der Verteilung P_g durch die Zufallsvariable $Z \sim P_Z$, welche einer beliebigen Verteilung folgt, als Eingabe und hat einer daher eine zufällige Ausgabe.

Anstatt jetzt die Zielfunktionen des Generators oder des Kritikers zu modifizieren, wird in dieser Arbeit die Allgemeingültigkeit des Ansatzes von [16] genutzt und sowohl der Kritiker als auch der Generator durch ein MRF realisiert. Damit werden für beide Funktionen Gibbs-Verteilungen genutzt und der Generator kann mittels PAM alle möglichen Datenpunkte erzeugen. Zusätzlich bietet ein PGM den Vorteil, das es die Struktur der Daten lernt und die Abhängigkeiten der Zufallsvariablen direkt modelliert werden können.

Nach der Kenntnis des Verfassers ist bisher noch nicht versucht worden sowohl Kritiker als auch Generator durch ein MRF zu realisieren. In [12] wurde ein *Gaussian Mixture Model* als Kritiker eingesetzt, dessen Zielfunktion über die Likelihood hergeleitet wurde, aber der Generator blieb ein neuronales Netz. Es ist keine Veröffentlichung bekannt, wo der Generator etwas anderes als ein neuronales Netz war, was in [2] als Grundlage für die

Theorie herangezogen wurde. Im folgenden Kapitel wird die Implementierung des GAN beschrieben, das die JS-Divergenz minimiert.

Kapitel 5

Implementierung

In dieser Arbeit wurde ein GAN entwickelt, bei dem der Generator als MRF realisiert ist und mittels PAM-Sampling die Datenpunkte erzeugt. Zur Bewertung der erzeugten Datenpunkte wurde der Kritiker ebenfalls als MRF erzeugt. Folgend wird die Implementierung der bisher vorgestellten Theorie erläutert.

Zu Beginn wird das verwendete Framework PX [33] vorgestellt. PX erzeugt die MRF und berechnet die Inferenz darüber. Daran schließt sich die Implementierung des PAM an und wie die Anbindung in PX erfolgte, hier wird insbesondere auf die Auswahl der zu verzerrenden Kanten eingegangen. Den Abschluss bildet die Vorstellung von PAM-GAN, welche Approximationen hier genutzt wurden, und wie sich daraus der Trainings-Algorithmus ergibt.

5.1 PX-Framework

Bei PX handelt es sich um ein Framework, welches in C++ geschrieben ist, um PGM zu entwickeln. Für die vorliegende Arbeit ist die Fähigkeit MRF einzusetzen von besonderer Bedeutung. PX bietet die Funktionalität, um direkt einen CSV-Datensatz einzulesen, den Graphen eines MRF zu lernen, um damit schließlich Inferenzen zu berechnen und Datenpunkte mit Gibbs-Sampling zu erzeugen. Als Strukturen eines Graphen liegen in PX gängige Basisstrukturen vor. Folgend eine Aufzählung der vorliegenden Strukturen, die im Rahmen dieser Arbeit eingesetzt wurden:

- Gitter-Graph
- Kette
- Stern
- ELEM-GM

Um die Randverteilungen, die MAP, des Graphen zu berechnen, wird in PX die BP eingesetzt. Diese liegt in einer parallelen Implementierung vor und erlaubt eine schnelle Berechnung. Für allgemeine Graphen liegt die LBP vor. Die Gewichte eines MRF werden mit ML optimiert.

PX wurde mit C++ implementiert und für die einzelnen Funktionaliäten liegen Basisklassen vor, von denen die verwendeten Berechnungen, wie die BP, und Datenstrukturen, die Graphstrukturen, abgeleitet wurden. Insgesamt ist das Framework einfach zu erweitern. Zusätzlich liegt eine eigene Sprache vor, mit der Experimente erstellt werden können, um ein MRF zu evaluieren.

In dieser Arbeit wurde PX um eine Klasse für das PAM-Sampling erweitert. Hierfür wurde eine Basisklasse SamplingAlgorithm abgeleitet, um eine einfache Implementierung weiterer Sampling-Algorithmen zu ermöglichen. Mit dem implementierten Sampling-Algorithmus wurde eine Klasse für das PAM-GAN eingefügt. Die Implementierung im Rahmen dieser Arbeit wird in den folgenden Kapiteln erläutert.

Experimente mit PX

Ein wichtiger Teil ist die Syntax der Code-Dateien, um ein Experiment mit PX zu starten. Im Folgenden wird die Syntax kurz vorgestellt. Mit den Code-Dateien werden PX Parameter übergeben, welche die Struktur des Graphen oder die Optimierungsmethode beim Lernen der Gewichte des MRF angeben. Dementsprechend gibt es für jeden Parameter in der Code-Datei auch eine zugeordnete Methode oder zugeordnete Variablen in PX. In der Code-Datei können bedingte Sprünge gesetzt werden, wodurch sich Schleifen realisieren lassen und verschiedene Parametereinstellungen einfach getestet werden können. In einer Code-Datei müssen die Argumente einer Funktion vorher angelegt werden. Wenn Argumente fehlen sollten oder Syntaxfehler vorhanden sein sollten, gibt PX eine Fehlermeldung aus. Für die GAN-Experimente ist die Codedatei in Listing 5.1 angegeben.

Listing 5.1: Basisvariante der Codedatei der Experimente.

```
#Used dataset
DFN "./data/mnist_train.csv"
#DFN "./data/religious20PercentInput.csv"
#SEP " "; # Character where the csv is splitted
HED false; # Did the datafile get a header row?
LDX DPT; # Load the datafile into PX

#Used graph types
GRA GRID; #Chosen graph structure
#GRA CHAIN;
#GRA ELEMGM;
```

```
#PEL 0.001; # Percentage of saved edges, parameter for ELEMGM

LDX GPT; # Load the graph into PX
CREATE; # Calculate the structure

#Numbers of edges and vertices are printed to the running console
NECHO GEX; # Prints the number of edges
NECHO GVX; # Prints the number of vertices

#Sampling parameter for the Gibbs sampling
GBR 10; # Gibbs burn-in iterations
GRE 10; # Iterations before parameter is saved
GNU 10; # This parameter is used to determine the number of samples created,
# GNU is also used for the PAM-Sampling

#Used parameters in GAN experiment
GP1 5; #MF - Batchsize critic
GP2 5; #MG - Batchsize generator
GP3 10; #Number of iterations for the GAN training
GP4 1;  #Every GP4 number of round the parameters are saved
GP5 0.01; #ALPHA - Step size critic
GP6 10.0; #h
GP7 0.001; #BETA - Step size generator
GP8 0; # 0 : Sample Integer values, 1: Sample vertex marginals
GP9 1; # 0 : Random edge decision, 1 : Greedy edge decision

#Calls the GAN procedure
GAN;

#Save the sampled data points
DFN "GAN-SAMPLES/correctGradTest.sample"
STORE DPT; # Save the data to the file given by DFN
\label{lst:ganexp}
```

In Tabelle 5.1 sind die verwendeten Variblen von PX kurz dargestellt. Die verwendeten Methoden sind in Tabelle 5.2 dargestellt, wo die erwarteten Argumente und das Resultat der Methode genannt sind.

Tabelle 5.1: Syntaxelemente.

Variablenname	Datentyp	Verwendung
DFN	String	Pfad zur Datei mit erzeugten oder verwendeten Daten
GFN	String	Pfad der Struktur eines MRF
MFN	String	Pfad eienes kompletten Modells
HED	Boolean	Enthält die Trainingsdatendatei einen Header?
SEP	Character	Trennsymbol in einer CSV-Datei
OVW	Boolean	Dateien überschreibbar
GPS	String	String-Variable
GP[0, 9]	Reell	Reellwertige Variablen
DPT	Zeiger	Zeiger auf den verwendeten Datensatz
GPT	Zeiger	Zeiger auf den aktuellen Graphen
PEL	Reell	Konstante zur Berechnung des ELEM-GM-Schätzers
MPT	Zeiger	Zeiger auf das aktuelle Modell
LDX	Zeiger	Biegt den Zeiger auf die vorher angegebenen Daten
GRA	Identifier	Erwartet den Typ des betrachteten Graphen
GNU	Ganzzahlig	Anzahl erzeugter Samples
GBR	Ganzzahlig	Iterationen in Burn-In Phase des Gibbs-Sampling
GRE	Ganzzahlig	Resampling-Runden des Gibbs-Samplings
GVX	Ganzzahlig	Hält die Anzahl der Knoten des Graphen
GEX	Ganzzahlig	Hält die Anzahl der Kanten des Graphen

Tabelle 5.2: Befehle von PX.

Methodenname	Erwartete Parameter	Effekt
CREATE	Graphtypen oder Adjazenzmatrix	Erzeugt einen Graphen und berechnet einen Teil der Parameter des Modells
ESTIMATE	Modell	Lernt die Gewichte des Graphen
SAMPLE	Modell	Zieht eine Stichprobe aus dem MRF
NECHO	Variable	Ausgabe des Variablenwertes mit neuer Zeile
GAN	Datenzeiger, Modellzeiger auf Generator	Berechnet den Generator eines GAN

5.2 PAM-Sampling

Das PAM-Sampling verzerrt die Gewichte mit Werten unabhängig verteilter Zufallsvariablen, die der Gumbel-Verteilung folgen, und berechnet damit das MAP des Graphen. In Kapitel 3 wurde gezeigt, wie dieser Schritt das Problem löst Datenpunkte zu generieren, die der Verteilung des MRF folgen.

Wie in [31] gezeigt wurde, führt die vollständige Verzerrung von $\boldsymbol{\theta} \in \mathbb{R}^{|\theta^g|}$ mit $Z \sim \mathcal{G}(\mu = 0, x)$, $Z \in \mathbb{R}^{|\theta^g|}$ zu einer Verletzung der Eigenschaften des MRF. Daher wird nur eine Teilmenge von $\boldsymbol{\theta}$ verzerrt. Die Experimente in [31] waren auf Bilder zugeschnitten, für die ein MRF als planarer Graph mit einer gerade Anzahl an Kanten vorlag. Es wurde der Ansatz verfolgt die gegenseitige Information (MI) zwischen den Knoten zu berechnen. Die gegenseitige Information für zwei Knoten v_1 und v_2 ist gegeben durch:

$$MI(v_1, v_2) = \sum_{v_1} \sum_{v_2} P(v_1, v_2) \log \frac{P(v_1, v_2)}{P(v_1) P(v_2)} \tag{5.1}$$

Mit der MI wird in der Statistik ein Maß angegeben, wie weit eine Zufallsvariable durch eine Andere berechnet werden kann. Sie wird 0, wenn die Variablen unabhängig sind. Die MI wurde dann als Präferenz der Knoten des MRF für ein stabiles Matching genutzt. In dieser Arbeit fand diese Idee keine Anwendung, da die Bedingungen, welche für ein stabiles Matching benötigt werden, nicht garantiert werden können.

Stattdessen wurden zwei Alternativen zum Matching implementiert und getestet. Für beide Ansätze gilt, dass bei der Instanziierung des PAM-Samplers einmal die zu verzerrenden Kanten bestimmt und in einer Indexliste gespeichert werden. Beim Sampling wird dann nur noch auf die Indexliste zurückgegriffen.

Als erster Ansatz werden die Kanten durch einen Zufallsprozess ausgewählt, dafür wird der Parameter $\lambda \in [0, 1]$ als Schwellenwert gewählt. Für jede Kante wird ein Wert aus der Gleichverteilung $u \sim \mathcal{U}[0, 1]$ gezogen; sollte $u > \lambda$ sein, wird die Kante zum Verzerren ausgewählt. Für die Experimente wurde $\lambda = 0.5$ gesetzt, was im Erwartungswert jede zweite Kante verzerrt. Die Abweichungen von der Gibbs-Verteilung sollten so höher ausfallen.

Der zweite Ansatz ist ein Greedy-Verfahren. Für jeden Knoten wird die gegenseitige Information $MI(v_i, Nb(v_i))$ zu seinen Nachbarn berechnet. Der Knoten v_i wählt dann die Kante zu dem Nachbarn $Nb(v_i)_j$ aus, für den die gegenseitige Information maximal ist; $\max_j MI\left(v_i Nb(v_i)_j\right)$. Knoten, die über keine ausgehende Kante verfügen, werden hierbei bedingt durch die fehlenden Kantengewichte nicht zum Verzerren berücksichtigt. Mit diesem Ansatz kann nicht ausgeschlossen werden, dass die Gewichte eines Knotens mehrfach verzerrt werden, wodurch das Modell ungenauer wird als in [31].

5.3 PAM-GAN

Mit PX wird der Generator g als MRF erzeugt und die Parameter mit ML initialisiert. Um einen Datenpunkt zu generieren, wird das PAM-Sampling mit g genutzt, was die Neuheit an dieser Arbeit darstellt. Das GAN minimiert die JS-Divergenz zwischen der Verteilung des Generators P_g und der unbekannten Verteilung von D, P_{data}, indem es iterativ die Gewichte $\boldsymbol{\theta}^g$ verbessert.

Der Generator g besitzt eine Struktur, welche in Abhängigkeit von dem Datensatz gewählt wurde, siehe Tabelle 5.3. Sei $G = (V, E)$ der Graph des MRF des Generators mit der Knotenmenge V und der Kantenmenge E. Die Knotenmenge V ergibt sich aus D, die Kantenmenge bestimmt sich durch die verwendete Struktur. Die Anzahl der Gewichte $\boldsymbol{\theta}_{e_{ij}} \in \mathbb{R}^{d^g}$ für eine Kante e_{ij} zwischen v_i und v_j ergibt sich aus dem kombinierten Zustandsraum beider Knoten. Für eine Kante ergeben sich damit auch $\mid \phi\left(e_{ij}\right) \mid = d^g$ verschiedene Zustände, welche die Verteilung kodieren.

Tabelle 5.3: Auswahl der Struktur des Graphen.

Datensatz	Graphtyp
Bilder	Gitter, ELEM-GM
Text	Kette

Den Gegenpart zum Generator spielt der Kritiker. Der Kritiker muss die Entscheidung treffen, ob ein Datenpunkt der Verteilung P_{data} folgt oder P_g. In [2] wurde gezeigt, dass ein perfekter Kritiker die Optimierung eines GAN zum Erliegen bringen kann. Damit dem genutzten Kritiker keine optimale Trennung gelingt, wurde in dieser Arbeit ein Kritiker gewählt, der den Naive Bayes Ansatz verfolgt. Beim Naive Bayes wird die Annahme getroffen, dass die Features, zum Beispiel die Messungen $\boldsymbol{x}$, unabhängig sind, wenn die Klasse gegeben ist. Dadurch kann die Wahrscheinlichkeit für eine bestimmte Klasse als Produkt der Features berechnet werden, was diese stark vereinfacht. Die Klasse kann dann mit dem Bayes-Klassifikator berechnet werden und erzielt trotz dieser Annahme weiterhin gute Ergebnisse [19].

Die Abhängigkeitsstruktur eines Graphen kann bei dem Naive Bayes Ansatz als Stern dargestellt werden. Von einem Knoten, der mit der Klasse assoziiert ist, gehen direkt die Kanten zu Knoten, die mit den Features assoziiert sind.

In Abbildung 5.1 ist der Aufbau des Kritikers c dargestellt. Den Zufallsvariablen des MRF c wird die suffiziente Statistik $\phi^g\left(\boldsymbol{X} = \boldsymbol{x}\right)$ des Generators zugewiesen in der Form, dass jeder Knoten des Kritikers v_c, welche die Features darstellen, einer Kante e_g des Generators entspricht. Da der Kritiker bewerten soll, ob der Generator Datenpunkte erzeugt, die P_{data} folgen, bekommt er $\phi^g\left(\boldsymbol{X} = \boldsymbol{x}\right)$ des Generators direkt übergeben. Dadurch ist die Verteilung P_g, welche als Exponentialfamilie vorliegt, kodiert.

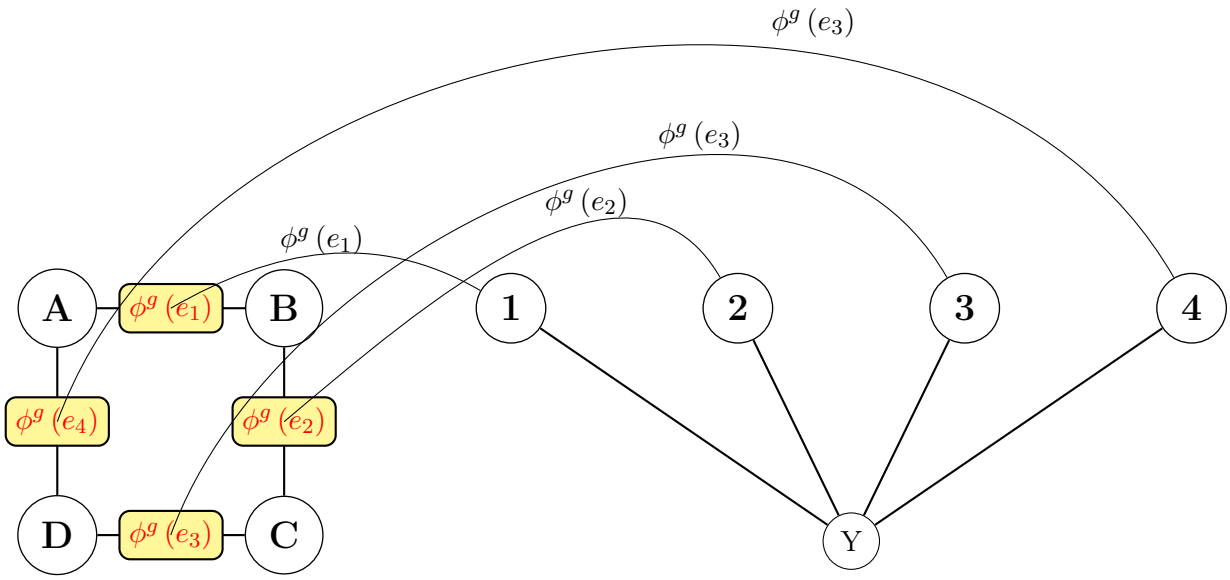


Abbildung 5.1: Der Generator kodiert die Datenpunkte. Die Kantengewichte des Generators werden für die Feature-Knoten des Kritikers genutzt.

Eine Kante $e_{ij} \in E$, für $v_i, v_j \in V$ des MRF g beschreibt die möglichen Realisierungen der adjazenten Knoten. Dabei enthält $\phi^g(e_{ij})$ die Kodierungen der Realisierungen der Kante e_{ij}. Jede mögliche Realisierung ist durch eine Indikatorfunktion gegeben, $\phi^g(e_{ij})_l$ ist Indikatorfunktion für die l mögliche Realisierung der Kante e_{ij}. Wenn zum Beispiel in Abbildung 5.1 der Knoten A in Realisierung 2 vorliegt und der Knoten B in Realisierung 1, dann ist für Kante 1 die Indikatorfunktion $\mathbb{1}_{A=2,B=1} = 1$ und jede andere Indikatorfunktion ist nicht erfüllt und daher 0. Im Knoten 1 des Kritikers wird dann die Realisierung gewählt, die der Indikatorfunktion $\mathbb{1}_{A=2,B=1} = 1$ entspricht. Für den Kritiker ergibt sich damit $\phi^c(\phi^g(D), Y = y)$, wobei $Y = \{0, 1\}$ angibt, ob es sich um P_g oder P_{data} handelt.

