Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Quantenunterstütztes Clustering mit hybriden neuronalen Netzen

Philipp Wundrack

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h.c. Frank Leymann
Betreuer/in:	Dr. phil. Johanna Barzen Benjamin Weder, M.Sc.
Beginn am:	12. November 2020
Beendet am:	12. Mai 2021

Kurzfassung

Maschinelles Lernen und Quantencomputer sind zwei aktuelle Forschungsthemen, die großes Potenzial haben. Aktuell wird erforscht, wie diese beiden Gebiete kombiniert werden können, um voneinander zu profitieren. In diesen Bereich fällt die vorliegende Arbeit. In dieser Arbeit wird untersucht, ob hybride neuronale Netze genutzt werden können, um die Ergebnisse von Clustering-Algorithmen zu verbessern. Hierzu wird auf den Daten Dimensionsreduktion mit hybriden Autoencodern durchgeführt, bevor die Daten den Clustering-Algorithmen übergeben werden. Als Ergebnis konnte festgestellt werden, dass für bestimmte Datensätze Clustering-Algorithmen bessere Cluster erstellen können, wenn Dimensionsreduktion mit hybriden Autoencodern durchgeführt wurde, anstatt mit klassischen Autoencodern oder PCA.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Motivation

Immer mehr Systeme werden durch Computer gesteuert, die mehr und mehr Daten generieren und speichern [Alp16]. Diese Daten möchte man nutzen. *Künstliche Intelligenz* wird als eine zentrale Zukunftstechnologie gesehen, in die Firmen investieren müssen, um bestehen zu können [SP18]. Zur künstlichen Intelligenz zählen alle Aufgaben, in denen ein Computer Entscheidungen trifft [Ser21]. Eine Art von künstlicher Intelligenz ist das *maschinelle Lernen* [Ser21]. Dabei geht es darum, dass ein Computer Entscheidungen trifft, basierend auf Daten [Ser21]. Früher mussten Programmierer dem Computer "beibringen", wie er bestimmte Aufgaben lösen kann. Mit maschinellem Lernen können die wachsenden Datenmengen genutzt werden, damit der Computer selber "lernt", wie er die Aufgaben lösen kann. Maschinelles Lernen ist ein wichtiges Werkzeug geworden für unterschiedliche Bereiche [For19] mit vielen aktiven Forschungsthemen [GBC16].

Eine häufig angewandte Art des maschinellen Lernens ist *Clustering*, welches im Besonderen dann eingesetzt werden kann, wenn für einen Datensatz nicht bekannt ist, welche Datenpunkte zu welchen Klassen gehören [Bra20]. Dabei werden automatisch Cluster, also Gruppen, von ähnlichen Dingen gebildet [Har12]. Die einzelnen Datenpunkte werden Klassen zugeordnet, ähnlich wie bei der Klassifikation. Allerdings werden die Klassen, anders als bei der Klassifikation automatisch bestimmt, da diese nicht im Datensatz vorgegeben sein müssen. Clustering kann z. B. verwendet werden, um einen Datensatz zusammenzufassen, um Strukturen im Datensatz zu finden [For19] oder um bisher unbekannte Klassen zu identifizieren [Bra20].

Eine weitere Zukunftstechnologie, die gerade Realität wird, ist die des *Quantencomputers*. Quantencomputer sind eine neue Art von Computer, die Quantenmechanik nutzt, um Informationen zu speichern und zu verarbeiten [RP11]. Da Quantencomputer bestimmte Aufgaben schneller lösen als klassische Computer, gilt es zu untersuchen, ob sie auch beim maschinellen Lernen Vorteile erzielen können [SP18]. Da Quantencomputer Zustände und damit auch Muster erzeugen können, die mit klassischen Computern zu aufwendig wären in der Berechnung kann man erwarten, dass Quantencomputer auch beim maschinellen Lernen Muster erkennen, die klassische Computer nur mit enormem Aufwand erkennen würden [BWP+17; ROA17]. Das könnte sich in einer höheren Genauigkeit bemerkbar machen. Da aktuelle Quantencomputer noch stark verrauscht sind und sie weniger als 100 Qubits zur Verfügung haben [LC20], wird meist nur ein Teil der Berechnungen auf einem Quantencomputer ausgeführt und der Rest auf einem klassischen Computer [WSS+19]. Verfahren, die diesen Ansatz verfolgen, werden als hybrid bezeichnet [LB20].

In dieser Arbeit soll untersucht werden, ob Clustering-Algorithmen durch den Einsatz von hybriden neuronalen Netzen bessere Cluster erzeugen können. Hybride neuronale Netze sind Kombinationen aus klassischen neuronalen Netzen und Quantennetzen. Sie werden hier in der Form von *Autoencodern* genutzt. Autoencoder sind eine Art von neuronalem Netz, das genutzt werden kann für Dimensionsreduktion. Diese Dimensionsreduktion soll auf den Daten angewandt werden, bevor diese für Clusteringverfahren verwendet werden.

Zuerst wird in Kapitel 2 eingeführt, was maschinelles Lernen ist, wofür es verwendet werden kann und was für Arten es gibt. Außerdem werden die Themen, die für die späteren Experimente wichtig sind, erklärt. In Kapitel 3 werden dann die fundamentalen Konzepte der Quantencomputer und ihre Besonderheiten beschrieben. Daraufhin wird in Kapitel 4 erläutert, wie Quantencomputer für maschinelles Lernen genutzt werden können. Kapitel 5 fokussiert sich auf die Durchführung von 5 Experimenten. Hierbei wird überprüft, ob hybride Autoencoder geeignet sind, Dimensionsreduktion als Vorverarbeitungsschritt für Clustering-Algorithmen durchzuführen. Das letzte Kapitel gibt eine Zusammenfassung und einen Ausblick auf Fragen, die in weiteren Forschungsarbeiten geklärt werden können.

2 Maschinelles Lernen

Beim maschinellen Lernen geht es um das automatisierte Lernen aus Daten [Alp16; CANS21; GBC16; Har12; Ser21]. Das Wissen soll nicht formal durch Menschen spezifiziert werden, wie es bei der Entwicklung herkömmlicher Programme der Fall ist [Alp16; GBC16]. Stattdessen soll ein Programm Wissen in Form von Mustern aus rohen Daten extrahieren [Alp16; GBC16]. Ziel ist es, die menschliche Leistung in der jeweiligen Aufgabe möglichst genau zu erreichen [Bra20]. Allerdings ist es eine große Herausforderung, das formlose Wissen, das in den Daten enthalten ist, zu extrahieren [GBC16].

Einige Anwendungsbeispiele für maschinelles Lernen sind:

- Objekterkennung in Bildern [RDGF16; ZZXW19]
- Spracherkennung [AAA+16]
- Spam-Mails erkennen [CKP+15]
- Robotik [Pet08]
- Verarbeitung natürlicher Sprache [Ols09]
- Handschrifterkennung [BSOM05]

Eine Art von maschinellem Lernen ist *Supervised Learning*. Hierbei bestehen die Daten aus Vektoren mit Messungen, die sogenannten *Feature Vectors*, und Zielwerte, die vorhergesagt werden sollen [Alp16; Bra20; CANS21; Har12]. Zum Beispiel kann ein *Feature Vector* Eigenschaften eines Autos enthalten (Höchstgeschwindigkeit, PS, Ausstattung etc.) und der Zielwert kann der Verkaufspreis sein. Da der Zielwert in diesem Fall ein numerischer Wert ist, spricht man von Regression [Alp16; Har12]. Ist der Zielwert stattdessen ein nominaler Wert (z.B. Hersteller des Autos), so spricht man von Klassifikation [CANS21; Har12].

Eine andere Art von maschinellem Lernen ist *Unsupervised Learning*. Hierbei sind in den Daten keine Zielwerte enthalten, stattdessen wird das Problem alleine durch die Verteilung der Daten spezifiziert [Bra20]. Eine Schwierigkeit hierbei ist, dass nicht offensichtlich ist, wie das Ergebnis des maschinellen Lernens beurteilt werden kann, da man es nicht mit einem Zielwert vergleichen kann [Bra20]. Eine Art von *Unsupervised Learning* ist Dimensionsreduktion. Hierbei wird die Anzahl der Werte pro *Feature Vector* reduziert, damit diese von anderen Algorithmen leichter verarbeitet werden können [Bra20; Har12]. Eine andere Art ist Clustering. Dabei werden die Datenpunkte (z.B. Autos) aus den Daten in Gruppen (Cluster) aufgeteilt [Bra20; Har12].

2.1 Terminologie

Im Bereich des maschinellen Lernens werden viele Fachbegriffe verwendet. Die wichtigsten, die in dieser Arbeit vorkommen, sollen hier kurz definiert werden.

- Ein *Attribut* ist eine relevante Information, z. B. das Baujahr eines Autos [GBC16].
- Ein *Datenpunkt* ist eine Menge an Attributen, die z. B. zu einem Objekt gehören, z. B. alle Attribute, die über ein bestimmtes Auto bekannt sind [Alp16; Har12].
- Ein *Feature Vector* ist die Darstellung eines Datenpunktes als Vektor.
- Ein *Datensatz* ist eine Menge an Datenpunkt, z. B. eine Tabelle mit Attributen von vielen Autos, wobei jede Zeile einem Auto entspricht.
- Ein *Modell* ist eine Menge an Regeln, mit deren Hilfe man Vorhersagen treffen kann, z. B. eine Funktion, die als Eingabe die Attribute eines Autos bekommt und als Ausgabe den geschätzten Verkaufspreis vorhersagt [Ser21].
- Die *Parameter* eines Modells sind Werte, die das Verhalten des Modells bestimmen und verändert werden können.
- Die *Objective Function* ist eine Funktion, die angibt, wie gut ein Ergebnis ist.
- Das *Training* ist ein repetitiver und inkrementeller Prozess, bei dem die Parameter eines Modells optimiert werden bezüglich einer *Objective Function* [Alp16; CANS21].
- Der *Trainingsdatensatz* ist der Teil des Datensatzes, der für das Training des Modells verwendet wird [CANS21; Har12].
- Der *Testdatensatz* ist der Teil des Datensatzes, der für das Testen des Modells verwendet wird. Er darf keine Schnittmenge mit dem Trainingsdatensatz haben [Har12]
- Ein *Hyperparameter* ist kein Parameter, der beim Training optimiert wird, sondern er muss vom Anwender spezifiziert werden. Dazu gehört z. B. welches Modell genutzt wird, welcher Optimierer genutzt wird, welche Lernrate der Optimierer nutzt, etc.

2.2 Clustering

Clustering ist eine Art von *Unsupervised Learning*. Daher müssen für den verwendeten Datensatz keine Zielwerte bekannt sein [Bra20]. Diese Technik wird genutzt, um die Datenpunkte aus dem Datensatz in Gruppen (Cluster) von ähnlichen Dingen aufzuteilen [Bra20; Har12]. Da keine Zielwerte vorhanden sein müssen, ist es schwierig, ein Maß anzugeben, wie gut die gefundenen Cluster sind [For19]. Üblicherweise werden die Cluster dahingehend bewertet, wie gut sie für den jeweiligen Anwendungsfall sind [For19].

2.2.1 Algorithmen

In diesem Abschnitt werden mehrere Clustering Algorithmen vorgestellt. Diese Algorithmen wurden ausgewählt aufgrund ihrer Popularität oder Eigenschaften.

k-means

K-means ist einer der bekanntesten Clustering Algorithmen und wird häufig eingesetzt. Bei diesem Algorithmus muss der Anwender selber festlegen, in wie viele Cluster die Daten aufgeteilt werden sollen [Bra20; Mac+67]. Es wird die Annahme getroffen, dass die Cluster kugelförmig sind und es sollen die Mittelpunkte der Cluster gefunden werden [For19; Har12]. Die Datenpunkte werden dann dem Cluster zugewiesen, dessen Mittelpunkt am nächsten ist [For19]. Das Ergebnis ist aber im Allgemeinen nicht die optimale Lösung [Mac+67]. Um trotzdem eine gute Lösung zu finden, kann k-means mehrfach ausgeführt werden mit unterschiedlichen zufälligen Startpunkten [Bra20].

DBSCAN

Bei DBSCAN gibt es, anders als bei k-means, nicht die Einschränkung, dass die Cluster kugelförmig sein müssen. Es muss aber wie bei k-means nur ein Parameter angegeben werden, allerdings nicht die Anzahl der Cluster, sondern ϵ [EKX96]. Die Anzahl der Cluster bestimmt DBSCAN selber. Es ist ein Dichte-basierter Algorithmus, daher sind Cluster, Regionen, die eine höhere Dichte haben als ihre Umgebung. Ab welcher Dichte eine Region als Cluster infrage kommt, wird durch den ϵ Parameter festgelegt. Da es nur auf die Dichte ankommt, können die Cluster beliebige Formen haben. Eine weitere Eigenschaft von DBSCAN ist, dass es eine gute Effizienz hat, selbst bei großen Datensätzen.

OPTICS

OPTICS erzeugt nicht direkt Cluster, sondern eine Reihenfolge der Datenpunkte, aus der sich entsprechende dichtebasierte Cluster erzeugen lassen [ABK99]. Der Algorithmus lässt sich als eine Erweiterung von DBSCAN betrachten, aber anders als bei DBSCAN wird kein Parameter benötigt. Die Ausgabe von OPTICS enthält genügend Informationen, um Cluster für jedes beliebige ϵ effizient zu generieren.

Affinity Propagation

Bei Affinity Propagation muss ebenfalls die Anzahl der Cluster nicht angegeben werden. Als Eingabe werden Ähnlichkeitswerte für jedes Paar von Datenpunkten benötigt [FD07]. Zwischen den Datenpunkten werden Nachrichten ausgetauscht mit reellen Werten, bis gute *Exemplare* und zugehörige Cluster gefunden wurden. Ein *Exemplar* ist dabei ein Datenpunkt aus dem Datensatz, der den Mittelpunkt eines Clusters repräsentiert. Die Anzahl der Cluster, die gefunden werden sollen, müssen nicht angegeben werden, aber der Anwender kann die Datenpunkte unterschiedlich stark gewichten und damit beeinflussen, welche Datenpunkte bevorzugt als Exemplare gewählt werden.

Agglomerative Clustering

Agglomerative Clustering hat die Besonderheit, dass eine hierarchische Struktur von Clustern erzeugt wird, aus der der Anwender schließlich die geeigneten Cluster auswählen kann [For19]. Es wird zuerst jeder Datenpunkt als eigenes Cluster betrachtet und dann mit anderen Clustern verschmolzen, bis gute Cluster entstanden sind. Eine Schwierigkeit dabei ist, dass man eine gute Methode benötigt, die Distanz zwischen Clustern zu messen. Während des Verfahrens werden immer die zwei Cluster, die die geringste Distanz zueinander haben, verschmolzen.

2.2.2 Bewertung von Clustering-Ergebnissen

Möchte man bestimmen, wie gut die Ergebnisse eines Clustering-Algorithmus sind, gibt es einige Maße, die genutzt werden können [SPG+17]. Wenn man weiß, welche Cluster man als Ergebnis erwartet, kann man externe Qualitätsmaße verwenden. Zu diesen zählt z. B. der Rand Index [Ran71] und Mutual Information [SPG+17]. Im Folgenden werden mit C und C' die zwei Clustering-Ergebnisse bezeichnet, die man vergleichen möchte.

Rand Index

Der Rand Index [Ran71] betrachtet alle Paare von Datenpunkten und ob sich die beiden Datenpunkte im gleichen Cluster befinden [SPG+17]. Wenn zwei Datenpunkte in C im gleichen Cluster liegen, dann sollten sie auch in C' im gleichen Cluster liegen. Definiert ist der Rand Index folgendermaßen [SPG+17]:

$$\mathcal{R}(C, C') = \frac{2(n_{11} + n_{00})}{n(n-1)} \tag{2.1}$$

Hierbei ist n die Gesamtzahl von Datenpunkten, n_{11} die Anzahl der Datenpunkt-Paare, die in beiden Clusterings im gleichen Cluster liegen und n_{00} die Anzahl der Datenpunkt-Paare, die in beiden Clusterings in unterschiedlichen Cluster liegen [SPG+17].

$\mathcal{R}$ kann Werte zwischen 0 und 1 annehmen. 0 bedeutet, dass sich beide Clusterings vollständig unterscheiden und 1 bedeutet, dass beide Clusterings identisch sind [SPG+17; WW07]. Der Wert hängt auch von der Anzahl der Cluster und Datenpunkten ab, was unerwünscht ist [SPG+17].

Mutual Information

Mutual Information, zu Deutsch, *Transinformation* oder *gegenseitige Information*, stammt aus der Informationstheorie und basiert auf der Entropie [WW07]. Es hat nicht den Nachteil, den der Rand Index hat und ist damit nicht sensibel bezüglich der Anzahl der Cluster oder Datenpunkte. Das Ergebnis der Formel beschreibt, wie viele Bit weniger nötig sind, um C' zu kodieren, wenn man C kennt [VEB10]. Es ist definiert als [WW07]:

$$\mathcal{I}(C, C') = \sum_{i=1}^{k} \sum_{j=1}^{l} P(i, j) \log_2 \frac{P(i, j)}{P(i)P(j)} \tag{2.2}$$

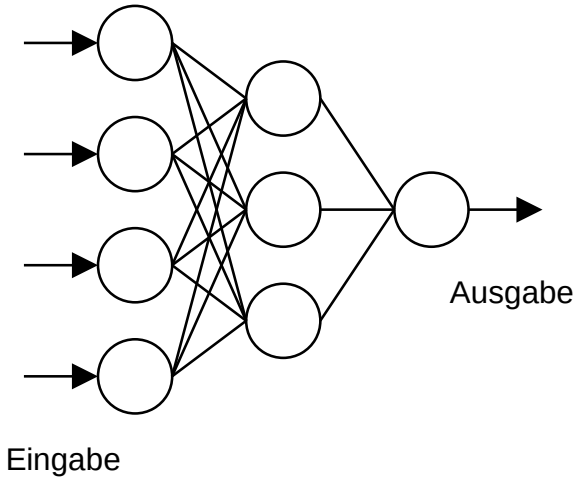

Abbildung 2.1: Ein neuronales Netz. Die Neuronen werden durch Kreise dargestellt. Die Verbindungen zwischen den Neuronen zeigen, welche Neuronen an welche anderen Neuronen Daten weitergeben. Eine Spalte Neuronen stellen eine Schicht dar. Darstellung inspiriert durch [For19].

mit

$$P(i,j) = \frac{|C_i \cap C'_j|}{n} \tag{2.3}$$

und

$$P(i) = \frac{|C_i|}{n} \tag{2.4}$$

wobei C_i das i-te Cluster in C ist, C'_j das j-te Cluster in C' und n die Anzahl der Datenpunkte.