Der Kritiker bewertet so nicht direkt die Daten, sondern die Abbildung der Datenpunkte aus D bzw. der generierten Datenpunkte auf das MRF des Generators, $\mathcal{X} \to \phi^g$. Die Realisierung der Knoten des Kritikers können so mit gegebenen $\phi^g(\boldsymbol{x})$ einfach gesetzt werden. Da die suffiziente Statistik ϕ^c in PX für die Kanten bestimmt wird, muss der Zustand von Y berücksichtigt werden. Sei

$$k(i) = \begin{cases} i & , 1 \leq i \leq d^g \\ i - d^g & , d^g + 1 \leq i \leq 2d^g \end{cases} \tag{5.2}$$

die Assoziierung des Index i des Kritikers mit dem zugehörigen Index des Generators. Durch den Knoten Y ist der Zustandsraum eines Knoten von c doppelt so groß wie der der assoziierten Kante von g. Damit können die Zustände des Kritikers gesetzt werden:

$$\phi^c\left(\phi^g\left(\boldsymbol{x}\right), y\right)_i = \begin{cases} \phi^g\left(\boldsymbol{x}\right)_{k(i)} \mathbb{1}_{y=0} & , 1 \leq i \leq d^g \\ \phi^g\left(\boldsymbol{x}\right)_{k(i)} \mathbb{1}_{y=1} & , d^g + 1 \leq i \leq 2d^g \end{cases} \tag{5.3}$$

Mit d^g ist die Dimension der Kante des Generators gemeint. Zu beachten ist, dass die Hälfte der Gewichte einer Kante des Kritikers zum Erkennen von P_{data} genutzt wird und die andere für das Erkennen von P_g, vgl. (5.3).

5.4 Grundlagen des PAM-GAN Trainings

Der Trainings-Algorithmus des GAN von [16] baut darauf auf die Gewichte der optimalen neuronalen Netzwerke zu lernen. In dieser Arbeit wird die Idee für die Gewichte der Exponentialfamilie von MRF angewandt. Während des Trainings wird die Zielfunktion (4.3) optimiert, durch den Gradientenabstieg des Kritikers (4.4) und des Generators (4.5).

Für den MRF-Kritiker ergibt sich hierbei

$$\log f\left(\boldsymbol{X} = \boldsymbol{x}\right) = \log p^c\left(\phi^g\left(\boldsymbol{X} = \boldsymbol{x}\right), y\right) = \left\langle \phi^c\left(\phi^g\left(\boldsymbol{X} = \boldsymbol{x}\right), y\right), \theta^c \right\rangle - A\left(\boldsymbol{\theta}^c\right) \ . \tag{5.4}$$

Der Generator erzeugt die Datenpunkte $\boldsymbol{X} = \boldsymbol{x}'$ mit PAM-Sampling und gibt $\phi^g\left(\boldsymbol{X} = \boldsymbol{x}'\right)$ während des Trainings zurück:

$$g\left(\boldsymbol{z}\right) = \phi^g\left(arg\max_{\boldsymbol{x} \in \mathcal{X}^g} \left\langle \phi^g\left(\boldsymbol{X} = \boldsymbol{x}\right), \boldsymbol{\theta}^g + \boldsymbol{z} \right\rangle\right) = \phi^g\left(\boldsymbol{X} = \boldsymbol{x}'\right) \ . \tag{5.5}$$

Der Kritiker gibt für ein $\phi^g\left(\boldsymbol{x}\right)$ an, ob es sich um einen Datenpunkt aus p_{data} handelt mit $y = 1$ oder um P_g für $y = 0$. Es gilt $p_{\boldsymbol{\theta}^c}\left(\phi^g\left(\boldsymbol{X} = \boldsymbol{x}\right), y = 1\right) + p_{\boldsymbol{\theta}^c}\left(\phi^g\left(\boldsymbol{X} = \boldsymbol{x}\right), y = 0\right) = 1$, daher wird der zweite Summand von (4.3), $\log\left(1 - f\left(g\left(\boldsymbol{z}\right)\right)\right)$, zu:

$$\log p^c\left(\phi^g\left(\boldsymbol{x}'\right), y = 0\right) \tag{5.6}$$

Mit (5.6) berechnet der Kritiker die Wahrscheinlichkeit, dass der generierte Datenpunkt $\boldsymbol{x}'$ aus p_g stammt. Mit dem ersten Term der Zielfunktion

$$\log p^c\left(\phi^g\left(\boldsymbol{x}\right), y = 1\right) \tag{5.7}$$

berechnet der Kritiker die Wahrscheinlichkeit, dass $\boldsymbol{x} \in D$ aus p_{data} stammt. In den folgenden Punkten werden die Gradienten des GAN hergeleitet und die angewandten Verfahren zur Berechnung dieser, anschließend ist der implementierte Algorithmus angegeben.

5.4.1 Gradient Kritiker

Die Gewichte des Kritikers sind in beiden Summanden der Zielfunktion vorhanden, zur Berechnung wird der Erwartungswert durch den Mittelwert approximiert. Setzt man die Terme des Kritikers (5.7) und (5.6) in die Zielfunktion ein, ergibt sich folgender Kritiker:

$$\frac{1}{m_f}\sum_{i=1}^{m_f}\underbrace{\left\langle \phi^c\left(\phi^g\left(\boldsymbol{x}_i\right), y=1\right), \boldsymbol{\theta}^c\right\rangle - A\left(\boldsymbol{\theta}^c\right)}_{\text{I (x,y=1)}} + \underbrace{\left\langle \phi^c\left(\phi^g\left(\boldsymbol{x}_i'\right), y=0\right), \boldsymbol{\theta}^c\right\rangle - A\left(\boldsymbol{\theta}^c\right)}_{\text{II(x',y=0)}} \tag{5.8}$$

Die beiden Summanden $I\left(x, y=1\right)$ und $II\left(x, y=0\right)$ sind bis auf das y und den verwendeten Datenpunkt identisch, daher wird die Ableitung für I betrachtet und für II ergibt sie sich analog. Der Gradient ergibt sich zu:

$$\frac{\partial}{\partial \boldsymbol{\theta}^c} I\left(\boldsymbol{x}, y=1\right) = \underbrace{\frac{\partial}{\partial \boldsymbol{\theta}^c}\phi^c\left(\phi^g\left(\boldsymbol{x}\right), y=1\right)^T \boldsymbol{\theta}^c}_{\text{a}} - \underbrace{\frac{\partial}{\partial \boldsymbol{\theta}^c} A\left(\boldsymbol{\theta}^c\right)}_{\text{b}} \tag{5.9}$$

Für a ergibt sich durch die Produktregel

$$\frac{\partial}{\partial \boldsymbol{\theta}^c}\phi^c\left(\phi^g\left(\boldsymbol{x}\right), y=1\right)^T \boldsymbol{\theta}^c + \phi^c\left(\phi^g\left(\boldsymbol{x}\right), y=1\right)\frac{\partial}{\partial \boldsymbol{\theta}^c}\boldsymbol{\theta}^c = \phi^c\left(\phi^g\left(\boldsymbol{x}\right), y=1\right)\ . \tag{5.10}$$

Da der Kritiker durch die Zustände des Generators gegeben ist, welche sich aus den Datenpunkten ergeben, die mit PAM gezogen werden, hat $\boldsymbol{\theta}^c$ keine Auswirkung und der erste Summand wird 0. Beim zweiten Summanden bleibt ϕ^c und ergibt das Differential von a. Für b wurde in [37] gezeigt, das sich das Differential der Log Partition Function als Erwartungswert der suffizienten Statistik über das Modell ergibt: $\mathbb{E}_{\theta^c}\left[\phi^c\left(\phi^g\left(\boldsymbol{x}\right), y\right)\right]$. Bei jeder Aktualisierung der Gewichte, was im Training in jeder Iteration geschieht, müsste der Erwartungswert erneut ausgerechnet werden.

Anstatt in jeder Iteration den Erwartungswert erneut zu berechnen, wird er einmal zu Beginn mit den Trainingsdaten D über die Mittelwerte von $\phi^c\left(\phi^g\left(D\right), Y=y\right)$ berechnet, was der Approximation des Erwartungswerts durch den Mittelwert für den optimalen Generator entspricht. Der optimale Generator erzeugt Datenpunkte $\boldsymbol{x'}$, für die der Kritiker nicht entscheiden kann, ob diese P_g oder P_{data} folgen, was bedeutet, dass die Kodierung im Kritiker sich für beide Verteilungen nicht unterscheidet.

So werden die Gewichte benachteiligt, die häufig beim optimalen Kritiker gesetzt sind, was den Fehler des Kritikers erhöht und die optimale Trennung des Kritiker zusätzlich erschwert. Gleichzeitig wird die Rechenzeit reduziert, da der Mittelwert nur einmal zu Beginn berechnet wird und in einer Lookup-Tabelle gespeichert wird.

Für das j-te Element von ϕ^c ergibt sich der Mittelwert $\bar{\phi^c}$ über D wie folgt:

$$\bar{\phi^c}_j = \frac{1}{n}\sum_{i=1}^{n}\phi^c\left(\phi^g\left(\boldsymbol{x}_i\right), y\right)_{j=1} \tag{5.11}$$

Damit ergibt sich als Gradient für $I(x, y=1)$ in der Dimension w

$$\frac{\partial}{\partial \theta_w^c} I(x, y=1) = \phi^c\left(\phi^g(\boldsymbol{x}), y=1\right)_w - \bar{\phi^c}_w \ . \tag{5.12}$$

Ist der Kritiker benachteiligt beim Erkennen von realen Datensätzen und schafft er es nicht die generierten Daten perfekt von den realen Daten zu trennen, so sollte kein mode collapse auftreten.

Für die Ableitung des Kritikers ergibt sich

$$\frac{1}{m_g} \sum_{i=1}^{m_g} \left\{ \left[\phi^c\left(\phi^g(\boldsymbol{x}_i), y=1\right) - \bar{\phi^c}\right] + \left[\phi^c\left(\phi^g\left(\boldsymbol{x}_i'\right), y=0\right) - \bar{\phi^c}\right] \right\} \ . \tag{5.13}$$

Der Gradient bestimmt für reale Datenpunkte $\boldsymbol{x}$ und für generierte Datenpunkte $\boldsymbol{x}'$ die zugehörigen ϕ^c und subtrahiert davon den assoziierten doppelten Mittelwert des perfekten Generators $\bar{\phi^c}$.

5.4.2 Gradient Generator

Nur im letzten Summanden der Zielfunktion werden die generierten Datenpunkte $g(\boldsymbol{z})$ bewertet und nur in diesem Term tauchen die Gewichte des Generators $\boldsymbol{\theta}^g$ auf. Ausgehend von (5.6) ergibt sich der Term wie folgt:

$$\log p_{\boldsymbol{\theta}^c}\left(\phi^g\left(\boldsymbol{x}'\right), y=0\right) \tag{5.14}$$

$$= \langle \phi^c\left(\phi^g\left(arg\max_{\boldsymbol{x}\in\mathcal{X}^g} \langle \phi^g(\boldsymbol{x}), \boldsymbol{\theta}^g + \boldsymbol{z}\rangle\right), y=0\right), \boldsymbol{\theta}^c\rangle - A(\boldsymbol{\theta}^c) \tag{5.15}$$

Zur besseren Lesbarkeit wird die Funktion des Generators im weiterem Verlauf wieder als $\phi^g(\boldsymbol{x}')$ notiert. Von (5.15) ist die Ableitung nach $\boldsymbol{\theta}^g$ gesucht:

$$\frac{\partial}{\partial \boldsymbol{\theta}^g} \phi^c\left(\phi^g\left(\boldsymbol{x}'\right), y=0\right)^T \boldsymbol{\theta}^c + \underbrace{\phi^c\left(\phi^g\left(\boldsymbol{x}'\right), y=0\right)^T \frac{\partial}{\partial \boldsymbol{\theta}^g} \boldsymbol{\theta}^c}_{=0} + \underbrace{\frac{\partial}{\partial \boldsymbol{\theta}^g} A(\boldsymbol{\theta}^c)}_{=0} \tag{5.16}$$

Nur der erste Summand bleibt erhalten, durch die Kettenregel ergibt sich:

$$\boldsymbol{\theta}^c \underbrace{\frac{\partial}{\partial \phi^g(\boldsymbol{x}')} \phi^c\left(\phi^g\left(\boldsymbol{x}'\right), y=0\right)^T}_{\mathrm{I} \in d^c \times d^g} \underbrace{\frac{\partial}{\partial \boldsymbol{\theta}^g} \phi^g\left(\boldsymbol{x}'\right)}_{\mathrm{II} \in d^g \times d^g} \tag{5.17}$$

Der Term I nach $\phi^g(\boldsymbol{x})$ abgeleitet ergibt:

$$\frac{\partial}{\partial \phi^g(\boldsymbol{x}')_j} \phi^c\left(\phi^g\left(\boldsymbol{x}'\right), y=0\right)_i = \begin{cases} 0 & , j \neq k(i) \\ \mathbb{1}_{y=0} & , j = k(i) \wedge 1 \leq i \leq d^g \end{cases} \tag{5.18}$$

Die Ableitung in (5.18) gibt eine binäre Matrix zurück, wobei eine 1 einem Gewicht des Kritikers entspricht, das genutzt wird, um die Wahrscheinlichkeit für generierte Datenpunkte zu berechnen. Multipliziert man die Ableitung mit $\boldsymbol{\theta}^c$, erhält man nur die Gewichte, die zum Erkennen von P_g benötigt werden.

Der Term II ist die suffiziente Statistik ϕ^g des generierten Datenpunkts. Da diese nur aus Indikatorfunktionen besteht, welche nicht differenzierbar sind, wird die Ableitung approximiert.

Der Generator ist gegeben durch:

$$\phi^g\left(\boldsymbol{x}'\right) = \phi^g\left(arg\max_{x\in\mathcal{X}^g}\langle\phi^g\left(\boldsymbol{x}\right), \boldsymbol{\theta}^g + \boldsymbol{z}\rangle\right) \ , \tag{5.19}$$

wovon der Gradient gesucht ist:

$$\frac{\partial}{\partial\theta_i^g}\phi^g\left(\boldsymbol{x}'\right) \ . \tag{5.20}$$

Die erste Ableitung ist definiert durch

$$f'\left(x\right) = \lim_{h\to 0}\frac{f\left(x\right) - f\left(x+h\right)}{h} \ . \tag{5.21}$$

Damit beschreibt die Ableitung eine Sekante, die an $f\left(x\right)$ angelegt wird und den Graphen der Funktion an Punkt $f\left(x+h\right)$ wieder schneidet. Betrachtet man die Taylor-Reihe, welche eine Funktion durch eine Potenzreihe darstellt, von der Funktion der Ableitung

$$\frac{f\left(x\right) - f\left(x+h\right)}{h} = f'\left(x\right) + \frac{hf''\left(x\right)}{2!} + \frac{h^2f'''\left(x\right)}{3!} + ... \ , \tag{5.22}$$

erhält man eine Approximation der Ableitung zu

$$f'\left(x\right) \approx \frac{f\left(x\right) - f\left(x+h\right)}{h} \ . \tag{5.23}$$

Dieser Ansatz heißt Finite-Differenzen-Methode (FDM) [28] und wird als Grundlage für die Ableitung verwendet. Der Quotient ist in der Literatur auch als Differenzenquotient bekannt.

Überträgt man diesen Ansatz auf den Generator, ist die Dimension i des Gradienten gegeben als

$$\frac{\partial}{\partial\theta^g{}_i}\phi^{g'}\left(g\left(\boldsymbol{z}\right)\right) = \frac{\phi^g\left(g\left(\boldsymbol{z}\right)\right) + \phi^g\left(g\left(\boldsymbol{z} + \boldsymbol{eL}h\right)\right)}{h} \tag{5.24}$$

mit $\boldsymbol{eL} = \left(0, ..., \underbrace{1}_{\text{i-tes Element}}, ...0\right)^T$. Der Gradient der Dimension i ist gegeben durch einen Vektor, der nur aus den Gewichten $\{-1/h, 0, 1/h\}$ besteht. Um den vollständigen

Gradienten zu berechnen, müsste die Differenz in Gleichung 5.24 für den kompletten Gewichtsvektor $\boldsymbol{\theta}^g$ berechnet werden, was eine Matrix der Dimension $\mid \boldsymbol{\theta}^g \mid \times \mid \boldsymbol{\theta}^g \mid$ ergibt.

Das Problem an diesem Ansatz ist, dass die Matrix quadratisch in der Dimension von ϕ^g ist, was für einfache Probleme zu einer großen Matrix führt. Diese Matrix müsste in einer Iteration für jedes Element des Batches des Generators berechnet werden. Die Berechnung des Gradienten wäre so nicht praktikabel, weshalb für den Generator die Koordinatenabstiegs-Methode [30] verwendet wird, anstatt eines stochastischen Gradientenabstiegs. Die hier vorgestellte Methode wurde dort als Random Coordinate Descent Method (RCDM) eingeführt.

Die Idee ist es anstatt des vollen Gradienten $\frac{\partial f}{\partial \boldsymbol{x}}$ zu berechnen, wird nur der partielle Gradient $\frac{\partial f}{\partial x_i}$ für eine Dimension berechnet. Dadurch wird die Berechnung des Gradienten stark vereinfacht, ebenso die Aktualisierung der Zielvariablen. Beim Generator ist der Gradient gegeben durch

$$\boldsymbol{\theta}^c \frac{\partial}{\partial \boldsymbol{\theta}^g} \phi^c \left(\phi^g \left(\boldsymbol{x}' \right), y = 0 \right) \ . \tag{5.25}$$

Durch die Kettenregel ergibt sich für den partiellen Gradienten

$$\boldsymbol{\theta}^c \frac{\partial}{\partial \phi^g \left(\boldsymbol{x}' \right)} \phi^c \left(\phi^g \left(\boldsymbol{x}' \right), y = 0 \right) \frac{\partial}{\partial \theta_i^g} \phi^g \left(\boldsymbol{x}' \right) \ . \tag{5.26}$$

Der Gradientenabstieg führt eine Anzahl von Iterationen aus, in denen jeweils eine Dimension i aktualisiert wird. Im einfachsten Fall wird der Index in jeder Iteration gleichverteilt gezogen. Für Iteration k ergibt sich der Index für den Generator zu $u_k \sim \mathcal{U}\left(1, \mid \boldsymbol{\theta}^g \mid\right)$. Der optimale Schritt in einer Iteration k ergibt sich zu

$$\boldsymbol{\theta}_{k+1}^g = \boldsymbol{\theta}_k^g - \beta \boldsymbol{\theta}^c \frac{\partial}{\partial \phi^g \left(\boldsymbol{x}' \right)} \phi^c \left(\phi^g \left(\boldsymbol{x}' \right), y = 0 \right) \frac{\partial}{\partial \theta_i^g} \phi^g \left(\boldsymbol{x}' \right) \ . \tag{5.27}$$

β ist die Schrittweite des Gradientenabstiegs und ist in [30] mit der Inversen der Lipschitzkonstanten der Koordinate i, $\frac{1}{L_i}$, angegeben.