I hat keinen konstanten Wert als Oberschranke, aber es existieren normalisierte Varianten von *Mutual Information*, die den Wert auf einen Bereich zwischen 0 und 1 beschränken [WW07].

2.3 Neuronale Netze

Neuronale Netze sind eine der meist genutzten Arten von maschinellem Lernen, die in vielen Anwendungsbereichen zur Zeit die besten Ergebnisse liefern [Ser21].

Neuronale Netze bestehen aus einzelnen Einheiten, den Neuronen [For19]. Jedes Neuron hat eine Menge an Eingaben und Parametern und produziert mit einer nicht-linearen Funktion eine Ausgabe. Neuronen sind in Schichten angeordnet und die Neuronen einer Schicht sind meist nur mit Neuronen der nächsten Schicht verbunden [For19], es existieren allerdings auch andere Varianten [Ser21]. Eine Menge von Schichten bildet das neuronale Netz [For19]. In Abbildung 2.1 ist ein neuronales Netz mit zwei Schichten (die Eingabeschicht wird nicht mitgezählt), vier Eingabe-Neuronen und einem Ausgabe-Neuron dargestellt.

Während des Trainings werden die Parameter so angepasst, dass das neuronale Netz die gewünschten Werte ausgibt [For19]. Dies geschieht meist durch Gradient Descent. Hierfür muss eine *Objective Function* angegeben werden, die bestimmt, wie gut die aktuellen Ausgaben im Vergleich zu den Zielwerten sind. Das neuronale Netz wird zusammen mit der *Objective Function* als eine einzige mathematische Funktion betrachtet und für diese wird der Gradient berechnet bezüglich der Parameter. Mithilfe des Gradienten können dann die Parameter so angepasst werden, dass der Fehler minimiert wird. Wie das im funktioniert, wird in Abschnitt 2.4 beschrieben.

2.4 Optimierer

Bei der Optimierung geht es um das Minimieren einer *Objective Function*, damit das Modell, welches optimiert wird, die gewünschten Werte ausgibt [SW18]. Zur Minimierung kann ein Optimierer die Parameter eines Modells (z. B. ein neuronales Netz) anpassen, bis ein gutes Ergebnis erreicht wurde [SW18]. Eine *Objective Function* kann z. B. der quadratische Fehler sein, zwischen der Ausgabe eines neuronalen Netzes und einem Zielwert [Kri06].

2.4.1 Gradientenbasiert

Gradientenbasierte Optimierer verwenden den Gradienten der *Objective Function*, bezüglich der Parameter des Modells [Kri06]. Sie werden oft verwendet, da sie in der Praxis geringere Rechenleistung benötigen als andere Verfahren [SW18]. Diese Verfahren haben allerdings auch mehrere Nachteile. Zum einen bleiben sie oft in einem lokalen Minimum hängen, erreichen also nicht die optimale Lösung. Zum anderen können flache Plateaus in der *Objective Function* die Optimierung stark verlangsamen. Diese Nachteile lassen sich aber in der Praxis oft umgehen [SW18]. Um den Gradienten zu bestimmen, werden im folgenden zwei Möglichkeiten vorgestellt. Backpropagation und die Finite-Differenzen-Methode.

Backpropagation

Backpropagation ist ein effizientes Verfahren [LBOM98], mit dem sich der Gradient eines neuronalen Netzes berechnen lässt, wenn dieses differenzierbar ist [Kri06]. Die Schichten des neuronalen Netzes werden dabei als eigenständige Funktionen betrachtet, die die Ausgaben von anderen Schichten als Eingabe bekommen können [For19; LBOM98]. Die Gradienten dieser verschachtelten Funktionen lassen sich mithilfe von der Kettenregel bestimmen [For19; Kri06]. Die Zwischenergebnisse hiervon können für die einzelnen Funktionen (Schichten) wiederverwenden werden, wodurch der Algorithmus effizient wird [For19].

Finite-Differenzen-Methode

Die Finite-Differenzen-Methode ist ein Verfahren, mit dem sich die Ableitung einer Funktion approximieren lässt [LeV98]. Dafür ist es lediglich nötig, die Funktion an unterschiedlichen Stellen auswerten zu können. Dies ist wichtig, wenn sich die Ableitung nicht analytisch berechnen lässt. Es gibt verschiedene Varianten dieser Methode, die unterschiedlich genau sind und unterschiedlich

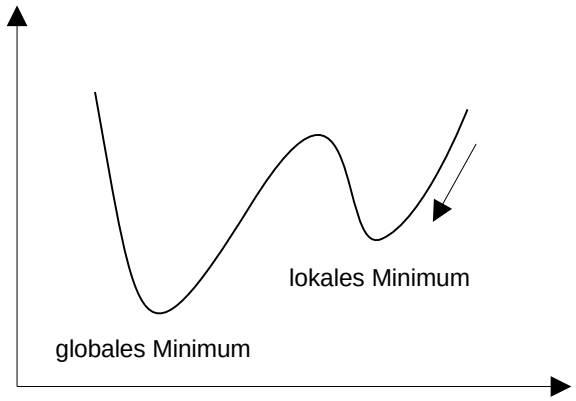

Abbildung 2.2: Darstellung einer Funktion, die minimiert werden soll. Auf der y-Achse wird der Wert der Funktion dargestellt und auf der x-Achse der Wert eines Parameters. Es gibt ein globales Minimum und ein lokales Minimum. Der Startpunkt liegt ganz rechts und durch *Gradient Descent* wird der Parameter angepasst, sodass man im lokalen Minimum landet. Dies wird durch den Pfeil angedeutet. Darstellung inspiriert durch [SP18].

viele Funktionsauswertungen benötigen. Eine der einfachsten Varianten ist folgendermaßen definiert [LeV98]:

$$D_+u(x) = \frac{u(x+h) - u(x)}{h} \tag{2.5}$$

Hierbei ist u die Funktion, deren Ableitung man approximieren möchte, D_+u ist die approximierte Ableitung, x ist die Stelle, an der man ableiten möchte und h ist ein kleiner Wert, der bestimmt, wie genau die Approximation ist [LeV98]. Der Fehler dieser Approximation ist ungefähr proportional zu h. Diese Methode lässt sich natürlich nicht nur verwenden, um eine Ableitung zu approximieren, sondern auch um den Gradienten einer Funktion zu approximieren [MBK20].

Gradient Descent

Unter den gradientenbasierten Optimierern ist *Gradient Descent* der einfachste [LBOM98]. Hierbei wird der Gradient mit einem Faktor, der sogenannten Lernrate, multipliziert und von den Parametern subtrahiert. Formal ist dies so definiert [LBOM98]:

$$\theta^{(n+1)} = \theta^{(n)} - \eta_n \nabla f(\theta^{(n)}) \tag{2.6}$$

Mit θ wird der Vektor der Parameter bezeichnet, der hochgestellte Wert gibt die Iteration an (n ist der aktuelle Wert, n + 1 der neue), η ist die Lernrate und ∇ ist der Gradient. Im einfachsten Fall ist die Lernrate eine skalare Konstante [LBOM98], aber es gibt auch komplexere Varianten, wie zum Beispiel ADAM, was als nächstes vorgestellt wird.

In Abbildung 2.2 ist beispielhaft eine Funktion abgebildet, die mit *Gradient Descent* minimiert werden kann. Oftmals wird dabei aber nur das lokale Minimum erreicht [SW18].

ADAM

ADAM ist eine Variante von *Gradient Descent*, die ebenfalls einfach zu implementieren und effizient ist [KB14]. Die Konvergenz ist dabei schneller als bei *Gradient Descent* und anderen Varianten. Der Unterschied zu *Gradient Descent* ist, dass es für jeden Parameter eine eigene Lernrate gibt. Diese werden währen des Trainings automatisch angepasst nach jeder Iteration.

SPSA

SPSA [Spa+92; Spa87] ist ein Optimierungsalgorithmus, der zwar einen Gradienten verwendet, diesen aber selber approximiert auf eine effiziente Weise [BPP13]. Wie bei der Finite-Differenzen-Methode ist es nicht nötig den Gradienten analytisch bestimmen zu können, da man lediglich die Funktion an bestimmten Stellen auswerten muss [Spa87]. Der große Vorteil von SPSA ist, dass für die Approximation lediglich zwei Funktionsauswertungen nötig sind, egal wie viele Parameter die Funktion besitzt [Spa87]. Bei der Finite-Differenzen-Methode hingegen ist die Anzahl der benötigten Funktionsauswertungen proportional zu der Anzahl der Parameter [Spa87]. Wenn der Gradient approximiert wurde, werden die Parameter ähnlich wie bei Gradient Descent aktualisiert [BPP13].

2.4.2 Gradientenfrei

Wie der Name schon vermuten lässt, benötigen gradientenfreie Optimierer nicht den Gradienten der Funktion, die sie optimieren sollen [RS13]. Diese Optimierer können daher auch benutzt werden, wenn der Gradient nicht analytisch bestimmt werden kann. Sie sind auch besser geeignet als die Finite-Differenzen-Methode, wenn die Auswertung der Funktion teuer oder verrauscht ist.

Nelder-Mead

Nelder-Mead ist ein häufig verwendeter Algorithmus, der Funktionswerte an unterschiedlichen Punkten vergleicht [NM65]. Wenn die Funktion n Parameter hat, vergleicht er die Funktionswerte an n + 1 verschiedenen Punkten, die ein n-Dimensionales Simplex definieren. Anschließend wird der Punkt mit dem höchsten Funktionswert ausgetauscht durch einen neuen Punkt. Dieser neue Punkt wird bestimmt, indem der schlechteste Punkt um den geometrischen Schwerpunkt des Simplexes herum transformiert wird mittels verschiedener Operationen [RS13].

COBYLA

COBYLA [Pow94; Pow98] ist eine Optimierungsmethode, die wie Nelder-Mead Funktionswerte an n + 1 verschiedenen Punkten verwendet [Pow07]. Diese werden aber dafür genutzt, eine lineare Approximation der *Objective Function* zu erstellen. Der Gradient dieser Approximation wird dann dafür genutzt einen neuen Punkt zu berechnen, an dem die zu optimierende Funktion ausgewertet werden soll. Einer der alten Punkte wird dann durch den neuen Punkt ersetzt.

NFT

NFT ist ein Optimierungsverfahren für quanten-klassische hybride Algorithmen [NFT20]. Diese Algorithmen verwenden eine Kombination aus Quantenschaltkreisen, die in Kapitel 3 beschrieben werden, und klassischen Algorithmen. NFT optimiert immer nur einen einzigen Parameter gleichzeitig, da in diesem Fall die *Objective Function* eine einfache Sinuskurve ist und sie sich bezüglich dieses Parameters exakt minimieren lässt. Dieses Verfahren hat mehrere Vorteile. Die *Objective Function* muss nur dreimal ausgewertet werden, um einen Parameter zu optimieren, er hat keine Hyperparameter, er konvergiert schnell, die Wahl der Parameter zu Beginn hat einen kleineren Einfluss und er ist robust gegenüber statistischer Fehler. Damit NFT genutzt werden kann, müssen der Quantenschaltkreis und die *Objective Function* mehrere Voraussetzungen erfüllen, was die Anwendbarkeit einschränkt.

2.5 Vorverarbeitung von Daten

Die Daten, die man verwenden möchte für maschinelles Lernen, sollten vorverarbeitet werden, bevor sie den Algorithmen übergeben werden [For19]. Dies ist wichtig, um gute Ergebnisse zu erhalten. Die Attribute können unterschiedliche Typen haben, die unterschiedlich vorverarbeitet werden müssen [HKP11]. Numerische Attribute sollten skaliert werden und kategorielle Attribute müssen als Zahlen kodiert werden, damit sie überhaupt verwendet werden können. Beides wird in den nächsten Abschnitten beschrieben.

2.5.1 Numerische Werte

Numerische Werte sind messbare Größen, die als Ganzzahl oder reelle Zahl dargestellt werden [HKP11]. Sie sollten skaliert werden, bevor sie verwendet werden, um die Ergebnisse und die Effizienz zu verbessern.

Skalierung

Bei der Skalierung geht es darum, die Werte so zu transformieren, dass sie z.B. in einem festen Intervall liegen (z. B. [-1, 1]) oder dass sie einen Durchschnitt von 0 und eine Varianz von 1 haben [HKP11]. Das ist für viele Algorithmen des maschinellen Lernens von Vorteil. Im Folgenden werden drei Varianten vorgestellt.

Min-Max Normalisierung Die Min-Max Normalisierung ist eine lineare Transformation der Werte [HKP11]. Die Werte werden so transformiert, dass sie in einem bestimmten Intervall liegen. Formal ist das folgendermaßen formuliert [HKP11]:

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A \tag{2.7}$$

v_i ist ein Wert des Attributes A, v_i' ist der neue Wert, min_A ist das Minimum, dass das Attribut A annehmen kann, max_A ist das Maximum, new_max_A das neue Maximum und new_min_A das neue Minimum.

Zero-Mean Normalisierung Bei der Zero-Mean Normalisierung werden die Werte bezüglich des Durchschnitts und der Standardabweichung normalisiert [HKP11]. Das ist von Vorteil gegenüber der Min-Max Normalisierung, wenn das Maximum und Minimum eines Attributs nicht bekannt sind oder es viele Ausreißer gibt. Formal ist das folgendermaßen formuliert [HKP11]:

$$v_i' = \frac{v_i - \overline{A}}{\sigma_A} \tag{2.8}$$

$\overline{A}$ ist der Durchschnitt von A und σ_A ist die Standardabweichung.

Robuste Skalierung Die robuste Skalierung ist eine Variante der Zero-Mean Normalisierung, die noch robuster gegen Ausreißer ist [Sci21]. Statt dem Durchschnitt wird hier der Median genutzt und statt der Varianz wird die Differenz zwischen dem 25% Quantil und dem 75% Quantil genutzt.

2.5.2 Kategorielle Werte

Kategorielle Werte oder auch nominale Werte genannt, sind Symbole oder "Namen von Dingen" [HKP11]. Die Werte besitzen keine bedeutsame Reihenfolge und mathematische Operationen können auch nicht sinnvoll auf diese angewendet werden. Ein Beispiel für einen kategoriellen Wert ist die Marke eines Autos.

One-Hot Kodierung

One-Hot Kodierung ist die meist genutzte Kodierung von kategoriellen Werten aufgrund ihrer Einfachheit und Effektivität [RBGE18]. Hierbei wird ein Vektor erzeugt, der so viele Elemente hat, wie es verschiedene Werte gibt für ein bestimmtes Attribut [CANS21]. Jedes Element ist 0, außer eins, das auf 1 gesetzt wird. Welches der Elemente auf 1 gesetzt wird, hängt von dem Wert ab, den man kodiert. Der Vektor könnte z. B. so aussehen:

$$\vec{x} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.9}$$

Hier ist das zweite Element 1, das bedeutet, dass damit der zweite Wert aus der Menge der Werte, die das Attribut annehmen kann, kodiert ist.

2.5.3 Dimensionsreduktion

Bei der Dimensionsreduktion geht es um die Reduzierung der Anzahl der Attribute, die verwendet werden [HKP11]. Hierfür gibt es mehrere Gründe. Der Datensatz lässt sich einfacher benutzen, der Rechenaufwand von vielen Algorithmen verringert sich, Rauschen wird reduziert, Klassifikationsgenauigkeit wird verbessert, Daten lassen sich besser visualisieren und die Ergebnisse werden einfacher verständlich [Bra20; For19; Har12].

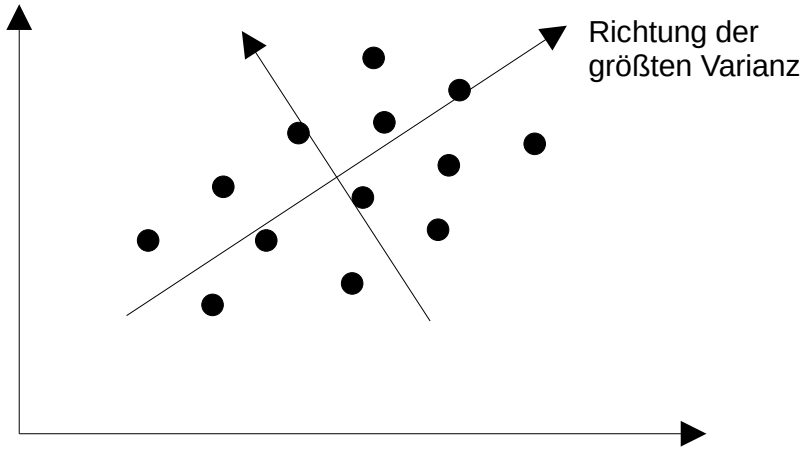

Abbildung 2.3: Vektoren, die durch PCA bestimmt werden. Die orangenen Punkte sind die Datenpunkte. Die Pfeile in der Mitte sind die orthogonalen Vektoren, die PCA gefunden hat. Der Vektor, der nach rechts oben zeigt, zeigt in die Richtung der größten Varianz [Bra20].

PCA

PCA ist eine sehr bekannte Dimensionsreduktionsmethode, die versucht, möglichst viel Varianz von den Daten zu erhalten, aber mit möglichst wenigen Attributen [Bra20]. Der Grund hierfür ist, dass in der Varianz meist die nützlichsten Informationen enthalten sind. Zunächst werden die Daten normalisiert, damit Attribute mit größeren Wertebereichen nicht stärker gewichtet werden als die anderen Attribute [HKP11]. Dann werden orthonormale Vektoren bestimmt, die die Daten am besten repräsentieren. Diese werden dann ihrer Signifikanz nach sortiert und die Schwächsten können verworfen werden. Die Daten werden dann als Linearkombination dieser Vektoren dargestellt und haben eine geringere Anzahl an Elementen, da im vorherigen Schritt die Anzahl der Vektoren verringert wurde.

Autoencoder

Ein Autoencoder ist eine Art von neuronalem Netz, das aus zwei Teilen besteht, einem Encoder und einem Decoder [For19]. Der Encoder produziert aus der Eingabe eine Ausgabe, die üblicherweise aus weniger Werte besteht und reduziert damit die Anzahl der Dimensionen. Der Decoder bekommt die Ausgabe des Encoder als Eingabe und produziert eine Ausgabe mit genau so vielen Werten, wie der Encoder ursprünglich als Eingabe bekommen hat. Fasst man den Encoder und Decoder zu einem einzigen neuronalen Netz zusammen, entsteht der Autoencoder. Dieser hat in der Mitte üblicherweise einen Flaschenhals, da der Encoder die Dimensionen reduziert. Dies ist in Abbildung 2.4 dargestellt. Der Autoencoder wird darauf optimiert, möglichst genau das auszugeben, was er als Eingabe bekommen hat [Kra91]. Da in der Mitte der Flaschenhals ist, muss der Autoencoder lernen, die Eingabedaten im Encoder zu komprimieren und im Decoder zu dekomprimieren [GBC16]. Die Werte, die der Encoder erzeugt, also die komprimierten Daten, werden als *Embedding* bezeichnet [Dah18].