L_i gibt die maximale Änderung der Funktion f für den partiellen Gradienten in Dimension i an. Für den Fall, dass die abgeleitete Dimension gleichverteilt gezogen wird, ergibt sich als Lipschitzkonstante für den gesamten Gradienten $L = \mid \boldsymbol{\theta}^g \mid$. In dieser Arbeit wurde die Schrittweite mit Hilfe von L gebildet, wodurch diese kleiner als die optimale Schrittweite gewählt wurde:

$$\beta = \frac{1}{\mid \boldsymbol{\theta}^g \mid} \tag{5.28}$$

Die Konvergenz des Verfahrens ist identisch zu der schlechtesten Konvergenz eines vollständigen Gradientenabstiegs, aber da die einzelnen Iterationen einfacher zu berechnen sind, ist RCDM eine Alternative, die in dieser Arbeit angewandt wird.

5.4.3 PAM-GAN Training

Der Trainings-Algorithmus 3 ist an den von [16] angelehnt und die zugehörigen Ableitungen werden wie ausgeführt gebildet. Da $\phi^g(D)$ nur eingesetzt wird, wird dies vor dem Aufruf berechnet und als Trainingsdaten eingesetzt. Dasselbe gilt für $\bar{\phi}^c$.

Data : Trainingsdaten $\phi^g(D)$, $\bar{\phi}^c(\phi^g(D))$, Anzahl Iterationen $ITER$, Batchsize Kritiker m^c, Batchsize Generator m^g, Schrittweite Kritiker α, Schrittweite Generator β, FDM-Parameter h

Result : Parameter $\boldsymbol{\theta}^g$

Initialisiere $\boldsymbol{\theta}^c$, $\boldsymbol{\theta}^g$;

for $k = 1...ITER$ **do**

$\phi^g(D^{m_c})$ = Ziehe m^c Datenpunkte aus $\phi^g(D)$;

$\phi^g\left(D^{m_c}_{gen}\right)$ = Erzeuge m^c Datenpunkte mit dem *Generator*;

Berechne $\phi^c\left(\phi^g\left(D^{m^c}\right), y = 1\right)$;

Berechne $\phi^c\left(\phi^g\left(D^{m^c}_{gen}\right), y = 0\right)$;

Bestimme $\bar{\phi}^c\left(\phi^g\left(D^{m^c} \cup D^{m^c}_{gen}\right)\right)$;

$$\nabla_{\boldsymbol{\theta}^c_k} = \frac{1}{m^c}\sum_{i=1}^{m^c}[\left(\phi^c\left(\phi^g\left(\boldsymbol{x}_i \in D^{m^c}\right), y = 1\right) - \bar{\phi}^c\left(\phi^g\left(\boldsymbol{x}_i \in D^{m^c}\right)\right)\right) + \left(\phi^c\left(\phi^g\left(\boldsymbol{x}_i \in D^{m^c}_{gen}\right), y = 0\right) - \bar{\phi}^c\left(\phi^g\left(\boldsymbol{x}_i \in D^{m^c}_{gen}\right)\right)\right)]$$

$\boldsymbol{\theta}^c_{k+1} = \boldsymbol{\theta}^c_k + \nabla_{\boldsymbol{\theta}^c_k}$;

$\boldsymbol{\theta}^c_g = \boldsymbol{\theta}^c_{y=0}$;

$\boldsymbol{e_L}_{i=1}$, $i \sim \mathcal{U}(1, \mid \boldsymbol{\theta}^g \mid)$;

$z'^{m^g} \sim \mathcal{G}(\mu = 0, x)$: Ziehe m^g Verzerrungsvektoren;

$\phi^g\left(D^{m^g}_{gen}\right) = Generator\left(z'^{m^g}\right)$: Erzeuge mit m^g Verzerrungsvektoren Datenpunkte;

$z''^{m^g} = z'^{m^g} + \boldsymbol{e_L}h$;

$\phi^g\left(D^{m^g}_{genDiff}\right) = Generator\left(z''^{m^g}\right)$: Erzeuge mit m^g Verzerrungsvektoren Datenpunkte als Differenz vom unverzerrten generierten Datenpunkt;

$\frac{\partial}{\partial\theta^g_{e_L}}\phi^g\left(D^{m^g}_{genDiff} \cup D^{m^g}_{gen}\right) = \frac{\phi^g\left(\boldsymbol{x}_i \in D^{m^g}_{gen}\right) - \phi^g\left(\boldsymbol{x}_i \in D^{m^g}_{genDiff}\right)}{h}$;

$\nabla_{\theta^g_k e_l} = \frac{1}{m^g}\sum_{i=1}^{m^g}\frac{\partial}{\partial\theta^g_{e_L}}[i] \cdot \boldsymbol{\theta}^c_g[i]$;

$\boldsymbol{\theta}^g_{k+1} = \boldsymbol{\theta}^g_k + \beta\nabla_{\theta^g_k e_l}$;

end

Algorithmus 3 : PAM-GAN Training.

Mit $\boldsymbol{e_L}_{i=1}$ wird die Dimension des Generators bestimmt, welche durch die FDM abgeleitet wird.

Das Training des PAM-GAN ist aufgebaut wie das des klassischen GAN. Über eine feste Anzahl an Iterationen wird die Zielfunktion des GAN durch die gegenseitige Verbesserung des Kritikers und des Generators optimiert. Mit dem PAM-Sampling werden die Datenpunkte erzeugt und anschließend mit dem Kritiker verbessert. Insgesamt ist der Trainingsalgorithmus einfach aufgebaut, es muss keine Inferenz mit dem Kritiker erfolgen, da für das Training nur $\boldsymbol{\theta}^c$ von Belang ist. Ein Unterschied zum originären Training ist, dass der Kritiker immer nur einmal aktualisiert wird und nicht noch zusätzlich in einer Schleife verbessert wird. Bevor das PAM-GAN Training startet, wird ein Generator erstellt und die Struktur des Graphen gelernt, für den Kritiker wird ein Stern als Naive Bayes erstellt. Vor der Berechnung des Trainings wird $\phi^g(D)$ berechnet und zwischengespeichert, so müssen diese nur noch in einer Tabelle nachgeschlagen werden. Dasselbe gilt für $\bar{\phi}^c$. Die Initialisierung der Gewichte wird in dem nachfolgenden Kapitel, bei den Experimenten, behandelt. Hierbei ist zu sagen, dass die Gewichte des Kritikers immer der Standardnormalverteilung $\mathcal{N}(0,1)$ folgen. Nachfolgend werden die geschilderten Verfahren evaluiert.

Kapitel 6

Experimente

6.1 Forschungsfragen

Um das PAM-GAN zu evaluieren, wird auf einen Standard in der GAN-Literatur zurückgegriffen [16] [3] [4], die Erzeugung von Ziffern mit den MNIST-Daten [27] als Trainingsdatensatz. Vorher sind zwei Punkte zu testen, die Qualität des Generators, gegeben durch das PAM-Sampling, und die Bestimmung eines guten Parameters h der FDM. Ein weiterer Punkt ist der perfekte Kritiker, so dass der Gradient des Generators, gegeben durch die Ableitung des Kritikers, immer 0 ist und dieser sich nicht mehr verbessern kann. Bedingt durch die Approximationen sollte dieser Fall nicht eintreten und eine stetige Verbesserung des Generators möglich sein, was ebenfalls evaluiert wird. Ein weiterer Test für das PAM-GAN ist die Erzeugung von Texten, wofür zwei Datensätze ausgewählt werden [32] [24] und die Verteilung gelernt wird. Zusammenfassend ergeben sich die folgenden Forschungsfragen und die damit einhergehenden Experimente:

1. Erzeugt der PAM-Sampler gute Datenpunkte aus einem MRF und ist so in der Lage als Generator zu dienen?

2. Was ist ein guter Parameter h, mit dem gute Ergebnisse beim Lernen erzielt werden können?

3. Ist das PAM-GAN in der Lage die Verteilung der MNIST-Daten zu erlernen?

4. Tritt der Fall des mode collapse auf und verbessert sich der Generator deshalb nicht weiter?

5. Kann das PAM-GAN komplexere Verteilungen als die des MNIST-Datensatzes lernen und Texte erzeugen?

6.2 Datensätze

Bevor es zur Auswertung der Forschungsfragen kommt, werden zuerst die verwendeten Datensätze vorgestellt. Die drei Datensätze, die in dieser Arbeit betrachtet werden, sind der MNIST-Datensatz [27], ein Datensatz aus dem relNet-Projekt [32], der aus Posts eines religiösen Forums besteht, sowie ein Datensatz von Überschriften von Artikeln von Reuters aus dem Jahr 2017 [24].

6.2.1 MNIST

Es ist dem Verfasser kein Artikel bekannt, wo eine neue GAN-Methode vorgestellt wurde und diese nicht auf dem MNIST-Datensatz getestet wurde. Der Datensatz besteht aus handschriftlichen Ziffern von 0-9, gespeichert als Bild mit 28×28 Pixeln, in dessen Zentrum die Ziffer dargestellt ist, wobei jeder Pixel Grauwerte aus dem Intervall $[0, 255]$ annehmen kann. Der Datensatz besteht aus einem Trainingsdatensatz mit $D = 60.000$ Bildern und einen Testdatensatz mit 10.000 Bildern.

Zur Generierung wurde der Trainingsdatensatz verwendet, eine Beschränkung der Auswahl der Bilder wurde nicht vorgenommen. Ein Bild besteht aus Pixeln und hat durch die Nachbarschaften der Pixel eine eigene Struktur gegeben. Eine Zufallsvariable des PGM entspricht einem Pixel im Bild und kann in der Aufgabe die Werte 0 oder 1 annehmen. Dies ist in der einfachsten Form durch ein Gitter über das komplette Bild darzustellen. In der Vorverarbeitung wurden nur binäre Werte betrachtet, wobei die Binarisierung pro Pixel mit der Funktion Bin erfolgte, welche den Farbwert des Pixels x angibt nach dem folgenden Schema:

$$Bin\,(x) = \begin{cases} 1 & \text{, wenn } Bin\,(x) > 127 \\ 0 & \text{, sonst} \end{cases}. \tag{6.1}$$

Die Zufallsvariablen sind bei MNIST die 784 Pixel und für jede Zufallsvariable gilt, dass sie nur einen aus zwei Werten annehmen kann, $\mathcal{X}_i = \{0, 1\}$ mit $i = 1, ..., 784$. Ein Datenpunkt $\boldsymbol{x} \in D$ besteht aus einem Vektor von 784 binären Zufallsvariablen.

6.2.2 relNet-Projekt

Der Datensatz aus dem relNet-Projekt besteht aus Posts aus einem religiösen Forum, wo sich Menschen muslimischen Glaubens treffen und sich auf Englisch miteinander austauschen. Es liegen $D = 37.893$ Posts vor, wovon jeder 12 Tokens lang ist. Es liegen 28.305 unterschiedliche Tokens im Datensatz vor. Ein Token ist ein String im Post und kann zum Beispiel ein Wort, eine URL, ein Satzzeichen, ein Bild oder Emoticon sein.

Zur Generierung von Sätzen wurde der Datensatz vorverarbeitet. Zuerst wurden alle Tokens gefiltert, die nicht ausschließlich aus Buchstaben bestehen, was die Anzahl an

Tokens zu 24.450 reduzierte. Von den Tokens wurden die Häufigkeiten gezählt und zur Synthese der Sätze nur die häufigsten 20% der Tokens verwendet, was insgesamt 4.890 Tokens ergab.

Am häufigsten trat das Token *the* mit 11.021 Vorkommen auf, das seltenste war *couples*, welches 5 mal auftrat. Es wurden keine Stopwords oder häufige Wörter entfernt, in der Hoffnung, dass so leichter gute Sätze erzeugt werden können. Dabei bedeutet gut hier nicht korrekt, sondern, ob die Sätze in einem englischsprachigen Forum auftreten könnten. Damit die Sätze die Länge von 12 Tokens behalten, wurden die ersetzten Wörter durch das Token *FILL-IN* ersetzt.

6.2.3 Reuters 2017 Dataset

Der Datensatz besteht aus den Überschriften von Reuters in englischer Sprache mit einem zugehörigen Zeitstempel aus dem Jahr 2017. Für die Generierung wurde der zugehörige Zeitstempel nicht beachtet. Insgesamt gibt es 799.921 Artikelüberschriften mit 214.642 verschiedenen Tokens. Die Anzahl an Tokens pro Überschrift schwankte von 1-33, mit der größten Gruppe bei 10 Tokens und 136.047 Überschriften. Die Teilmenge an Überschriften mit 6 Tokens hatte ein Größe von 36.816 und wurde aufgrund der Kürze für die Experimente ausgewählt. Duplikate in den Überschriften wurden aus dem Datensatz entfernt, was 20.378 Überschriften ergab.

Der Datensatz verfügte über $D = 24.728$ unterschiedliche Tokens, der häufigste mit 2.235 Aufkommen war *profit* und die seltensten traten einmal auf, zum Beispiel *brief-travelers*. Zur Vereinfachung wurden ebenfalls die häufigsten 20% der Tokens genutzt, was am Ende 4.946 Tokens ergab. Ersetzte Tokens wurden weiterhin durch *FILL-IN* ersetzt.

6.3 Ergebnisse

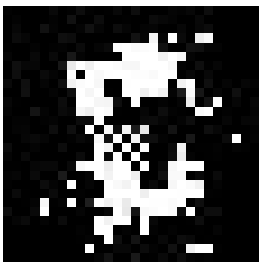
6.3.1 Evaluation des PAM-Sampling

MNIST-Daten

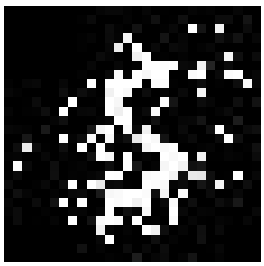
Der MNIST-Datensatz ist einfach strukturiert und wird für die Evaluation von GAN häufig verwendet. Wegen der Binarisierung der Daten liegen diese als einfaches Schwarz-Weiß-Bild vor und Änderungen am Ergebnis können am Bild erkannt werden. Da der Generator die Bilder der Ziffern erzeugen können muss, wurde dieser in einem ersten Schritt getestet.

Ein GAN arbeitet klassisch auf dem vollständigen MNIST-Datensatz, daher wurde zuerst mit dem vollständigen MNIST-Datensatz als Trainingsdatensatz ein Gitter-Graph und ein ELEM-GM-Graph konstruiert, anschließend wurden die Gewichte $\boldsymbol{\theta}$ mittels ML gelernt. Die Datenpunkte wurden mittels PAM-Sampling erzeugt, wobei die zu verzerrenden Gewichte $\boldsymbol{\theta}$ durch den Greedy-Algorithmus bestimmt wurden.

Abbildung 6.1: Vergleich des PAM-Samplings mit Gitter- und ELEM-GM-Graphen auf dem vollständigen MNIST-Datensatz.



(a) Gitter-Graph.



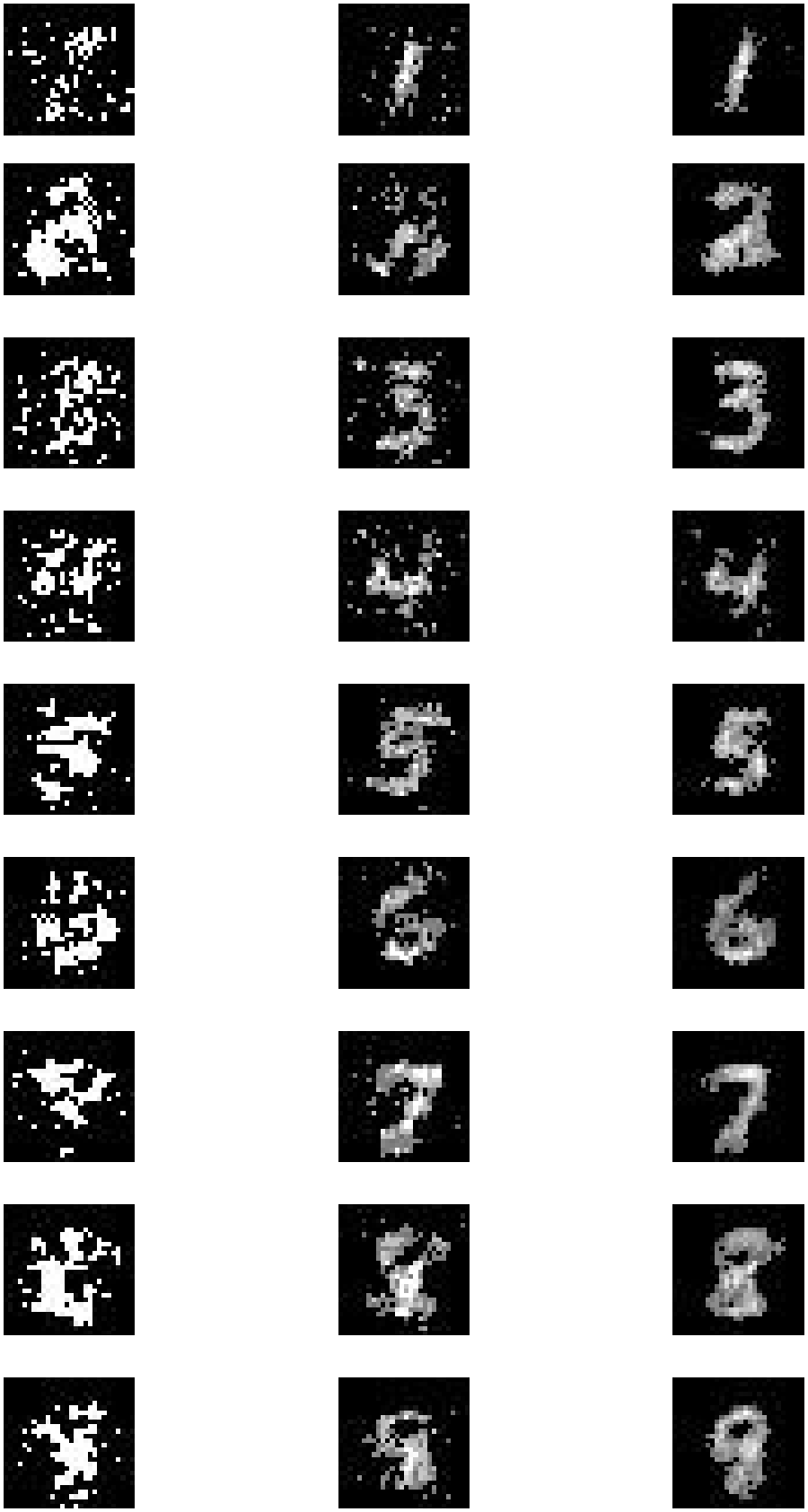
(b) ELEM-GM-Graph.