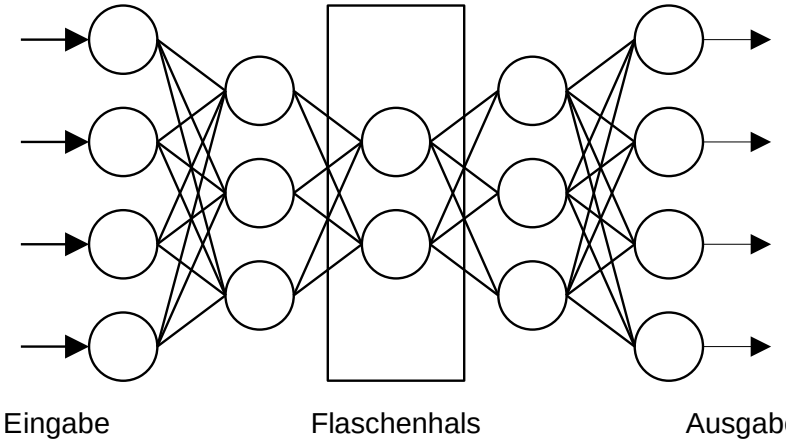

Abbildung 2.4: Ein Autoencoder mit dem Encoder zwischen Eingabe und Flaschenhals, und dem Decoder zwischen Flaschenhals und Ausgabe. Darstellung inspiriert durch [CWZ+14].

Embeddings haben mehrere nützliche Eigenschaften, die sie interessant machen. Zum einen liegen Embeddings von ähnlichen Eingabedaten nahe zusammen, wenn man sie als Punkte in einem Raum betrachtet [MCCD13]. Zum anderen können in manchen Fällen sogar einfache algebraische Operationen auf ihnen ausgeführt werden und sinnvolle Ergebnisse erzielt werden. Beispielsweise wenn man einen Autoencoder auf Text anwendet, kann man Embedding(“König”) - Embedding(“Mann”) + Embedding(“Frau”) berechnen und das Ergebnis liegt am nächsten an dem Embedding von dem Wort “Königin”.

Da die Embeddings von ähnlichen Objekten beieinanderliegen, können diese Embeddings auch für Clustering genutzt werden [GLZY17]. Die Embeddings zu clustern kann bessere Ergebnisse erzielen, als die Eingabedaten direkt zu verwenden [XGF16], und sie können auch bessere Ergebnisse erzielen als wenn PCA zur Dimensionsreduktion genutzt wird [Dah18].

3 Quantencomputer

Quantencomputer sind eine neue Art von Computer, die als kleinste Einheit für Speicher und Verarbeitung nicht das Bit benutzen, sondern das Quanten-Bit, auch Qubit genannt [RP11]. Quantencomputer können bestimmte Aufgaben schneller lösen als klassische Computer, teilweise sogar exponentiell schneller. Dies macht Quantencomputer zu einem spannenden Forschungsthema.

Die Quantencomputer, die es momentan gibt, werden als *Noisy-Intermediate Scale Quantum (NISQ) Computer* bezeichnet [WSS+19]. Bei ihnen treten viele Fehler durch Rauschen auf und sie haben noch weniger als 100 Qubits [LC20; Pre18]. Um mit den Fehlern umzugehen, gibt es zwei verschiedene Ansätze [WSS+19]: (i) während der Ausführung werden die auftretenden Fehler korrigiert, (ii) nur ein Teil der Berechnungen wird auf einem Quantencomputer ausgeführt, da bei kürzeren Berechnungen weniger Fehler auftreten. Der Rest des Algorithmus wird dann auf einem klassischen Computer ausgeführt. Für den zweiten Ansatz wurden Verfahren entwickelt wie z. B. der *Quantum Approximation Optimization Algorithm* (QAOA) [FGG14] und der *Variational Quantum Eigensolver* (VQE) [PMS+14].

In diesem Kapitel werden die Grundlagen des Rechnens auf Quantencomputern kurz beschrieben und im nächsten Kapitel wird darauf eingegangen, wie sich Quantencomputer für maschinelles Lernen nutzen lassen.

3.1 Qubits

Qubits sind die Einheiten, auf denen ein Quantencomputer rechnet [RP11]. Der Wert eines Qubits wird als zwei-dimensionaler Vektor $v = \begin{pmatrix} a \\ b \end{pmatrix}$ beschrieben. Der Vektor v muss dabei eine Länge von 1 haben. Die Werte a und b können komplexe Zahlen sein und werden als Amplituden bezeichnet.

3.1.1 Dirac-Notation

Die Dirac-Notation wird viel genutzt, um Quantenzustände wie z.B. den Wert eines Qubits, und deren Transformation zu beschreiben [RP11]. $|x\rangle$ steht für einen Vektor, der einen Quantenzustand repräsentiert. Das x ist dabei ein beliebiger Name für diesen Quantenzustand. Um das Qubit aus dem vorherigen Abschnitt in dieser Notation zu beschreiben, kann man es als Linearkombination von Vektoren formulieren: $|v\rangle = a|0\rangle + b|1\rangle$. Hierbei werden $|0\rangle$ und $|1\rangle$ als die Standardbasis bezeichnet.

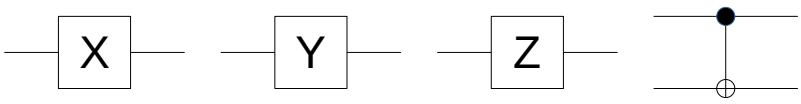

Abbildung 3.1: Von links nach rechts: Pauli-X-Gatter, Pauli-Y-Gatter, Pauli-Z-Gatter, CNOT-Gatter [RP11].

3.2 Messungen

Um Informationen über einen Quantenzustand zu bekommen, muss dieser gemessen werden [RP11]. Durch eine Messung erhält man allerdings nicht alle Informationen über einen Quantenzustand. Misst man beispielsweise ein einzelnes Qubit in der Standardbasis, erhält man als Ausgabe lediglich $|0\rangle$ oder $|1\rangle$. Welchen der beiden Werte man erhält, ist zufällig und die Wahrscheinlichkeit hängt von den Amplituden ab. Bei dem Quantenzustand $|v\rangle = a|0\rangle + b|1\rangle$ ist die Wahrscheinlichkeit $|0\rangle$ zu messen $|a|^2$ und $|1\rangle$ zu messen $|b|^2$.

3.3 Quantenregister

Mehrer Qubits können zusammengefasst werden zu einem Quantenregister. Eine Besonderheit von Quantenzuständen ist, dass ihr Zustandsraum exponentiell wächst mit der Anzahl der Qubits [RP11]. Ein Quantenzustand mit n Qubits wird durch einen 2^n-dimensionalen Vektor repräsentiert.

3.3.1 Quantenverschränkung

Eine weitere Eigenheit ist, dass viele der Quantenzustände nicht durch die Zustände der einzelnen Qubits ausgedrückt werden können [RP11]. Diese Zustände werden als verschränkt bezeichnet. Auf klassischen Computern lässt sich dieses Phänomen nicht effizient simulieren, was sich ausnutzen lässt, bestimmte Probleme auf Quantencomputern schneller zu lösen als auf klassischen.

3.4 Gatter und Schaltkreise

Berechnungen auf Quantencomputer sind Transformationen der Quantenzustände [RP11]. Diese Transformationen bilden einen Quantenzustand auf einen anderen ab. Messungen zählen deshalb nicht zu diesen Transformationen. Es können nur bestimmte Transformationen ausgeführt werden, die als unitär bezeichnet werden. Eine unitäre Transformation wird als Matrix U dargestellt und hat die Eigenschaft $U^\dagger = U^{-1}$. Dabei bezeichnet $U^\dagger$ die adjungierte Matrix, also die transponierte und konjugierte Matrix von U. Aufgrund dieser Bedingung, dass die Transformationen unitär sein müssen, sind sie auch immer umkehrbar.

Jede beliebige unitäre Transformation kann ausgeführt werden durch eine Reihe von Transformationen auf nur ein und zwei Qubits [RP11]. Diese Transformationen, die auf wenigen Qubits arbeiten, werden als Quantengatter bezeichnet. Wenn man mehrere solche Quantengatter kombiniert, wird dies als Quantenschaltkreis bezeichnet.

Ein paar einfache ein-Qubit Gatter sind die Pauli-Gatter [RP11]:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.1}$$

und das Hadamard-Gatter:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.2}$$

Geometrisch betrachtet sind alle Transformationen auf Quantenzuständen Rotationen im entsprechenden Zustandsraum [RP11].

Ein wichtiges zwei-Qubit Gatter ist das CNOT [RP11]. Es flippt das zweite Bit, wenn das erste 1 ist.

$$C_{not} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.3}$$

In Abbildung 3.1 sind die grafischen Repräsentationen dieser Gatter dargestellt.

3.5 Kodierung von Informationen

Bevor man Daten in einem Quantencomputer verarbeiten kann, müssen diese zunächst als Quantenzustand kodiert werden. Es gibt verschiedene Verfahren dafür, die unterschiedliche Eigenschaften haben, z. B. bezüglich der Anzahl der benötigten Qubits und der Robustheit gegenüber Rauschen [LC20].

Das wahrscheinlich einfachste Verfahren wird als *Angle Encoding* oder *Qubit Encoding* bezeichnet [LC20]. Für aktuelle Quantencomputer ist es von Vorteil, wenn Quantenschaltkreise eine möglichst geringe Tiefe haben, da mit steigender Tiefe das Rauschen zunimmt. Da dieses Verfahren so einfach ist, hat der Schaltkreis, der die Daten kodiert auch nur eine geringe Tiefe.

Die Attribute, die kodiert werden sollen, müssen reelle Werte sein und pro Attribut wird ein Qubit benötigt. Der Quantenzustand soll dann so aussehen [LC20]:

$$|\mathbf{x}\rangle = \bigotimes_{i=1}^{N} \cos(x_i)|0\rangle + sin(x_i)|1\rangle \tag{3.4}$$

Die x_i sind dabei die einzelnen Attribute. Um diesen Zustand zu erzeugen, muss auf jedem Qubit nur eine Transformation ausgeführt werden, die folgendermaßen definiert ist [LC20]:

$$U_i := \begin{pmatrix} \cos(x_i) & -\sin(x_i) \\ \sin(x_i) & \cos(x_i) \end{pmatrix} \tag{3.5}$$

4 Maschinelles Lernen auf Quantencomputern

In den beiden vorherigen Kapiteln wurde einerseits das maschinelle Lernen beschrieben, mit dessen Hilfe aus Daten automatisiert gelernt wird, Muster zu erkennen und Probleme zu lösen [SP18]. Andererseits wurden Quantencomputer beschrieben, die auf eine völlig neue Weise Informationen verarbeiten, die mit klassischen Computern nicht effizient simuliert werden kann. Im maschinellen Lernen auf Quantencomputern geht es darum, diese beiden Themen miteinander zu vereinen. Wissenschaftler untersuchen, ob neue Verfahren des maschinellen Lernens für Quantencomputer entwickelt werden können und ob bereits existierende Verfahren, die sich auf klassischen Computern bewährt haben, sich für Quantencomputer anpassen lassen.

Da Quantencomputer bestimmte Probleme schneller lösen können als klassische, möchte man untersuchen, ob sie auch beim maschinellen Lernen schneller sein können [SP18]. Beispielsweise können bestimmte klassische Routinen durch quantenbasierte Alternativen ersetzt werden, die schneller sind. Dazu gehören unteranderem die *Quantum Basic Linear Algebra Subroutines* (qBLAS), Fourier Transform, das Finden von Eigenvektoren und Eigenwerten und das Lösen von linearen Gleichungen [BWP+17; ROA17]. Geschwindigkeit ist aber nicht das Einzige, worin sie besser sein könnten. Da Quantencomputer Zustände und damit auch Muster erzeugen können, die mit klassischen Computern zu aufwendig wären zu berechnen, könnten damit vielleicht auch Muster erkannt werden, die klassische Computer nur mit enormem Aufwand erkennen würden [BWP+17; ROA17]. Es können noch viele weitere Fragen untersucht werden, z. B. ob die neuen Verfahren besser mit weniger Daten umgehen können oder ob Rauschen weniger Einfluss hat [SP18].

4.1 Arten

Es gibt verschiedene Arten von maschinellem Lernen, in denen Quantencomputer verwendet werden. Es kann z. B. unterschieden werden, ob die Daten aus klassischen oder Quantencomputern stammen und ob die Verarbeitung auf einem klassischen oder Quantencomputer stattfindet [SP18]. Verwendet man diese Einteilung, kommt man auf vier unterschiedliche Arten von maschinellem Lernen. In Abbildung 4.1 ist dies veranschaulicht.

Werden klassische Daten auf einem klassischen Computer verarbeitet, entspricht dies dem klassischen maschinellen Lernen und es kommen keine Quantencomputer zum Einsatz [SP18]. Werden Quanteninformationen auf klassischen Computern verarbeitet mit Verfahren des maschinellen Lernens, geht es darum, Quantencomputern zu helfen, indem z. B. mit möglichst wenigen Messungen möglichst viel über den inneren Zustand in Erfahrung gebracht werden soll. Verwendet man Quantencomputer, um die Daten zu verarbeiten, gibt es wiederum die Möglichkeit, klassische oder Quanteninformationen zu verwenden. Bei klassischen Informationen müssen diese erst mit einem Quantenschaltkreis kodiert werden, sodass der Quantencomputer mit dem entstandenen Quantenzustand arbeiten kann.

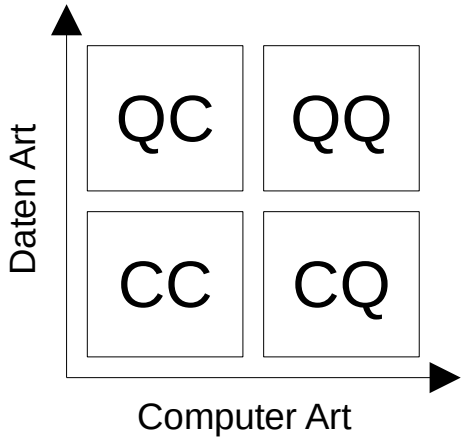

Abbildung 4.1: Vier verschiedene Arten von maschinellem Lernen. Die Daten können klassisch sein oder von Quantencomputern stammen. Die Verarbeitung kann auf klassischen Computern oder auf Quantencomputern stattfinden [SP18].

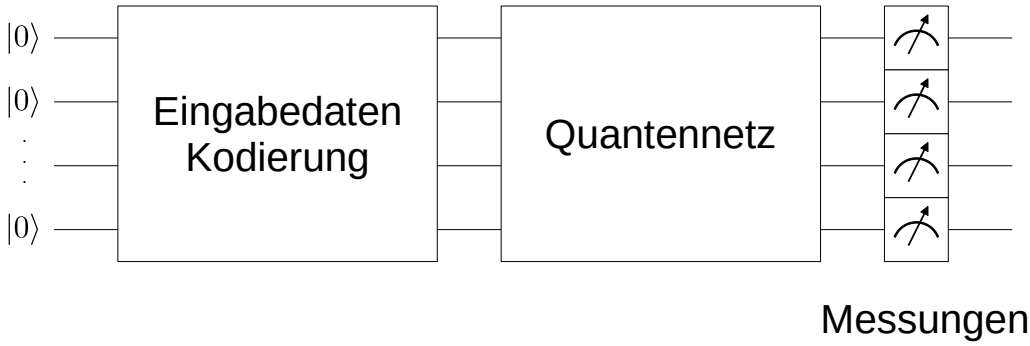

Abbildung 4.2: Kodierung der Eingabedaten, Ausführung des Quantennetzes und Messung der Ergebnisse [ASZ+20].

In dieser Arbeit werden klassische Datensätze betrachtet und quantenunterstützte Verfahren, die aus diesen Daten lernen sollen.

4.2 Quantennetze

Quantennetze sind eine Art von neuronalen Netzen, die darauf ausgelegt sind, auf Quantencomputern ausgeführt zu werden. Genauer gesagt sind es parametrisierte Quantenschaltkreise, deren Parameter während des Trainings optimiert werden [ASZ+20]. Bevor ein solches Quantennetz ausgeführt werden kann, muss zunächst ein Quantenzustand präpariert werden mit Daten, die das Quantennetz als Eingabe bekommen soll. Auf diesem Quantenzustand wird dann der Quantenschaltkreis ausgeführt, der das Quantennetz repräsentiert. Zum Schluss werden dann die Qubits gemessen. Diese Werte werden dann auf einem klassischen Computer in eine *Objective Function* gegeben und berechnet, wie gut das Ergebnis ist. Die Parameter des Quantennetzes werden dann über einen klassischen Algorithmus optimiert, um das Ergebnis der *Objective Function* zu verbessern [SP18]. In Abbildung 4.2 wird die Ausführung eines Quantennetzes skizziert.

4.3 Hybride neuronale Netze

Da zurzeit nur wenige Qubits pro Quantencomputer zur Verfügung stehen, bietet es sich an, hybride neuronale Netze zu verwenden. Diese bestehen aus klassischen neuronalen Netzen, wie sie in Abschnitt 2.3 beschrieben wurden und aus Quantennetzen, wie sie im vorherigen Abschnitt beschrieben wurden [MBI+20]. Sie haben den Vorteil, dass der klassische Teil die Eingabedaten vorverarbeiten kann und auch die Dimensionen reduzieren kann, bevor das Quantennetz auf den kleineren Daten arbeiten kann. Hybride neuronale Netze sind ein Kompromiss, um Daten verwenden zu können, die mehr Dimensionen haben, als Qubits zur Verfügung stehen. Wie auch bei klassischen neuronalen Netzen und Quantennetzen werden hybride neuronale Netze mit klassischen Optimierern optimiert, z. B. mit denen, die in Abschnitt 2.4 vorgestellt wurden.

4.4 Parametershift

Möchte man Quantenschaltkreise, also auch Quantennetze, mit einem gradientenbasiert Optimierer optimieren, muss man den Gradienten dieses Quantenschaltkreises berechnen oder approximieren können [SBG+19]. Dazu kann die Finite-Differenzen-Methode verwendet werden. Da heutigen Quantencomputern allerdings viele Fehler machen und verrauschte Ergebnisse ausgeben, kann es passieren, dass die Finite-Differenzen Methode keine guten Ergebnisse liefert.

Parametershift ist eine Alternative zur Finite-Differenzen-Methode, um den Gradient eines Quantenschaltkreises exakt zu berechnen, falls man exakte Erwartungswerte für den Quantenschaltkreis bestimmen kann [SBG+19]. Da die Ausgaben eines Quantenschaltkreises stochastisch sind, können die Erwartungswerte nur approximiert werden, indem der Quantenschaltkreis mehrfach ausgeführt wird und der Durchschnitt berechnet wird. Dadurch wird der Gradient auch zu einer Approximation.

Parametershift muss für jeden Parameter den Schaltkreis zweimal ausführen mit verschiedenen Werten für die Parameter [SBG+19]. Je nachdem, welche Gatter der Schaltkreis verwendet, kann der Schaltkreis entweder unverändert verwendet werden oder er muss leicht angepasst werden.

5 Experimente

Quantencomputer versprechen für das maschinelle Lernen viele Vorteile gegenüber klassischen Computern. Ob diese jedoch eingehalten werden können, soll in diesem Kapitel ausprobiert werden für einen Anwendungsfall [1]. Ziel ist es, die Ergebnisse von klassischen Clusteringverfahren zu verbessern, indem die Dimensionen der Daten vorher mithilfe eines hybriden Autoencoders reduziert werden. Dieser hybride Autoencoder besteht aus Schichten eines klassischen neuronalen Netzes und aus Schichten von Quantennetzen. Die Ergebnisse werden verglichen mit denen eines rein klassischen Autoencoders und PCA, um zu sehen, ob die Nutzung von quantenmechanischen Eigenschaften für diesen Anwendungsfall Vorteile mit sich bringen.

In Abschnitt 5.1 wird die Frage diskutiert, welche Optimierer gute Ergebnisse hervorbringen, wenn sie Quantennetze optimieren müssen. Dafür werden verschiedene Varianten von Quantennetzen vorgestellt und anschließend werden einige Optimierer genutzt, um eine Variante zu optimieren für einen einfachen Testfall.

In Unterabschnitt 5.2.3 geht es um die Frage, wie gut ein hybrider Autoencoder darin ist, Dimensionsreduktion durchzuführen, um das Clustern von Daten zu erleichtern. Dafür werden alle drei in Abschnitt 5.1 vorgestellten Quantennetze in hybriden Autoencodern genutzt und auf drei unterschiedliche Datensätze angewandt.

5.1 Quantennetze

In diesem Kapitel werden drei verschiedene Varianten von Quantennetzen vorgestellt und es wird verglichen, wie gut verschiedene Optimierer ein Quantennetz optimieren können. Ein Quantennetz ist letztendlich nichts anderes als ein parametrisierter Quantenschaltkreis. Diese Parameter können optimiert werden, indem der Quantenschaltkreis als eine mathematische Funktion betrachtet wird und eine *Objective Function* definiert wird, die angibt, wie gut die Ausgaben des Quantenschaltkreises sind. Ein Optimierer kann dann die Parameter so anpassen, dass die *Objective Function* minimiert wird.

Damit ein Quantennetz effizient genutzt werden kann, sollte die Anzahl an Gattern und Parametern maximal polynomiell steigen [ROA17]. Die Varianten, die im nächsten Abschnitt vorgestellt werden, haben diese Eigenschaft und bei einer Variante skaliert die Anzahl der Parameter sogar nur linear mit der Anzahl der Qubits. Das ist gerade für gradientenbasierte Optimierer von Vorteil, da häufig genutzte Verfahren zum Approximieren von Gradienten, wie die Finite-Differenzen Methode und Parametershift, pro Parameter 1 bis 2 Quantenschaltkreise ausführen müssen [LeV98; SBG+19].

[1] Quellcode unter https://github.com/PhilWun/MAProjekt verfügbar.

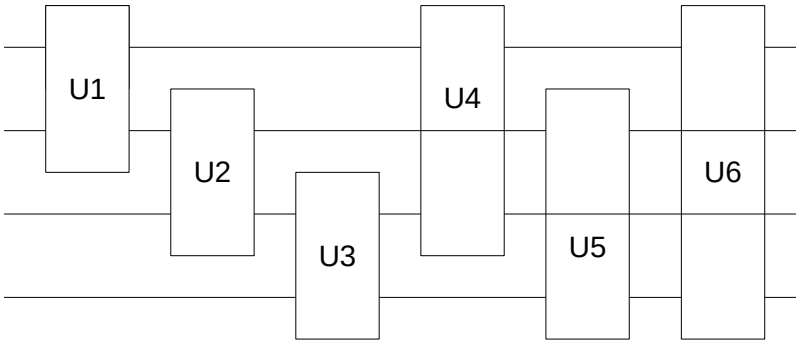

Abbildung 5.1: Erste Variante eines Quantennetzes, bestehend aus zwei-Qubit Gattern, die beliebige unitäre Operationen auf zwei Qubits ausführen können [ROA17].

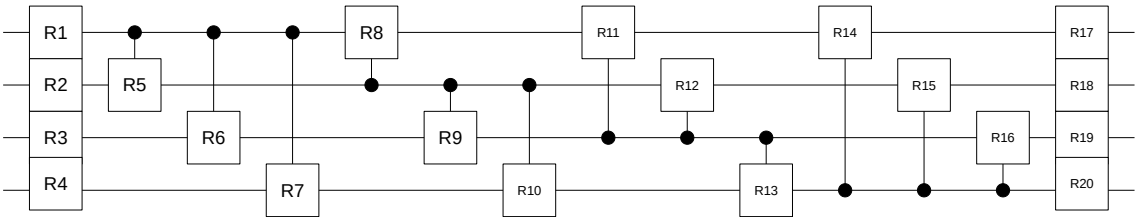

Abbildung 5.2: Zweite Variante eines Quantennetzes, bestehend aus ein-Qubit Rotationen und kontrollierten ein-Qubit Rotationen [ROA17].

5.1.1 Varianten

In diesem Abschnitt werden drei verschiedene Varianten von Quantennetzen vorgestellt, die in den nachfolgenden Experimenten genutzt wurden. Jede Variante verwendet ein- und zwei-Qubit Gatter, die sich leicht implementieren lassen [ROA17]. Außerdem steigt die Anzahl der Parameter quadratisch bzw. linear mit der Anzahl der Qubits. Damit wird die Voraussetzung erfüllt, dass sie maximal polynomiell steigen soll.

In Abbildung 5.1 ist die erste hier vorgestellte Variante dargestellt. Sie wurde ausgewählt, da sie in [ROA17] erfolgreich in einem Quantenautoencoder eingesetzt wurde. Auf jedes Paar von Qubits wird ein allgemeines zwei-Qubit Gatter angewendet, welches beliebige unitäre Operationen auf zwei Qubits ausführen kann. Diese unitäre Operation wird durch 15 Parameter definiert, wodurch sich die gesamte Anzahl an Parameter bei dieser Variante auf $15n(n-1)/2$ beläuft, wobei n die Anzahl der Qubits ist.

In Abbildung 5.2 ist eine weitere Variante dargestellt, ebenfalls aus [ROA17]. Hier werden parametrisierte Rotationen verwendet, einerseits auf einzelnen Qubits und andererseits kontrolliert durch ein zweites Qubit. Pro Rotation werden drei Parameter benötig, dadurch sind es insgesamt $3n(n-1) + 6n$ Parameter, die den Quantenschaltkreis bestimmen.

In Abbildung 5.3 ist die letzte Variante dargestellt. Diese wurde ausgewählt, da in [ASZ+20] gezeigt wurde, dass sie sich schneller optimieren lässt als ein vergleichbares klassisches neuronales Netz und auch einen geringeren Fehler erreicht. Sie besteht aus parametrisierten Y-Rotationen und CNOTs zwischen jedem möglichen Qubit-Paar. Für jede Y-Rotation wird nur ein Parameter benötigt, daher ergibt sich eine Anzahl von $2n$ Parameter für das ganze Quantennetz.

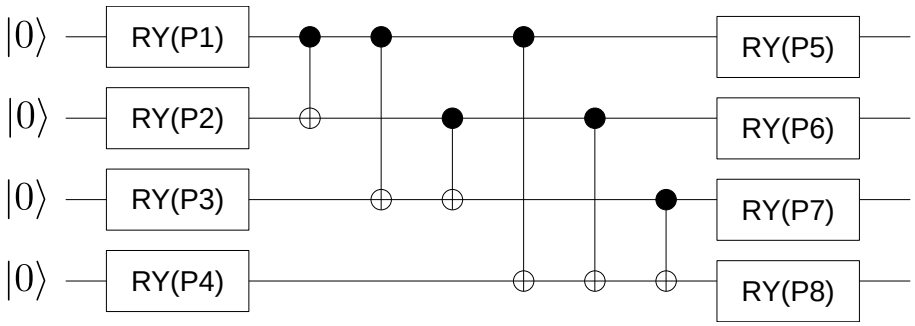

Abbildung 5.3: Dritte Variante eines Quantennetzes, bestehend aus Rotationen um die y-Achse und CNOTs [ASZ+20].

5.1.2 Optimierer

In Abschnitt 2.4 wurden bereits mehrere Optimierer vorgestellt. Nun stellt sich die Frage, welche dieser Optimierer am besten geeignet sind, um Quantennetze zu optimieren. Um dieser Frage nachzugehen, wurde ein Quantennetz der ersten Variante optimiert, welche in Unterabschnitt 5.1.1 vorgestellt wurde. Diese Variante wurde ausgewählt, da sie die meisten Parameter in Bezug auf die Anzahl der Qubits hat und damit voraussichtlich die größte Herausforderung darstellt. Ausgeführt wurden die Experimente mit dem QasmSimulator von Qiskit [AAA+19]. Das Ziel der Optimierung ist, dass die Messwerte des Quantennetzes jeweils einen Erwartungswert von 0.8 haben. Die Wahl dieses Wertes war zufällig und es wurde lediglich darauf geachtet, nicht 0, 0.5 oder 1 zu wählen, da diese Werte sehr einfach mit einem Identitäts-Gatter, Hadamard-Gatter oder Pauli-X-Gatter hätten erreicht werden können.

Die getesteten Optimierer sind in Tabelle 5.1 aufgelistet inklusive der verwendeten Argumente. Zu den Argumenten zählt die maximale Anzahl an Iterationen, die maximale Anzahl an Funktionsauswertungen, die Schrittweite für die Finite-Differenzen-Methode und Argumente, die spezifisch sind für den jeweiligen Optimierer. Es wurde jeweils die Implementierung genutzt, die in dem Qiskit Framework [AAA+19] enthalten ist. Alle lokalen Optimierer, die in Qiskit enthalten sind, wurden getestet, bis auf AQGD und P_BFGS, da diese abgestürzt sind bei der Ausführung. Bei bestimmten Argumenten wurde von den Standardwerten abgewichen, die von Qiskit voreingestellt waren. Die maximale Anzahl an Iterationen und Funktionsauswertungen wurde jeweils so gewählt, dass die Optimierer entweder genügend Zeit hatten zu konvergieren oder mindestens 1000 Funktionsauswertungen durchgeführt wurden, je nachdem, was zuerst eingetreten ist. Das Epsilon, welches die Schrittweite für die Finite-Differenzen Methode definiert, wurde immer auf 0.1 gesetzt, da bei kleineren Werten die approximierten Gradienten zu viel Rauschen enthielten und die Optimierung nicht erfolgreich war. Alle Toleranzen, welche genutzt werden, um die Optimierung vorzeitig zu terminieren, wurden auf 0 gesetzt, um eine zu frühe Terminierung zu vermeiden.

In Abbildung 5.4 sind die Ergebnisse des ersten Experiments dargestellt. Es wurde getestet, wie viele Funktionsauswertungen die Optimierer durchschnittlich benötigt haben, um zu konvergieren. Die Anzahl der Funktionsauswertungen entspricht auch der Anzahl der Quantenschaltkreise, die ausgeführt werden. Wenn der mittlere quadratische Fehler 0.01 erreicht hat, wurde das als konvergiert angesehen. Jeder Optimierer wurde genutzt, um 10-mal das Quantennetz zu optimieren, das mit zufälligen Parametern initialisiert wurde. Die durchschnittliche Anzahl an Funktionsauswertungen

Optimierer	Argumente
ADAM	maxiter=100, tol=0, lr=0.1, beta1=0.9, beta2=0.99, noise_factor=1e-08, eps=0.1, amsgrad=False
AMSGRAD	maxiter=100, tol=0, lr=0.1, beta1=0.9, beta2=0.99, noise_factor=1e-08, eps=0.1, amsgrad=True
CG	maxiter=1000, disp=False, gtol=0, tol=0, eps=0.1
COBYLA	maxiter=1000, disp=False, rhobeg=1.0, tol=0
L_BFGS_B	maxfun=10000, maxiter=10000, factr=10, iprint=-1, eps=0.1
GSLS	maxiter=1000, maxfun=1000, disp=False, sampling_radius=1e-06, sampling_size_factor=1, initial_step_size=0.01, min_step_size=1e-10, step_size_multiplier=0.4, armijo_parameter=0.1, min_gradient_norm=1e-08, max_failed_rejection_sampling=50
NELDER_MEAD	maxiter=1000, maxfun=1000, disp=False, xtol=0, tol=0, adaptive=False
NFT	maxiter=1000, maxfun=1000, disp=False, reset_interval=32
POWELL	maxiter=100, maxfun=100, disp=False, xtol=0, tol=0
SLSQP	maxiter=100, disp=False, ftol=0, tol=0, eps=0.1
SPSA	maxiter=1000, save_steps=1,last_avg=1, c0=0.6283185307179586, c1=0.1, c2=0.602, c3=0.101, c4=0, skip_calibration=False
TNC	maxiter=100, disp=False, accuracy=0, ftol=0, xtol=0, gtol=0, tol=0, eps=0.1

Tabelle 5.1: Die verwendeten Optimierer für die Quantennetze. Die Bezeichnungen der Optimierer sind so gewählt wie in der Dokumentation von Qiskit [AAA+19]. Es werden alle Argumente angegeben, mit denen die Optimierer initialisiert werden.

aus diesen Durchläufen wird im oberen Bereich auf der x-Achse dargestellt. Im unteren Bereich gibt die x-Achse an, wie oft der Optimierungsprozess nicht konvergiert ist. TNC und Nelder Mead sind nie konvergiert, weshalb sie nicht in dem Diagramm dargestellt sind.

Alle gradientenbasierten Optimierer außer POWELL haben deutlich mehr Funktionsauswertungen benötigt als die meisten Gradientenfreien. Das lässt sich damit erklären, dass in jeder Iteration eines gradientenbasierten Optimierers der Gradient approximiert werden muss. SPSA benötig allerdings deutlich weniger Funktionsauswertungen als alle anderen getesteten gradientenbasierten Optimierer und kommt in die Nähe von den gradientenfreien Optimierern COBYLA und NFT. Das liegt vermutlich an der effizienten Art, wie SPSA den Gradienten approximiert, da für diese Art nur zwei Funktionsauswertungen nötig sind, egal wie viele Parameter optimiert werden [Spa87]. Gradientenapproximierungsverfahren wie die Finite-Differenzen Methode und Parametershift benötigen hingegen 1-2 Funktionsauswertungen pro Parameter [LeV98; SBG+19]. GSLS ist in 80% der Durchläufe nicht konvergiert, was im Vergleich mit den anderen Optimierern sehr viel schlechter ist und TNC ist sogar nie konvergiert.

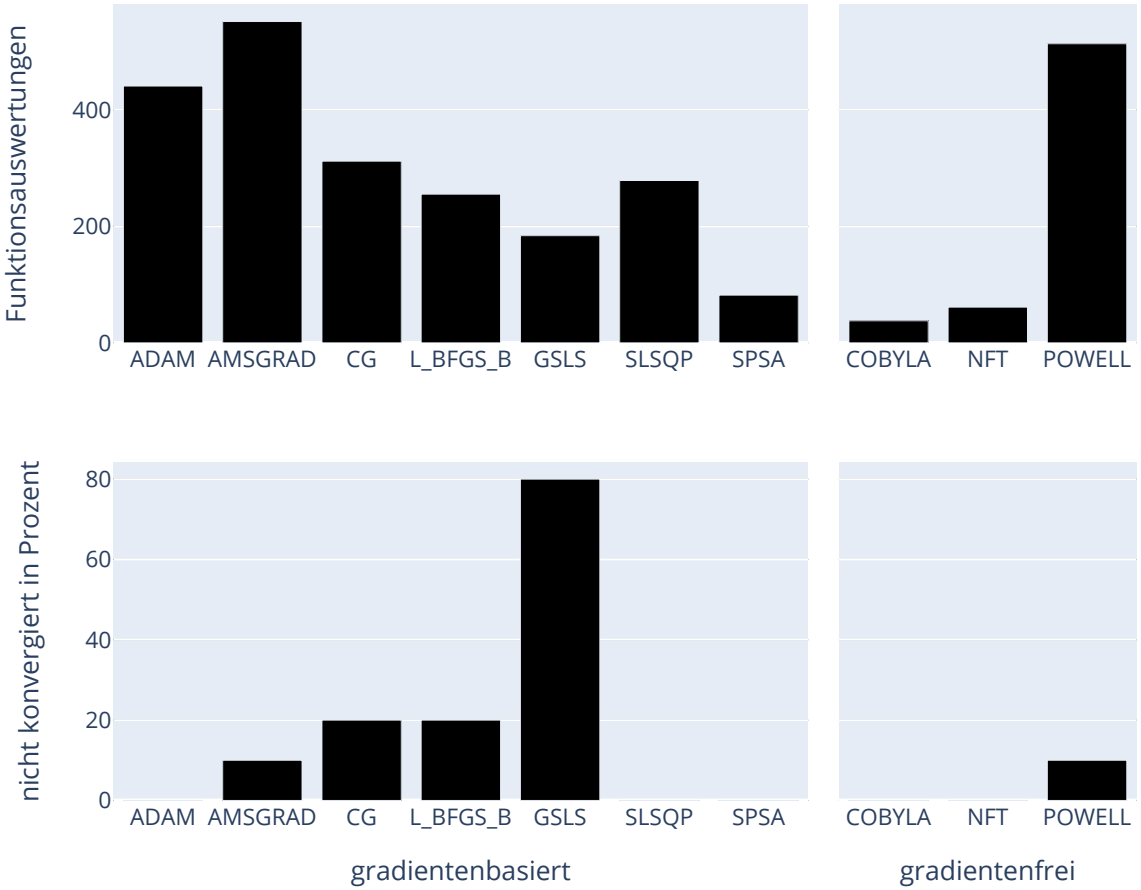

Abbildung 5.4: Durchschnittliche Anzahl an Funktionsauswertungen, die benötigt wurden, um einen mittleren quadratischen Fehler von 0.01 zu erreichen (oben) und Anteil der nicht konvergierten Durchläufe (unten). Es wurden jeweils 10 Durchläufe durchgeführt. Auf der linken Seite sind die gradientenbasierten Optimierer und auf der rechten die Gradientenfreien. TNC und Nelder Mead haben es nie geschafft, den Fehler so weit zu reduzieren und sind deshalb nicht dargestellt.