Die Bilder in Abbildung 6.1 zeigen die Resultate und erinnern nur grob überhaupt an Ziffern. Zur Vereinfachung wurden dann die Teilmengen der einzelnen Ziffern der MNIST-Trainingsdaten genommen und darüber ein MRF gelernt. In dem Artikel [31] wurden Segmentierungen auf Bildern berechnet, wofür der Mittelwert über 20 PAM-Sampling-Ergebnisse genutzt wurde. Der Ansatz wurde hier angewandt zur Generierung von weiteren Bildern, einmal wurde der Mittelwert über 5, einmal über 20 Bilder berechnet. In Abbildung 6.2 sind die Ergebnisse dargestellt. Bei den Bildern, die über den Mittelwert mehrerer Bilder gebildet wurden, sind nur die Pixel angezeigt, die mit einer relativen Häufigkeit von mindestens 40% auftraten. Die Gewichte des Gitter-Graphen wurden weiterhin mit ML optimiert.

Ein einfaches Sample zeigt höchstens einen Teil einer Ziffer und wäre ein unbefriedigendes Ergebnis, ist aber schnell berechnet. Das Ergebnis für die Mittelwertbildung über 5 Datenpunkte zeigt schon die Ziffer und für 20 Samples liegen gute Ergebnisse vor. Die Anwendung derselben Einstellungen für Strukturen des Graphen, die mit ELEM-GM erzeugt wurden, führten zu den Ergebnissen in Abbildung 6.3.

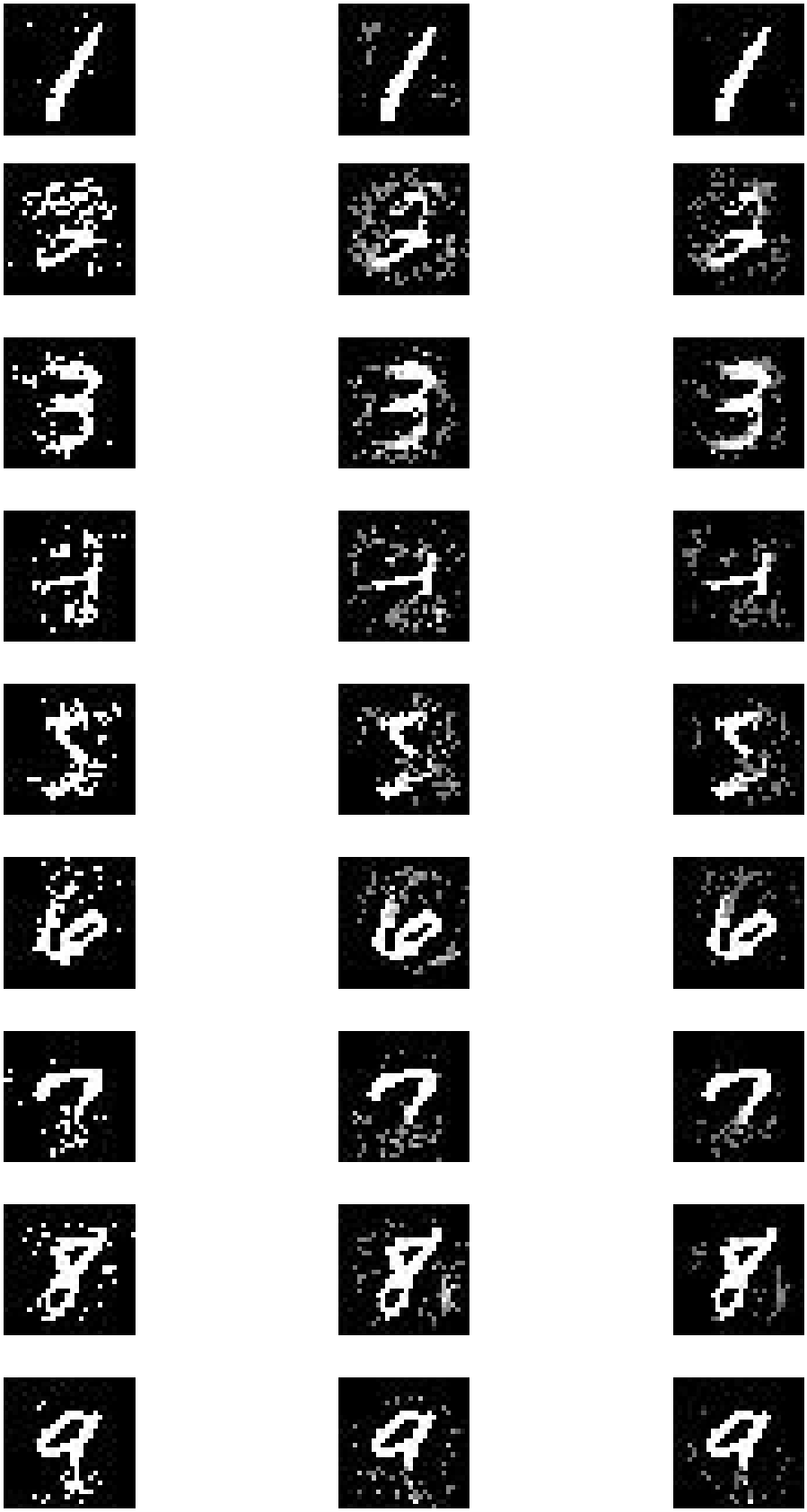
Die Ergebnisse des ELEM-GM-Graphen sind eindeutiger, da an der Helligkeit der Pixel erkennbar ist, dass die Pixel beim ELEM-GM-Graphen mit höherer Wahrscheinlichkeit vorliegen, als beim Gitter-Graphen. Die Ergebnisse sind schon für ein Sample besser und werden nur marginal besser, was aber nicht für alle Ziffern zutrifft. Der gelernte Graph bildet die Struktur der Ziffer gut ab, lernt aber viele Kanten, die einen konstanten Wert aufweisen und sich nicht ändern können.

Die MNIST-Daten lassen sich gut mit den PAM-Sampling generieren, wenn die Ergebnisse über Mittelwerte der erzeugten Datenpunkte verwendet werden. Während des Training des PAM-GAN wurden in den späteren Experimenten mit dem MNIST-Datensatz 1 und 5 Samples verwendet, um darüber mit dem Mittelwert ein Bild zu erzeugen.



(a) 1 Sample **(b)** 5 Samples **(c)** 20 Samples

Abbildung 6.2: Die dargestellten Ziffern wurden mit PAM-Sampling über einen Gitter-Graphen erzeugt. In jeder Zeile ist eine Ziffer dargestellt, die aus dem Graphen erzeugt wurde, mit der entsprechenden Anzahl an Bildern, die in die Mittelwertbildung einflossen.



(a) 1 Sample **(b)** 5 Samples **(c)** 20 Samples

Abbildung 6.3: Die dargestellten Ziffern wurden mit PAM-Sampling über einen ELEM-GM-Graphen erzeugt. In jeder Zeile ist eine Ziffer dargestellt, die aus dem Graphen erzeugt wurde, mit der entsprechenden Anzahl an Bildern, die in die Mittelwertbildung einflossen.

• FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN,FILL-IN
• whats,the,difference,between,the,two,in,simple,english,FILL-IN,plz,jzk
• is,it,me,FILL-IN,or,does,he,look,FILL-IN,like,FILL-IN,FILL-IN
• i,dunno,if,u,can,FILL-IN,i,think,its,part,of,FILL-IN
• i,guess,thats,why,i,dont,understand,it,FILL-IN,it,how,FILL-IN
• can,i,ask,someone,to,help,or,is,that,considered,as,FILL-IN

Tabelle 6.1: Ergebnisse des PAM-Samplings mit einem Graphen für den relNet-Datensatz, der eine Ketten-Struktur hat und dessen Gewichte mit ML trainiert wurden.

• israeli,police,FILL-IN,netanyahu,over,FILL-IN
• brief-gladstone,land,acquires,FILL-IN,in,colorado
• FILL-IN,confident,sunderland,can,avoid,relegation
• paul,ryan,re-elected,speaker,of,house
• suns,hand,heat,6th,straight,loss
• FILL-IN,century,announces,change,of,chairman

Tabelle 6.2: Ergebnisse des PAM-Samplings für den Reuters-Datensatz mit einem Graphen, der eine Ketten-Struktur hat und dessen Gewichte mit ML trainiert wurden.

Textdaten

Um die Frage der Qualität der erzeugten Texte durch das PAM-Sampling zu beantworten, wurden auch diese getestet. PX ist in der Lage Strings direkt zu verarbeiten, daher wurden diese direkt übergeben. Als Struktur des Graphen wurde eine Kette genutzt und deren Gewichte mit ML optimiert. Zur Generierung der Sätze wurde nur ein Datenpunkt genutzt und kein Mittelwert gebildet. Der Datensatz aus dem relNet-Projekt führte zu Ergebnissen, die in Tabelle 6.1 gezeigt sind.

Bei den Ergebnissen fehlen viele Wörter. Die erzeugten Datenpunkte sind Sätze, die aus dem Datensatz gezogen wurden, und das PAM-Sampling ist nicht in der Lage mit den gelernten Gewichten neue Sätze zu erzeugen.

Für den Reuters-Datensatz sind die Ergebnisse in Tabelle 6.2 dargestellt. Auch hier gilt, mit den gelernten Gewichten erzeugt das PAM-Sampling keine neuen Sätze, sondern zieht nur aus dem gegebenen Datensatz. Für Textdatensätze zeigt sich, dass die gelernten Gewichte nur den Datensatz wiedergeben. Mittels des PAM-GAN-Trainings besteht die Hoffnung, dass mit zufällig initialisierten Gewichten Ergebnisse vorliegen, die nicht nur aus dem Datensatz stammen.

6.3.2 Parametereinstellung h

Beim FDM ergibt sich die Ableitung einer Funktion durch die Gewichte der Differenz zweier Funktionen (5.23):

$$\frac{\partial}{\partial \boldsymbol{\theta}^g} \phi^g \left(g \left(z \right) \right) = \frac{\phi^g \left(g \left(\boldsymbol{z} \right) \right) - \phi^g \left(g \left(\boldsymbol{z} + \boldsymbol{eL} h \right) \right)}{h}. \tag{6.2}$$

In $\frac{\partial}{\partial \boldsymbol{\theta}^g} \phi^g \left(g \left(z \right) \right)$ ist die Differenz zweier mit PAM-Sampling generierter Datenpunkte enthalten. Diese unterscheiden sich in einem einzigen Gewicht θ_i^g und stellen den partiellen Gradienten dar. Die Differenz ist gegeben als Vektor von Gewichten der Form $\{-1/h, 0, +1/h\}$. Im Gradienten des Generators werden mit dem partiellen Gradienten die $\boldsymbol{\theta}^c$ gewichtet, welche in die Aktualisierung des Generators einfließen. In [28] wird die Annahme getroffen, dass h klein zu wählen ist, aber wenn das h klein ist, unterscheiden sich die generierten Datenpunkte unter Umständen nicht und der Gradient wäre 0. Deshalb soll h so gewählt werden, dass die Unterschiede der beiden generierten Datenpunkte größtmöglich ausfallen.

Um dies zu bewerten, wird für die Ziffer 3 des MNIST-Datensatzes der vollständige Gradient berechnet. Der Gradient ist als Matrix gegeben, wo in einer Spalte i die Differenz des Referenz-Datenpunktes und des Datenpunktes, für den $\theta_i + h$ gilt, angegeben wird. Es soll das h gefunden werden, für das die Anzahl der Elemente ungleich 0 am größten ist.

Die Auswirkungen durch das Verzerren mit h variieren bei den eingesetzten Graphen. In einem Gitter-Graphen ist jeder Knoten, der nicht am Rand liegt, mit drei weiteren verbunden. Wenn das Gewicht einer Kante geändert wird und dadurch die Knoten ihre Realisierung ändern, betrifft dies auch ihre Nachbarn und kann sich nach dem gleichen Muster durch den Graphen ziehen. Bei ELEM-GM ist die Anzahl der Nachbarn eines Knoten nicht von vornherein bekannt und eine Entsprechung der benachbarten Pixel wie bei einem Gitter-Graphen ist nicht gegeben. Für die beiden Graphenstrukturen wurden verschiedene h getestet und die Auswahl nach den genannten Kriterien getroffen. In der Optimierung des Parameters wurde nur die Differenz $\phi^g \left(\boldsymbol{x}' \right) - \phi^g \left(\boldsymbol{x}'' \right)$ betrachtet, da der Quotient für die Berechnung irrelevant ist.

Gitter-Graph

In Tabelle 6.3 sind die Häufigkeiten der Differenz $\{-1, 0, +1\}$ für verschiedene h-Werte gegeben. Für den Datensatz der Ziffer 3 ergibt sich ein Graph mit 784 Knoten und 1.512 Kanten, die über 4.712 Gewichte verfügen und zu einer Matrix mit 22.202.944 Zellen führen. Für $h = 15$ sind die meisten Zellen ungleich 0.

In Abbildung 6.4 ist der Gradient $\frac{\partial}{\partial \boldsymbol{\theta}^g} \phi^g \left(\boldsymbol{x}' \right)$ für $h = 15, 0$ geplottet. Auf der y-Achse ist pro Zeile $\phi^g \left(\boldsymbol{x}' \right) - \phi^g \left(g \left(\boldsymbol{z} + \boldsymbol{eL} h \right) \right)$ mit $\boldsymbol{eL} = (0, ..., 1, ...0)^T$ abgetragen, wobei die 1 an der Stelle steht, die der Zeile der Matrix entspricht. Auf der x-Achse ist das jeweilige

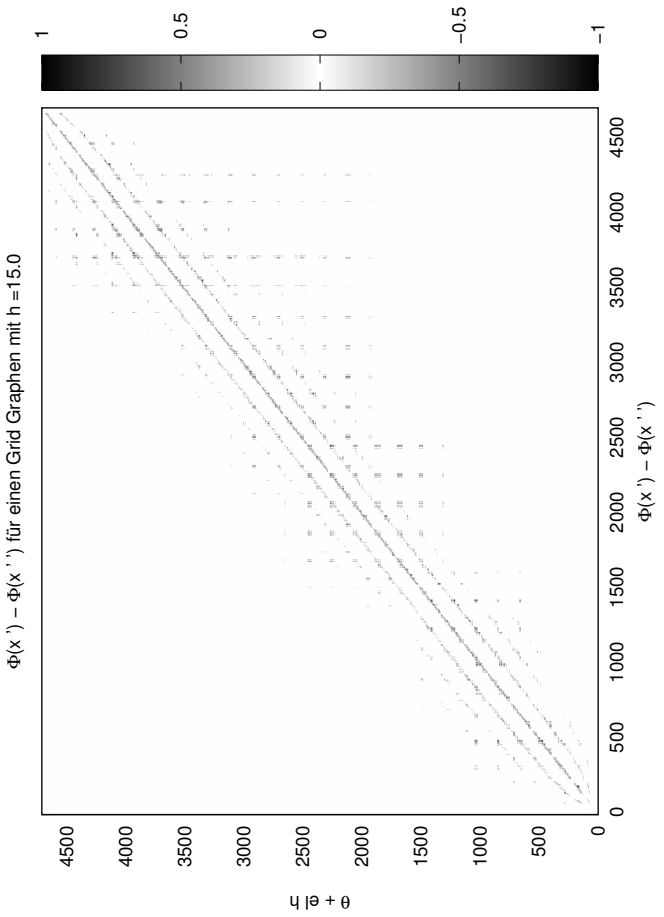


Abbildung 6.4: Differenzmatrix zu h = 15. Auf der y-Achse ist der zweite erzeugte Datenpunkt abgetragen, der sich durch das verzerrte z ergibt. Auf der x-Achse ist die Differenz abgetragen. Die schwarzen Flächen sind Knoten, die ihren Wert nicht geändert haben. Die weißen Punkte stellen unsymmetrische Änderungen der erzeugten Datenpunkte dar.

Tabelle 6.3: Häufigkeit der Differenz $\phi^g(\boldsymbol{x}') - \phi^g(\boldsymbol{x}'')$ angewandt auf die Ziffer 3 des MNIST-Datensatzes für einen Gitter-Graphen.

h	-1	0	+1
1,5	13.108	22.176.728	13.108
2,0	17.044	22.168.856	17.044
2,5	21.043	22.160.858	21.043
5,0	34.087	22.134.770	34.087
5,5	35.913	22.131.118	35.913
6,0	37.569	22.127.806	37.569
6,5	39.110	22.124.724	39.110
7,0	40.183	22.122.578	40.183
7,5	41.024	22.120.896	41.024
10,0	43.172	22.116.600	43.172
15,0	**43.844**	**22.115.256**	**43.844**

Ergebnis der Differenz abgezeichnet. Dabei fällt auf, dass auf den Elementen der Hauptdiagonalen, welche assoziiert sind mit der Kante, zu der das Gewicht zählt, fast immer ein Unterschied vorliegt. Die geringste Anzahl an Unterschieden zweier Datenpunkte war gegeben für die Knoten der Kante und deren Nachbarschaft, wodurch immer wenigstens 8 Kanten anders kodiert waren. Es ist zu beachten, das das Flippen bei $\phi^g(\boldsymbol{x}'')$ immer mit einer symmetrischen Änderung im Vergleich zu $\phi^g(\boldsymbol{x}')$ einher geht und $\{-1, +1\}$ immer ausgeglichen sind.

Bis auf in wenigen Fällen ist immer eine Änderung zu sehen. Zum Vergleich sei $h = 1,5$ in Abbildung 6.5 gegeben, wo die Änderungen auf der Hauptdiagonalen erkennbar sind, aber diese nicht deutlich hervortreten. Zusätzlich ist die Gitterstruktur nicht so gut zu erkennen und die möglichen Änderungen des Gradienten $\frac{\partial}{\partial \boldsymbol{\theta}^g}\phi^g$ wären dabei geringer als für $h = 15,0$. Daher wurde für die nachfolgenden Experimente mit dem Gitter-Graph $h = 15.0$ gewählt.

ELEM-GM-Graph

Für den MNIST-Datensatz wurde auch der Graph verwendet, der sich durch den ELEM-GM-Schätzer ergibt. Der Graph hatte insgesamt 2.176 Kanten und 8.712 Gewichte. Die Häufigkeiten der Differenzen für ELEM-GM sind in Tabelle 6.4 dargestellt.

Beim Gitter-Graphen stieg die Anzahl der Zellen, die ungleich 0 waren, bis $h = 15$, was bei einem ELEM-GM-Graphen nicht passierte. Bei $h = 10,0$ ergab sich die stärkste Änderung, dabei ist in Abbildung 6.6 auf der Hauptdiagonalen die Änderung zu erkennen, aber eine weitere Struktur ist nicht erkennbar. Die zugrunde liegende Struktur des Graphen

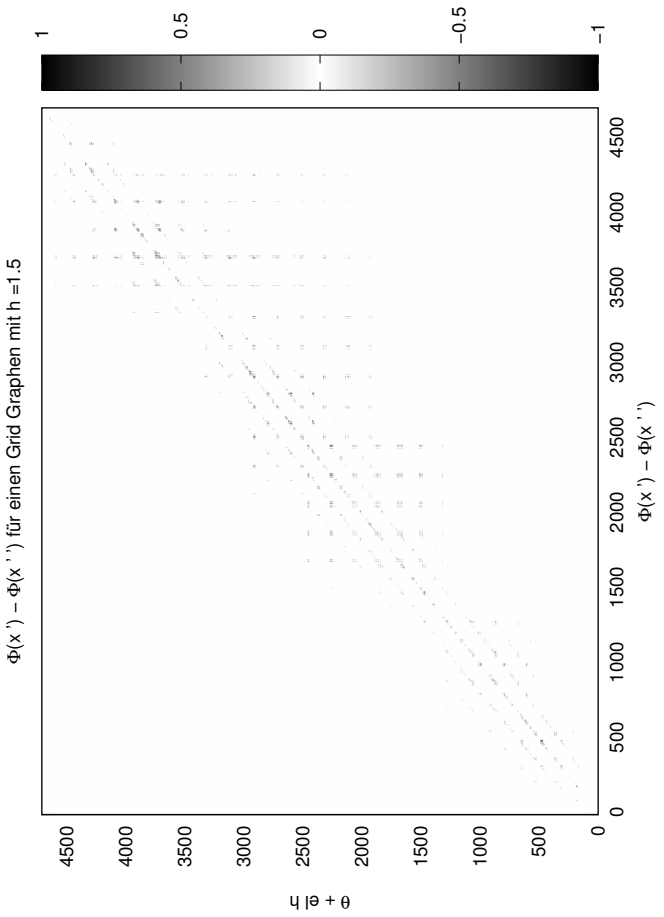


Abbildung 6.5: Differenzmatrix zu h = 1,5. Auf der y-Achse ist der zweite erzeugte Datenpunkt abgetragen, der sich durch das verzerrte z ergibt. Auf der x-Achse ist die Differenz abgetragen. Die schwarzen Flächen sind Knoten, die ihren Wert nicht geändert haben. Die weißen Punkte stellen unsymmetrische Änderungen der erzeugten Datenpunkte dar.

Tabelle 6.4: Häufigkeit der Differenz $\phi^g(\boldsymbol{x}') - \phi^g(\boldsymbol{x}'')$ angewandt auf die Ziffer 3 des MNIST-Datensatzes für einen ELEM-GM-Graphen mit 2.176 Kanten und 8.712 Gewichten.

h	-1	0	+1
0,0001	4.004.007	67.751.602	4.004.007
0,001	4.033.115	67.693.386	4.033.115
0,01	4.024.867	67.709.882	4.024.867
0,1	3.136.897	69.485.822	3.136.897
1,0	3.344.804	69.070.008	3.344.804
1,5	4.271.446	67.216.724	4.271.446
2,5	4.385.036	66.989.544	4.385.036
5,0	4.515.917	66.727.782	4.515.917
10,0	**4.616.671**	**66.526.274**	**4.616.671**
15,0	3.814.291	68.131.034	3.814.291

ist nicht von vornherein bekannt. Aber die Kanten waren nicht als Gitter gegeben und so beeinflusste die Änderung eines Gewichts Kanten an unterschiedlichster Stelle des Graphen.

Zum Vergleich ist in Abbildung 6.7 die Differenzmatrix für $h = 0,001$ abgebildet. Hier sind die Kanten auf der Hauptdiagonalen nicht immer betroffen. Daher wurde für die nachfolgenden Experimente $h = 10,0$ gewählt, da an der dem verzerrten Gewicht θ_i assoziierten Kante eine Änderung auftrat und noch weitere Kanten ihren Zustand änderten.

6.3.3 PAM-GAN-Training MNIST-Daten

In diesem Kapitel wird der PAM-GAN Algorithmus 3 evaluiert, der ein MRF zur Generierung von Datenpunkten aus dem MNIST-Datensatz lernt. Die generierten Datenpunkte können visuell evaluiert werden, wodurch die Unterschiede in den Parametern direkt erkennbar sind. Es wurden für verschiedene Parametereinstellungen MRF mit Gitter-Struktur und ELEM-GM getestet.

Als erstes Expxeriment wurde, wie beim PAM-Sampling, der gesamte Datensatz für das Training des GAN genutzt. Das h wurde entsprechend dem evaluierten Wert für die jeweilige Struktur genommen und $\alpha = 0,01$, $\beta = \frac{1}{|\boldsymbol{\theta}^g|}$ gewählt. So ist die Schrittweite für beide Gradienten klein genug, um das Optimum des jeweiligen Gradienten zu finden. Die initialen Gewichte des Generators wurden mit der ML-Methode initialisiert. Für den Gittergraphen wurde das PAM-GAN Training 50.000 Iterationen laufen lassen, für den ELEM-GM-Graphen waren es 20.000 Iterationen.

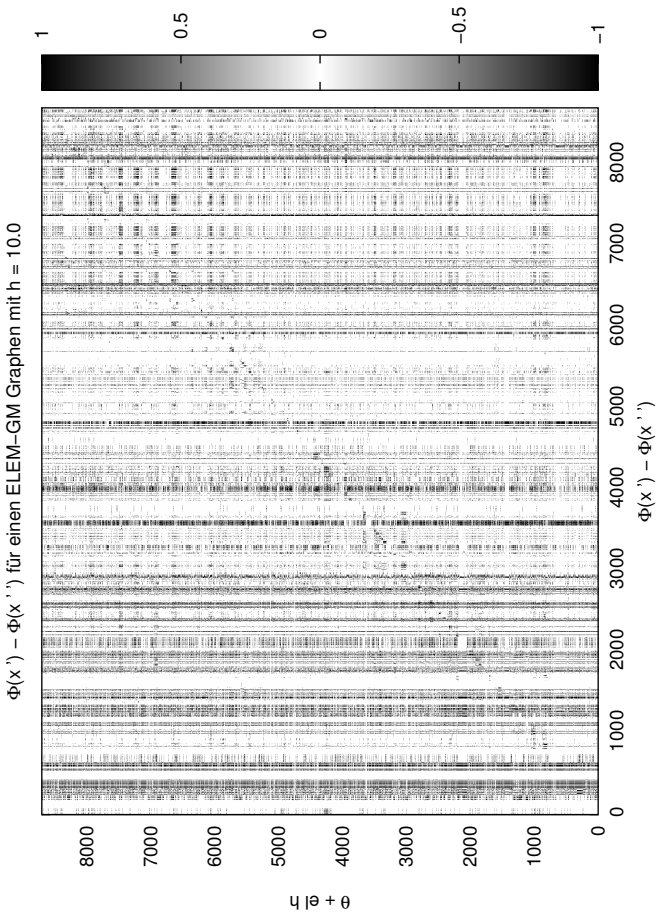


Abbildung 6.6: Differenzmatrix zu h = 10,0. Auf der y-Achse ist der zweite erzeugte Datenpunkt abgetragen, der sich durch das verzerrte z ergibt. Auf der x-Achse ist die Differenz abgetragen. Die schwarzen Flächen sind Knoten, die ihren Wert nicht geändert haben. Die weißen Punkte stellen unsymmetrische Änderungen der erzeugten Datenpunkte dar.

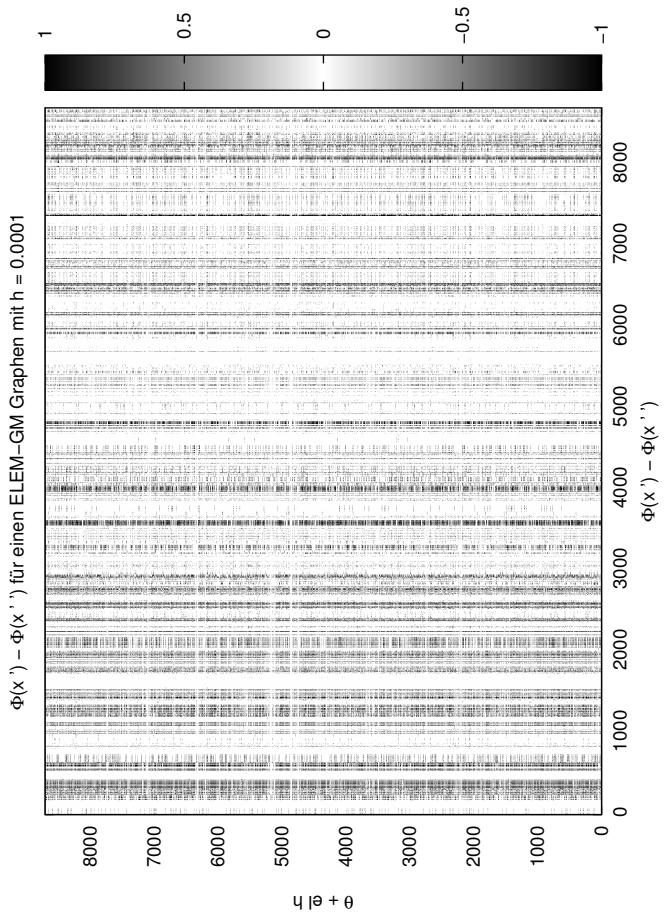
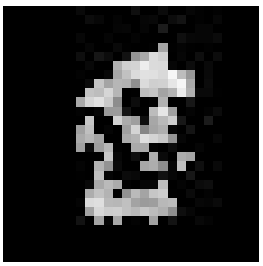
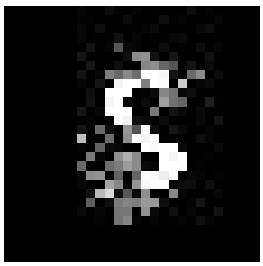


Abbildung 6.7: Differenzmatrix zu h = 0,0001. Auf der y-Achse ist der zweite erzeugte Datenpunkt abgetragen, der sich durch das verzerrte z ergibt. Auf der x-Achse ist die Differenz abgetragen. Die schwarzen Flächen sind Knoten, die ihren Wert nicht geändert haben. Die weißen Punkte stellen unsymmetrische Änderungen der erzeugten Datenpunkte dar.

Abbildung 6.8: Vergleich des PAM-GAN mit Gitter- und ELEM-GM-Graphen auf dem vollständigen MNIST-Datensatz.



(a) Gitter-Graph.



(b) ELEM-GM-Graph.

Für den ELEM-GM-Graphen ergibt sich ein Bild, das bis auf marginale Änderungen konstant bleibt und der Abbildung 6.1b) ähnlich sieht. Das GAN war nicht in der Lage die einzelnen Ziffern zu erzeugen, weswegen für die Teilmenge der Ziffern ein Graph mit dem PAM-GAN optimiert wurde.

Gitter-Graph

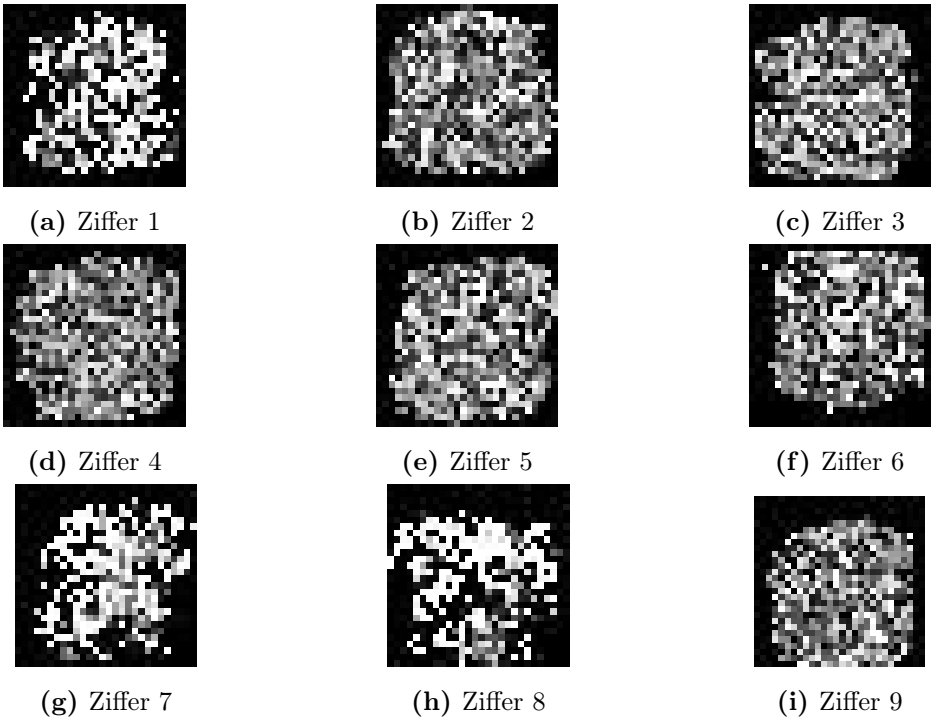
Ein Gitter-Graph lernt keine Struktur der Daten und verfügt nur über konstante Kanten, die in den Daten vorliegen, wie der schwarze Rand bei den MNIST-Daten. Für den Gitter-Graphen wurden die in Tabelle 6.5 gegebenen Parametereinstellungen eingesetzt.

Tabelle 6.5: Parametereinstellungen für die Experimente mit dem Gitter-Graphen.

Nr.	Init $\boldsymbol{\theta}^g$	Init $\boldsymbol{\theta}^c$	Iterationen	β	α	M_f	M_g	h
1	$\mathcal{N}(0,1)$	$\mathcal{N}(0,1)$	50.000	0,01	0,01	5	5	15.0
2	$ML(D)$	$\mathcal{N}(0,1)$	50.000	$\frac{1}{\lvert\boldsymbol{\theta}^g\rvert}$	0,01	5	5	15.0
3	$ML(D)$	$\mathcal{N}(0,1)$	20.000	$\frac{1}{\lvert\boldsymbol{\theta}^g\rvert}$	0,01	5	5	15.0

Da der ELEM-GM-Graph mit einer zufälligen Initialisierung funktionierte, wurde dies auch hier getestet, vgl. Parametereinstellung 1. Für die ersten beiden Parametereinstellungen in Tabelle 6.5 wurde der Algorithmus wie im Kapitel 5.3 eingesetzt. In der dritten Spalte bekam der Kritiker den Mittelwert von 5 erzeugten Bildern übergeben, wie in der zweiten Parametereinstellung bei ELEM-GM, siehe Tabelle 6.6. Durch diesen Schritt wird der Kritiker mit besser generierten Datenpunkten trainiert. Nach der Evaluation des PAM-Samplers besteht die Annahme, dass das PAM-GAN für einen Gitter-Graphen in der Lage ist das anfängliche Bild zu verbessern.

Für die einzelnen Ziffern ergab sich mit Parametereinstellung 1 das Resultat von Abbildung 6.9. Die Ergebnisse zeigen für den gesamten Trainingsverlauf ein Rauschen, in dem nicht die Ziffer zu erkennen ist. Für unterschiedliche β und α ergaben sich vergleich-



(a) Ziffer 1 **(b)** Ziffer 2 **(c)** Ziffer 3

(d) Ziffer 4 **(e)** Ziffer 5 **(f)** Ziffer 6

(g) Ziffer 7 **(h)** Ziffer 8 **(i)** Ziffer 9

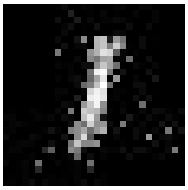
Abbildung 6.9: PAM-GAN Ergebnisse für einen Gitter-Graphen und standardnormalverteilter Initialisierung aller Gewichte.

bar verrauschte Bilder, in denen nichts zu erkennen ist. Mit den entsprechend gewählten Startgewichten ist das PAM-GAN nicht in der Lage die Ziffern zu generieren.

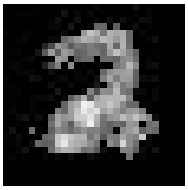
Die Initialisierung des Generators hat einen entscheidenen Einfluss auf das Resultat. Mit den Parametereinstellungen 2 und 3 aus Tabelle 6.5 wurde zu Beginn $\boldsymbol{\theta}^g$ mit ML initialisiert. Während des Trainings bekam der Kritiker einzelne PAM-Samples zu sehen, was nur grob der Form der jeweiligen Ziffer ähnelte. Die finalen Bilder sind ein Mittelwert über 10 Datenpunkte, die visuell besten Ergebnisse sind in Abbildung 6.10 zu sehen.

Das Training lief immer 50.000 Iterationen und brach dann ab. Es wurde keine Zielfunktion ausgewertet, um die Qualität zu bewerten und früher zu stoppen, daher ist in den Bildern immer die Iteration angegeben. Während des Trainings veränderten sich die Ziffern durchgängig, sodass die Annahme, dass der Gradient des Kritikers nie 0 ist, aufgestellt werden kann, aber genauer evaluiert werden muss. Bedingt durch den Mittelwert erzeugt der Generator schon zu Beginn bei manchen Ziffern gute Werte, die aber im Verlauf besser wurden. Für die Ziffer 2 ist in Abbildung 6.11 der Verlauf angegeben, jedes Bild entspricht den Parametern nach 5.000 Iterationen, wobei das erste Bild mit den initialen Parametern des Generators erzeugt wurde. Die Bilder entstanden durch die Mittelwertbildung über 10 generierte Datenpunkte.

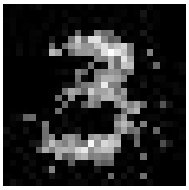
Zu Beginn ist bloß die grobe Struktur erkennbar, der obere Bogen ist noch eine Gerade und der Fuß der Zahl ist eine graue Fläche ohne weitere Struktur. Bis Iteration 35.000



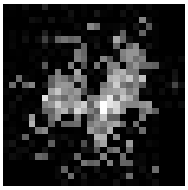
(a) Ziffer 1, 50.000 Iter.



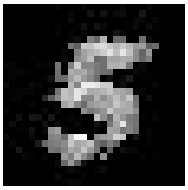
(b) Ziffer 2, 40.000 Iter.



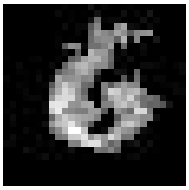
(c) Ziffer 3, 40.000 Iter.



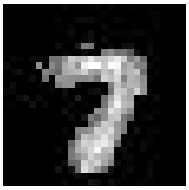
(d) Ziffer 4, 20.000 Iter.