Bei den gradientenfreien Optimierern haben COBYLA und NFT im Vergleich sehr viel weniger Funktionsauswertungen benötigt als die meisten anderen Optimierer. POWELL hat allerdings sehr viel mehr benötigt und gehört auch im Vergleich mit den gradientenbasierten Optimierern zu den langsamsten. Da bei diesen Verfahren kein Gradient approximiert werden muss, können dadurch viele Funktionsauswertungen eingespart werden, was sich bei COBYLA und NFT bemerkbar gemacht hat. Nelder Mead ist in den 10 Durchläufen nie konvergiert und ist deshalb nicht in Abbildung 5.4 dargestellt.

In Abbildung 5.5 ist exemplarisch der mittlere quadratische Fehler von vier Optimierern während des Trainings abgebildet. Bei den beiden gradientenfreien Optimierern ist zu sehen, dass der Fehler früher sinkt als bei den beiden anderen, die gradientenbasiert sind. NFT schafft es aber nicht, einen so niedrigen Fehler zu erreichen, wie die anderen dargestellten Optimierer.

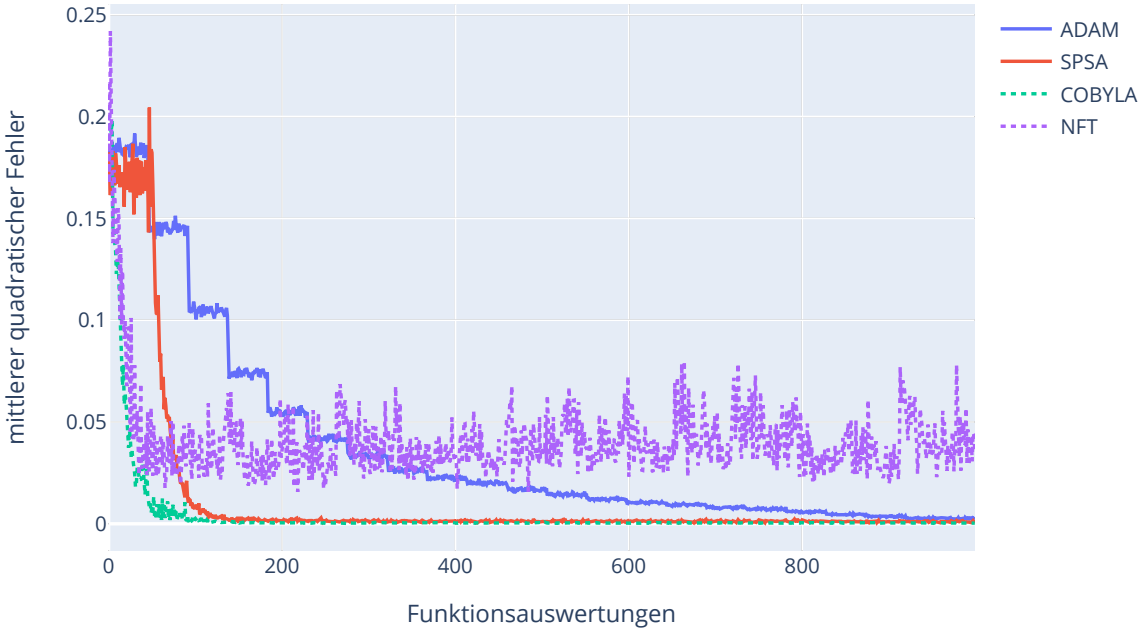

Abbildung 5.5: Mittlerer quadratischer Fehler während des Trainings. Hier werden zwei gradientenbasierte und zwei gradientenfreie Optimierer verglichen. Die gradientenbasierten werden mit durchgezogenen Linien dargestellt und die gradientenfreien mit gestrichelten. Auf der y-Achse ist der mittlere quadratische Fehler und auf der x-Achse die Anzahl der bisherigen Funktionsauswertungen.

Bei ADAM wird vor jedem Schritt der Gradient mit der Finite-Differenzen Methode approximiert. Das wird durch den treppenförmigen Abstieg sichtbar. Während die Finite-Differenzen Methode ausgeführt wird, werden die aktuellen Werte der Parameter leicht verändert [LeV98], was durch die horizontale und leicht schwankende Bewegung der Linie zu sehen ist. Ist der Gradient approximiert, wird mit dem Optimierer ein Optimierungsschritt ausgeführt und der Fehler sinkt stark. Dies ist auch im Diagramm zu sehen anhand der sprunghaften Änderung des Fehlers.

Bei SPSA sieht man zu Beginn ein ähnliches Verhalten wie bei ADAM, obwohl der Gradient nicht mit der Finite-Differenzen Methode approximiert wird [Spa87]. Das liegt daran, dass die Implementierung von Qiskit zunächst eine Kalibrierungsphase hat, die 50 Funktionsauswertungen benötigt [AAA+19; KMT+17]. Das dient dazu die Schrittweite festzulegen für den *Gradient Descent* Schritt von SPSA [KMT+17].

5.2 Hybride Autoencoder

In Abschnitt 2.5.3 wurde bereits beschrieben, dass ein Autoencoder eine Art von neuronalem Netz ist. Er kann verwendet werden, um die Dimension von Daten zu reduzieren, was auch genutzt werden kann, um die Daten besser zu clustern. Da Quantencomputer für maschinelles Lernen mehrere Vorteile bieten können, wie in Kapitel 4 beschrieben, sollen hier Quantenschaltkreise verwendet werden in Autoencodern. Diese Schaltkreise werden aber nicht auf echter Quantenhardware ausgeführt, sondern in Simulatoren. In Abschnitt 4.3 wurde bereits beschrieben, dass es nötig

ist, für große Dimensionen hybride neuronale Netze zu verwenden, wenn nicht genügend Qubits zur Verfügung stehen. Das wird hier in den Experimenten genutzt, indem die Eingabe mit einer Schicht von einem klassischen neuronalen Netz auf 3 Dimensionen reduziert wird und mit einem Quantennetz weiter reduziert wird auf 2 Dimensionen. Anschließend wird mit einem Quantennetz wieder auf 3 Dimensionen erweitert und mit einer Schicht von einem klassischen neuronalen Netz auf die ursprünglich Anzahl an Dimensionen.

Im nächsten Abschnitt geht es um den genauen Aufbau des verwendeten hybriden Autoencoders. Danach werden die verwendeten Datensätze vorgestellt und schließlich die Experimente, die mit dem hybriden Autoencoder durchgeführt wurden für Dimensionsreduktion.

5.2.1 Aufbau

In Abbildung 5.6 ist der Aufbau des hybriden Autoencoders dargestellt, der in den nachfolgenden Experimenten genutzt wurde. Zuerst werden die Eingabedaten in ein klassisches neuronales Netz gegeben, welches die Dimensionen reduziert. Dieses besteht aus einer einzelnen, vollständig verbundenen Schicht und einer *Sigmoid-Aktivierungsfunktion*. Das ist eine nicht-lineare Funktion, die einen Wertebereich von [-1, 1] hat. Die Ausgabe des klassischen neuronalen Netzes wird mit *Angle Encoding* kodiert, sodass die Werte als Eingabe für ein Quantennetz genutzt werden können [LC20]. Der Quantenzustand nach dem Quantennetz wird gemessen, aber nicht alle Werte verwendet, da die Dimensionen weiter reduziert werden sollen. Diese Werte sind dann die Embeddings. Danach werden die Dimensionen wieder erhöht, erst durch ein Quantennetz, dann wieder durch ein klassisches neuronales Netz. Als Quantennetz kann ein beliebiger parametrisierter Quantenschaltkreis genutzt werden. In Unterabschnitt 5.1.1 wurde drei verschiedene Varianten vorgestellt.

5.2.2 Datensätze

Um realistische Ergebnisse zu erhalten, wurden hier echte Daten verwendet und keine synthetisch generierten. Dafür wurden die nachfolgenden Experimente mit drei verschiedenen Datensätzen durchgeführt.

Der erste Datensatz ist das *Hearth Disease Data Set* aus dem *UCI Machine Learning Repository* [DG17]. Hier wurde allerdings eine Version genutzt, die auf einer Teilmenge der Attribute basiert, da diese die Einzigen sind, die in veröffentlichten Experimenten genutzt wurden [JSP+88]. Dieser Datensatz enthält Daten von Patienten mit und ohne Herzkrankheit. In dieser Version von dem Datensatz befinden sich 14 Attribute, wovon eins der zugehörten Klasse entspricht. Die Klasse beschreibt in diesem Fall, ob die Werte zu einem gesunden Patienten gehören oder zu einem Patienten mit einer Herzkrankheit. Manche der Attribute sind numerische Werte, wie z. B. das Alter oder Blutdruck. Andere Attribute sind kategoriell wie z. B. das Geschlecht oder die Art der Brustschmerzen. Die kategoriellen Daten wurden mit *One-Hot Encoding* kodiert und mit der Zero-Mean Normalisierung normalisiert. Die numerischen Werte wurden ebenfalls mit der Zero-Mean Normalisierung normalisiert.

Der zweite Datensatz ist das *Breast Cancer Wisconsin (Diagnostic) Data Set* ebenfalls aus dem *UCI Machine Learning Repository* [DG17; WSM92]. Dieser Datensatz enthält Daten von Patienten mit gutartigem und bösartigem Brustkrebs. Der Datensatz hat 32 Attribute, wovon eines eine ID ist und

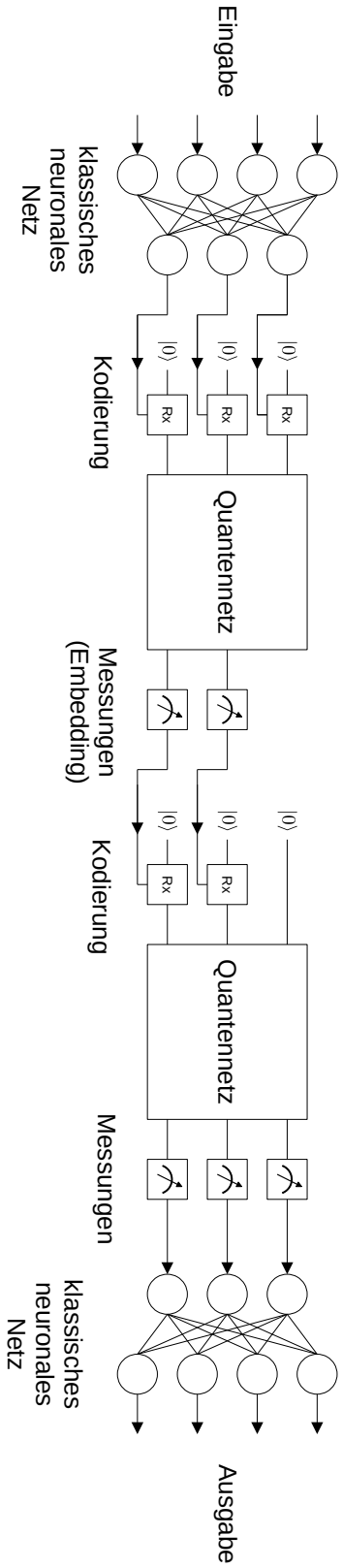

Abbildung 5.6: Hybrider Autoencoder. Darstellung inspiriert durch [LLW+19].

damit uninteressant ist und ein Attribut ist die Diagnose, welches die Klasse ist. Die Diagnose kann entweder “gutartig” oder “bösartig” sein in Bezug auf den Brustkrebs. Die restlichen Attribute sind numerische Werte, die die Eigenschaften von Zellkernen aus Gewebeproben beschreiben. Diese Werte werden mit Zero-Mean Normalisierung normalisiert.

Der dritte Datensatz ist der *Iris Species* Datensatz ebenfalls aus dem *UCI Machine Learning Repository* [DG17; Fis36]. Der Datensatz enthält Daten von drei verschiedenen Arten von Iris Pflanzen. Die Attribute sind die Länge und Breite von Kelchblatt und Kronblatt der Pflanze in Zentimetern und die Art der Iris Pflanze, welche der Klasse entspricht. Die Klasse wird weggelassen, dadurch bleiben bei diesem Datensatz nur numerische Attribute übrig, die ebenfalls mit Zero-Mean Normalisierung normalisiert werden.

Die Attribute der Datensätze, die die Klasse repräsentieren, werden für das Training immer weggelassen und später beim Clustering verwendet, um zu bewerten, wie gut die entstandenen Cluster sind, um die verschiedenen Verfahren miteinander vergleichen zu können.

5.2.3 Dimensionsreduktion

Hier werden die Ergebnisse präsentiert von den Experimenten mit dem vorgestellten hybriden Autoencoder. Wie gut der hybride Autoencoder im Vergleich mit einem klassischen Autoencoder abschneidet, soll anhand von zwei verschiedenen Dingen überprüft werden. Zum einen wird der Rekonstruktionsfehler betrachtet. Dieser gibt an, wie nahe die Ausgabewerte an den Eingabewerten liegen. Dies zeigt, wie gut ein Autoencoder gelernt hat, die Eingabedaten zu komprimieren und wieder zu dekomprimieren. Zum anderen wird betrachtet, wie gut sich die Embeddings clustern lassen, die der Autoencoder erzeugt. Dies ist auch das Ziel dieser Arbeit zu prüfen, ob hybride neuronale Netze genutzt werden können, um Clustering-Ergebnisse zu verbessern.

Rekonstruktionsfehler

Für die folgenden Experimente wurden vier Autoencoder miteinander verglichen.

Der Erste ist ein rein klassischer Autoencoder. Dieser besteht aus insgesamt vier vollständig verbundenen Schichten. Zwischen diesen Schichten ist jeweils eine Sigmoid-Aktivierungsfunktion. Die ersten beiden Schichten sind der Encoder und die anderen beiden der Decoder. Die erste Schicht hat so viele Eingabewerte, wie der vorverarbeitete Datensatz an Dimensionen hat. Ausgabewerte hat diese Schicht drei, die als Eingabe für die zweite Schicht dienen. Die zweite Schicht reduziert die Anzahl der Werte und damit auch Dimensionen auf zwei. Die Schichten des Decoders arbeiten genau andersherum.

Die anderen drei Autoencoder sind hybride Autoencoder, so wie am Anfang dieses Kapitels beschrieben. In der ersten Hälfte reduziert eine Schicht eines klassischen neuronalen Netzes die Daten auf drei Dimensionen und ein Quantennetz reduziert diese weiter auf zwei Dimensionen. Zwischen der klassischen und der Quantenschicht ist eine Sigmoid-Aktivierungsfunktion. Die zweite Hälfte, der Decoder, ist genauso aufgebaut, nur andersherum. Als Quantennetz werden die drei Varianten verwendet, die in Unterabschnitt 5.1.1 beschrieben wurden.

Die äußeren beiden Schichten sind bei allen Autoencoder gleich. Es sind Schichten von klassischen neuronalen Netzen mit Sigmoid Aktivierungsfunktion. Die Autoencoder unterscheiden sich nur bezüglich der inneren beiden Schichten. Beim Klassischen sind es auch klassische Schichten und bei den hybriden sind es Quantennetze. Die beiden inneren klassischen Sichten haben zusammen 12 Gewichte und 5 Biases, also 17 Parameter. Zwei Quantennetze der ersten Variante haben zusammen 90 Parameter, zwei der zweiten Variante haben 72 Parameter und zwei der dritten Variante haben 12 Parameter.

Die Datensätze werden vorverarbeitet, indem kategorielle Attribute One-Hot kodiert werden und alle Werte mit Zero-Mean Normalisierung normalisiert werden. Die Datensätze werden jeweils in Trainingsdaten und Testdaten aufgeteilt. 70% der Datenpunkte werden zufällig ausgewählt und werden als Trainingsdaten genutzt, die restlichen 30% werden als Testdaten genutzt. Trainiert wurden die Autoencoder jeweils für 100 Epochen. Eine *Epoche* bedeutet, dass alle Datenpunkte aus dem Trainingsdatensatz einmal zur Optimierung genutzt wurde. Der Rekonstruktionsfehler wurde berechnet als der mittlere quadratische Fehler zwischen den Eingabedaten und den Ausgabedaten des jeweiligen Autoencoders. Zur Optimierung wurde der Adam Optimierer genutzt, der in Abschnitt 2.4.1 beschrieben wurde. Er ist zwar langsamer als andere, wie in Unterabschnitt 5.1.2 gezeigt, aber er ist immer konvergiert und vor allem ist er in PyTorch [PGM+19] enthalten, was genutzt wurde für den klassischen Autoencoder und in Zusammenspiel mit Pennylane [BIS+20] für den hybriden Autoencoder verwendet wurde. Die Hyperparameter für Adam wurden so gewählt, wie sie in PyTorch standardmäßig festgelegt sind. Das bedeutet eine Lernrate von 0.001 und betas = (0.9, 0.999). Für die Quantennetze wurde die Lernrate allerdings auf 0.1 gesetzt, da das Training sonst zu langsam wäre. Bei den klassischen Bestandteilen hatten Tests mit einer höheren Lernrate zu schlechteren Ergebnissen geführt, deshalb wurde sie bei 0.001 belassen. Jede Variante von Autoencoder wurde für jeden Datensatz jeweils 5-mal trainiert, wobei jedes Mal die Parameter zufällig initialisiert wurden und auch der Datensatz zufällig in Trainingsdaten und Testdaten aufgeteilt wurde.

In Abbildung 5.7 sind die Ergebnisse für den *Hearth Disease Data Set* dargestellt. Deutlich zu sehen ist, dass der Rekonstruktionsfehler des klassischen Autoencoders deutlich größer ist als bei den hybriden Autoencodern mit den drei verschiedenen Quantennetzen. Der hybride Autoencoder mit der dritten Variante des Quantennetzes ist mit Abstand am besten. Die mit der ersten und zweiten Variante sind etwas schlechter, aber der Rekonstruktionsfehler ist trotzdem deutlich tiefer als beim klassischen Autoencoder.

In Abbildung 5.8 sind die Ergebnisse für den *Breast Cancer Wisconsin (Diagnostic) Data Set* dargestellt. Hier sind die hybriden Autoencoder wieder deutlich besser als der Klassische. Bei den hybriden Autoencodern ist der mit dem Quantennetz der zweiten Variante etwas besser als die mit den anderen Varianten, aber es schwankt etwas stärker als bei der dritten Variante. Mit der ersten Variante ist der Rekonstruktionsfehler schlechter als mit den anderen und schwankt auch deutlich stärker.

In Abbildung 5.9 sind die Ergebnisse für den *Iris Species* Datensatz dargestellt. Wie auch bei den beiden anderen Datensätzen sind die hybriden Autoencoder deutlich besser. Der mit der zweiten Variante des Quantennetzes ist wie beim vorherigen Datensatz am besten gefolgt von der dritten Variante. Wieder weist der Rekonstruktionsfehler des hybriden Autoencoders mit der dritten Variante die geringsten Schwankungen auf.