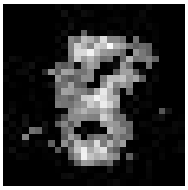
(e) Ziffer 5, 25.000 Iter.



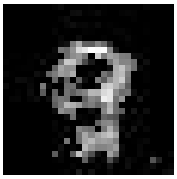
(f) Ziffer 6, 25.000 Iter.



(g) Ziffer 7, 35.000 Iter.

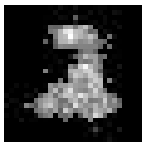


(h) Ziffer 8, 10.000 Iter.

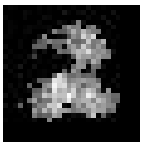


(i) Ziffer 9, 20.000 Iter.

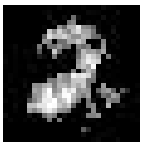
Abbildung 6.10: PAM-GAN Ergebnisse für einen Gitter-Graphen, für den der Generator mit ML und der Kritiker mit $\mathcal{N}(0,1)$ initialisiert wurde. Der Algorithmus lief immer mit 50.000 Iterationen und alle 5.000 Iterationen wurden die aktuellen Parameter gespeichert. In den Bildern sind die Pixel dargestellt, die zu mindestens 40% im Mittel vorlagen, so wurde das Rauschen im Bild unterdrückt.



(a) 0 Iterationen

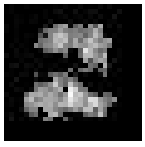


(b) 5.000 Iterationen

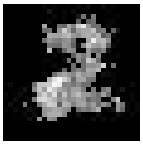


(c) 10.000 Iterationen

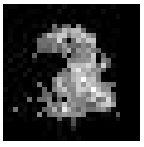
(d) 15.000 Iterationen



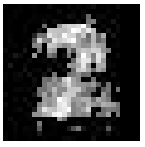
(e) 20.000 Iterationen



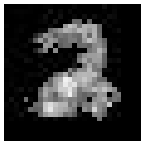
(f) 25.000 Iterationen



(g) 30.000 Iterationen

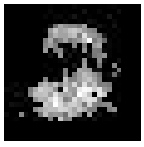


(h) 35.000 Iterationen



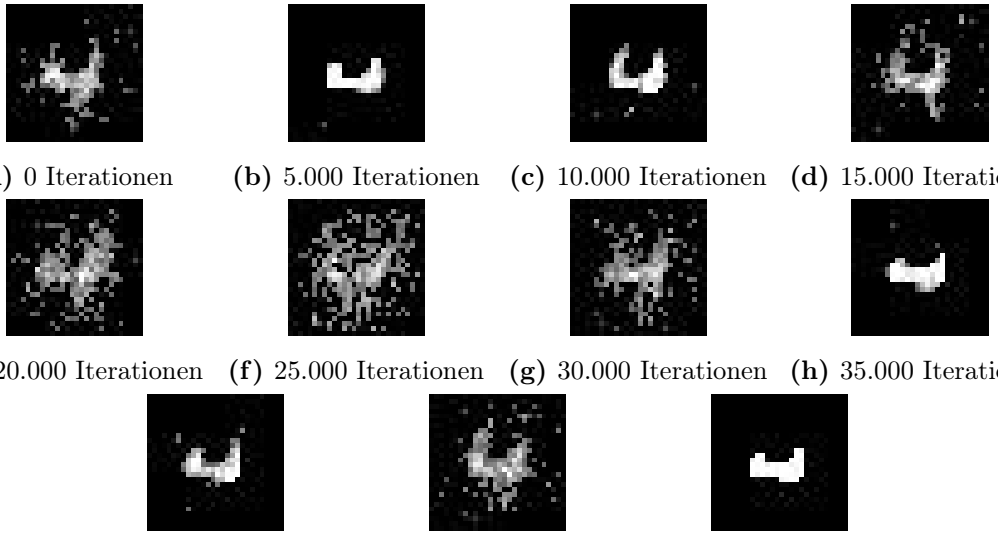
(i) 40.000 Iterationen

(j) 45.000 Iterationen



(k) 50.000 Iterationen

Abbildung 6.11: Entwicklung der Ziffer 2 durch PAM-GAN mit einem Gitter-Graphen. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$, die durch ML optimiert wurden.



(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen

(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung 6.12: Entwicklung der Ziffer 4 durch PAM-GAN mit einem Gitter-Graphen. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$, die durch ML optimiert wurden.

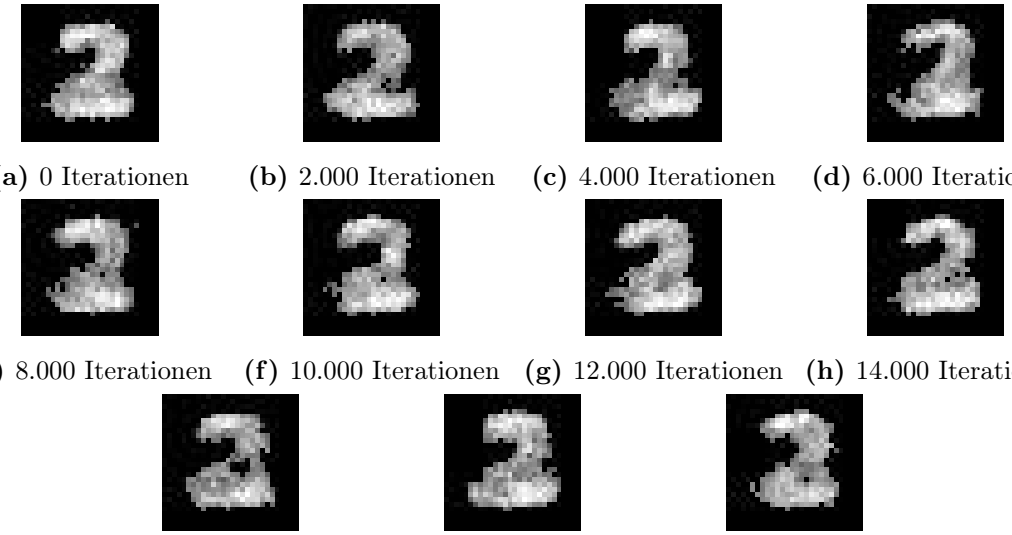
verbessert sich das Resultat mit geringen Ausreißern, um dann anschließend wieder schlechter zu werden. Hier könnte das PAM-GAN Training angepasst werden und das Training vorzeitig stoppen, wobei damit eine spätere mögliche Verbesserung ausgeschlossen wird.

Der ELEM-GM-Graph hatte bei einigen Ziffern Probleme mit bestimmten Features, da er für diese keine Kanten gelernt hat. Der Gitter-Graph ist hier nicht eingeschränkt und kann die Darstellung der Ziffer besser lernen, siehe die Entwicklung der Ziffer 4 in Abbildung 6.12, wo die Ziffer verschwommen zu Beginn zu erkennen ist und zwischen 15.000 bis 20.000 Iterationen ihr bestes Ergebnis erreicht. Zusätzlich ist in der Entwicklung zu sehen, das diese wieder schlechter wird und sich die Ergebnisse wiederholen.

Der Gitter-Graph ist in der Lage die Features des Bildes zu lernen, aber insgesamt oszilliert der Generator zwischen dem mittleren Balken der 4 und einer kompletten Darstellung der Ziffer. Mit genügend Iterationen könnte das Oszillieren auch für die anderen Ziffern auftreten. Insgesamt zeigt sich, dass mit weniger Iterationen zwischen den Aktualisierungen der Parameter das optimale Bild gefunden würde. Die weiteren Ziffern sind im Anhang A.1 zu finden.

Die dritte Parametereinstellung in Tabelle 6.5 wurde genutzt, wenn während des Trainings Mittelwerte über 5 Bilder gebildet wurden. Die Iterationen wurden gesenkt, um einen besseren Vergleich zum ELEM-GM-Graphen zu gewährleisten. Im Folgenden ist die Entwicklung für die Ziffer 2 in Abbildung 6.13 gegeben, zwischen den einzelnen Bildern liegen jetzt 2.000 Iterationen.

Insgesamt zeigt sich in der Entwicklung der Ziffer 2, dass ein gutes Ergebnis nach 2.000 Iterationen vorliegt, um sich anschließend nur noch wenig zu ändern. Bessere Bilder im



(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen

(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung 6.13: Entwicklung der Ziffer 2 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$, die durch ML optimiert wurden.

Training des Kritikers führten dazu, dass dieser seine Gewichte nur gering änderte und dies auch beim Generator passierte. In Anhang A.2 sind die weiteren Ziffern dargestellt, es zeigte sich für viele Ziffern ein gutes Bild, aber bei der 8 und der 9 führte dies zu schlechteren Ergebnissen, die sich nicht mehr verbesserten. Für manche Parametereinstellungen waren die Ergebnisse mit einem durch ein Sample generierten Bild im Training besser.

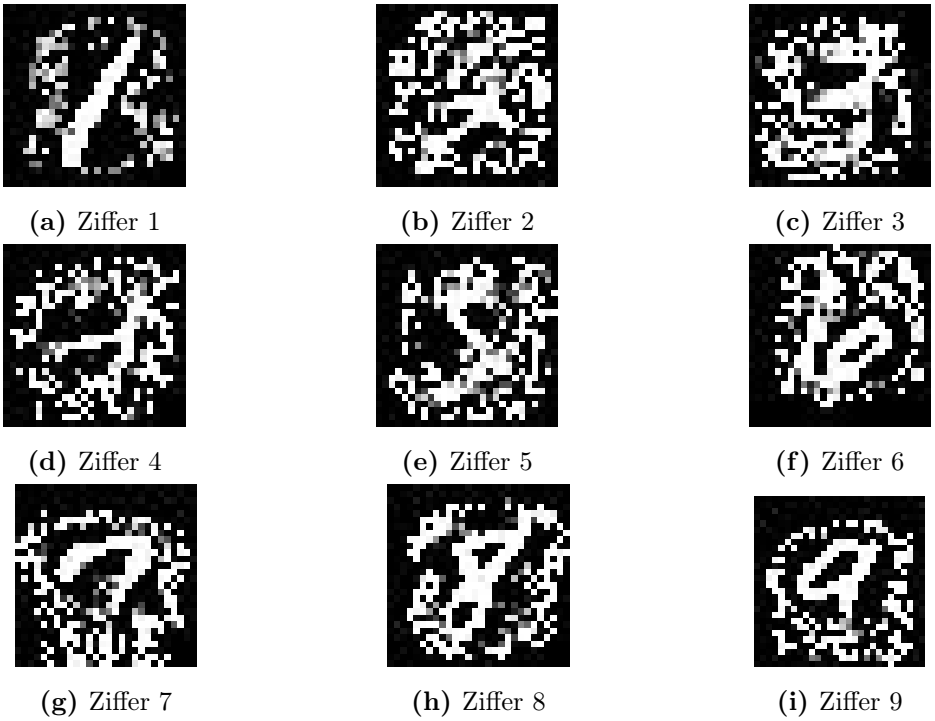
ELEM-GM-Graph

Der ELEM-GM-Graph wurde mit den in Tabelle 6.6 genannten Parametern getestet.

Tabelle 6.6: Parametereinstellungen für die Experimente mit dem ELEM-GM-Graphen.

Nr.	Init $\boldsymbol{\theta}^g$	Init $\boldsymbol{\theta}^c$	Iterationen	β	α	M_f	M_g	h
1	$\mathcal{N}(0,1)$	$\mathcal{N}(0,1)$	20.000	0,001	0,01	5	5	10.0
2	$ML(D)$	$\mathcal{N}(0,1)$	20.000	$\frac{1}{\lvert\boldsymbol{\theta}^g\rvert}$	0,01	5	5	10.0

Für beide Parametereinstellungen wurde die Struktur der Graphen gelernt, und mit einer unterschiedlichen Initialisierung des Generators gestartet. Die zweite Parametereinstellung diente als Referenz zum Gitter-Graphen, hier wurden ebenfalls 5 Datenpunkte generiert und der Mittelwert gebildet. In Abbildung 6.14 sind die Ergebnisse der ersten Parametereinstellung abgebildet, hierbei wurde nur ein Datenpunkt mit dem PAM-Sampling erzeugt und der Mittelwert über 10 Datenpunkte ausgegeben. Ein Pixel wurde auf 1 gesetzt, wenn er in mindestens 40% der erzeugten Datenpunkte auftrat.



(a) Ziffer 1 **(b)** Ziffer 2 **(c)** Ziffer 3

(d) Ziffer 4 **(e)** Ziffer 5 **(f)** Ziffer 6

(g) Ziffer 7 **(h)** Ziffer 8 **(i)** Ziffer 9

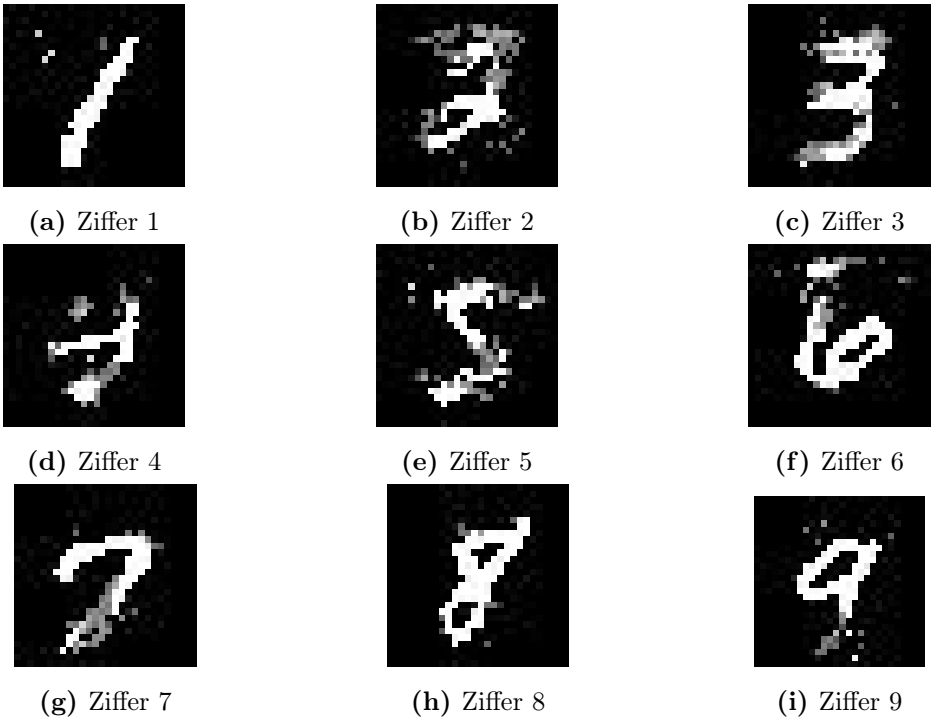
Abbildung 6.14: PAM-GAN Ergebnisse für ELEM-GM-Graphen mit Parametereinstellung Nr. 1.

In den einzelnen Bildern sind die Ziffern grob zu erkennen. Da der Graph über viele Kanten verfügt, die nur einen Zustand besitzen, ändert sich das initiale Bild nur gering. Dadurch sind die Ergebnisse direkt zu Beginn gut, aber haben keine Chance sich grundlegend zu ändern, denn die adjazenten Knoten haben nur eine mögliche Realisierung. Auch wenn der Kritiker im GAN-Training ein Bild als Mittelwert von 5 generierten Punkten erhält, bleibt diese Eigenschaft des ELEM-GM-Graphen erhalten, siehe Abbildung 6.15. In Abbildung 6.15 sind zwar einige Ziffern besser zu erkennen als in Abbildung 6.14, aber eine wirkliche Änderung trat nicht ein.

6.3.4 Mode Collapse

Bei der Generierung der Ziffern änderten sich die Ziffern durchgängig für die aktuellen Gewichte. Allerdings war das PAM-GAN nicht in der Lage mit einer zufälligen Initialisierung die Gewichte zu lernen, wodurch der direkte Vergleich fehlt.

Als Beispiel sei $|| \nabla_{\boldsymbol{\theta}^c} ||_2$ für das Training der Ziffer 4 mit einem Gitter-Graphen gegeben. Die Gewichte wurden mit der ML-Methode initialisiert und es wurden die Bilder im Training der Ziffer mit einen einfachem PAM-Sampling erstellt. Das Training lief 20.000 Iterationen.



(a) Ziffer 1 **(b)** Ziffer 2 **(c)** Ziffer 3

(d) Ziffer 4 **(e)** Ziffer 5 **(f)** Ziffer 6

(g) Ziffer 7 **(h)** Ziffer 8 **(i)** Ziffer 9

Abbildung 6.15: PAM-GAN Ergebnisse für ELEM-GM-Graphen mit Parametereinstellung Nr. 2.

Wenn der perfekte Kritiker gefunden wird, kann es passieren, dass der Gradient 0 ist. Um die Entwicklung des Gradienten zu bewerten, wurde die $L2$-Norm verwendet:

$$|| \boldsymbol{X} ||_2 = \sqrt{\sum_{i=1}^{m} X_i^2} \tag{6.3}$$

Diese wurde in jeder Iteration berechnet und gespeichert. Dies erfolgte für den Gradienten des Kritikers und den Gradienten des Generators. Für den Kritiker ergibt sich der Graph in Abbildung 6.16 und für den Gradienten des Generators der Graph in Abbildung 6.17.

Durch die mit der ML-Methode optimierten Gewichte $\boldsymbol{\theta}^g$ ist der Gradient des Generators zu Beginn sehr gering und nur wenige Änderungen ergeben sich. Da im Training nur wenige Gewichte beim Generator verändert werden, hat dies auch nur geringe Auswirkungen auf den Generator. Erst wenn der Generator bei seinem optimalen Bild ankommt, verringert sich der Gradient des Kritikers und gleichzeitig steigt der Gradient des Generators. Für die Ziffer 4 ergibt sich das an dem Punkt, an dem die Ergebnisse wieder schlechter werden.

Abbildung 6.16: Graph der $L2$-Norm des Gradienten des Kritikers.

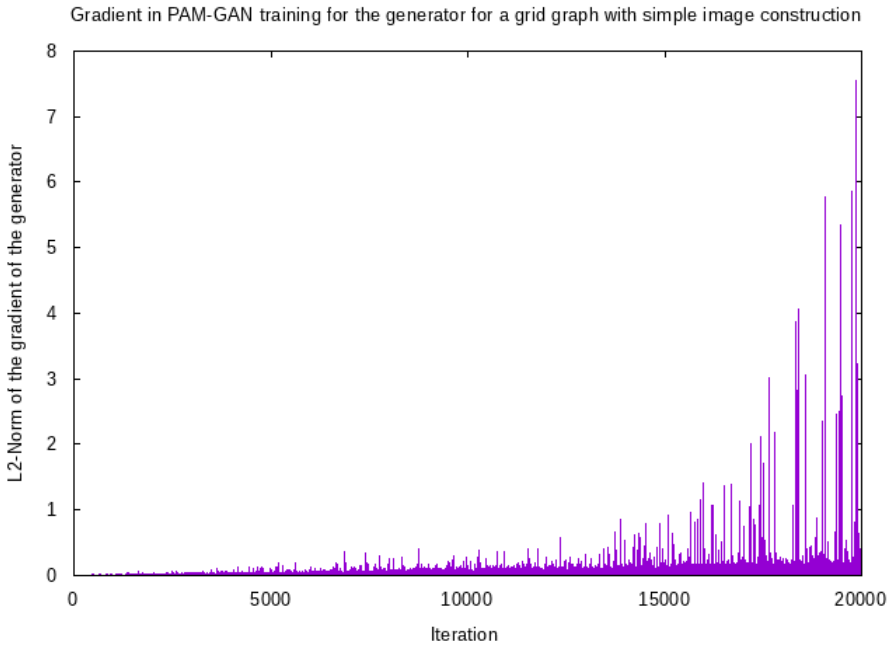


Abbildung 6.17: Graph der $L2$-Norm des Gradienten des Generators.

6.3.5 PAM-GAN-Training Textdaten

In dem PAM-GAN Verfahren wurden auch zwei Textdatensätze getestet, in Tabelle 6.7 ist die Parametereinstellung dargestellt.

Tabelle 6.7: Parametereinstellungen für die Experimente mit den Textdatensätzen.

Nr.	Init $\boldsymbol{\theta}^g$	Init $\boldsymbol{\theta}^c$	Iterationen	β	α	M_f	M_g	h
1	$\mathcal{N}(0,1)$	$\mathcal{N}(0,1)$	200	$\frac{1}{\lvert\boldsymbol{\theta}^g\rvert}$	0.001	5	5	10

Der Parameter h wurde nicht wie für den MNIST-Datensatz ermittelt. Für den relNet-Datensatz liegt die Größe des Zustandsraumes eines Graphen mit Ketten-Struktur bei $\mid \mathcal{X} \mid = 108.895.155$, wofür die zugehörige Matrix nicht gespeichert werden konnte. Für den Reuters-Datensatz war die Größe des Zustandsraumes noch $\mid \mathcal{X} \mid = 25.778.185$, was ebenfalls nicht gespeichert werden konnte. Insgesamt zeigte sich, dass für große Zustandsräume die Implementierung zu langsam war und daher nur 200 Iterationen durchgeführt wurden. Ein Testlauf mit 1.000 Iterationen dauerte für den relNet-Datensatz länger als zwei Wochen. Für diese Parametereinstellung und 200 Iterationen lagen für beide Datensätze Ergebnisse vor, in Tabelle 6.8 sind die Ergebnisse des relNet-Datensatzes dargestellt und in Tabelle 6.9 die des Reuters-Datensatzes.

Durch die Initialisierung mit standardnormalverteilten Gewichten werden zufällige Sätze und Überschriften generiert und es liegen keine Ergebnisse wie bei den ML-optimierten Gewichten vor. Aber bei den MNIST-Daten war schon zu sehen, dass mit dem PAM-GAN keine Verbesserung eintrat, weshalb dies vermutlich auch für die Textgenerierung gilt. Hier fehlt eine sinnvolle Initialisierung der Gewichte, wo nicht nur Daten aus D generiert werden. Ebenfalls ist die Bildung eines Mittelwerts nicht möglich, wodurch an der Qualität der Sätze zu zweifeln ist.

Tabelle 6.8: Ergebnisse des PAM-GAN Trainings für den relNet-Datensatz nach 200 Iterationen.

- everywhere,points,manage,adobe,face,invite,daal,schools,reveal,flash,bottom,christ
- ad,bet,intended,adobe,face,invite,daal,schools,reveal,flash,bottom,christ
- sunni,alarm,kaaba,cell,educated,hes,fine,happens,fitnah,japan,pious,pbuh
- sunni,alarm,kaaba,cell,educated,hes,fine,happens,smells,let,dentist,symbol
- sunni,alarm,kaaba,cell,educated,install,high,talha,hands,loooool,designs,anywhere
- ad,bet,intended,adobe,face,invite,daal,schools,reveal,flash,bottom,christ
- alot,starting,slapped,ki,send,highest,took,victim,vkd,akhy,sources,cheese
- calm,presume,imma,worried,users,ideas,posters,bakr,makkah,everybody,sources,cheese
- working,tend,lightly,step,tovah,ti,daddy,donate,wacky,deny,community,competition
- ad,bet,moderators,century,excuse,traditional,contest,b,worrying,lips,sources,cheese

Tabelle 6.9: Ergebnisse des PAM-GAN Trainings für den Reuters-Datensatz nach 200 Iterationen.

• drug,gain,brighton,bank,carlo,snapchat
• woods,vegas,demand,reiterates,software,blues
• brief-volaris,panel,liberty,2,dec,ruling
• woods,communications,foundation,retreat,french,action
• bid,victims,adjusted,approval,policy,standings
• iraq,kills,rating,minnesota,dec,ruling
• europe's,am,receives,debut,buy-back,johnson
• south,audi,bridge,2,dec,ruling
• house,poll,major,2,dec,ruling
• blood,triggers,start,opposes,communications,$ 1.75

6.4 Diskussion

In den vorherigen Abschnitten wurde die Qualität des PAM-Samplings und des entwickelten PAM-GAN evaluiert. Das PAM-Sampling lieferte, wie bereits in der Literatur zu sehen war, für die Generierung von Bildern gute Ergebnisse für einen Graphen, dessen Gewichte mit ML optimiert wurden. Mit dem implementierten Ansatz von [31] wurden keine neuen Texte generiert und hierfür hat die implementierte Prozedur kein erfolgreiches Ergebnis geliefert.

Zu beachten ist der Unterschied zwischen den unterschiedlichen Graphenstrukturen. Für einen Gitter-Graphen erzeugte das PAM-GAN Ergebnisse, wo die einzelnen Pixel nicht mit einer hohen Sicherheit vorlagen und entsprechend grauer in den Bildern dargestellt sind. Dafür ist der Gitter-Graph in der Lage sein Bild zu variieren, sodass unterschiedliche Ziffern erzeugt werden können. Für einen ELEM-GM-Graphen hingegen konnten einfach gute Ergebnisse erzielt werden, welche die Ziffer gut darstellen. Die Ziffern sind immer zu erkennen und der zugrunde liegende Graph sollte für diskriminative Aufgaben gut geeignet sein und könnte einen alternativen Kritiker bilden.

Das PAM-GAN erzielte in Abhängigkeit von dem Graphen unterschiedlich gute Ergebnisse, aber mit einem Graphen konnten nicht alle Ziffern erzeugt werden. Im direkten Vergleich ist das PAM-GAN den klassischen Verfahren auf Basis neuronaler Netz in der Bildqualität unterlegen und generiert verrauschte Bilder. Mit dem ELEM-GM-Graphen entwickelte sich die Ziffer nicht mehr, da viele Kanten zwischen Knoten mit einer einzigen möglichen Realisierung vorlagen und so schon die Ziffer dargestellt wurde. Fehlten allerdings manche Teile der Ziffer, konnten diese nicht durch das PAM-GAN gefunden werden.

Bei den erzeugten Bildern des Gitter-Graphen hat es eine Auswirkung auf die Qualität, wenn der Datenpunkt als Mittelwert über mehrere generierte Datenpunkte gebildet wird. Dies wurde auch im Training angewandt und in jeder Iteration bekam der Kritiker ein Bild,

für das 5 Bilder mit PAM-Sampling generiert wurden, über die anschließend der Mittelwert gebildet wurde. Hierbei zeigte sich, dass die dargestellten Ziffern nur eine geringe Varianz aufwiesen, aber insgesamt die Sicherheit der Pixel stieg. Aber bei der 8 führte dieser Ansatz dazu, dass die untere Schlaufe ebenfalls weiß blieb und sich nicht mehr änderte.

Bei der 8 wurde mit dem einfach generierten Datenpunkt ein besseres Ergebnis erzielt, so dass die Schlaufen zu erkennen sind. Beim PAM-Sampling zeigte sich, dass die einfachen Bilder eine höhere Varianz aufwiesen und höchstens Teile der Ziffern darstellten, wodurch aber in einer Iteration bestimmte Teile einer Ziffer gesehen wurden und beim Verbessern des Generators sich dann auch entsprechend auswirken konnten. Wenn $\mid \mathcal{X} \mid$ deutlich größer ist als beim MNIST-Datensatz, ist die aktuelle Implementierung des PAM-GAN zu langsam, was ein zusätzliches Problem des Trainings darstellte.

Die Betrachtung der $L2$-Norm der Gradienten zeigte, dass die $L2$-Norm des Kritikers klein wurde, wenn sich das Bild dem Optimum annäherte. Im Gegensatz dazu verschlechterte sich die $L2$-Norm des Generators und streute stärker. Dies ist nur ein erster Eindruck, könnte aber den Punkt anzeigen, an dem das Training stoppen sollte, um ein optimales Ergebnis zu finden.

Auch wenn in fast jeder Iteration der Gradient des Generators ungleich 0 ist, kann der Punkt eintreten, dass sich das Ergebnis nicht ändert. Die Änderungen haben einen direkten Einfluss auf die Pixel. Wenn die Wahrscheinlichkeit eines Pixels sich nicht stark genug ändert, wird dieser seinen Zustand nicht ändern, da er von seiner Nachbarschaft beeinflusst wird. Wenn die Knoten in einer Clique alle weiß sind und das Gewicht für die weißen Pixel steigt, ändert dies das Ergebnis nicht mehr, siehe zum Beispiel die Ziffer 8 im Anhang A.2. Der Kritiker bekommt Bilder des Generators zu sehen, die wenig Konturen enthalten, und lernt entsprechend seine Parameter. Der Generator kann das Ergebnis nicht ändern und es entsteht derselbe Effekt, als ob der Gradient 0 wäre.

Auch wenn die Ergebnisse für den MNIST-Datensatz nicht denen eines GAN entsprechen, das mit neuronalen Netzen gelernt wurde, zeigt sich, dass ein MRF in der Lage ist bei geschickter Initialisierung seiner Gewichte $\boldsymbol{\theta}$ die Verteilung des Trainingsdatensatzes P_{data} zu lernen und zu generieren. Hierbei ist eine geschickte Wahl der Struktur des Graphen wichtig, denn die Änderungen am Ergebnis müssen auch in der Struktur möglich sein. Wenn D keine numerischen Werte enthält, muss der Generator damit umgehen können, das einfache PAM-Sampling kann es nicht.

Kapitel 7

Ausblick

In dieser Arbeit wurde ein GAN entwickelt, das sowohl den Generator als auch den Kritiker durch ein MRF realisiert. Dafür wurden einige Approximationen eingeführt und teilweise wurde auf die gegebenen Funktionalitäten von PX zurückgegriffen, wie der Loopy Belief Propagation. Die Ergebnisse der Experimente sind nicht so gut wie bei dem klassischen GAN, aber es bieten sich noch Möglichkeiten diese zu verbessern.

Einige der Verfahren besitzen Alternativen, an denen man ansetzen könnte, um das PAM-GAN weiterzuentwickeln. Diese sind zum Beispiel das Verfahren, um das MAP zu bestimmen. Im PAM-Sampling wurde nur die LBP eingesetzt. Schon in [31] wurde das MAP-Verfahren anders gelöst. Manche Verfahren können besser mit den verzerrten Gewichten arbeiten, wie zum Beispiel Graph Cuts [8]. Aber ein weiterer Punkt ist die Mittelwertbildung: Anstatt diese mit den erzeugten Datenpunkten durchzuführen, könnte dies direkt auf Basis der Zufallsvariablen geschehen, wie es bei [20] beschrieben wurde. Dafür würden Kopien der Knoten angelegt werden und darüber der Mittelwert gebildet werden, was zu besseren Ergebnissen führen könnte und direkt durch das MAP-Verfahren gelöst würde.

Es zeigte sich, dass die Initialisierung der Gewichte ein entscheidender Faktor war, hier müssten weitere Ansätze zur Initialisierung getestet werden, gerade für den Kritiker. Durch die hohen Gewichte des Kritikers waren die Vorhersagen der Klasse schlecht, was im Training allerdings nur indirekt eine Rolle spielte, da nur ϕ^c und $\boldsymbol{\theta}^c$ genutzt wurden. Aber wenn die Gewichte anders initialisiert würden, könnte dies zu besseren Ergebnissen führen, da der Kritiker näher an der perfekten Vorhersage wäre.

In der FDM soll das gewählte h eigentlich möglichst klein gewählt werden, um den Fehler gering zu halten [28]. Wenn dies erfolgt, könnte anstatt

$$\frac{\phi^g\left(x'\right) - \phi^g\left(x''\right)}{h}$$

auch die zentrierte Form (7.1) genutzt werden:

$$\frac{\phi^g\left(x+h\right) - \phi^g\left(x-h\right)}{2h}. \tag{7.1}$$

Die FDM mit der zentrierten Form hat einen geringeren Fehler als die verwendete Formel. Hier ist zu beachten, dass h quadratisch in den Fehler einfließt, daher würde ein großer Wert in h das Resultat stärker abweichen lassen. Wie weit die beiden Graphen sich unterscheiden und daher den Term beeinflussen, müsste evaluiert und bewertet werden.

Das vorgestellte GAN minimiert die JS-Divergenz. Eine Alternative wäre es eine andere Distanz zu minimieren, wie die Wasserstein-1 Distanz [3]. Dafür müsste der Prozess nur minimal angepasst werden und es sollte kein Mode-Collapse mehr auftreten können. Allerdings schränkt diese Prozedur den Kritiker bei seinen Änderungen ein, was evaluiert werden müsste.

Als Fazit dieser Arbeit kann festgehalten werden, dass das PAM-GAN weniger gute Ergebnisse lieferte als die Ansätze mit neuronalen Netzen. Aber die Ergebnisse zeigen, dass die Parameter des MRF sich verändern und bis zu einer Iteration, die in Abhängigkeit des betrachteten Datensatzes steht, besser werden. Mit den vorgeschlagenen Änderungen können hoffentlich Verbesserungen des Ergebnisses erzielt werden.

Literatur

[1] Christophe Andrieu u. a. “An Introduction to MCMC for Machine Learning”. In: *Machine Learning* 50.1 (Jan. 2003), S. 5–43.

[2] Martín Arjovsky und Léon Bottou. “Towards Principled Methods for Training Generative Adversarial Networks”. In: *CoRR* abs/1701.04862 (2017). URL: http://arxiv.org/abs/1701.04862.

[3] Martín Arjovsky, Soumith Chintala und Léon Bottou. “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, S. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html.

[4] Sanjeev Arora u. a. “Generalization and Equilibrium in Generative Adversarial Nets (GANs)”. In: *CoRR* abs/1703.00573 (2017). [Abgerufen am 30.11.2017]. URL: http://arxiv.org/abs/1703.00573.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[6] Léon Bottou. “Large-Scale Machine Learning with Stochastic Gradient Descent”. In: *Procs. of the 19th International Conference on Computational Statistics*. Hrsg. von Yves Lechevallier und Gilbert Saporta. Paris, France: Springer, 2010, S. 177–187.

[7] Stephen Boyd und Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[8] Yuri Boykov, Olga Veksler und Ramin Zabih. “Fast Approximate Energy Minimization via Graph Cuts”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 23.11 (Nov. 2001), S. 1222–1239. URL: https://doi.org/10.1109/34.969114.

[9] Thomas H. Cormen u. a. *Introduction to Algorithms*. 3. Aufl. McGraw-Hill Higher Education, 2001.

[10] James Coughlan. “A Tutorial Introduction to Belief Propagation”. [Abgerufen am 12.03.2018]. 2009. URL: http://computerrobotvision.org/2009/tutorial_day/crv09_belief_propagation_v1.pdf.

[11] Richard O. Duda, Peter E. Hart und David G. Stork. *Pattern Classification (2nd Ed)*. Wiley, 2001.

[12] Hamid Eghbal-zadeh und Gerhard Widmer. "Probabilistic Generative Adversarial Networks". In: *CoRR* abs/1708.01886 (2017). [Abgerufen am 30.11.2017]. URL: http://arxiv.org/abs/1708.01886.

[13] Andrew Gelman u. a. *Bayesian Data Analysis*. 3. Aufl. CRC Press, 2013.

[14] Stuart Geman und Donald Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *Readings in Uncertain Reasoning*. Hrsg. von Glenn Shafer und Judea Pearl. San Mateo CA: Morgan Kaufmann, 1990, S. 452–472.

[15] Ian J. Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. [Abgerufen am 5.1.2018]. 2017. URL: http://arxiv.org/abs/1701.00160.

[16] Ian Goodfellow u. a. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Hrsg. von Z. Ghahramani u. a. Curran Associates, Inc., 2014, S. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[17] John M. Hammersley und Peter Clifford. *Markov fields on finite graphs and lattices*. [Abgerufen am 28.4.2018]. 1971. URL: http://www.statslab.cam.ac.uk/~grg/books/hammfest/hamm-cliff.pdf.

[18] Joachim Hartung, Bärbel Elpelt und Karl-Heinz Klösner. *Statistik. Lehr- und Handbuch der angewandten Statistik*. 15. Aufl. Oldenbourg, 2009.

[19] Trevor Hastie, Robert Tibshirani und Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2017.

[20] Tamir Hazan und Tommi Jaakkola. "On the Partition Function and Random Maximum A-Posteriori Perturbations". In: *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*. Hrsg. von John Langford und Joelle Pineau. ICML '12. New York, NY, USA: Omnipress, Juli 2012, S. 991–998. ISBN: 978-1-4503-1285-1.

[21] Assembling Intelligence. *Generative Adversarial Nets*. [Abgerufen am 5.1.2018]. 2017. URL: https://assemblingintelligence.wordpress.com/2017/05/02/generative-adversarial-nets/.

[22] Adam Kalai und Santosh Vempala. "Efficient algorithms for online decision problems". In: *Journal of Computer and System Sciences* 71.3 (2005), S. 291–307.

[23] Daphne Koller und Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[24] Rohit Kulkarni. *Global News-Wire Archive, 2007-2018*. [Abgerufen am 1.3.2018]. 2017. URL: https://www.kaggle.com/therohk/reuters-news-wire-archive/version/4.

[25] Solomon Kullback und Richard A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (1951), S. 79–86.

[26] Dima Kuzmin und Manfred K. Warmuth. "Optimum Follow the Leader Algorithm". In: *Proceedings of the 18th Annual Conference on Learning Theory*. Hrsg. von Peter Auer und Ron Meir. COLT'05. Berlin, Heidelberg: Springer-Verlag, 2005, S. 684–686.

[27] Yann LeCun und Corinna Cortes. *MNIST handwritten digit database*. [Abgerufen am 5.1.2018]. 2010. URL: http://yann.lecun.com/exdb/mnist/.

[28] Randall LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial und Applied Mathematics, 2007.