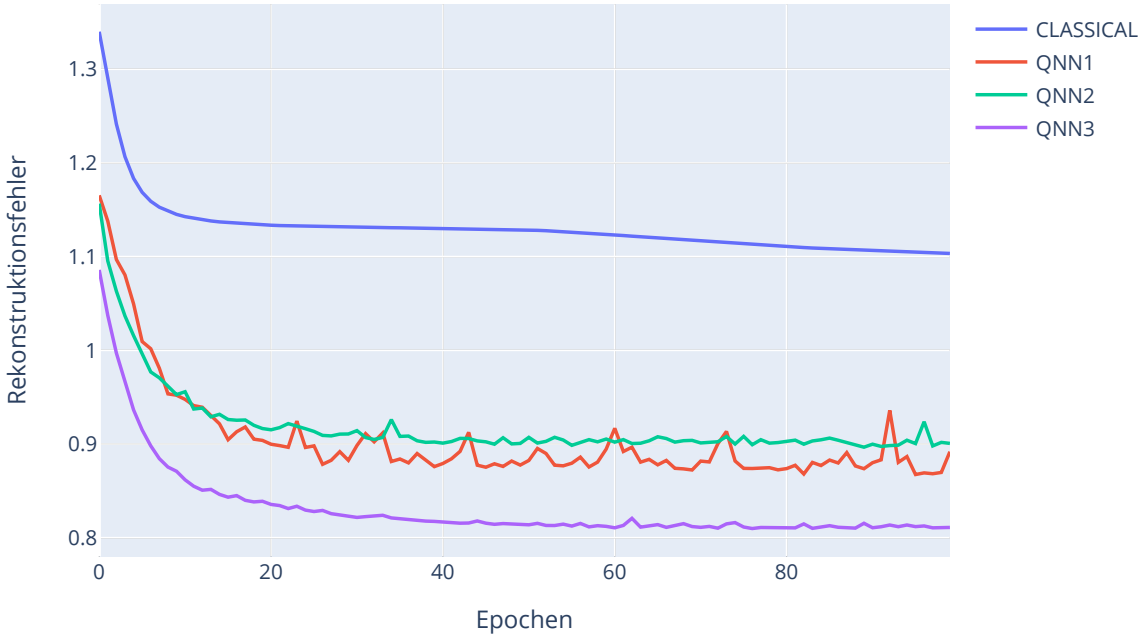

Abbildung 5.7: Rekonstruktionsfehler von klassischen und hybriden Autoencodern während des Trainings auf dem *Hearth Disease Data Set*. Dargestellt sind die Ergebnisse von dem klassischen Autoencoder und drei hybriden Autoencoder jeweils mit einer der drei Varianten von Quantennetzen, die vorgestellt wurden.

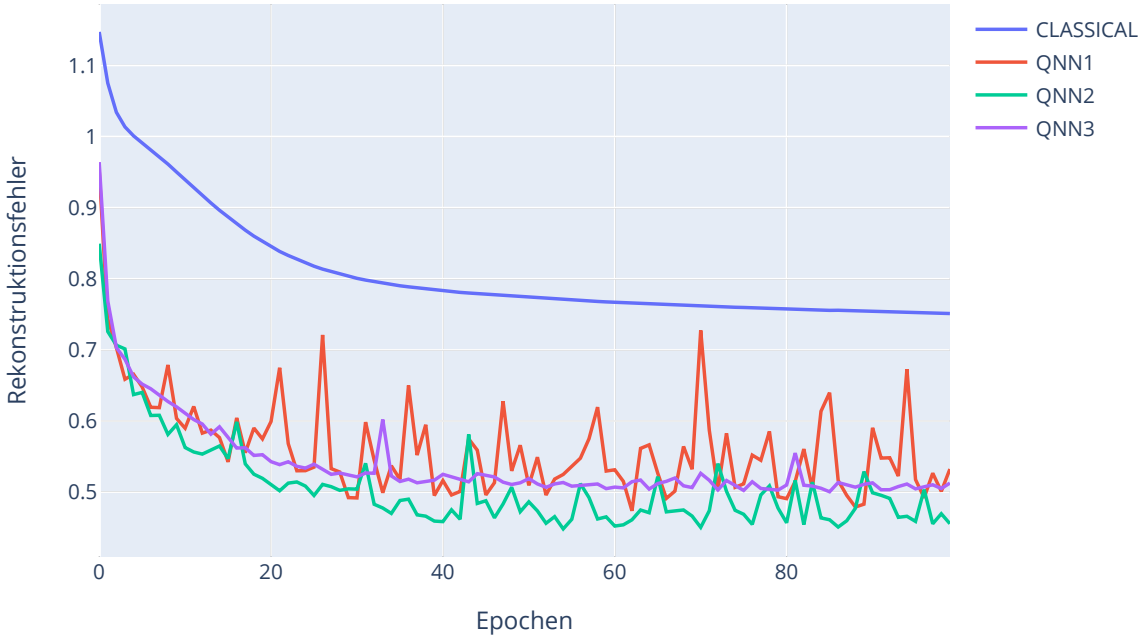

Abbildung 5.8: Rekonstruktionsfehler von klassischen und hybriden Autoencodern während des Trainings auf dem *Breast Cancer Wisconsin (Diagnostic) Data Set*. Dargestellt sind die Ergebnisse von dem klassischen Autoencoder und drei hybriden Autoencoder jeweils mit einer der drei Varianten von Quantennetzen, die vorgestellt wurden.

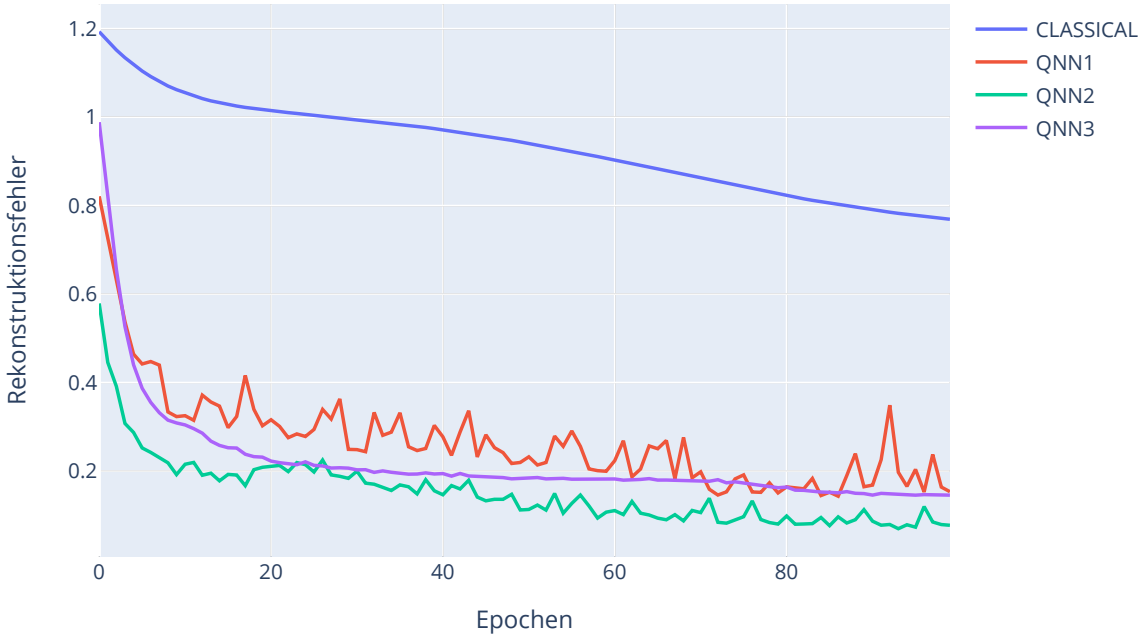

Abbildung 5.9: Rekonstruktionsfehler von klassischen und hybriden Autoencodern während des Trainings auf dem *Iris Species* Datensatz. Dargestellt sind die Ergebnisse von dem klassischen Autoencoder und drei hybriden Autoencoder jeweils mit einer der drei Varianten von Quantennetzen, die vorgestellt wurden.

Die Experimente haben gezeigt, dass hybride Autoencoder besser darin sein können, den Rekonstruktionsfehler zu reduzieren als klassische. Mit allen drei verwendeten Varianten von Quantennetzen waren die Ergebnisse besser auf allen drei Datensätzen. An der Anzahl der Parameter kann es nicht liegen, da sowohl Varianten mit mehr als auch mit weniger Parametern genutzt wurden als bei dem klassischen Autoencoder.

Visuelle Darstellung der Embeddings

Um eine Vorstellung davon zu bekommen, wie die Embeddings aussehen, die die hybriden Autoencoder produzieren, werden hier einige dargestellt. Es wurde ein hybrider Autoencoder mit Quantennetz der dritten Variante genutzt auf dem *Hearth Disease Data Set*. Da die Embeddings jeweils aus zwei reellen Werten bestehen, lassen sie sich als Punkte in einem zwei-dimensionalen Raum darstellen. In dem Datensatz sind die Datenpunkte zwei Klassen zugeordnet. Diese Informationen wurden für das Training nicht verwendet, werden aber in den Diagrammen dargestellt, damit man einen Eindruck davon bekommen kann, ob die Punkte der beiden Klassen sich in unterschiedlichen Regionen ansammeln, was ein Hinweis darauf wäre, dass sich die Punkte gut clustern lassen und die entstanden Cluster sinnvoll sind.

In Abbildung 5.10 sind die Embeddings nach einer Epoche Training dargestellt. Die Punkte der zwei verschiedenen Klassen sind noch sehr durcheinander und sie ließen sich voraussichlich nicht gut clustern.

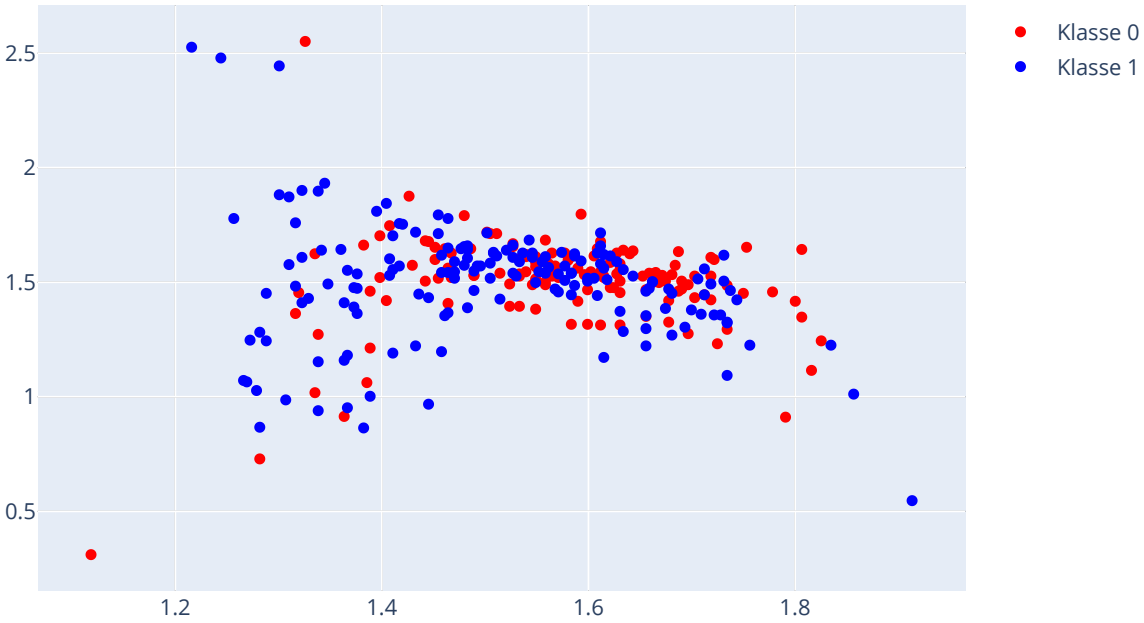

Abbildung 5.10: Embeddings aus einem hybriden Autoencoder, der eine Epoche lang mit dem *Hearth Disease Data Set* trainiert wurde. Die Klassenzuteilung ist aus dem Datensatz und wurde nicht für das Training verwendet.

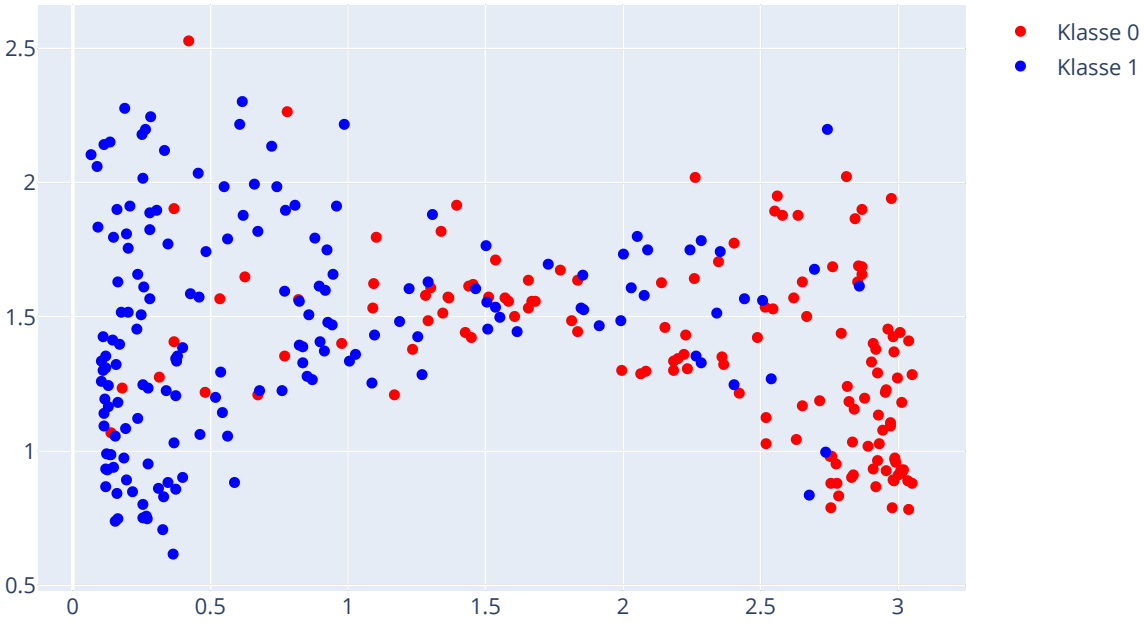

Abbildung 5.11: Embeddings aus einem hybriden Autoencoder, der 10 Epochen lang mit dem *Hearth Disease Data Set* trainiert wurde. Die Klassenzuteilung ist aus dem Datensatz und wurde nicht für das Training verwendet.

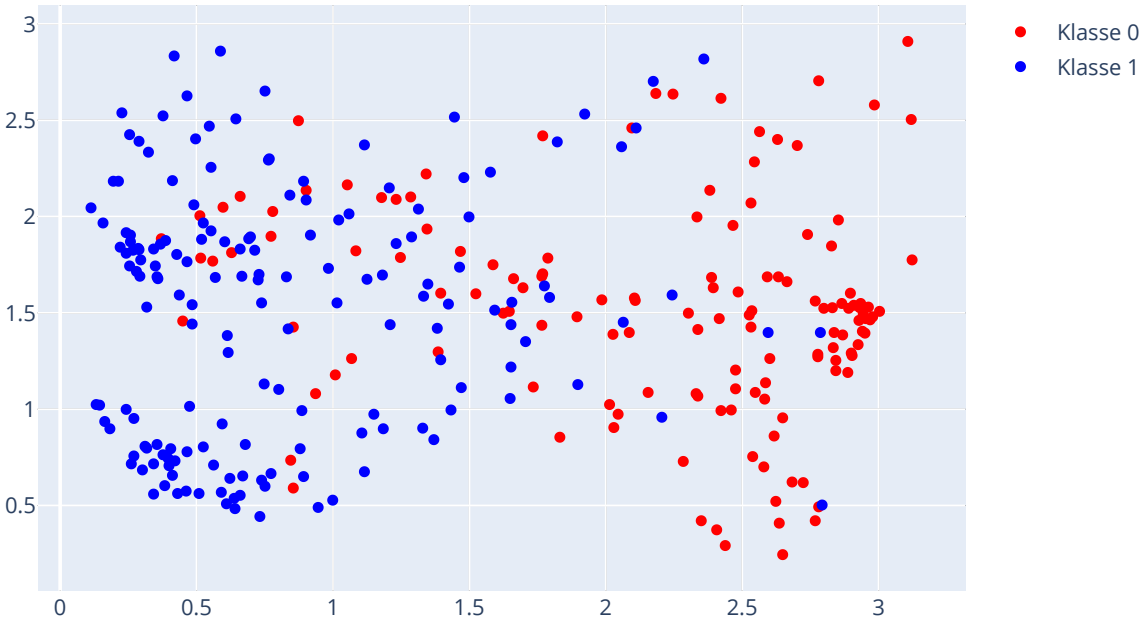

Abbildung 5.12: Embeddings aus einem hybriden Autoencoder, der 100 Epochen lang mit dem *Hearth Disease Data Set* trainiert wurde. Die Klassenzuteilung ist aus dem Datensatz und wurde nicht für das Training verwendet.

In Abbildung 5.11 wurde der Autoencoder schon 10 Epochen lang trainiert und es ist schon deutlich zu erkennen, dass auf der linken Seite eher die Embeddings der Klasse 1 liegen und auf der rechten Seite die der Klasse 0.

In Abbildung 5.12 sind schon 100 Epochen vergangen, aber die Aufteilung sieht nicht besser aus als nach 10 Epochen, aber die Embeddings der Klasse 1 scheinen sich in zwei Gruppen aufgeteilt zu haben.

Clustering

Hier geht es um das Ziel dieser Arbeit. Es soll anhand von Experimenten geprüft werden, ob hybride neuronale Netze Clustering-Ergebnisse verbessern können. Das wird getestet, indem die Anzahl der Dimensionen der Datensätze reduziert werden mit klassischen und hybriden Autoencodern. Die Embeddings, die von diesen Autoencodern berechnet werden, werden dann mit Clustering-Algorithmen geclustert. Wie gut die Cluster sind wird mit *Adjusted Mutual Information* (zu Deutsch "angepasste gegenseitige Information") gemessen, der die entstandenen Cluster vergleicht mit den Klassen, zu denen die Datenpunkte gehören. Die Informationen über die Klassen sind in den Datensätzen enthalten, wurden aber nicht für das Training der Autoencoder genutzt. Die angepasste gegenseitige Information wurde als Maß gewählt, da sie bei einer zufälligen Zuteilung von Clustern einen Wert von 0 ausgibt, einen maximal Wert von 1 hat und nicht sensibel ist bezüglich der Anzahl der Cluster oder Datenpunkte [VEB10].