[29] Kevin P. Murphy, Yair Weiss und Michael I. Jordan. "Loopy belief propagation for approximate inference: An empirical study". In: *UAI'99 Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, S. 467–475.

[30] Yuri Nesterov. "Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems". In: *SIAM Journal on Optimization* 22.2 (2012), S. 341–362.

[31] George Papandreou und Alan L. Yuille. "Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models". In: *2011 IEEE International Conference on Computer Vision (ICCV)*. 2011, S. 193–200.

[32] Lukas Pfahler u. a. "Learning Low-Rank Document Embeddings with Weighted Nuclear Norm Regularization". In: *Proceedings of the 4th IEEE International Conference on Data Science and Advanced Analytics*. 2017. URL: https://doi.org/10.1109/dsaa.2017.46.

[33] Nico Piatkowski. *PX*. Abgerufen am 16.11.2017. 2018. URL: http://sfb876.tu-dortmund.de/px.

[34] Scott Rome. *An Annotated Proof of Generative Adversarial Networks with Implementation Notes*. [Abgerufen am 20.12.2017]. 2017. URL: http://srome.github.io/An-Annotated-Proof-of-Generative-Adversarial-Networks-with-Implementation-Notes/.

[35] Lucas Theis, Aäron van den Oord und Matthias Bethge. "A note on the evaluation of generative models". In: *International Conference on Learning Representations*. [Abgerufen am 2.4.2018]. Apr. 2016. URL: http://arxiv.org/abs/1511.01844.

[36] Martin J. Wainwright, Tommi S. Jaakkola und Alan S. Willsky. “MAP estimation via agreement on trees: message-passing and linear programming”. In: *IEEE Transactions on Information Theory* 51.11 (2005), S. 3697–3717.

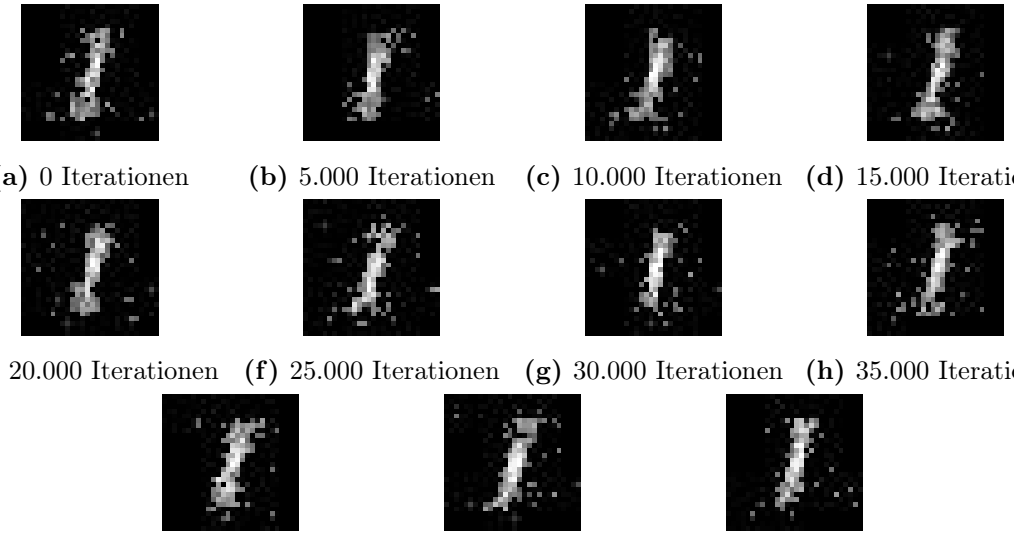
[37] Martin J. Wainwright und Michael I. Jordan. “Graphical Models, Exponential Families, and Variational Inference”. In: *Foundations and Trends in Machine Learning* 1.1–2 (2008), S. 1–305. URL: http://dx.doi.org/10.1561/2200000001.

[38] Eunho Yang, Aurelie C. Lozano und Pradeep Ravikumar. “Elementary Estimators for Graphical Models”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Dez. 2014, S. 2159–2167. URL: http://papers.nips.cc/paper/5318-elementary-estimators-for-graphical-models.

Anhang A

Ergebnisse

A.1 MNIST Ergebnisse - Gitter-Graph Einzelner PAM Aufruf



(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen

(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung A.1: Entwicklung der Ziffer 1 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

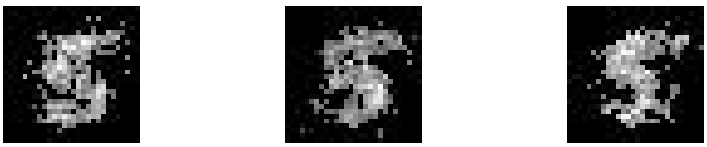
(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen

(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung A.2: Entwicklung der Ziffer 3 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

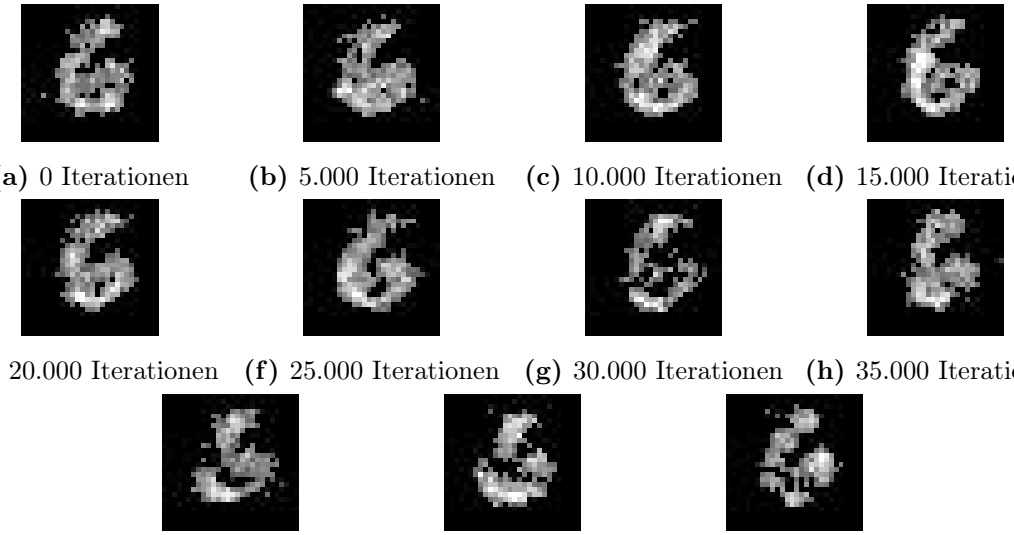
(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen



(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung A.3: Entwicklung der Ziffer 5 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

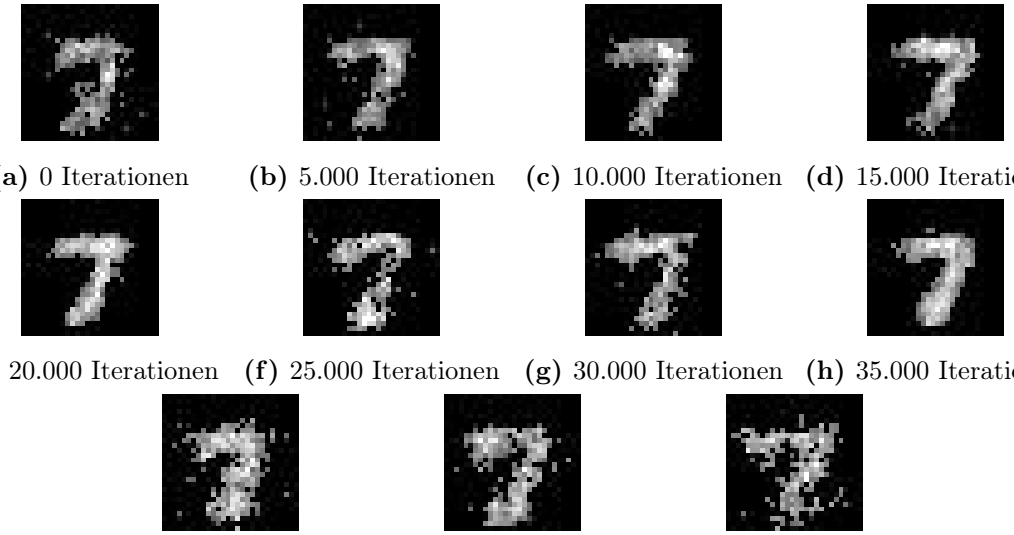


(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen

(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung A.4: Entwicklung der Ziffer 6 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.



(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

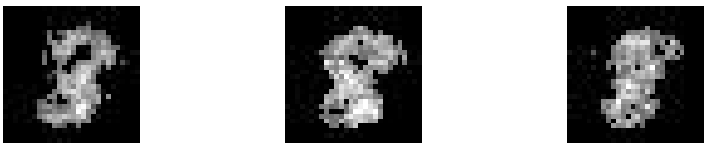
(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen

(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung A.5: Entwicklung der Ziffer 7 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen



(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung A.6: Entwicklung der Ziffer 8 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

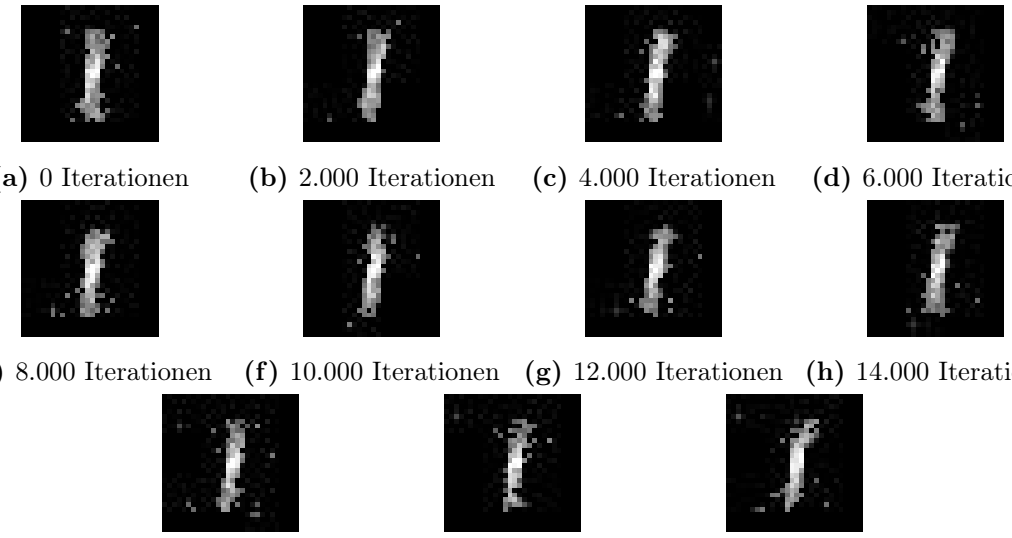
(a) 0 Iterationen **(b)** 5.000 Iterationen **(c)** 10.000 Iterationen **(d)** 15.000 Iterationen

(e) 20.000 Iterationen **(f)** 25.000 Iterationen **(g)** 30.000 Iterationen **(h)** 35.000 Iterationen

(i) 40.000 Iterationen **(j)** 45.000 Iterationen **(k)** 50.000 Iterationen

Abbildung A.7: Entwicklung der Ziffer 8 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

A.2 MNIST Ergebnisse - Gitter-Graph Mittelwerte im Training des Kritikers

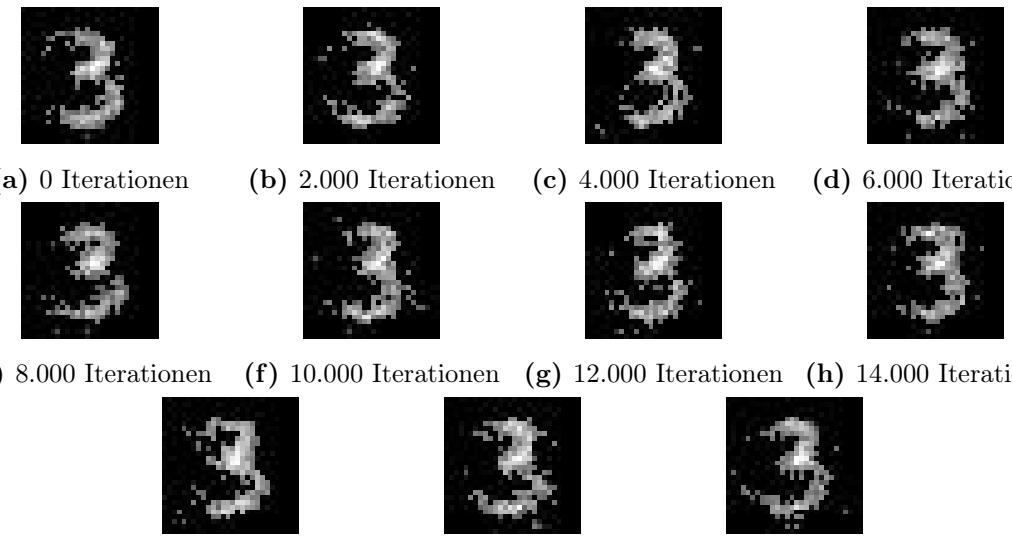


(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen

(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.8: Entwicklung der Ziffer 1 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.



(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

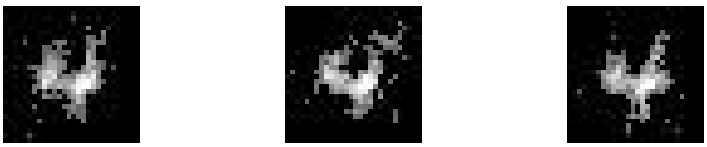
(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen

(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.9: Entwicklung der Ziffer 3 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen

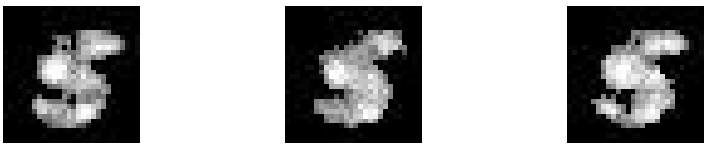


(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.10: Entwicklung der Ziffer 4 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

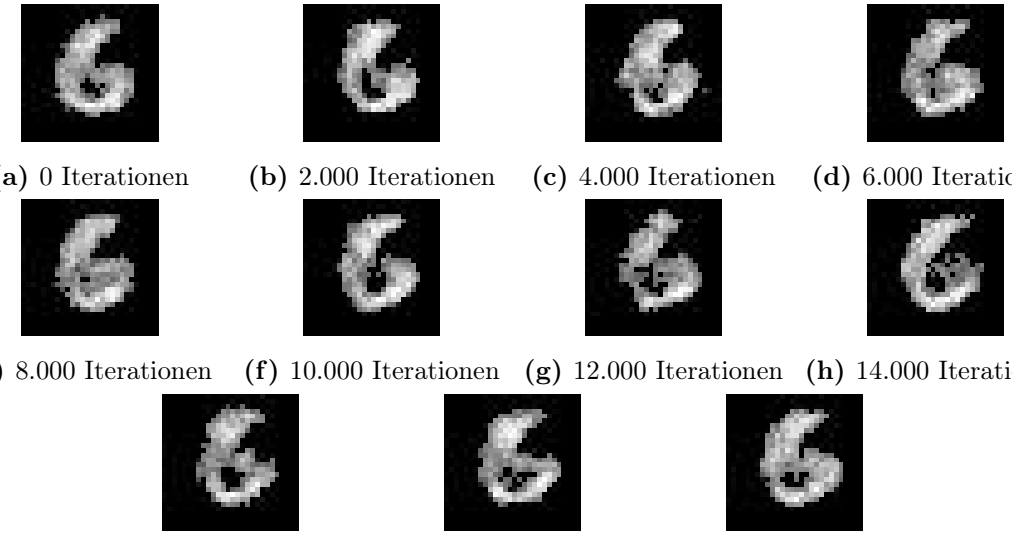
(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen



(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.11: Entwicklung der Ziffer 5 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

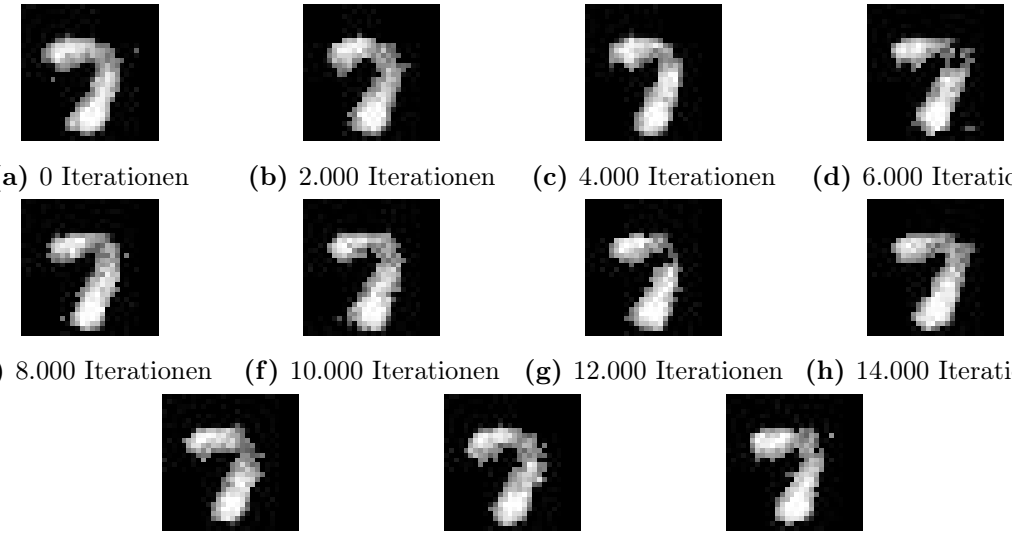


(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen

(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.12: Entwicklung der Ziffer 6 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.



(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

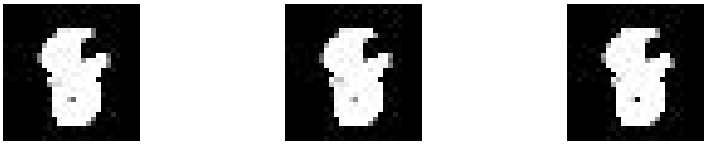
(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen

(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.13: Entwicklung der Ziffer 7 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen



(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.14: Entwicklung der Ziffer 8 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.

(a) 0 Iterationen **(b)** 2.000 Iterationen **(c)** 4.000 Iterationen **(d)** 6.000 Iterationen

(e) 8.000 Iterationen **(f)** 10.000 Iterationen **(g)** 12.000 Iterationen **(h)** 14.000 Iterationen

(i) 16.000 Iterationen **(j)** 18.000 Iterationen **(k)** 20.000 Iterationen

Abbildung A.15: Entwicklung der Ziffer 9 durch PAM-GAN. Das Training lief 50.000 Iterationen und begann mit $\boldsymbol{\theta}^g$ die durch ML optimiert wurden.