Im ersten Experiment sollte herausgefunden werden, welcher Clustering-Algorithmus am besten die Embeddings clustern kann, damit dieser für die nachfolgenden Experimente genutzt werden kann. Dafür wurde der *Hearth Disease Data Set* genutzt und der hybride Autoencoder mit der dritten

Clustering Algorithmus	Argumente
Affinity Propagation	damping=0.5, max_iter=200, convergence_iter=15, preference=None, affinity=euclidean
Agglomerative Clustering	n_clusters=2, affinity=euclidean, linkage=average, distance_threshold=None
DBSCAN	eps=0.5, min_samples=5, metric=euclidean, p=None
k-Means	n_clusters=2, init=k-means++, n_init=10, max_iter=300, tol=1e-4, algorithm=auto
OPTICS	min_samples=5, max_eps=np.inf, metric=minkowski, p=2, cluster_method=xi, eps=None, xi=0.05, min_cluster_size=None

Tabelle 5.2: Verwendete Clustering-Algorithmen und die genutzten Argumente.

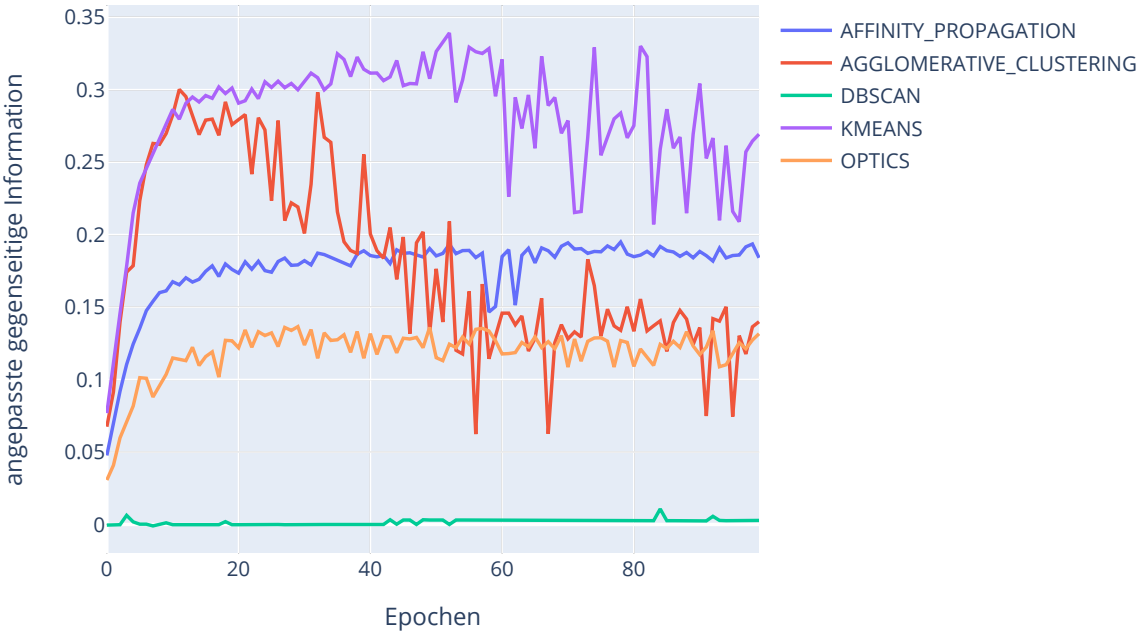

Abbildung 5.13: Vergleich verschiedener Clustering-Algorithmen auf den Embeddings eines hybriden Autoencoders mit einem Quantennetz der dritten Variante. Für jede Epoche, die die Autoencoder auf dem *Hearth Disease Data Set* trainiert wurden, wurden die Embeddings erstellt und Cluster berechnet. Der Durchschnitt von fünf Durchläufen ist hier dargestellt.

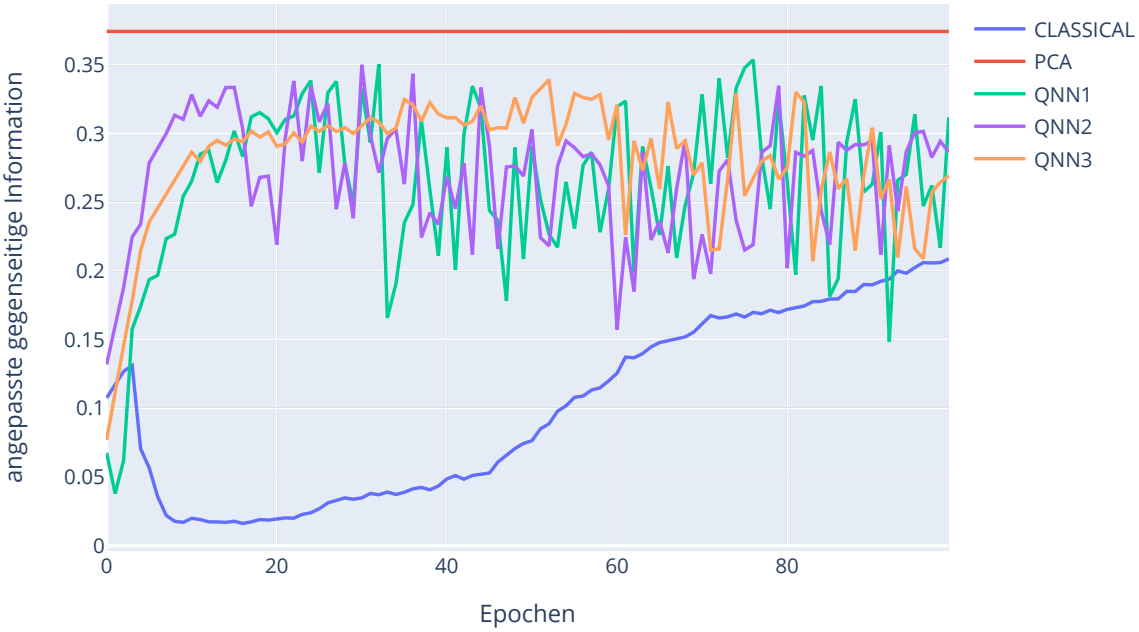

Abbildung 5.14: Clustering Ergebnisse auf dem *Hearth Disease Data Set*. Die angepasste gegenseitige Information wurde für die k-Means Cluster der Embeddings bestimmt. Außerdem wurde als zusätzlicher Vergleich noch das Ergebnis mit PCA dargestellt.

Variante von dem Quantennetz, da der Rekonstruktionsfehler sehr niedrig war im Vergleich mit dem klassischen Autoencoder und es weniger Schwankungen gab als mit den anderen Varianten von Quantennetzen.

In Tabelle 5.2 sind die verwendeten Clustering-Algorithmen aufgelistet, inklusive der Argumente, die verwendet wurden. Zu den Argumenten gehören unteranderem die Anzahl der Cluster, die Distanzmetrik, die Iterationen und die Mindestanzahl an Elementen pro Cluster. Die Argumente wurden auf den Standardeinstellungen belassen, wie sie in der Scikit-learn [PVG+11] Softwarebibliothek verwendet wurden, da die Implementierungen aus dieser genutzt wurden. Lediglich die Anzahl der Cluster wurde auf die Anzahl der Klassen im jeweiligen Datensatz gesetzt, wenn dies möglich war.

In Abbildung 5.13 sind die Ergebnisse dargestellt. Während des Trainings der Autoencoder wurden nach jeder Epoche die Embeddings erstellt, geclustert und mit der angepassten gegenseitigen Information bewertet. Man sieht, dass DBSCAN es nicht geschafft hat, sinnvolle Cluster zu erstellen, da der Wert immer nahe der 0 ist. OPTICS und Affinity Propagation haben es geschafft, Cluster zu finden, die besser sind als eine zufällige Zuteilung und die Werte blieben auch zum Ende hin relativ konstant. Agglomerative Clustering hat erst deutlich bessere Ergebnisse geliefert, ist dann aber wieder stark eingebrochen. K-Means zu Anfang genauso gute Ergebnisse geliefert, ist aber nicht so stark eingebrochen, aber starke Schwankungen gab es trotzdem zum Ende hin. Da k-Means während des gesamten Trainings einer der besten bzw. der beste Algorithmus war, wurde er für die weiteren Experimente genutzt, da es zu aufwendig gewesen wäre, alle Experimente mit allen Clustering-Algorithmen durchzuführen.

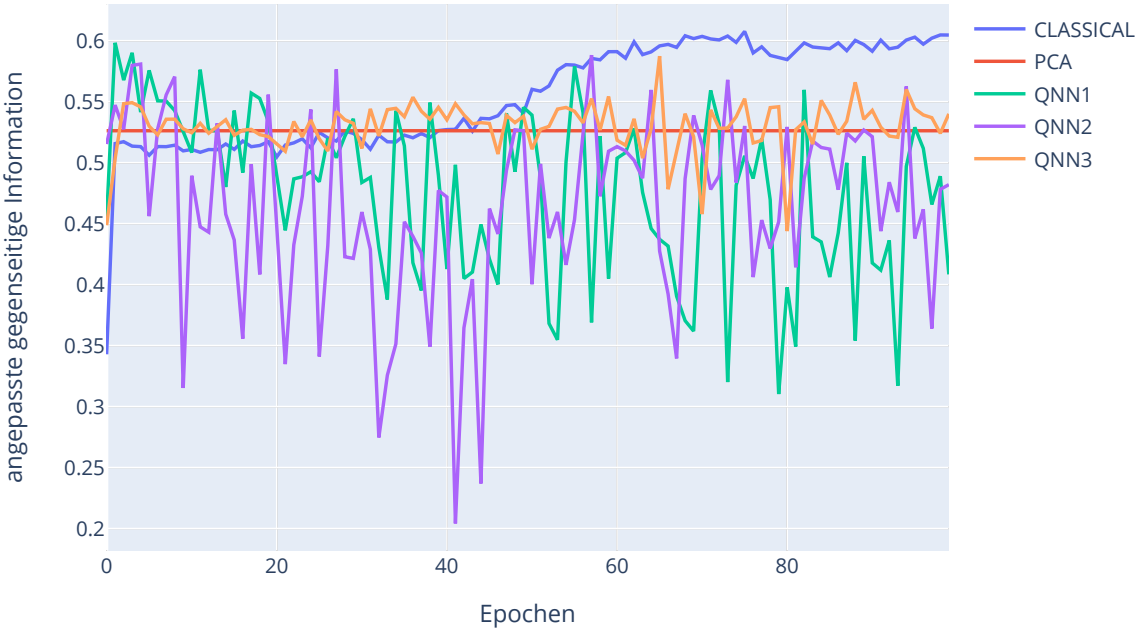

Abbildung 5.15: Clustering Ergebnisse auf dem *Breast Cancer Wisconsin (Diagnostic) Data Set*. Die angepasste gegenseitige Information wurde für die k-Means Cluster der Embeddings bestimmt. Außerdem wurde als zusätzlicher Vergleich noch das Ergebnis mit PCA dargestellt.

In Abbildung 5.14 sind die Cluster Ergebnisse für das *Hearth Disease Data Set* dargestellt. Es wurde die angepasste gegenseitige Information berechnet für die Cluster die k-Means erstellt hat aus den Embeddings und aus der Dimensionsreduktion mit PCA. PCA wurde als zusätzlicher Vergleich eingefügt, um einen Vergleich zu haben mit einem Algorithmus, der nicht auf Autoencodern basiert. Für die Ergebnisse mit den Embeddings wurden die Embeddings nach jeder Epoche, die die Autoencoder trainiert wurden, berechnet und geclustert. Da es für PCA nur einen Wert gibt, wurde dieser als horizontale Linie eingezeichnet, damit es sich besser mit den anderen vergleichen lässt. Hier ist der klassische Autoencoder zu Anfang sehr schlecht im Vergleich zu den anderen und erholt sich zwar im Laufe des Trainings, bleibt aber schlechter als die anderen. Die hybriden Autoencoder sind hier besser als der Klassische, schwanken aber stark und liegen unter PCA.

In Abbildung 5.15 sind die Cluster Ergebnisse für das *Breast Cancer Wisconsin (Diagnostic) Data Set* dargestellt. Man sieht, dass der klassische Autoencoder die besten Ergebnisse erbringt gegen Ende des Trainings. Die Ergebnisse der hybriden Autoencoder schwanken stark und die mit den Quantennetzen der zweiten und dritten Variante liegen meist unter PCA. Der mit der dritten Variante des Quantennetzes schwankt am wenigsten im Vergleich mit den anderen beiden und liegt oft über PCA, aber immer noch deutlich unter dem klassischen Autoencoder.

In Abbildung 5.16 sind die Cluster Ergebnisse für den *Iris Species* Datensatz. Hier sieht es mit dem klassischen Autoencoder ähnlich aus wie beim ersten Datensatz. Erst bricht es stark ein, dann erholt es sich wieder ein bisschen, ist aber trotzdem noch schlechter als alle anderen. Die hybriden Autoencoder sind schnell besser als PCA, aber sinken dann wieder. Untereinander sind die hybriden Autoencoder relativ ähnlich vom Ergebnis her und durch die Schwankungen lässt sich nicht eindeutig sagen, welche Version am besten ist.

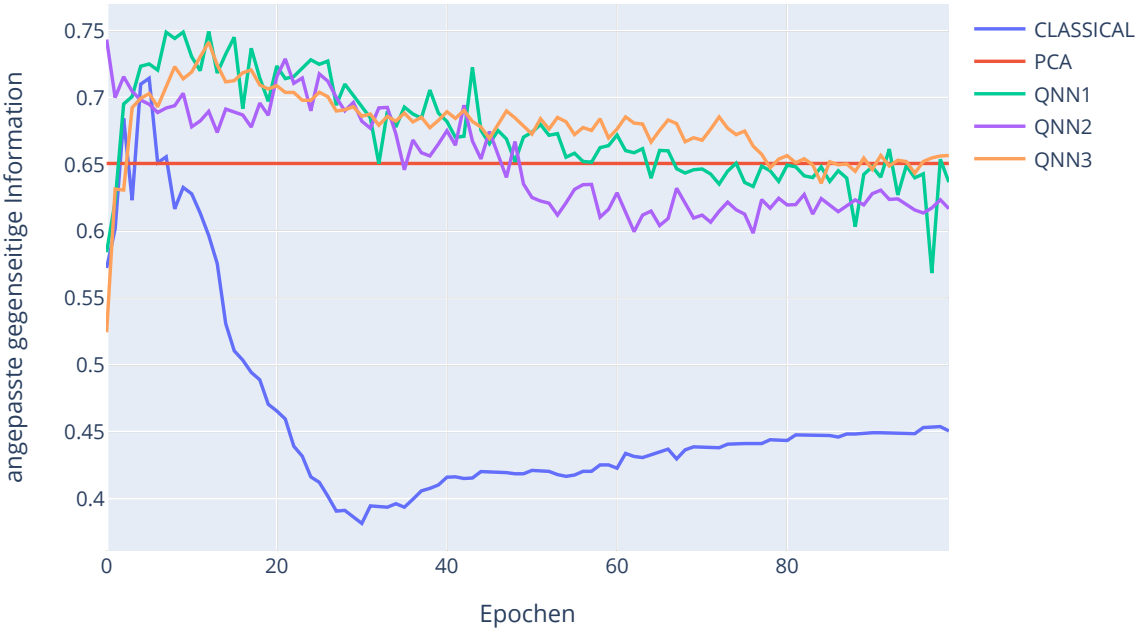

Abbildung 5.16: Clustering Ergebnisse auf dem *Iris Species* Datensatz. Die angepasste gegenseitige Information wurde für die k-Means Cluster der Embeddings bestimmt. Außerdem wurde als zusätzlicher Vergleich noch das Ergebnis mit PCA dargestellt.

Zusammenfassend lässt sich sagen, dass sich die Embeddings aus den hybriden Autoencodern nutzen lassen, um sie mit klassischen Clustering Algorithmen zu clustern. Die Cluster sind, gemessen an der angepassten gegenseitigen Information, bei zwei von drei Datensätzen besser mit den Embeddings aus den hybriden Autoencodern als aus dem klassischen Autoencoder. Beim ersten Datensatz war PCA besser als alle Autoencoder und beim zweiten Datensatz war der klassische Autoencoder zwar besser als PCA, aber die hybriden nur stellenweise. Beim dritten Datensatz hingegen konnten die hybriden Autoencoder nach wenigen Epochen schon PCA übertreffen, sind dann aber wieder schlechter geworden.

Wenn man die Rekonstruktionsfehler mit den Clustering-Ergebnissen vergleicht, sieht man, dass ein besserer Rekonstruktionsfehler nicht unbedingt bedeutet, dass auch die Cluster besser sind. Beispielsweise sind bei dem *Breast Cancer Wisconsin (Diagnostic) Data Set* die Rekonstruktionsfehler der hybriden Autoencoder deutlich besser als der Rekonstruktionsfehler des klassischen Autoencoders. Trotzdem sind die Cluster, die k-Means aus den Embeddings des klassischen Autoencoders erstellt, besser als die aus den hybriden Autoencodern. Das kann ein Hinweis darauf sein, dass es *overfitting* gibt. In diesem Fall würde das bedeutet, dass sich die hybriden Autoencoder zu stark an die Aufgabe anpassen, den Rekonstruktionsfehler zu minimieren, sodass sie schlechtere Ergebnisse beim Clustering erzielen.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde untersucht, ob hybride neuronale Netze genutzt werden können, um die Ergebnisse von klassischen Clustering Verfahren zu verbessern. Es wurden verschiedene Varianten von Quantennetzen vorgestellt, die genutzt werden können als Alternative zu klassischen neuronalen Netzen. Mit einer dieser Varianten wurden verschiedene Optimierer getestet, um zu sehen, wie lange diese brauchen, um das Quantennetz so zu optimieren, dass es einen bestimmten Wert ausgibt. Dabei wurden sowohl gradientenbasierte als auch gradientenfreie Optimierer verwendet. Unter den gradientenbasierten Optimierern hat SPSA besonders gut abgeschnitten, da es den Gradienten selbst approximiert auf eine effiziente Weise und es immer konvergiert ist. Bei den gradientenfreien Optimierern waren COBYLA und NFT besonders schnell und auch schneller als alle gradientenbasierten.

Es wurde gezeigt, wie sich ein hybrider Autoencoder aus klassischen neuronalen Netzen und Quantennetzen aufbauen lässt. Dieser hybride Autoencoder wurde mit den drei Varianten von Quantennetzen getestet und auch mit einem klassischen Autoencoder verglichen. Der Rekonstruktionsfehler war für alle Datensätze bei den hybriden Autoencodern niedriger war als beim klassischen Autoencoder.

Die Embeddings eines hybriden Autoencoders wurden im zwei-dimensionalen Raum dargestellt. Dabei konnte man erkennen, dass die Embeddings sich so im Raum verteilen, dass Embeddings die zu der gleichen Klasse gehören, durchschnittlich näher beieinander liegen als Embeddings, die zu unterschiedlichen Klassen gehören.

Wie gut sich die Embeddings clustern lassen wurde auf drei verschiedenen Datensätzen getestet. Dabei wurden die hybriden Autoencoder wieder mit dem klassischen Autoencoder und zusätzlich mit PCA verglichen. Bei einem Datensatz war PCA am besten, bei einem der klassische Autoencoder und beim letzten waren die hybriden Autoencoder am besten. Der letzte Datensatz war derjenige mit den wenigsten Attributen im Vergleich mit den anderen. Außerdem enthält er nur numerische Attribute und keine kategoriellen Attribute, im Gegensatz zu den anderen. Ein weiterer Unterschied ist, dass er drei verschiedene Klassen enthält, anstatt zwei wie die anderen Datensätze. Ob diese Eigenschaften einen Einfluss darauf hatten, dass der hybride Autoencoder auf diesem Datensatz die besten Ergebnisse erzielt hat, ist eine Frage, die zukünftige Forschungsarbeiten beantworten können.

Aus den Experimenten lässt sich folgern, dass hybride Autoencoder in bestimmten Fällen die Clustering Ergebnisse verbessern können im Vergleich zu klassischen Autoencodern und PCA. Es gibt aber auch noch Verbesserungspotenzial. Beispielsweise könnte getestet werden, wie es sich auswirkt, mehrere Quantennetze hintereinander zuschalten oder größere Quantennetze mit mehr Qubits zu verwenden oder mehr klassische Schichten zur Vorverarbeitung zu verwenden.

Literaturverzeichnis

[AAA+16] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al. „Deep speech 2: End-to-end speech recognition in english and mandarin“. In: *International conference on machine learning*. PMLR, 2016, S. 173–182 (zitiert auf S. 13).

[AAA+19] H. Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110 (zitiert auf S. 35, 36, 38).

[ABK99] M. Ankerst, M. M. Breunig, H.-P. Kriegel. „OPTICS: Ordering Points To Identify the Clustering Structure“. In: *ACM Sigmod record* 28.2 (1999), S. 49–60 (zitiert auf S. 15).

[Alp16] E. Alpaydin. *Machine learning: the new AI*. en. MIT Press essential knowledge series. Cambridge, MA: MIT Press, 2016. ISBN: 978-0-262-52951-8 (zitiert auf S. 11, 13, 14).

[ASZ+20] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, S. Woerner. „The power of quantum neural networks“. In: *arXiv:2011.00027 [quant-ph]* (Okt. 2020). arXiv: 2011.00027. URL: http://arxiv.org/abs/2011.00027 (zitiert auf S. 30, 34, 35).

[BIS+20] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, K. McKiernan, J. J. Meyer, Z. Niu, A. Száva, N. Killoran. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2020. arXiv: 1811.04968 [quant-ph] (zitiert auf S. 42).

[BPP13] S. Bhatnagar, H. Prasad, L. Prashanth. *Stochastic Recursive Algorithms for Optimization*. Bd. 434. Lecture Notes in Control and Information Sciences. Springer London, 2013. ISBN: 978-1-4471-4284-3. DOI: 10.1007/978-1-4471-4285-0. URL: http://link.springer.com/10.1007/978-1-4471-4285-0 (zitiert auf S. 20).

[Bra20] U. Braga-Neto. *Fundamentals of Pattern Recognition and Machine Learning*. en. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-27655-3 978-3-030-27656-0. DOI: 10.1007/978-3-030-27656-0. URL: http://link.springer.com/10.1007/978-3-030-27656-0 (besucht am 29. 01. 2021) (zitiert auf S. 11, 13–15, 22, 23).

[BSOM05] F. Bortolozzi, A. de Souza Britto Jr, L. S. Oliveira, M. Morita. „Recent advances in handwriting recognition“. In: *Document analysis* (2005), S. 1–31 (zitiert auf S. 13).

[BWP+17] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd. „Quantum Machine Learning“. In: *Nature* 549.7671 (Sep. 2017). arXiv: 1611.09347, S. 195–202. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature23474 (zitiert auf S. 11, 29).

[CANS21] K. Chaudhury, A. Ashok, S. Narumanchi, D. Shankar. *Math and Architectures of Deep Learning*. Manning, 2021 (zitiert auf S. 13, 14, 22).

[CKP+15] M. Crawford, T. M. Khoshgoftaar, J. D. Prusa, A. N. Richter, H. A. Najada. „Survey of review spam detection using machine learning techniques“. In: *Journal of Big Data* 2.11 (Dez. 2015), S. 1–24. ISSN: 2196-1115. DOI: 10.1186/s40537-015-0029-9 (zitiert auf S. 13).

[CWZ+14] F.-q. Chen, Y. Wu, G.-d. Zhao, J.-m. Zhang, M. Zhu, J. Bai. „Contractive Denoising Auto-Encoder“. In: *Intelligent Computing Theory*. Hrsg. von D.-S. Huang, V. Bevilacqua, P. Premaratne. Bd. 8588. Lecture Notes in Computer Science. Springer International Publishing, 2014, S. 776–781. ISBN: 978-3-319-09332-1. DOI: 10.1007/978-3-319-09333-8_84. URL: http://link.springer.com/10.1007/978-3-319-09333-8_84 (zitiert auf S. 24).

[Dah18] P. Dahal. „Learning Embedding Space for Clustering From Deep Representations“. In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, S. 3747–3755. DOI: 10.1109/BigData.2018.8622629 (zitiert auf S. 23, 24).

[DG17] D. Dua, C. Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml (zitiert auf S. 39, 41).

[EKX96] M. Ester, H.-P. Kriegel, X. Xu. „A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise“. In: 96.34 (1996), S. 226–231 (zitiert auf S. 15).

[FD07] B. J. Frey, D. Dueck. „Clustering by Passing Messages Between Data Points“. In: *Science* 315.5814 (Feb. 2007), S. 972–976. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1136800 (zitiert auf S. 15).

[FGG14] E. Farhi, J. Goldstone, S. Gutmann. „A Quantum Approximate Optimization Algorithm“. In: *arXiv:1411.4028 [quant-ph]* (Nov. 2014). arXiv: 1411.4028. URL: http://arxiv.org/abs/1411.4028 (zitiert auf S. 25).

[Fis36] R. A. Fisher. *Iris Data Set*. https://www.kaggle.com/uciml/iris. 1936 (zitiert auf S. 41).

[For19] D. Forsyth. *Applied Machine Learning*. en. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-18113-0 978-3-030-18114-7. DOI: 10.1007/978-3-030-18114-7. URL: http://link.springer.com/10.1007/978-3-030-18114-7 (besucht am 29. 01. 2021) (zitiert auf S. 11, 14–18, 21–23).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016 (zitiert auf S. 11, 13, 14, 23).

[GLZY17] X. Guo, X. Liu, E. Zhu, J. Yin. „Deep clustering with convolutional autoencoders“. In: *International conference on neural information processing*. Springer. 2017, S. 373–382 (zitiert auf S. 24).

[Har12] P. Harrington. *Machine learning in action*. Manning Publications Co., 2012 (zitiert auf S. 11, 13–15, 22).

[HKP11] J. Han, M. Kamber, J. Pei. „Data mining concepts and techniques third edition“. In: *The Morgan Kaufmann Series in Data Management Systems* 5.4 (2011), S. 83–124 (zitiert auf S. 21–23).

[JSP+88] A. Janosi, W. Steinbrunn, M. Pfisterer, R. Detrano, D. W. Aha. *Heart Disease Data Set*. https://www.kaggle.com/ronitf/heart-disease-uci. 1988 (zitiert auf S. 39).

[KB14] D. P. Kingma, J. Ba. „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980* (2014) (zitiert auf S. 20).

[KMT+17] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, J. M. Gambetta. „Hardware-efficient Variational Quantum Eigensolver for Small Molecules and Quantum Magnets“. In: *Nature* 549.7671 (Sep. 2017). arXiv: 1704.05018, S. 242–246. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature23879 (zitiert auf S. 38).

[Kra91] M. A. Kramer. „Nonlinear Principal Component Analysis Using Autoassociative Neural Networks“. In: *AIChE Journal* 37.2 (1991), S. 11 (zitiert auf S. 23).

[Kri06] D. Kriesel. „Neuronale Netze“. In: *Rheinische Friedrich-Wilhelms-Universität* (2006) (zitiert auf S. 18).

[LB20] F. Leymann, J. Barzen. „The bitter truth about gate-based quantum algorithms in the NISQ era“. In: *Quantum Science and Technology* 5.4 (2020), S. 044007 (zitiert auf S. 11).

[LBOM98] Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller. „Efficient backprop“. In: *Neural networks: Tricks of the trade*. Springer, 1998, S. 9–50 (zitiert auf S. 18, 19).

[LC20] R. LaRose, B. Coyle. „Robust data encodings for quantum classifiers“. In: *Physical Review A* 102.3 (Sep. 2020). arXiv: 2003.01695, S. 032420. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.102.032420 (zitiert auf S. 11, 25, 27, 39).

[LeV98] R. J. LeVeque. „Finite difference methods for differential equations“. In: *Draft version for use in AMath* 585.6 (1998), S. 112 (zitiert auf S. 18, 19, 33, 36, 38).

[LLW+19] J. Liu, K. H. Lim, K. L. Wood, W. Huang, C. Guo, H.-L. Huang. „Hybrid Quantum-Classical Convolutional Neural Networks“. In: *arXiv:1911.02998 [quant-ph]* (Nov. 2019). arXiv: 1911.02998. URL: http://arxiv.org/abs/1911.02998 (zitiert auf S. 40).

[Mac+67] J. MacQueen et al. „Some methods for classification and analysis of multivariate observations“. In: 1.14 (1967), S. 281–297 (zitiert auf S. 15).

[MBI+20] A. Mari, T. R. Bromley, J. Izaac, M. Schuld, N. Killoran. „Transfer learning in hybrid classical-quantum neural networks“. In: *Quantum* 4 (Okt. 2020). arXiv: 1912.08278, S. 340. ISSN: 2521-327X. DOI: 10.22331/q-2020-10-09-340 (zitiert auf S. 31).

[MBK20] A. Mari, T. R. Bromley, N. Killoran. „Estimating the gradient and higher-order derivatives on quantum hardware“. In: *arXiv:2008.06517 [quant-ph]* (Aug. 2020). arXiv: 2008.06517. URL: http://arxiv.org/abs/2008.06517 (zitiert auf S. 19).

[MCCD13] T. Mikolov, K. Chen, G. Corrado, J. Dean. „Efficient Estimation of Word Representations in Vector Space“. In: *arXiv:1301.3781 [cs]* (Sep. 2013). arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781 (zitiert auf S. 24).

[NFT20] K. M. Nakanishi, K. Fujii, S. Todo. „Sequential minimal optimization for quantum-classical hybrid algorithms“. In: *Physical Review Research* 2.4 (Okt. 2020). arXiv: 1903.12166, S. 043158. ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.2.043158 (zitiert auf S. 21).

[NM65] J. A. Nelder, R. Mead. „A simplex method for function minimization“. In: *The computer journal* 7.4 (1965), S. 308–313 (zitiert auf S. 20).

[Ols09] F. Olsson. „A literature survey of active machine learning in the context of natural language processing“. In: (2009) (zitiert auf S. 13).

[Pet08] J. Peters. „Machine learning for motor skills in robotics“. In: *KI-Künstliche Intelligenz* 2008.4 (2008), S. 41–43 (zitiert auf S. 13).

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems 32*. Hrsg. von H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett. Curran Associates, Inc., 2019, S. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (zitiert auf S. 42).

[PMS+14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O'Brien. „A variational eigenvalue solver on a quantum processor“. In: *Nature Communications* 5.1 (Sep. 2014). arXiv: 1304.3061, S. 4213. ISSN: 2041-1723. DOI: 10.1038/ncomms5213 (zitiert auf S. 25).

[Pow07] M. J. Powell. „A view of algorithms for optimization without derivatives“. In: *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications* 43.5 (2007), S. 170–174 (zitiert auf S. 20).

[Pow94] M. J. D. Powell. „A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation“. In: *Advances in Optimization and Numerical Analysis*. Hrsg. von S. Gomez, J.-P. Hennart. Mathematics and Its Applications. Springer Netherlands, 1994, S. 51–67. ISBN: 978-94-015-8330-5. DOI: 10.1007/978-94-015-8330-5_4. URL: https://doi.org/10.1007/978-94-015-8330-5_4 (zitiert auf S. 20).

[Pow98] M. J. D. Powell. „Direct search algorithms for optimization calculations“. In: *Acta Numerica* 7 (Jan. 1998), S. 287–336. ISSN: 0962-4929, 1474-0508. DOI: 10.1017/S0962492900002841 (zitiert auf S. 20).

[Pre18] J. Preskill. „Quantum Computing in the NISQ era and beyond“. In: *Quantum* 2 (Aug. 2018), S. 79. DOI: 10.22331/q-2018-08-06-79 (zitiert auf S. 25).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830 (zitiert auf S. 48).

[Ran71] W. M. Rand. „Objective criteria for the evaluation of clustering methods“. In: *Journal of the American Statistical association* 66.336 (1971), S. 846–850 (zitiert auf S. 16).

[RBGE18] P. Rodríguez, M. A. Bautista, J. Gonzàlez, S. Escalera. „Beyond one-hot encoding: Lower dimensional target embedding“. In: *Image and Vision Computing* 75 (Juli 2018), S. 21–31. ISSN: 02628856. DOI: 10.1016/j.imavis.2018.04.004 (zitiert auf S. 22).

[RDGF16] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. „You Only Look Once: Unified, Real-Time Object Detection“. In: IEEE, Juni 2016, S. 779–788. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.91. URL: http://ieeexplore.ieee.org/document/7780460/ (zitiert auf S. 13).

[ROA17] J. Romero, J. P. Olson, A. Aspuru-Guzik. „Quantum autoencoders for efficient compression of quantum data“. In: *Quantum Science and Technology* 2.4 (Dez. 2017). arXiv: 1612.02806, S. 045001. ISSN: 2058-9565. DOI: 10.1088/2058-9565/aa8072 (zitiert auf S. 11, 29, 33, 34).

[RP11] E. Rieffel, W. Polak. *Quantum computing: a gentle introduction*. Scientific and engineering computation. The MIT Press, 2011. ISBN: 978-0-262-01506-6 (zitiert auf S. 11, 25–27).

[RS13] L. M. Rios, N. V. Sahinidis. „Derivative-free optimization: a review of algorithms and comparison of software implementations“. In: *Journal of Global Optimization* 56.3 (Juli 2013). Company: Springer Distributor: Springer Institution: Springer Label: Springer number: 3 publisher: Springer US, S. 1247–1293. ISSN: 1573-2916. DOI: 10.1007/s10898-012-9951-y (zitiert auf S. 20).

[SBG+19] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, N. Killoran. „Evaluating analytic gradients on quantum hardware“. In: *Physical Review A* 99.3 (März 2019). arXiv: 1811.11184, S. 032331. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.99.032331 (zitiert auf S. 31, 33, 36).

[Sci21] Scikit. *Scikit RobustScaler*. Accessed: 2021-02-26. 2021. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html (zitiert auf S. 22).

[Ser21] L. G. Serrano. *Grokking Machine Learning*. Manning, 2021 (zitiert auf S. 11, 13, 14, 17).

[SP18] M. Schuld, F. Petruccione. *Supervised Learning with Quantum Computers*. en. Quantum Science and Technology. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-96423-2 978-3-319-96424-9. DOI: 10.1007/978-3-319-96424-9. URL: http://link.springer.com/10.1007/978-3-319-96424-9 (besucht am 07. 12. 2020) (zitiert auf S. 11, 19, 29, 30).

[Spa+92] J. C. Spall et al. „Multivariate stochastic approximation using a simultaneous perturbation gradient approximation“. In: *IEEE transactions on automatic control* 37.3 (1992), S. 332–341 (zitiert auf S. 20).

[Spa87] J. C. Spall. „A stochastic approximation technique for generating maximum likelihood parameter estimates“. In: *1987 American control conference*. IEEE, 1987, S. 1161–1167 (zitiert auf S. 20, 36, 38).

[SPG+17] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, C.-T. Lin. „A review of clustering techniques and developments“. en. In: *Neurocomputing* 267 (Dez. 2017), S. 664–681. ISSN: 09252312. DOI: 10.1016/j.neucom.2017.06.053. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231217311815 (besucht am 11. 11. 2020) (zitiert auf S. 16).

[SW18] J. A. Snyman, D. N. Wilke. *Practical Mathematical Optimization*. Bd. 133. Springer Optimization and Its Applications. Springer International Publishing, 2018. ISBN: 978-3-319-77585-2. DOI: 10.1007/978-3-319-77586-9. URL: http://link.springer.com/10.1007/978-3-319-77586-9 (zitiert auf S. 18, 19).

[VEB10] N. X. Vinh, J. Epps, J. Bailey. „Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance“. In: *The Journal of Machine Learning Research* 11 (2010), S. 2837–2854 (zitiert auf S. 16, 46).

[WSM92] W. H. Wolberg, W. N. Street, O. L. Mangasarian. *Breast Cancer Wisconsin (Diagnostic Data Set)*. https://www.kaggle.com/uciml/breast-cancer-wisconsin-data. 1992 (zitiert auf S. 39).

[WSS+19] M. Watabe, K. Shiba, M. Sogabe, K. Sakamoto, T. Sogabe. „Quantum Circuit Parameters Learning with Gradient Descent Using Backpropagation“. In: *arXiv:1910.14266 [physics, physics:quant-ph]* (Nov. 2019). arXiv: 1910.14266. URL: http://arxiv.org/abs/1910.14266 (zitiert auf S. 11, 25).

[WW07] S. Wagner, D. Wagner. *Comparing clusterings: an overview*. Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007 (zitiert auf S. 16, 17).

[XGF16] J. Xie, R. Girshick, A. Farhadi. „Unsupervised deep embedding for clustering analysis“. In: *International conference on machine learning*. PMLR. 2016, S. 478–487 (zitiert auf S. 24).

[ZZXW19] Z.-Q. Zhao, P. Zheng, S.-t. Xu, X. Wu. „Object Detection with Deep Learning: A Review“. In: *arXiv:1807.05511 [cs]* (Apr. 2019). arXiv: 1807.05511. URL: http://arxiv.org/abs/1807.05511 (zitiert auf S. 13).

Alle URLs wurden zuletzt am 20. 03. 2021 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift