HAW HAMBURG

Bachelorarbeit

Leo Alexander König

Simulation der Steuerung eines optischen Pulsformers mit Deep Reinforcement Learning

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Leo Alexander König

Simulation der Steuerung eines optischen Pulsformers mit Deep Reinforcement Learning

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus Jünemann
Zweitgutachter: Dr. Tim Laarmann

Eingereicht am: 16. Juni 2023

Leo Alexander König

Thema der Arbeit

Simulation der Steuerung eines optischen Pulsformers mit Deep Reinforcement Learning

Stichworte

Reinforcement Learning, Pulsformer, Simulation, Regelungstechnik

Kurzzusammenfassung

Es werden die Umsetzung und verschiedene Trainingskonfigurationen vorgestellt um einen möglichst kurzen Puls zu formen. Der Puls soll anhand der Informationen seines Spektrogramms geformt werden. Die getesteten Reinforcement Learning Lösungen, führen zu einer Verringerung der Pulsbreite, jedoch erreichen sie im Gegensatz zur bisher angewandten Methode nicht die gewünschte Pulsebreite. Es werden Ansätze zur Verbesserung der Simulation und der Reinforcement Learning Lösung vorgestellt.

Leo Alexander König

Title of Thesis

Simulation of the control of a optical pulse shaper with Deep Reinforcement Learning

Keywords

Reinforcement Learning, Pulseshaping, Simulation, Control Technology

Abstract

The implementation and different training configurations to shape a pulse as short as possible are presented. The pulse is to be shaped based on the information of its spectrogram. The tested Reinforcement Learning solutions lead to a reduction of the pulse width but unlike the method used so far they do not achieve the desired pulse width. Approaches to improve the simulation and the Reinforcement Learning solution are presented.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

DFT Diskrete Fourier Transformation.

FT Fourier Transformation.

GDD Group Delay Dispersion.

ML Machine Learning.

MSE Mean Squared Error.

RL Reinforcement Learning.

TOD Third Order Dispersion.

1 Zielsetzung der Arbeit

In dieser Arbeit geht es darum, die ultrakurzen Pulse eines multichromatischen Lasers zu formen. In den letzten Jahren ist es Forscherteams gelungen, Pulse mit einer Dauer von wenigen Femtosekunden herzustellen.

Mit Hilfe solcher Pulse lässt sich eine ganze Reihe wichtiger ultraschneller Prozesse studieren, die von den kohärenten Anregungen in Halbleitern über das ultraschnelle Verhalten von Supraleitern bis zu den chemischen und biologischen Elementarreaktionen reicht.

In der Molekularforschung werden die geformten Laserpulse unter anderem dazu benutzt, eine chemische Reaktion zu starten, um eine gewünschte Substanz herzustellen.

Ultrakurze Pulse sind stark von Dispersionseffekten betroffen. Die Verbreiterung des Pulses durch die Dispersionseffekte soll durch das entsprechende Formen des Pulses verhindert werden.

Bisher wird für das Erzeugen und Formen von möglichst kurzen Pulsen ein Optimierungsverfahren nach dem Evolutionsprinzip verwendet. Ziel ist es, herauszufinden ob Deep Reinforcement Learning ein möglicher Ansatz für diese Aufgabenstellung ist. Die Deep Reinforcement Learning Lösung wird mit dem Optimierungsverfahren in Bezug auf die erreichte Pulsbreite und Anzahl der Zyklen zum Einstellen der Parameter verglichen.

Eine Reduzierung der Zyklen ist wünschenswert, da das Messen und Auswerten der Daten zum Anpassen der Parameter sehr zeitaufwendig ist.
Der neue Ansatz des Deep Reinforcement Learning soll nur anhand des aktuellen Spektrogramms die Parameter so einstellen, dass ein möglichst kurzer Puls erzeugt wird.

Wenn es gelingt, nur mit den Informationen des Spektrogramms den Puls entsprechend zu formen, ist es wahrscheinlich, dass dies auch für weitere erwünschte Pulsformen möglich ist.

2 Grundlagen

2.1 Fourier Transformation

In Bezug auf Laserlicht kann die Fourier Transformation (FT) verwendet werden, um die spektrale Zusammensetzung des Lichts zu analysieren. Ein Laser erzeugt Licht, das in einem sehr engen Frequenzbereich konzentriert ist. Im Gegensatz dazu, ist natürliches Licht, also das Licht der Sonne, über einen breiteren Frequenzbereich verteilt.

Die FT ermöglicht es, die auftretenden Frequenzen im Laserlicht zu untersuchen. Das Ergebnis der FT des Laserlichts ist ein Spektrum, aus dem ein Spektrogramm erzeugt wird. Das Spektrogramm zeigt die Amplituden der auftretenden Frequenzen im Verlauf der Zeit.

Ein Computer kann nur mit diskretisierten Werten arbeiten und benutzt daher die Diskrete Fourier Transformation (DFT). Das Transformationspaar zum Umrechnen eines Signals der Länge N in den Zeitbereich x_n bzw. in den Frequenzbereich X_k wird durch folgende Formeln beschrieben:

$$x_n = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X_k \cdot e^{\frac{i2\pi kn}{N}} \tag{2.1}$$

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi kn}{N}} \tag{2.2}$$

2.2 Dispersion

Dispersion bezieht sich auf die Eigenschaft von verschiedenen Frequenzkomponenten im Licht, die sich unterschiedlich schnell fortbewegen, wenn sie durch bestimmte Materialien wie Glas oder Kristalle geleitet werden. Dies führt zu einer Verbreiterung des Pulses.

2.2.1 Ursprung der Dispersion

Der Brechungsindex, n, eines Materials ist definiert als:

$$n(\omega) = \frac{c_0}{c} \tag{2.3}$$

wobei c_0 die Lichtgeschwindigkeit im Vakuum und c die Lichtgeschwindigkeit in dem jeweiligen Material ist. In allen Materialien ist $c < c_0$ und somit $n > 1$. Außerdem ist der Brechungsindex n eines Materials frequenzabhängig. Das ist der physikalische Ursprung der Dispersion. Wenn verschiedene Farben des Lichts unterschiedliche Brechungsindizes aufweisen, dann können höherfrequente blaue Farben sich in dem Material nur langsamer ausbreiten als die roteren Farben mit niedrigeren Frequenzen. Der Fall in dem sich Licht mit höheren Frequenzen langsamer ausbreitet, wird als normale Dispersion bezeichnet. Die umgekehrte Situation, in der sich höherfrequentes Licht schneller ausbreitet, wird als anomale Dispersion bezeichnet.
In beiden Fällen führt die Dispersion dazu, dass ein kurzer Puls, der viele Farben enthält, mit der Zeit immer breiter wird, wenn er ein Medium durchläuft. [22]

Bei herkömmlichen Laserpulsen (10^{-9} Sekunden) ist Dispersion in der Regel nicht problematisch, da die Bandbreite so gering ist, dass die Auswirkungen der Dispersion im Wesentlichen zu vernachlässigen sind. Bei der Arbeit mit ultrakurzen Pulsen (10^{-15} Sekunden) wird die Dispersion zu einem kritischen Faktor. Der Zusammenhang zwischen zeitlicher und spektraler Breite wird durch die Fourier-Transformations-Limitierung beschrieben.

2.2.2 Bandbreitenlimitierter Puls

Die Fourier-Transformations-Limitierung ist das Produkt aus zeitlicher und spektraler Breite. Sie besagt, dass ein ultrakurzer Puls eine große Bandbreite hat.

$$\Delta t \cdot \Delta v \geq const. \tag{2.4}$$

Es ist nicht die kurze zeitliche Breite, die dazu führt, dass ultrakurze Pulse stärker von Dispersion betroffen sind. Vielmehr sind breitbandige Pulse stärker von der Dispersion betroffen und ultrakurze Pulse sind notwendigerweise breitbandig.
Ein bandbreitenlimitierter Puls oder auch fourierlimitierter Puls (engl. Transform-limited pulse) ist ein optischer Puls, der eine minimale zeitliche Ausdehnung aufweist und somit die kürzestmögliche Pulsdauer für einen bestimmten spektralen Bereich hat. Bei einem bandbreitenlimitiertem Puls treten alle vorhandenen Frequenzkomponenten zur gleichen Zeit auf. Er wird auch als Fourier-limited pulse bezeichnet, da er die Fourier-Transformations-Limitierung widerspiegelt. [22]

2.2.3 Chirp

Um die Auswirkungen der Dispersion auf einen ultraschnellen Puls qualitativ zu verstehen, kann man sich einen bandbreitenlimitierten Puls mit einer sehr großen Bandbreite, der die Frequenzkomponenten von Dunkelrot bis Tiefblau enthält, vorstellen.
Lässt man diesen Puls durch ein Stück Glas mit normaler Dispersion laufen, verlangsamen sich sowohl die blauen als auch die roten Frequenzkomponenten. Die roten Frequenzkomponenten verlangsamen sich aber weniger als die blauen. Wenn der Puls betrachtet wird, nachdem er das Glas durchquert hat, würden mehr rote Komponenten an der Vorderseite des Pulses zu sehen sein, und mehr blaue Komponenten an der Rückseite des Pulses. Die Farben kommen nicht mehr zur gleichen Zeit an, der Puls ist nicht mehr fourierlimitiert. Da die verschiedenen Frequenzkomponenten zeitlich gestreut wurden, muss sich der Puls verbreitert haben. Dieser Effekt ist stärker bei Pulsen mit größerer Bandbreite und bei Materialien mit einer höheren Dispersion.
Nach dem Durchgang durch ein Medium mit positiver Dispersion treffen die roten Frequenzkomponenten vor den blauen Frequenzkomponenten an. Dieser Fall ist in der Abbildung 2.1 dargestellt. An diesem Punkt spricht man von einem gechirpten Puls. [22]

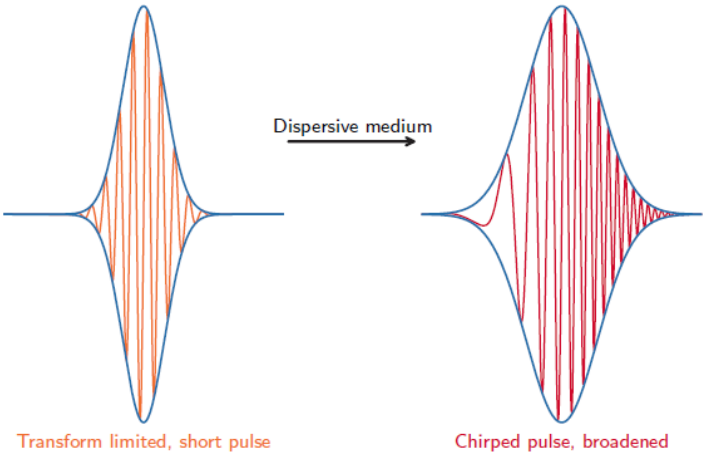

Abbildung 2.1: Gechirpter Puls [22]

2.2.4 Trägerwelle und Hüllkurve

Die Hüllkurve (engl. Pulse Envelope), hier die Gauß-Verteilung, stellt die zeitliche Form des Pulses dar und ist in der Abbildung 2.2 blau dargestellt. Die Trägerwelle (engl. Carrier Wave) ist die Überlagerung aller Frequenzkomponenten des Pulses und hier orange dargestellt. Das Produkt aus Trägerwelle und Hüllkurve stellt das elektrische Feld des Pulses dar. [22]

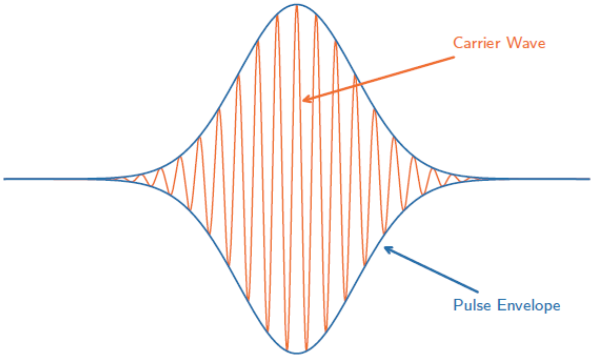

Abbildung 2.2: Trägerwelle und Hüllkurve [22]

2.2.5 Mathematischer Hintergrund der Dispersionseffekte

Das elektrische Feld im Zeitbereich kann durch die Fourier Transformation als die Überlagerung von allen vorkommenden Frequenzen ausgedrückt werden:

$$E(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \mathcal{E}(\omega) e^{i\omega t} d\omega \tag{2.5}$$

Der Term $e^{i\omega t}$ ist eine einfache Schwingung mit der Frequenz ω und dem Faktor $\mathcal{E}(\omega)$, der die Amplitude der jeweiligen Frequenz vorgibt.

Wenn ein Puls ein Material durchläuft, erfährt jede einzelne Frequenz eine Phasenverschiebung $\phi(\omega)$. Der Wert $\phi(\omega)$ wird als spektrale Phase bezeichnet und enthält die Phase von jeder Farbe des Farbspektrums in dem Puls.
Mathematisch betrachtet wird die Phasenverschiebung der Welle $e^{i\omega t}$ durch das multiplizieren mit dem Phasenfaktor $e^{i\phi(\omega)}$ ausgedrückt:

$$e^{i\omega t} \xrightarrow{\text{Ausbreitung}} e^{i\omega t} \cdot e^{i\phi(\omega)} \tag{2.6}$$

Das elektrische Feld des Pulses wird nun ausgedrückt als:

$$E(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \mathcal{E}(\omega) e^{i(\omega t + \phi(\omega))} d\omega \tag{2.7}$$

Die Spektrale Phase $\phi(\omega)$ kann als Taylor-Reihe um die Mittenfrequenz ω_0 ausgedrückt werden:

$$\phi(\omega) = \phi(\omega_0) + \frac{\delta\phi}{\delta\omega}\bigg|_{\omega=\omega_0} \cdot(\omega-\omega_0) + \frac{\delta^2\phi}{2\delta\omega^2}\bigg|_{\omega=\omega_0} \cdot(\omega-\omega_0)^2 + \frac{\delta^3\phi}{6\delta\omega^3}\bigg|_{\omega=\omega_0} \cdot(\omega-\omega_0)^3 + ... \tag{2.8}$$

Die Terme höherer Ordnung können in der Regel vernachlässigt werden. [22]
Im Folgenden wird die Bedeutung der einzelnen Terme erklärt:

Term nullter Ordnung

Dies ist die Phase der Mittenfrequenz und sorgt für die Hüllkurve.

Term erster Ordnung

Dieser Term wird Gruppenlaufzeit (engl. group delay) genannt und hat eine Zeiteinheit. Die Gruppenlaufzeit gibt die Zeit an, die der Puls braucht, um sich durch das Medium auszubreiten. Eine Änderung der Gruppenlaufzeit bewirkt eine Änderung der zeitlichen Position des Pulses, ändert aber nicht die Form der Pulshüllkurve, führt also nicht zu einer Verbreiterung. [22]

Term zweiter Ordnung

Dieser Term wird Gruppenlaufzeitsdispersion (engl. Group Delay Dispersion (GDD)) genannt und hat eine Zeiteinheit im Quadrat. GDD ist der Term niedrigster Ordnung, der für die Verbreiterung von Pulsen durch Dispersion verantwortlich ist. Er ist die Hauptquelle der durch Dispersion hervorgerufenen Verbreiterung des Pulses. Dieser Term stellt die quadratische Phase dar. Wenn er vorhanden ist bedeutet dies, dass die Phasendifferenz zwischen den Komponenten des Spektrums von der Frequenz abhängt. Es ist also nicht mehr möglich, die feste Phasenbeziehung zu erreichen, die alle Frequenzen gleichzeitig ankommen lässt und einen bandbreitenlimitierten Puls ergibt. [22]

Term dritter Ordnung

Dieser Term wird Dispersion dritter Ordnung (engl. Third Order Dispersion (TOD)) genannt und hat eine Zeiteinheit mit der dritten Potenz. TOD hat einen ähnlichen, aber schwächeren Effekt als GDD. [22]

2.3 Deep Learning

2.3.1 Was ist Deep Learning?

Deep Learning ist eine Unterkategorie des Machine Learning (ML), bei der künstliche neuronale Netze verwendet werden, um Muster in großen Datenmengen zu erkennen und automatisch zu lernen.

Im Gegensatz zum traditionellen maschinellen Lernen, das auf manuell erstellten Merkmalen basiert, kann Deep Learning automatisch Merkmale aus Rohdaten extrahieren.

Deep Learning hat in den letzten Jahren durch die Verfügbarkeit von großen Datensätzen und leistungsfähiger Hardware, enorm an Popularität gewonnen und hat in vielen Bereichen wie der Bild- und Spracherkennung, der automatischen Übersetzung und der Mustererkennung in medizinischen Daten signifikante Fortschritte ermöglicht. [15]

2.3.2 Unterschiede zwischen Supervised Learning, Unsupervised Learning und Reinforcement Learning

Supervised learning ist ein Bereich des ML, bei dem ein Algorithmus mit Trainingsdaten geschult wird, um eine Vorhersage oder Klassifikation für neue, unbekannte Daten zu treffen. Die Trainingsdaten enthalten sowohl Eingaben als auch die von einem Experten korrekt klassifizierten oder bezeichneten Ausgaben. Der Algorithmus lernt, indem er die Beziehung zwischen den Eingaben und Ausgaben analysiert und versucht, diese Beziehung auf neue, unbekannte Eingaben anzuwenden.

Unsupervised learning ist ein Bereich des ML, bei dem der Algorithmus keine korrekten Ausgaben für die Trainingsdaten hat. Stattdessen versucht der Algorithmus, strukturelle Beziehungen oder Muster in den Daten selbst zu entdecken. Beispiele für unsupervised learning-Verfahren sind Clustering und Dimensionenreduktion. [19]

Reinforcement Learning (RL) ist ein Bereich des ML, bei dem ein Algorithmus durch die Interaktion mit seiner Umgebung lernt, indem er Belohnungen oder Bestrafungen erhält. Der Algorithmus lernt durch die Ausführung von Aktionen und das Beobachten der Ergebnisse, um die Belohnung zu maximieren. RL findet häufig in realen Szenarien Anwendung, in denen ein Agent, z.B. ein Roboter oder ein computergestütztes System,

eine bestimmte Aufgabe ausführen muss, während er mit verschiedenen Umgebungsbedingungen interagiert. Anwendungsbeispiele sind selbstfahrende Autos, der Handel mit Aktien oder auch die Regelungstechnik in Industrie und Forschung. [16]

2.3.3 Struktur eines tiefen Neuronalen Netzes

Neuron

Das Neuron ist ein elementares Bauteil im ML. Ein Neuron hat beliebig viele Eingänge, die jeweils gewichtet sind. Die gewichteten Eingänge werden aufsummiert und zu einem weiteren Wert, dem sogenannten Bias addiert. Der berechnete Wert wird durch die Anwendung einer Aktivierungsfunktion zu dem Output des Neurons. In der Abbildung 2.3 wird eine Stufenfunktion als Aktivierungsfunktion eingesetzt. Der Bias ist in diesem Fall der negative Wert der Verschiebung in X-Richtung. Sobald die Summe von Eingängen und Bias größer als Null ist, wird eine Eins von dem Neuron ausgegeben. Andernfalls eine Null.

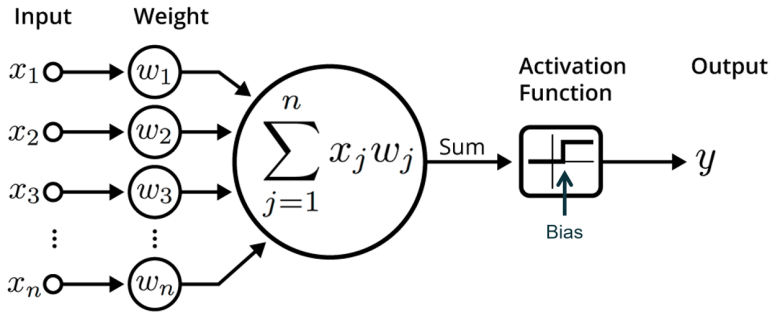

Abbildung 2.3: Aufbau eines Neurons [1]

Voll verbundene Schicht

Eine voll verbundene Schicht (engl. Dense Layer), beschreibt die Parallelschaltung von mehreren Neuronen. Die Anzahl der Neuronen kann beliebig gewählt werden. Jedes Neuron hat seine eigenen einstellbaren Parameter, also Gewichte und Bias. Die Eingangswerte der voll verbundenen Schicht sind an jedes Neuron über das entsprechende Gewicht gekoppelt. Die voll verbundene Schicht hat so viele Ausgangswerte wie sie Neuronen hat.

Aktivierungsfunktion

Eine Aktivierungsfunktion ist eine mathematische Funktion, die auf die Ausgabe eines Neurons oder einer Schicht in einem neuronalen Netzwerk angewendet wird. Die Aktivierungsfunktion dient dazu, eine Nichtlinearität in das Modell einzuführen. Das ermöglicht dem Netzwerk, komplexere Zusammenhänge zwischen den Eingabedaten und den Ausgaben zu modellieren.
Die im Abschnitt “Neuron“ beschriebene Sprungfunktion eignet sich nicht für das ML, da sie nur zwei Zustände annehmen kann und somit zu wenig Informationen überträgt.

Häufig wird die ReLU (Rectified Linear Unit) Aktivierungsfunktion aus Abbildung 2.4 benutzt.

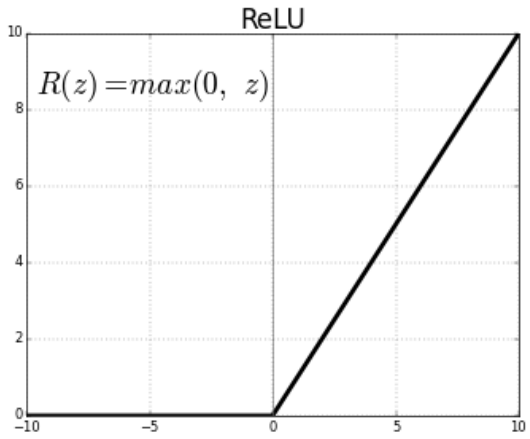

Abbildung 2.4: ReLU-Funktion

Die ReLU-Funktion ist definiert als $f(x) = max(0, z)$ wobei z der Ausgabewert des Neurons ist.

Eine weitere Aktivierungsfunktion ist die lineare Funktion, die einfach den Ausgangswert des Neurons weitergibt, ohne ihn zu verändern. Diese Aktivierungsfunktion wird im Bereich des RL angewendet.

Batch-Normalisierung

Die Batch-Normalisierung ist eine Methode zur Normalisierung der Eingaben eines neuronalen Netzwerks, um die Trainingszeit zu verkürzen und die Stabilität des Netzwerks zu erhöhen. Eine Batch-Normalisierungs-Schicht wird üblicherweise nach einer Schicht mit linearen Operationen, also einer voll verbundenen oder Faltungsschicht und vor einer Aktivierungsfunktion eingefügt.
Die Batch-Normalisierung funktioniert, indem sie die Eingaben der Schicht auf einen standardisierten Wertebereich bringt. Es wird der Durchschnitt und die Standardabweichung der Eingaben in einem Minibatch berechnet. Anschließend werden die Eingaben skaliert, verschoben und an die nächste Schicht weitergegeben.[15]

Dropout

Eine Dropout-Schicht ist eine Methode zur Regularisierung von neuronalen Netzwerken, die dazu beiträgt, Overfitting zu vermeiden.
Overfitting tritt auf, wenn ein Modell zu viele Parameter, im Verhältnis zu den verfügbaren Trainingsdaten hat. Das Modell passt sich dann zu sehr an die Trainingsdaten an und kann nicht gut verallgemeinern.
Die Idee hinter der Dropout-Schicht ist, dass während des Trainings eines neuronalen Netzwerks einige der Neuronen zufällig ausgeschaltet werden, indem ihre Ausgabe auf Null gesetzt wird. Dies wird in jeder Trainingsepoche mit einer bestimmten Wahrscheinlichkeit für jedes Neuron durchgeführt. Dadurch werden einige der Verbindungen im Netzwerk temporär unterbrochen, was dazu führt, dass das Netzwerk gezwungen wird, sich an verschiedene Merkmale der Daten anzupassen, anstatt sich auf bestimmte Merkmale zu spezialisieren.
Die Dropout-Schicht hilft dabei, das Problem des Overfitting zu lösen und verbessert die Leistung des Modells auf neuen Daten.

Für die Vorhersagen außerhalb des Trainings, wird die Dropout-Schicht deaktiviert und alle Neuronen werden aktiviert, um eine Vorhersage für neue Daten zu generieren. [15]

2.3.4 Faltungsschicht

Eine Faltungsschicht oder auch Convolutional Layer besteht aus einer beliebigen Anzahl von Filtern oder auch Feature Maps genannt. Ein Filter hat einen Filterkern, häufig in der Form eines 3x3 Blocks. Jedes Feld des Filterkerns hat eine Gewichtung.
Eine Faltungsschicht erwartet als Eingang ein zweidimensionales Bild, das entweder einen oder drei Farbkanäle hat. Der Filterkern jedes Filters fährt mit einer einstellbaren Schrittweite über das Eingangsbild hinweg. Bei jedem Schritt wird eine Faltung des Filterkerns mit dem Eingangsbild durchgeführt. Dabei werden die Gewichte des Filterkerns mit dem jeweils darunter liegenden Pixel des Bildes multipliziert und anschließend aufsummiert. Das Ergebnis der Faltung wird dann in das Ausgabebild des Filters geschrieben. Die Größe des Ausgangsbildes verkleinert sich je nach Größe des Filterkerns und der festgelegten Schrittweite, mit dem der Filter über das Bild fährt.

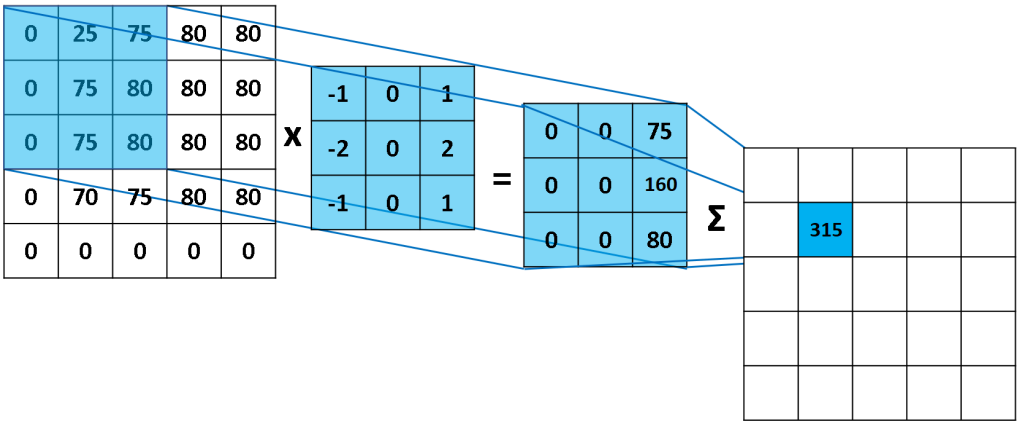

Abbildung 2.5: Filterkern einer Faltungsschicht [29]

Die Anzahl der einstellbaren Parameter einer Faltungsschicht ist deutlich geringer als die einer voll verbundenen Schicht.

Bei Bildern mit drei Farbkanälen verdreifacht sich entsprechend die Anzahl der Parameter, da jeder Farbkanal seine eigenen Filterkerne hat. [13]

Die Filter in der Faltungsschicht lernen während des Trainingsprozesses, bestimmte Merkmale des Eingangsbildes zu erkennen, wie z.B. Kanten, Formen oder Texturen.

Eine wichtige Eigenschaft von Faltungsschichten ist, dass sie das räumliche Muster der Eingabebilder beibehalten. Das bedeutet, dass die räumliche Information in der Eingabe auch in den Ausgabe-Feature-Maps erhalten bleibt.
Diese Eigenschaft ist besonders erwähnenswert, da bei voll verbundenen Schichten eine Verschiebung der Eingangswerte zu einem ganz anderen Verhalten und einer anderen Ausgabe führen würde. [15]

2.3.5 Pooling-Schicht

Um die Bildgröße und damit die Datenmenge in einem Faltungsnetzwerk zu reduzieren, sowie Overfitting zu vermeiden, werden Pooling-Schichten eingesetzt. Eine Pooling-Schicht hat einen Kern in beliebiger Größe, der genauso wie ein Filterkern einer Faltungsschicht, mit einer bestimmten Schrittweite über das Bild gefahren wird. Anstatt die Pixel zu gewichten, wird je nach Art der Pooling-Schicht, entweder der Durchschnitt oder der Maximalwert der Pixel gebildet. Dieser Wert wird nun als ein Pixel im Ausgabebild dargestellt. Bei einer Kerngröße von 2x2 und einer Schrittweite von 2 würden beispielsweise die Bildpixel um die Hälfte reduziert werden. [13]

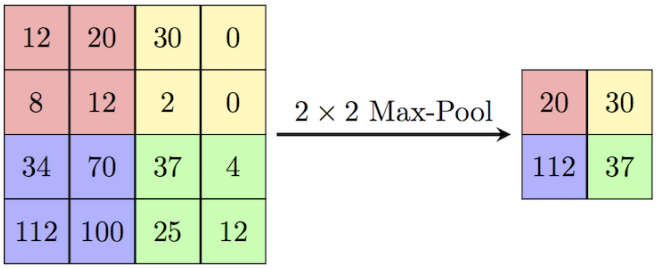

Abbildung 2.6: Beispiel Max-Pooling Schicht [6]

2.3.6 Netzwerkmodelle

Ein neuronales Netzwerk besteht aus mehreren aneinandergereihten Schichten. In dieser Arbeit sind voll verbundene und Faltungsnetzwerke relevant.
In den hier verwendeten voll verbundenen Netzwerken kommen voll verbundene Schichten, Aktivierungsfunktionen, Batch-Normalisierung und Dropout Schichten vor. Die Anzahl der Neuronen in der letzten voll verbundenen Schicht ist von der Aufgabenstellung abhängig. Falls es eine Klassifizierungsaufgabe ist, entspricht die Anzahl der Neuronen, der Anzahl von vorkommenden Klassen. Das Modell eines voll verbundenen Netzwerkes kann wie in Abbildung 2.7 gezeigt aussehen.

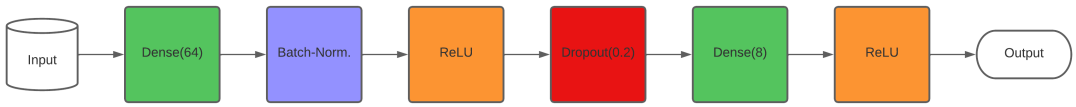

Abbildung 2.7: Beispiel voll verbundenes Netzwerk

In den hier verwendeten Faltungsnetzwerken (engl. Convolutional Neural Networks (Abk. CNNs)) kommen Faltungsschichten, voll verbundene Schichten, Aktivierungsfunktionen, Batch-Normalisierung, Dropout und Pooling Schichten vor. Das Modell eines CNNs kann wie in Abbildung 2.8 gezeigt aussehen.

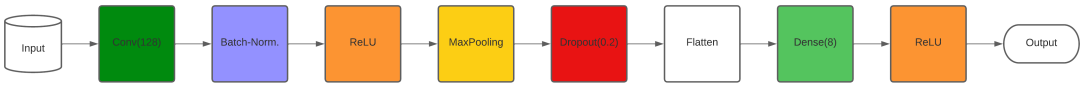

Abbildung 2.8: Beispiel CNN

Faltungsschichten erzeugen Bilder als Ausgabe. Da ein Bild bei Klassifizierungsaufgaben oder im RL keine Lösung darstellen, werden in CNNs Faltungsschichten und voll verbundene Schichten kombiniert. Durch die Flatten Schicht werden die einzelnen Bildpixel in ein Array der entsprechenden Länge umgewandelt. Dieses Array kann dann als Input für die voll verbundenen Schichten benutzt werden.
In der Bildverarbeitung spricht man von dem Feature Extractor, der alle Schichten bis hin zur Flatten Schicht enthält und dem Klassifizierer, der die restlichen Schichten enthält und letztendlich den gewünschten Output liefert. [13]

2.3.7 Trainingsprozess

Der Trainingsprozess eines neuronalen Netzwerks besteht aus mehreren Schritten, beginnend mit der Initialisierung der Gewichte in jedem Neuron des Netzwerks.

Die zufällige Initialisierung ist das einfachste Verfahren zur Initialisierung von Gewichten in neuronalen Netzen. Hier werden die Gewichte zufällig aus einem bestimmten Bereich oder einer bestimmten Verteilung gewählt. Diese Methode ist einfach zu implementieren und kann in manchen Fällen gute Ergebnisse liefern. Allerdings kann die benötigte Trainingsdauer, bis das Netz konvergiert, recht hoch sein.

Die He-Initialisierung ist ein Verfahren, das speziell für Netze mit nicht-linearen Aktivierungsfunktionen wie der ReLU-Funktion entwickelt wurde. Bei der He-Initialisierung werden die Gewichte aus einer Normalverteilung mit einer Standardabweichung berechnet, die proportional zu $\sqrt{\frac{2}{n}}$ ist. Hierbei ist n die Anzahl der Eingänge des Neurons. Diese Methode soll dazu beitragen, das Problem der "explodierenden Gradienten" zu vermeiden, das bei der Verwendung von nicht-linearen Aktivierungsfunktionen auftreten kann. [15]

Nach der Initialisierung werden die Trainingsdaten in Form von Mini Batches in das Netzwerk eingespeist. Ein Mini Batch ist eine Teilmenge der Trainingsdaten. Das Netzwerk berechnet eine Ausgabe für jeden Eingabevektor. Die Ausgabe wird dann mit der gewünschten Ausgabe verglichen, um den Fehler des Netzwerks zu berechnen. Dazu ist es erforderlich, dass die Trainingsdaten ein dazugehöriges Label mit dem richtigen Ergebnis haben.

Um den Fehler zu berechnen wird eine Fehlerfunktion (engl. loss function) verwendet. Die Fehlerfunktion gibt an, wie gut die Vorhersage des Netzwerks mit der tatsächlichen Ausgabe übereinstimmt. Eine gängige Fehlerfunktion ist der Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i' - y_i)^2 \tag{2.9}$$

n: Anzahl der Trainingsdaten
y_i': Ausgabe des Netzwerks zu Trainingsbeispiel i
y_i: label, also richtiger Wert des Trainingsbeispiels i

Die Quadratfunktion sorgt für einen, in jedem Fall, positiven Fehlerwert und somit für eine bessere Vergleichbarkeit, unabhängig davon ob der Ausgabewert unter- oder überschritten wird.

Das Ziel des Trainingsprozesses besteht darin, die Gewichte in jedem Neuron so anzupassen, dass der Fehler des Netzwerks auf den Trainingsdaten minimiert wird.
Dazu wird der Backpropagation-Algorithmus verwendet, der durch Differenzierung der Fehlerfunktion jedes einzelnen Neurons die Steigung der Fehlerfunktion berechnet und die Gewichte so verschiebt, dass die Steigung minimiert wird, bis schließlich nach einigen Iterationen jedes Neuron sein globales Minimum erreicht hat. Dieser Vorgang nennt sich Gradientenabstieg (engl. Gradient Descent) und ist in Abbildung 2.9 dargestellt.

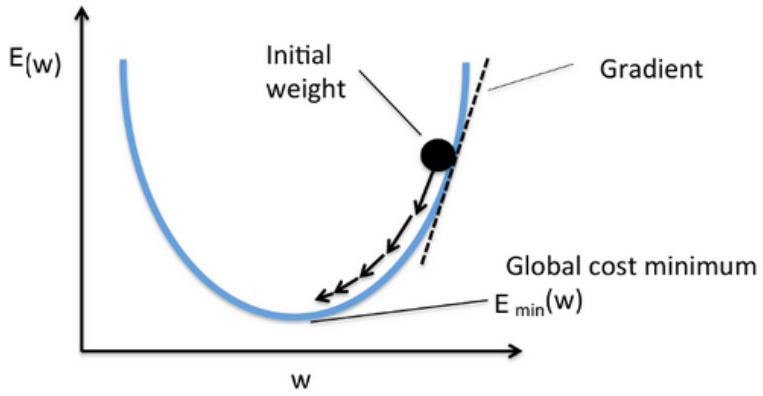

Abbildung 2.9: Gradientenabstieg [27]

Das Ausführen des Gradientenabstiegs mit Mini Batches, die beispielsweise 32 Datensätzen enthalten, ist effizienter als die einzelne Verarbeitung jedes Datensatzes. Außerdem wird der Trainingsprozess stabiler, da der Durchschnitt von mehreren Gradienten benutzt wird und somit die Varianz verringert wird.

Während des Trainingsprozesses werden die Trainingsdaten verwendet, um das Netzwerk zu trainieren. Um sicherzustellen, dass das Netzwerk tatsächlich lernt und nicht nur die Trainingsdaten auswendig lernt, wird das Netzwerk auch mit Testdaten getestet. Diese Testdaten sind wie die Trainingsdaten aufgebaut und enthalten auch ein Label mit dem richtigen Ergebnis, aber sie werden nicht zum Trainieren des Netzwerks verwendet. Stattdessen werden sie verwendet, um sicherzustellen, dass das Netzwerk nicht im Overfitting ist und auch auf neuen Daten gut performt.

Der Trainingsprozess eines neuronalen Netzwerks ist ein iterativer Prozess, bei dem das Netzwerk schrittweise durch Anpassung seiner Gewichte lernt, auf neue Daten zu reagieren. [15]

2.3.8 Optimizer

Es gibt verschiedene Optimierungsalgorithmen um ein neuronales Netzwerk zu trainieren. Der zuvor erklärte Gradientenabstieg ist einer von den sogenannten "Optimizern".

Der stochastische Gradientenabstieg (engl. Stochastic Gradient Descent) ist eine Variation des Gradientenabstiegsalgorithmus, der im Allgemeinen schneller ist und besser mit großen Datensätzen umgehen kann. Im Gegensatz zum Gradientenabstieg, bei dem der Gradient über den gesamten Datensatz berechnet wird, wird bei der stochastischen Variante der Gradient nur für eine zufällige Teilmenge der Daten berechnet.

RMSprop ist ein Gradientenabstiegs-Optimierungsalgorithmus, der die Schrittweite an die Bedingungen des Gradienten anpasst, um eine schnellere Konvergenz und Stabilität des Lernprozesses zu erreichen.
Der RMSprop-Algorithmus funktioniert, indem er einen gewichteten, gleitenden Durchschnitt der Quadratsummen der Gradienten berechnet. Anschließend wird der neue Gradient durch die Quadratwurzel des zuvor berechneten Durchschnitts geteilt.
Diese Methode sorgt dafür, dass die Schrittweite unterschiedlich groß für verschiedene Dimensionen ist. Die Schrittweite wird erhöht, wenn der Gradient um lokale Minima oder auch Sattelpunkte wie in Abbildung 2.10 abflacht, um diese zu überwinden.[17]

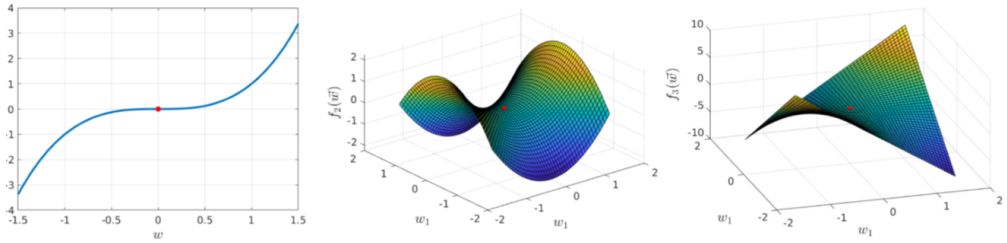

Abbildung 2.10: Lokale Minima bzw. Sattelpunkte [17]

Der Adam (Adaptive Moment Estimation) Optimizer ist ein Optimierungsverfahren, das die Vorteile des stochastischen Gradientenabstiegs mit denen des RMSprop-Algorithmus verbindet.

Adam verwendet adaptive Lernraten für jedes Gewicht, was bedeutet, dass jeder Parameter eine unterschiedliche Lernrate hat, die sich im Laufe des Trainings anpasst. Dies ist hilfreich, um schnellere Konvergenz zu erreichen und Overfitting zu vermeiden. Es berechnet auch einen schrittweisen Mittelwert und eine schrittweise Standardabweichung der Gradienten, um den Algorithmus zu stabilisieren und Rauschen zu reduzieren. Adam ist eine beliebte Wahl für viele Machine-Learning-Modelle, da es in der Regel schnell konvergiert und gute Leistungen erzielt.
Der Adam Optimizer ist bekannt für seine Effizienz bei der Anpassung an unterschiedliche Problemstellungen und Datensätze und hat sich als eines der am häufigsten verwendeten Optimierungsverfahren im Deep Learning etabliert. Es kann jedoch nicht garantiert werden, dass er immer das beste Ergebnis liefert, da die Leistung auch von anderen Faktoren wie dem Modell, den Daten und den Hyperparametern abhängt. [15]

2.4 Reinforcement Learning

Das grundlegende Modell ist bei jeder RL Lösung gleich und ist in Abbildung 2.11 dargestellt. Es gibt einen Agenten, der darauf trainiert werden soll möglichst gute Aktionen auszuführen und es gibt eine Umgebung (engl. Environment) die auf die Aktionen des Agenten reagiert und alle Regeln und Vorschriften des Szenarios enthält.
Es ist vergleichbar mit einem Brettspiel, in dem einer der Spieler der Agent ist, dessen Aufgabe es ist das Spiel zu meistern, ohne jegliche Vorkenntnisse über das Spiel zu haben. Die Umgebung umfasst das Spielbrett selbst, die eventuellen anderen Spieler, sowie alle Regeln des Spiels.
Der Lernprozess des Agenten beginnt damit, dass er von der Umgebung den aktuellen Zustand des Spielbretts bekommt. Da der Agent keine Spielerfahrung hat, wählt er zunächst eine zufällige Aktion aus die er durchführt. Diese Aktion wird der Umgebung mitgeteilt. Diese passt entsprechend den Zustand des Spielbretts an, indem sie die Züge anderer Mitspieler ausführt und eventuelle Regeln durchsetzt. Damit der Agent etwas lernen kann, wird ihm von der Umgebung nach jeder Aktion eine Belohnung bzw. Bestrafung gegeben.

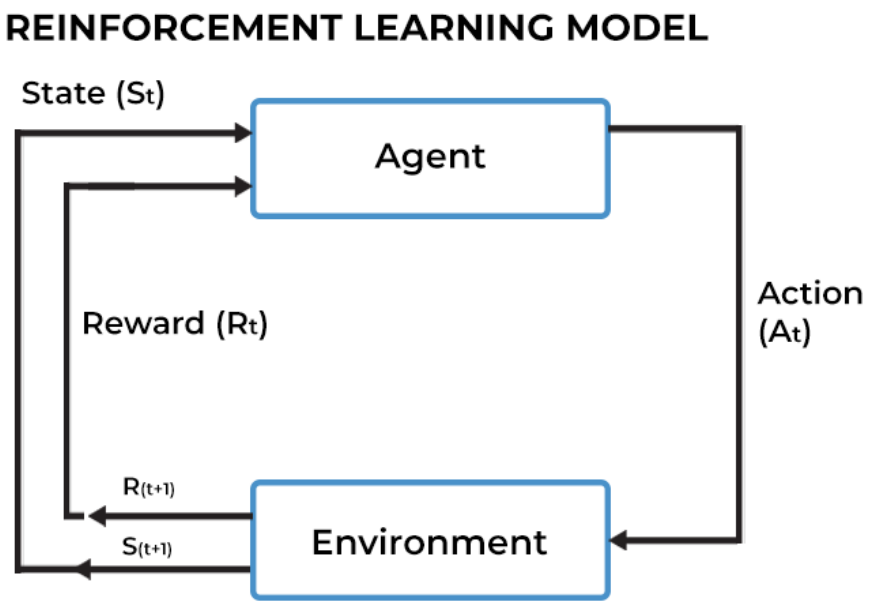

Abbildung 2.11: RL-Modell [18]

Dieser Prozess wird so lange wiederholt, bis das Spiel vorbei ist. Das Durchlaufen eines Spielzyklus wird im RL eine Episode genannt. Der Agent hat jetzt einen Teil der möglichen Zustände gesehen, aber jeweils nur die eine Aktion in jedem Zustand ausprobieren können. Damit der Agent das Spiel so gut beherrscht, dass er in jeder Situation die beste Aktion auswählt, werden während des Trainings viele Episoden hintereinander durchlaufen. So kann der Agent verschiedene Aktionen in den jeweiligen Zuständen ausprobieren und über die erhaltenen Belohnungen darauf schließen, welche die Beste ist.
Man unterscheidet zwischen Q-learning (Quality-learning) mit einer Q-Tabelle, in der alle Werte zum Auswählen der besten Aktion gespeichert sind und dem Deep Q-learning. Beim Deep Q-learning wird ein neuronales Netzwerk anstelle der Q-Tabelle eingesetzt, um die beste Aktion zu ermitteln. [16]

2.4.1 Q-Tabelle

Für jeden möglichen Zustand gibt es eine Zeile mit einem Q-Wert zu jeder Aktion. Die Spalten stehen für jeweils eine Aktion. Es wird die Aktion mit dem höchsten Q-Wert ausgeführt. Zu Anfang wird die Tabelle mit zufälligen Werten gefüllt. Mit jeder ausgeführten Aktion wird der Q-Wert der entsprechenden Zelle neu berechnet:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)}^{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \Big)}^{\text{temporal difference}}$$

Abbildung 2.12: Q-learning Formel

Die Lernrate α und der Faktor γ sind Parameter, die vor dem Training festgelegt werden müssen. Die Lernrate steuert in welchem Verhältnis der bisherige Q-Wert mit dem neuen verrechnet wird. Der Faktor $maxQ(s_{t+1,a})$ ist der maximale Q-Wert des nächsten Zustands der eingenommen wird. Dieser wird mit dem Discount Faktor γ, der zwischen Null und Eins liegt, skaliert.
Damit die Q-Werte nicht mit jedem Durchlauf weiter ansteigen, wird für die zeitliche Differenz (engl. temporal difference) der aktuelle Q-Wert von dem neu berechneten Wert abgezogen. [16]

2.4.2 Deep Q-learning

Bei einem großen Zustandsraum würde eine Q-Table sehr groß werden. Dies würde zu einem sehr langem Trainingsprozess führen, bei dem es auch zu Problemen mit dem verfügbaren Speicher kommen kann. Beim Deep Q-learning wird, wie in Abbildung 2.13 dargestellt, die Q-Table mit einem neuronalen Netzwerk ersetzt.

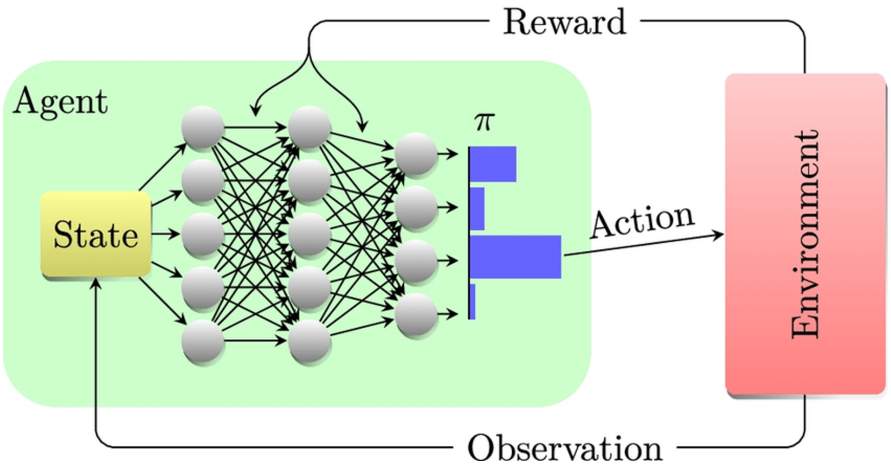

Abbildung 2.13: Ersetzen der Q-Table mit einem neuronalen Netzwerk [23]

Die Anzahl der Ausgabeneuronen des letzten Layers entspricht der Anzahl der verschiedenen möglichen Aktionen. Die ausgegebenen Werte für jede Aktion sind hier die Q-Werte. Die Q-Werte werden durch die selbe Formel wie in Abbildung 2.12 berechnet.

Die Gewichte des neuronalen Netzes werden im Training so angepasst, dass die Ausgabeneuronen die Q-Werte abbilden. Nach jeder Aktion soll jeweils nur ein Q-Wert angepasst werden. Dementsprechend müssen die Gewichte so verändert werden, dass die restlichen Q-Werte dabei unverändert bleiben. Sobald die zeitliche Differenz den Wert Null für jeden Q-Wert annimmt, ist das Training abgeschlossen und es kommt zu keiner Veränderung mehr.

Die zeitliche Differenz kann als Kostenfunktion während des Trainings genutzt werden. Eine Kostenfunktion wie der MSE hat im RL keine Aussagekraft, da es keine "richtigen" Werte gibt, die mit den Trainingswerten verglichen werden könnten. [16]

3 Deep Reinforcement Learning in der Physik

Deep RL kommt bereits in verschiedenen physikalischen Anwendungen zum Einsatz. Im Folgenden werden zwei Anwendungen aus der Steuer- und Regelungstechnik vorgestellt.

3.1 Kernfusion

Die Kernfusion mittels magnetischen Einschlusses, insbesondere in der Tokamak Konfiguration, ist ein vielversprechender Weg zur Gewinnung von Energie. Tokamaks sind torusförmige Aufbauten für die Kernfusionsforschung. Der Hauptteil der Forschung ist die Untersuchung von verschiedenen Plasmakonfigurationen, also die Form und Verteilung des Plasmas. Die unterschiedlichen Plasmakonfigurationen gilt es bezüglich Stabilität und Energieausstoß zu optimieren. Um ein Hochtemperaturplasma innerhalb des Tokamak-Gefäßes zu formen und zu erhalten, erfordert es eine hochdimensionale und hochfrequente Regelung von magnetischen Spulen. In Abbildung 3.1 ist der Aufbau eines Tokamaks sowie eine Plasmakonfiguration dargestellt.

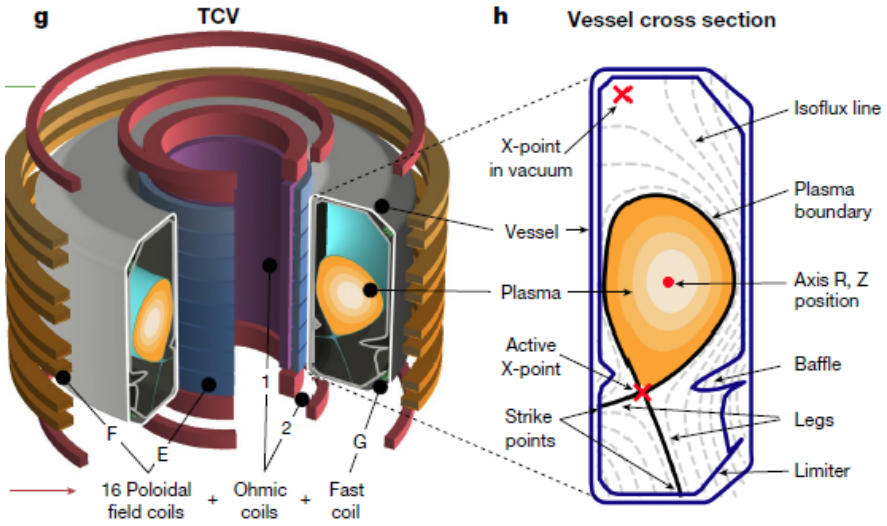

Abbildung 3.1: Aufbau eines Tokamaks mit einer Plasmakonfiguration [14]

Der bisherige Ansatz für dieses zeitlich veränderliche, nichtlineare, multivariate Steuerungsproblem besteht darin, die Spulenströme und -spannungen zu berechnen. Anhand der berechneten Werten werden dann PID-Regler entworfen, um die vertikale Position des Plasmas zu stabilisieren und die radiale Position und den Plasmastrom zu steuern. Die unterschiedlichen Größen dürfen sich dabei nicht gegenseitig beeinflussen.
Die Regler werden auf der Grundlage von linearisierten Modellen entworfen und erfüllen in der Regel auch ihre Aufgabe. Allerdings haben sie einen hohen Designaufwand und erfordern viel Fachwissen, falls die Plasmakonfiguration verändert werden soll.

Der neue Ansatz des RL bietet mehr Flexibilität und verringert den Entwicklungsaufwand für neue Plasmakonfigurationen erheblich. Anstelle einer Reihe von nichtlinearen Reglern, gibt es jetzt nur noch ein neuronales Netzwerk, das alle Aufgaben übernimmt. Um die Stabilität in der Realität zu gewährleisten, werden die Eigenschaften wie Verzögerung, Messrauschen oder Spannungs-Offsets variiert. Es sind Bereiche bekannt, in denen die Simulation nur sehr schlecht funktioniert. Diese Bereiche werden im Training durch entsprechende Abbruchbedinungen der Episoden gemieden und in der Realität in Form von maximalen Spulenströmen als Betriebsgrenzen gesetzt.
Es wird ein neuronales Netzwerk mit drei voll verbundenen Schichten für diese Anwendung benutzt. Die Eingangsdaten bestehen aus 92 Messwerten und die Ausgangsschicht gibt die Spannungen für alle 19 Spulen vor. [14]

3.2 Teilchenbeschleuniger

In der Fermilab Booster Teilchenbeschleuniger-Anlage soll die Steuerung der Stromversorgung der Magneten (gradient magnet power supply (GMPS)) von einem RL Agenten übernommen werden.

Teilchenbeschleuniger gehören zu den komplexesten technischen Systemen der Welt. Sie sind von entscheidender Bedeutung für die Untersuchung der elementaren Bestandteile der Materie und der Kräfte, die ihre Wechselwirkungen steuern. Das Abstimmen und Steuern von Beschleunigern ist herausfordernd und zeitaufwändig. Selbst geringfügige Verbesserungen können sich sehr effizient in eine verbesserte wissenschaftliche Ausbeute für ein experimentelles Teilchenphysikprogramm übersetzen, da die Laufzeit des Beschleunigers einen erheblichen Kostenfaktor darstellt.

Bisher bestand der gängigste Ansatz für die Steuerung von Beschleunigersystemen weitgehend aus manueller Abstimmung durch Experten und wurde so weit wie möglich aus physikalischen Prinzipien hergeleitet. Die Beschleunigerphysik ist jedoch komplex und hochgradig nichtlinear.

Die neuronalen Netze sollen für Steuerungszwecke auf Field-Programmable Gate Arrays (FPGAs) kompiliert werden können, da es auf einem FPGA keine unvorhergesehenen Latenzen gibt. Diese Fähigkeit ist für die Betriebsstabilität in komplizierten Umgebungen, wie einer Beschleuniger-Anlage, wichtig.

Es wird ein neuronales Netzwerk mit drei voll verbundenen Schichten mit jeweils 128 Neuronen für diese Anwendung benutzt. Die Eingangsdaten bestehen aus fünf Messwerten und die Ausgangsschicht gibt sieben Werte zur Steuerung aus.

Die Forscher versprechen sich von der RL-Lösung eine zehnfache Verbesserung der Präzision des Regelsystems. [24]

4 Optischer Pulsformer

4.1 Aufbau eines Pulsformers

Die Frequenzen der elektromagnetischen Welle in einem optischen Laserpuls sind zu hoch, um sie direkt zu steuern. Deswegen wird die Pulsformung im Frequenzraum durchgeführt. Der dazu benutzte Aufbau folgt typischerweise der sogenannten 4f-Geometrie. Eine Kombination aus einem Gitter und einem fokussierenden optischen Element (Linse oder Spiegel) mit der Brennweite f spaltet den Laserpuls in seine Frequenzkomponenten auf.
In der sogenannten Fourier-Ebene können dann Amplitude und Phase dieser Frequenzkomponenten manipuliert werden. Abschließend fügt ein symmetrischer Aufbau aus Linse und Gitter die modifizierten Frequenzkomponenten wieder zu einem nun veränderten Laserpuls zusammen. (A. Przystawik, persönliche Mitteilung)
Es gibt viele Möglichkeiten zur konkreten experimentellen Umsetzung dieses Prinzips der Pulsformung. Exemplarisch sind hier nur zwei Methoden erwähnt. Abbildung 4.1 zeigt die Nutzung eines akusto-optischen Modulators (AOM) als Amplitudenmaske.

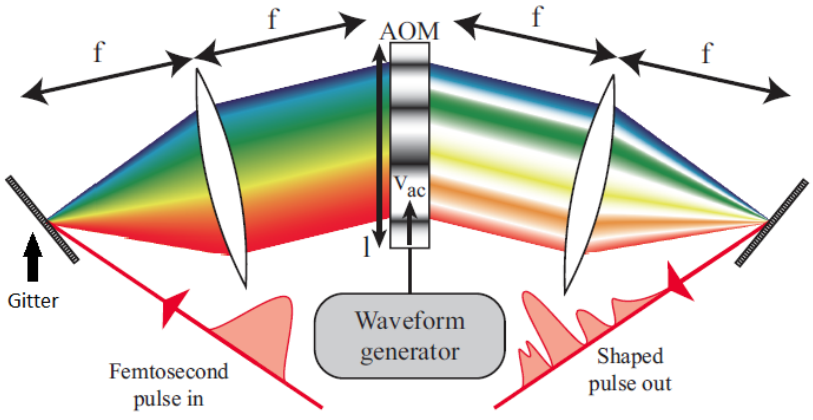

Abbildung 4.1: Aufbau eines Pulsformers [21]

Hier erzeugt eine hochfrequente Schallwelle (10 bis 200 MHz) ein Gitter, an dem die gewünschten Frequenzkomponenten gebeugt werden. Die unerwünschten Frequenzkomponenten laufen geradeaus weiter und gehen verloren. Durch Modulation der Amplitude und des Zeitverlaufs der Schallwelle im AOM können die Amplituden der optischen Frequenzkomponenten gesteuert werden.

Eine weitere Möglichkeit zur Manipulation des Lichts in der Fourier-Ebene stellen Modulatoren auf Flüssigkristall-Basis dar. In Abbildung 4.2 wird die Modulation schematisch dargestellt.

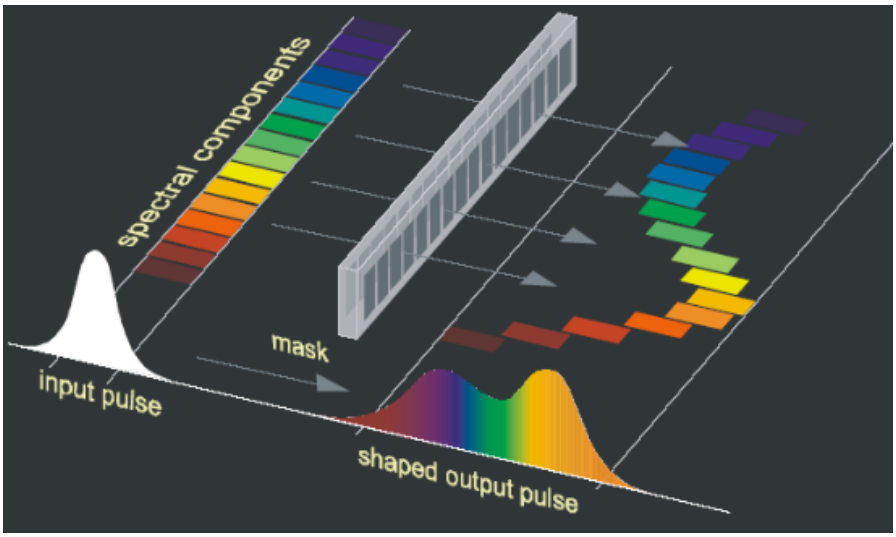

Abbildung 4.2: Phasenverschiebung der Frequenzen [28]

Durch Anlegen einer individuellen Spannung an die Pixel der Flüssigkristall-Matrix können wahlweise die Transmission oder die Phasenverzögerung über eine Änderung des Brechungsindex angepasst werden. Flüssigkristalle sind typischerweise doppelbrechend. Durch die unterschiedliche Ausrichtung ändert sich der Brechungsindex, was die Laufzeit durch den Pixel beeinflusst. (A. Przystawik, persönliche Mitteilung)
Der simulierte Modulator hat 640 Pixel, mit denen er jeweils die Phase einer bestimmten Bandbreite verschieben kann. Das Einstellen der 640 Spannungen soll durch den RL-Agenten erfolgen.

4.2 Frequency-resolved optical gating

Das Spektrogramm, mit dem der Agent arbeiten soll, wird durch Frequency-resolved optical gating (FROG) generiert.
Eine FROG-Messung liefert zeitaufgelöste Informationen über die spektralen Komponenten in einem Laserpuls. Dazu wird der zu vermessende Laserpuls in einem nichtlinearen Kristall mit einem sogenannten Gate-Puls überlagert und das dabei entstehende Summenfrequenz-Licht mit einem Spektrometer aufgezeichnet. Im einfachsten Fall ist der Gatepuls identisch zu dem zu vermessenden Puls, wie in Abbildung 4.3 dargestellt. (A. Przystawik, persönliche Mitteilung)

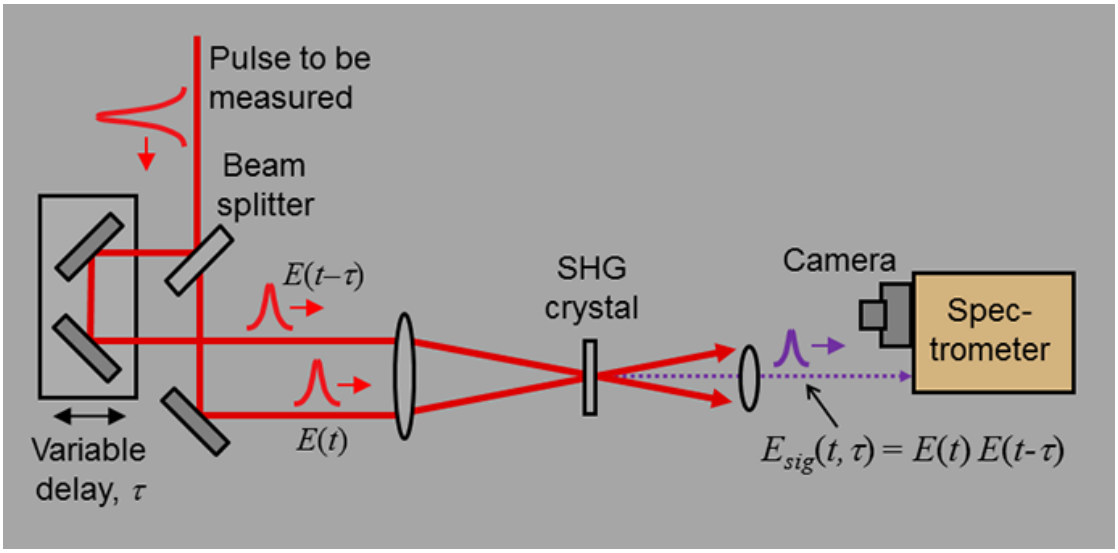

Abbildung 4.3: Aufbau der Autokorrelations-FROG Messung [25]

Der zu messende Puls wird durch einen beam splitter in zwei identische Pulse aufgeteilt. Einer dieser Pulse wird nun durch einen Umweg, der in der Länge variabel ist, verzögert. Man misst eine spektral aufgelöste Autokorrelation des Pulses bzw. ein Autokorrelations-FROG. Aus den sogenannten FROG-Spuren kann z.B. der zeitliche Verlauf der einhüllenden Kurve des Laserpulses und die spektrale Phase oder eben auch das Spektrogramm des Pulses rekonstruiert werden. [26]

Da die Pulse in dieser Arbeit nur simuliert werden, sind entsprechend auch die Messeinrichtung und die entstehenden Spektrogramme simuliert. Ein Spektrogramm des verzerrten Pulses kann wie in Abbildung 4.4 aussehen.

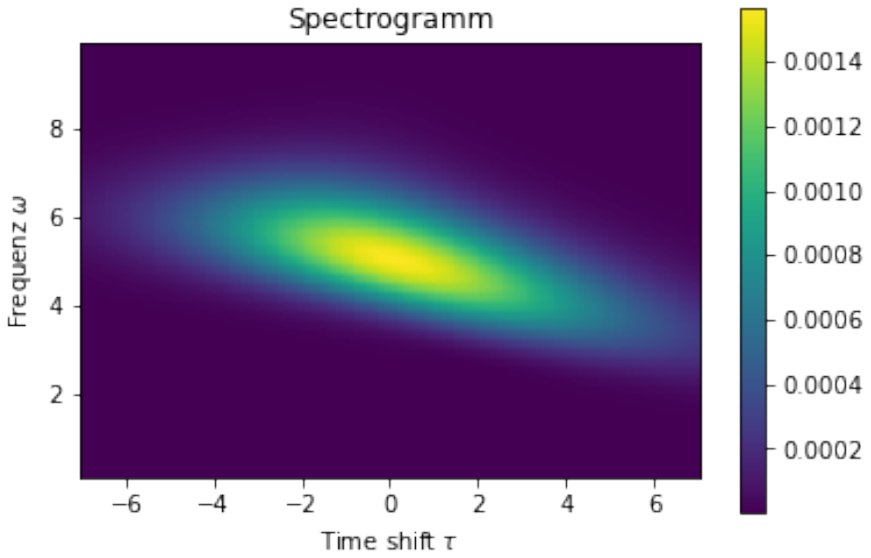

Abbildung 4.4: Spektrogramm eines verzerrten Pulses

Das Spektrogramm eines nahezu bandbreitenlimitierten Pulses ist in Abbildung 4.5 dargestellt. Merkmal eines bandbreitenbegrenzten Pulses ist, dass alle Frequenzkomponenten zur gleichen Zeit ankommen, es also keine diagonalen Verläufe über die Zeit gibt.

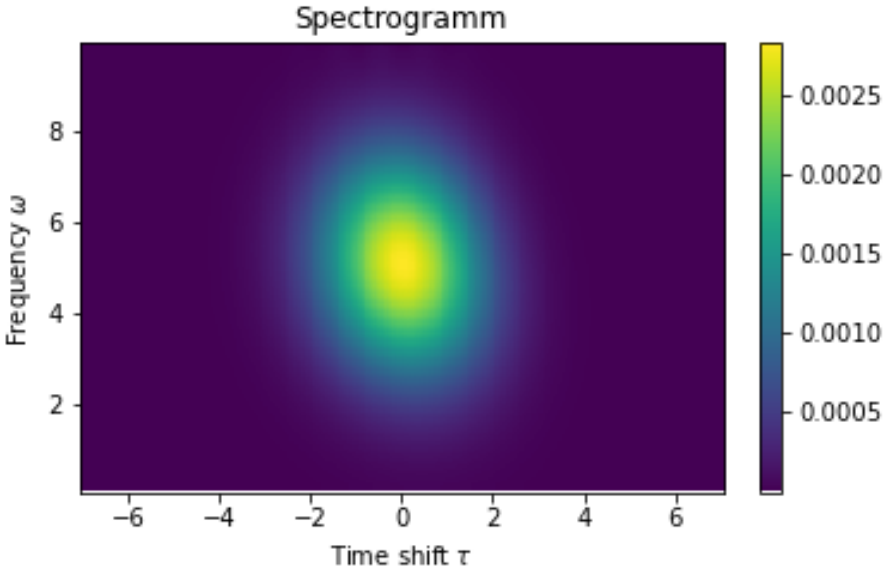

Abbildung 4.5: Spektrogramm eines Transformationsbegrenzten Pulses

5 Umsetzung der Steuerung eines optischen Pulsformers mit Deep Reinforcement Learning in Python

5.1 Bibliotheken

Framework TensorFlow und Keras

TensorFlow ist eine Open-Source-Softwarebibliothek für maschinelles Lernen und neuronale Netzwerke. Es wurde von Google entwickelt und ist derzeit eine der am häufigsten verwendeten Bibliotheken für maschinelles Lernen. TensorFlow bietet eine Vielzahl von Funktionen für das Erstellen, Trainieren und Bereitstellen von künstlichen neuronalen Netzwerken. Es unterstützt verschiedene Programmiersprachen wie Python, C++, Java und Go. [10]

Keras ist eine Open-Source-Softwarebibliothek für neuronale Netzwerke, die auf TensorFlow aufbaut. Keras wurde entwickelt, um die Erstellung und das Training von neuronalen Netzwerken zu vereinfachen. Es bietet eine benutzerfreundliche API, die es ermöglicht, schnell und einfach komplexe neuronale Netzwerke zu erstellen und zu trainieren. [4]

Im Wesentlichen ist TensorFlow eine Bibliothek für maschinelles Lernen und Keras ist eine benutzerfreundliche Schnittstelle für die Erstellung und das Training von neuronalen Netzwerken, die auf TensorFlow aufbaut. Zusammen bieten sie eine leistungsstarke und flexible Plattform für die Entwicklung und das Training von künstlichen neuronalen Netzwerken für verschiedene Anwendungen im Bereich des ML.

NumPy - mathematische Funktionen

NumPy (kurz für "Numerical Python") ist eine Python-Bibliothek, die hauptsächlich für numerische Berechnungen und wissenschaftliches Rechnen verwendet wird. Es bietet ein leistungsfähiges Array-Objekt, das N-dimensional ist. Dieses Array-Objekt ist wesentlich schneller als die standardmäßigen Python-Listen und bietet eine Vielzahl von Funktionen und Operationen, die speziell für numerische Berechnungen ausgelegt sind. Für diese Arbeit relevante Funktionen sind die Fourier Funktionen fft, ifft, fftshift sowie der Zufallsgenerator. [7]

Scipy - mathematische Funktionen

Scipy ist eine Python-Bibliothek, die auf NumPy aufbaut. Es bietet eine Vielzahl von Funktionen und Tools, um komplexe mathematische Probleme zu lösen. In dieser Arbeit wurden z.B. die Funktionen Integration, Optimierung und Interpolation genutzt. [8]

Matplotlib - Plots

Matplotlib ist eine Python-Bibliothek für die Erstellung von Statistikdiagrammen, Plots und Visualisierungen. Es ist eine der bekanntesten und am häufigsten verwendeten Visualisierungsbibliotheken in Python.
Matplotlib ist sehr flexibel und bietet eine Vielzahl von Einstellungen und Optionen um die Diagramme und Visualisierungen an die eigenen Bedürfnisse anzupassen. Es bietet eine Vielzahl von Farbschemata und Linienstilen sowie die Möglichkeit, Achsenbeschriftungen und Titel hinzuzufügen und das Layout des Diagramms anzupassen. [5]

Imagesc - Spektrogramm

Imagesc ist eine Funktion in der Python-Bibliothek matplotlib, die eine zweidimensionale Matrix als Eingabe nimmt und ein farbcodiertes Bild des Inhalts der Matrix darstellt. Die Farbcodierung der Bildpixel wird durch eine Farblegende an der Seite des Bildes dargestellt. Dies eignet sich, um Spekrogramme darzustellen. [3]

Deque

Ein Deque (Double-Ended Queue) ist eine Datenstruktur, die in der Verarbeitung von Erfahrungstupeln im Reinforcement Learning eingesetzt wird.
Im RL werden Erfahrungstupel gesammelt, um den Agenten zu trainieren. Ein Erfahrungstupel besteht aus einem Zustand, einer Aktion, einer Belohnung und dem darauffolgenden Zustand. Diese Tupel werden in einer Datenstruktur gespeichert, die als Erfahrungsgedächtnis oder Replay Memory bezeichnet wird. [2]

tqdm

tqdm ist eine Python-Bibliothek, die eine Fortschrittsanzeige in der Kommandozeile bereitstellt. Im RL kann tqdm verwendet werden, um den Fortschritt des Trainingsprozesses anzuzeigen.
Insbesondere kann tqdm verwendet werden, um den Fortschritt der Iterationen (Episoden) des RL-Algorithmus anzuzeigen. Dies ist besonders nützlich, da die Anzahl der Episoden oft sehr groß ist und es schwierig ist, den Fortschritt ohne eine entsprechende Anzeige zu verfolgen. Darüber hinaus bietet tqdm auch eine Schätzung der verbleibenden Trainingszeit. [11]

os - Betriebssystem

Die os Bibliothek ist eine Standardbibliothek in Python, die Funktionen zur Interaktion mit dem Betriebssystem bereitstellt. Die Bibliothek ermöglicht das Erstellen, Löschen, Umbenennen und Verschieben von Dateien und Verzeichnissen.

Tensorboard

TensorBoard ist ein Werkzeug zur Visualisierung und Überwachung von maschinellen Lernprozessen, das von der TensorFlow-Bibliothek unterstützt wird. Es ermöglicht, Daten wie Verlustfunktionen, Genauigkeitswerte, Modellarchitekturen, Grafiken und Histogramme zu visualisieren.
TensorBoard bietet eine interaktive Benutzeroberfläche, die es dem Benutzer ermöglicht, den Fortschritt des Trainingsprozesses von ML-Modellen in Echtzeit zu überwachen und

zu analysieren. Die Anzeige der Metriken und Grafiken kann auch dazu beitragen, Einblicke in den Trainingsprozess zu gewinnen und Fehler oder Probleme zu erkennen. [9]

5.2 Simulation der Pulse

In dieser Simulation werden alle Werte ohne Einheiten gehalten, um die Simulation zu vereinfachen.

Als Referenzwert wird zunächst ein unverzerrter Puls erzeugt. Die Amplituden eines unverzerrten Pulses im Frequenzbereich werden durch die folgende Funktion beschrieben:

$$A(\omega) = e^{-\frac{(\omega-\omega_p)^2}{2\cdot\sigma^2}} \tag{5.1}$$

Wobei ω_p der Mittelpunkt und σ die Breite der Frequenzfunktion ist. Die Dispersionseffekte werden durch die folgenden Elemente einer Taylorreihe beschrieben:

$$\phi(\omega) = -\frac{GDD}{2} \cdot (\omega - \omega_p)^2 + \frac{TOD}{6} \cdot (\omega - \omega_p)^3 \tag{5.2}$$

Diese Phasenfunktion wird nun zu dem unverzerrten Puls dazu addiert:

$$A(\omega) = e^{-\frac{(\omega-\omega_p)^2}{2\cdot\sigma^2}+i\cdot\phi(\omega)} \tag{5.3}$$

Die Breite des erhaltenen Pulses gilt es im Zeitbereich zu minimieren. Der Puls wird geformt, indem eine weitere Phasenfunktion $\phi(\omega)_{shaped}$ dazu addiert wird:

$$A(\omega)_{shaped} = amplitude(\omega) \cdot e^{i\cdot(\phi(\omega)+\phi(\omega)_{shaped})} \tag{5.4}$$

Die Phasenfunktion $\phi(\omega)_{shaped}$ wird durch die “CubicSpline“ Funktion der Bibliothek Scipy erstellt. Sie wird anhand von den gewählten Stützstellen geformt.

Die Transformation in den Zeitbereich, also die inverse DFT, erfolgt mit der Numpy Funktion “ifft“. Mit der Numpy Funktion “fftshift“ wird das Signal im Zeitbereich zentriert. Die Breite des Pulses im Zeitbereich wird durch die folgenden Formeln berechnet:

$$normFactor = \sum t \cdot y^2$$

Wobei t die hier einheitenlose Zeitskala ist und y der Betrag der dazugehörigen Y-Werte.

$$Moment_1 = \frac{1}{normFactor} \cdot \sum t \cdot y^2$$

$$Moment_2 = \frac{1}{normFactor} \cdot \sum t^2 \cdot y^2$$

$$Breite = \sqrt{Moment_2 - Moment_1^2} \tag{5.5}$$

5.3 Die Umgebung

Die Klasse “PulsShaperEnv“ stellt die Umgebung dar und hat mehrere Parameter und Funktionen.

5.3.1 “PulseShaper“ - Verwalten der Phasenfunktion

Die Verwaltung der Stützstellen und somit der Phasenfunktion wurde in die Klasse “PulseShaper“ ausgelagert.
Beim Erstellen eines Objekts der Klasse “PulseShaper“ werden die Stützstellen initialisiert und die Pulsbreite beim Start der Epoche berechnet. Die Klasse “PulseShaper“ hat die Funktion, die einzelnen Stützstellen zu bewegen. Die vom Agenten ausgewählten Aktionen werden an den “PulseShaper“ weitergegeben und dieser führt die entsprechende Verschiebung der Stützstellen aus.
Die Stützstellen repräsentieren eine bestimmte Phasenverschiebung zwischen null und zwei Pi. Der Wertebereich, in dem die Stützstellen liegen können, wird allerdings nicht auf zwei Pi beschränkt, denn bei dem Sprung einer Stützstelle, falls diese zwei Pi überschreitet, würden die spline Werte zwischen den Stützstellen sich um einen Wert ungeleich zwei Pi verändern. Eine Verschiebung über den zwei Pi Wertebereich hinaus, ändert mathematisch nichts aufgrund der zwei Pi Periodizität.

5.3.2 Parameter der Umgebung

Ein Parameter der Umgebung ist die Größe des Aktionsraumes. Die Anzahl der möglichen Aktionen entspricht der Anzahl der Stützstellen mit dem Faktor zwei multipliziert, da jede Stützstelle in positive oder negative Richtung verschoben werden kann.
Ein weiterer Parameter der Umgebung ist die Größe und Form des Zustands, der beobachtet werden kann. Der Zustand besteht in diesem Szenario aus dem Spektrogramm der Größe 47x100. Für ein voll verbundenes Netzwerk wird das Spektrogramm in ein 1-Dimensionales Array der Länge 4700 umgeformt.
Für ein Faltungsnetzwerk, das mit nur einem Farbkanal arbeitet, wird die Form von 47x100 bestehen bleiben. Für ein Faltungsnetzwerk, das mit drei Farbkanälen arbeitet, wird die Form 47x100x3 verwendet. Die zwei neuen Farbkanäle werden mit den gleichen Werten des einen Farbkanals des Spektrogramms gefüllt.
Die Klasse "PulseShaper" zur Verwaltung der Phasenfunktion hat die Schrittweite als Parameter. Sie legt fest, wie weit die Stützstellen jeweils bei einer Aktion verschoben werden und ist in Radian angegeben.

5.3.3 Funktionen der Umgebung

Zu Beginn einer Episode wird die Funktion *reset* der Umgebung ausgeführt. Diese Funktion erstellt ein neues Objekt der Klasse "PulseShaper" und setzt die Anzahl der Schritte innerhalb der Episode auf null zurück. Anschließend wird das Spektrogramm unter Berücksichtigung der neu initialisierten Stützstellen berechnet und als Beobachtung des Zustands zurückgegeben.
Die Funktion *step*(*action*) führt einen Schritt der Episode aus. Die Eingangsvariable *action*, die vom Agenten ausgewählt wurde, wird an den "PulseShaper" weitergegeben. Mit den aktualisierten Stützstellen wird nun das neue Spektrogramm, die Breite des Pulses und damit auch die Belohnung berechnet.
Die Funktion enthält zusätzlich eine Abbruchbedingung für die Episode. Wenn die festgelegte Anzahl von Schritten pro Episode erreicht wurde oder die Belohnung einen festgelegten Wert überschreitet und somit die Pulsbreite klein genug ist, wird die Episode beendet.
Die Funktion gibt das neue Spektrogramm, die Belohnung sowie die Information darüber, ob die Episode beendet ist, zurück.

Die Funktion *render*() stellt in einem Fenster die Plots der Phasenfunktion, des Pulses im Zeitbereich, sowie das Spektrogramm wie in Abbildung 5.1 gezeigt, dar. Diese Funktion kann aufgerufen werden, um den Trainingsfortschritt visuell zu verfolgen. Die Darstellung der Plots verlangsamt den Trainingsprozess jedoch deutlich. Daher ist es ratsam, die Funktion z.B. nur in der ersten Episode des Trainingsprozesses aufzurufen, um nachvollziehen zu können ob sich die Simulation wie erwartet verhält.

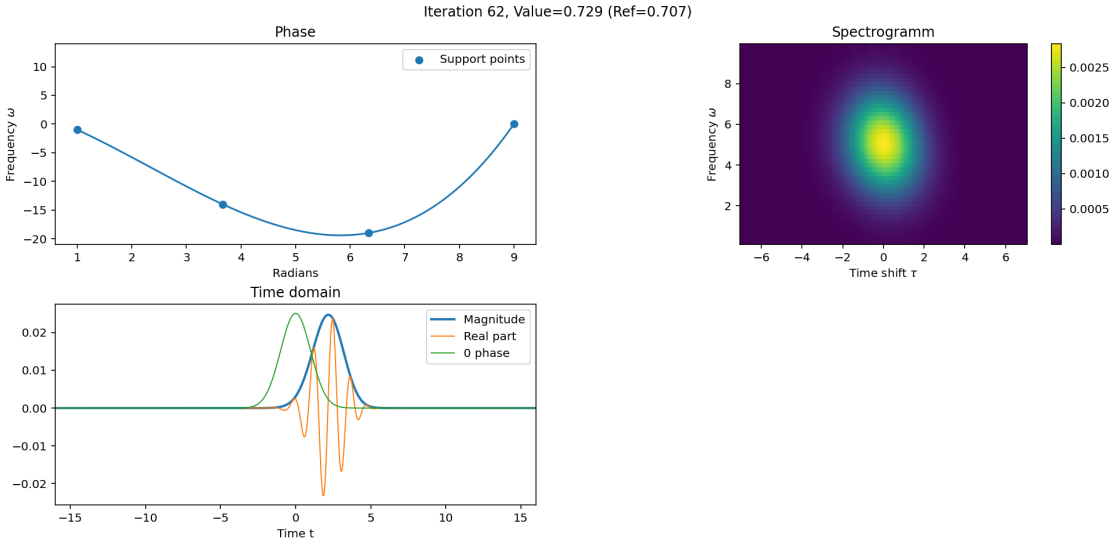

Abbildung 5.1: Visualisierung der Simulation durch die *render*-Funktion

5.4 Der Agent

Die Klasse “Agent“ enthält das zu trainierende Netzwerkmodell und das sogenannte “Replay Memory“.

5.4.1 Replay Memory

Das Replay Memory ist eine Technik, die beim RL verwendet wird, um Erfahrungen aus früheren Episoden zu speichern und später wiederzuverwenden. Das Replay Memory wird als Deque gespeichert.

Wenn der Agent eine neue Erfahrung sammelt, wird sie in dem Replay Memory gespeichert. Wenn das Memory voll ist, wird die älteste Erfahrung überschrieben. Der Agent

kann das Replay Memory verwenden, um aus früheren Erfahrungen zu lernen, indem er eine zufällige Stichprobe von Erfahrungen aus dem Speicher auswählt und diese erneut durchläuft. Dieses erneute Durchlaufen von Erfahrungen aus dem Replay Memory wird als Erfahrungswiederholung bezeichnet.

Das Replay Memory wird während des Trainings dafür genutzt, zufällige Mini Batches zu bilden.

5.4.2 Funktionen des Agenten

Die Funktion *createModel* erstellt das neuronale Netzwerk, das durch die Keras Bibliothek einfach zu konfigurieren ist. Hier wird auch der Optimizer für das Netzwerk festgelegt.

Die Initialisierungsfunktion erstellt mithilfe der *createModel* Funktion zwei Netzwerke mit unterschiedlich initialisierten Gewichten. Das "Target Netzwerk" sowie das "Main Netzwerk", das dazu dient, den Trainingsprozess zu stabilisieren. Außerdem wird das Replay Memory als ein Deque erstellt.

Die *getQs* Funktion gibt die Q-Werte des Main Netzwerks zu dem jeweiligen Zustand zurück.

5.4.3 Training des Agenten

Der Trainingsprozess des Agenten funktioniert wie folgt:
Zu Beginn einer Episode wird die Umgebung zurückgesetzt und somit ein Startzustand festgelegt. Als nächstes wird eine Aktion ausgewählt. Dazu wird unterschieden, ob eine zufällig generierte Zahl zwischen null und eins, größer als das momentane Epsilon ist.Ist sie größer als Epsilon, wird durch die Funktion *getQs* die Aktion mit dem höchsten Q-Wert ausgewählt. Andernfalls wird eine zufällige Aktion ausgewählt.
Nun wird die Funktion *step* der Umgebung mit der ausgewählten Aktion aufgerufen, um den neuen Zustand und die Belohnung zu erhalten.
Die "Erfahrung", bestehend aus Zustand, Aktion, Belohnung und neuem Zustand, wird nun in dem Replay Memory gespeichert.
Sobald eine bestimmte Anzahl von Erfahrungen gespeichert wurde, wird mit jedem

Schritt einer Episode ein zufälliger Mini Batch aus den bestehenden Erfahrungen gebildet. Zu jedem Zustand in dem Mini Batch werden die Q-Werte mit dem Main Netzwerk vorhergesagt. Die Q-Werte der jeweiligen Folgezustände werden von dem Target Netzwerk vorhergesagt.
Zu jeder Erfahrung wird nun der neue Q-Wert berechnet:

$$Q_{neu} = Belohnung + Discount \cdot maxQ(s_{t+1}, a) \tag{5.6}$$

Der Q-Wert der letzten Aktion in einer Episode entspricht nur der Belohnung, da es keinen Q-Wert für den Folgezustand gibt.
Die Formel zur Berechnung der Q-Werte weicht von der in Abbildung 2.12 ab, da der aktuelle Q-Wert sowie die Lernrate in dem Gradientenabstieg des Netzwerks vorkommen und hier nicht extra berücksichtigt werden müssen.
Es werden zwei Arrays erstellt. Eins mit den aktuellen Zuständen der Erfahrungen und eins mit den Q-Werten der Vorhersage des Main Netzwerks. Nur die Q-Werte der jeweils ausgewählten Aktion werden durch die neu berechneten Q-Werte ersetzt, die anderen bleiben unverändert.
Die fit Funktion von Keras, die die Gewichte an die gewünschten Ausgabewerte anpasst, wird von dem Main Netzwerk mit den beiden Arrays aufgerufen. Es wird also zunächst nur das Main Netzwerk angepasst.
Dieser Ablauf wird so lange wiederholt, bis die Episode beendet wurde. Die Schritte der aktuellen Episode sind nicht zwangsläufig die Erfahrungen, mit denen das Netzwerk zu dem Zeitpunkt trainiert wird, da die Mini Batches aus zufälligen Erfahrungen aus dem gesamten Replay Memory bestehen. Dieses Verhalten lässt sich über die Größe des Replay Memory sowie die Größe der Mini Batches steuern.

Das Vorhersagen der Q-Werte des nächsten Zustands mithilfe des Target Netzwerks anstelle des Main Netzwerks sorgt dafür, dass der Trainingsprozess stabiler abläuft. Es wird die Gefahr minimiert, dass eine Aktion aufgrund von zufälligen Initialisierungen und Ereignissen überbewertet wird und sich nur sehr langsam dem “wahren“ Q-Wert annähern würde.
Die Gewichte des Target Netzwerks werden nach einer bestimmten Anzahl von Episoden an die Gewichte des Main Netzwerks angepasst. Dabei können die Werte direkt übernommen werden, was den gewünschten Effekt minimieren würde oder sie werden durch die “Soft Update“ Methode angepasst:

$$W_{Target} = \tau \cdot W_{Main} + (1 - \tau) \cdot W_{Target} \tag{5.7}$$

τ ist hierbei ein einstellbarer Parameter zwischen null und eins.

Ein Ablaufdiagramm zu dem Trainingsprozess ist in den Abbildungen A.1 und A.2 im Anhang dargestellt.

5.5 Gespeicherte Daten

Um relevante Daten während des Trainingsprozesses zu speichern, wird die Klasse "ModifiedTensorboard" erstellt. Zu Beginn des Trainingsprozesses wird ihre Initialisierungsfunktion ausgeführt. Sie setzt den Episodenzähler auf eins und initialisiert den "FileWriter" mit dem gewünschten Dateipfad. Am Ende jeder Episode wird die Funktion *updateStats* ausgeführt. Sie erweitert die Datei mit den gewünschten Daten einer Episode in einem Format, sodass mit dem TensorBoard Plots ausgegeben werden können. Die gespeicherten Daten enthalten:

- den Wert des Parameters Epsilon in der Episode
- die Breite des Pulses mit dem die Episode beendet wurde
- die Anzahl der ausgeführten Schritte in der Episode
- die Veränderung der Pulsbreite $\Delta width$ in der Episode

Damit die Gewichte eines Netzwerks gespeichert werden, müssen einige Bedingungen erfüllt werden. Würde man beispielsweise die Gewichte aller Episoden speichern, wäre wahrscheinlich zu viel Speicherplatz erforderlich. Nur die Gewichte der letzten Epsiode zu speichern, ist häufig auch nicht sinnvoll, da ein längerer Trainingsprozess nicht immer zu besseren Ergebnissen führt. Somit würden eventuell Netzwerke mit besserer Performance verworfen werden. Eine Bedingung für das Speichern der Gewichte ist, dass die Belohnung bzw. die Pulsbreite am Ende der Episode den festgelegten Wert $MinReward$ erreicht hat. Außerdem muss die Verbesserung der Pulsbreite $\Delta width \geq 1$ sein, damit zufällig gut Initialisierte Stützstellen nicht als Lösen der Aufgabe gesehen werden. Um den Zufallsfaktor weiter zu reduzieren, ist eine weitere Bedingung, dass der Parameter Epsilon seinen Minimalwert erreicht hat. Somit wird sichergestellt, dass die guten Entscheidungen bewusst und nicht zufällig getroffen wurden.

5.6 Netzwerkarchitektur - Voll verbundenes oder Faltungsnetzwerk?

Im Vergleich zu Faltungsnetzwerken haben voll verbundene Netzwerke sehr viele Parameter und damit einen sehr hohen Trainingsaufwand. In Anwendungen wie der Spracherkennung wird auch mit Spektrogrammen gearbeitet und es werden meistens Faltungsnetzwerke eingesetzt. Dabei geht es eher um die Unterscheidung verschiedener Formen im Spektrogramm. Diese können aufgrund von verschiedenen Stimmlagen, an unterschiedlichen Orten im Spektrogramm auftauchen. Da die Faltungsnetzwerke bei Anwendungen mit Spektrogrammen gute Ergebnisse erzielen, liegt es nahe, diese Netzwerkart zuerst auszuprobieren.

5.6.1 Transfer learning

Ein Netzwerk komplett neu zu trainieren, lässt zwar individuelle Architekturen zu, stellt aber je nach komplexität des Netzwerkes einen hohen Rechenaufwand dar.

Eine Alternative zum neu trainieren des gesamten Netzwerks ist das sogenannte Transfer Learning. Hier wird ein bereits trainiertes Netzwerk für die eigene Anwendung genutzt. Für eine gute Performance sollte das Netzwerk auf ähnlichen Daten wie in der neuen Anwendung trainiert worden sein. Ein Netzwerk zur Klassifizierung von Hunderassen, kann wahrscheinlich auch andere Tiere gut klassifizieren.

Soll nun eins der gängigen Netzwerke, das auf Objekt-Klassifizierung trainiert wurde, für diese Anwendung benutzt werden, geht man folgendermaßen vor:
Der Feature Extractor, auch Convolutional Base genannt, enthält alle Schichten bis hin zur Flatten Schicht. Diese Convolutional Base wird mitsamt ihrer Gewichte aus dem trainierten Netzwerk übernommen. Der hintere Teil des Netzwerks mit den restlichen Schichten wird Classifier genannt. Dieser muss nur für die neue Anwendung trainiert werden. Bei dem Trainingsprozess werden nun also die Gewichte der Convolutional Base “eingefroren“ und nur die Gewichte des Classifiers angepasst.

Ein Problem bei der Nutzung eines Netzwerks, das zur Objekt-Klassifizierung auf Farbbildern trainiert wurde, ist dass diese Netzwerke drei Farbkanäle erwarten und ein Spektrogramm nur einen Farbkanal hat.

Ein Lösungsansatz für dieses Problem ist, die Eingangsschicht des vortrainierten Netzwerks mit einer Schicht zu ersetzen, die ein Bild mit nur einem Farbkanal erwartet. Dies würde allerdings zur Folge haben, dass sich das Verhalten der nachfolgenden Schichten ändern würde und der Feature Extractor nicht mehr wie gedacht funktioniert. Es müsste die Convolutional Base neu trainiert werden, was diese Lösung unbrauchbar macht.

Ein anderer Ansatz ist, den einen Farbkanal des Spektrogramms auf die anderen Farbkanäle zu kopieren. Dies würde zwar die Convolutional Base weiter funktionieren lassen, aber es würden keine farbbasierten Merkmale erkannt werden. Da es keine farbbasierten Merkmale in einem Spektrogramm gibt, wäre dies vermutlich kein Problem. Es wären dennoch redundante Informationen ohne Mehrwert für das Netzwerk.

Ein dritter Ansatz ist, ein vortrainiertes Netzwerk zu finden, dass auf Audiosignale trainiert wurde und somit auch auf Spektrogrammen trainiert ist. Dadurch wäre das Farbkanal-Problem gelöst. Problem hierbei ist, dass die Netzwerke ein Audiosignal als Eingang erwarten.

Wenn ein Netzwerk trainiert werden soll, ist es wichtig die Eingangsdaten in ein einheitliches Format zu bringen. Dieser Vorgang nennt sich Preprocessing. Dies beinhaltet in der Regel die Vorbereitung der Daten, die Anwendung von Augmentations-Techniken, die Normalisierung und die Skalierung auf einen gemeinsamen Wertebereich.

Beim Transfer Learning ist es besonders wichtig, die Input-Daten im gleichen Format zu haben, wie die Daten, mit denen das Netzwerk trainiert wurde. Wenn die Input-Daten nicht in einer gleichen Form vorliegen, haben die Gewichte des vortrainierten Netzwerks keine Aussagekraft bezüglich der neuen Daten, was zu schlechter Leistung und ungenauen Vorhersagen führt.

5.7 Hyperparameter

Hyperparameter im ML sind Parameter, die den Algorithmus eines ML-Modells beeinflussen, aber nicht direkt aus den Trainingsdaten gelernt werden können. Sie müssen manuell ausgewählt oder optimiert werden, um die Leistung des Modells zu verbessern.

Die Wahl der Hyperparameter hat einen großen Einfluss auf die Leistung des Modells und kann das Modell stark verbessern oder verschlechtern. Eine Möglichkeit Hyperparameter auszuwählen besteht darin, verschiedene Kombinationen von Hyperparameterwerten auszuprobieren und die Leistung des Modells zu bewerten. Eine andere Möglichkeit besteht darin, automatische Hyperparameteroptimierungstechniken wie Grid Search, Random Search oder die Funktion "HParams" von TensorFlow zu verwenden, um den besten Satz von Hyperparametern für das Modell zu finden.

Im RL gibt es neben den Hyperparametern des Netzwerks und Trainingsprozesses noch einstellbare Parameter für die Umgebung und für den Agenten.

5.7.1 Hyperparameter eines Netzwerkmodells

Zu den wichtigsten Hyperparametern gehören die Anzahl der Schichten im Modell, die Anzahl der Neuronen in jeder Schicht, die Aktivierungsfunktionen in den Schichten, die Lernrate und die Batch-Größe.

Die Anzahl der Schichten, sowie die Anzahl der Neuronen pro Schicht, beeinflussen direkt die Komplexität des Netzwerks. Ein tieferes Netzwerk kann in der Regel komplexere Muster erlernen, aber es kann auch zu Overfitting führen.
Schichten wie Dropout oder Pooling haben eigene Parameter die festgelegt werden müssen und einen Einfluss auf den Trainingsprozess haben.

Die Lernrate steuert, wie schnell das Modell lernt, während es trainiert wird. Eine zu hohe Lernrate kann dazu führen, dass das Modell zu schnell lernt und wichtige Muster überspringt, während eine zu niedrige Lernrate dazu führen kann, dass das Modell zu langsam lernt oder beim Gradientenabstieg in einem lokalen Minimum feststeckt.

5.7.2 Hyperparameter der Simulation

Anzahl der Stützstellen

Eine hohe Anzahl von gut gewählten Stützstellen führt zu einer besseren Modulation der Phasenfunktion. Jedoch ist es für den Trainingsprozess von Nachteil, wenn der Aktionsraum zu groß ist. Der Aktionsraum wächst durch die Anzahl der Stützstellen mit dem Faktor zwei (für die Verschiebung positiv und negativ).

Initialisierung der Stützstellen

Es ist sinnvoll, die Stützstellen für jede Episode mit zufälligen Werten anstelle von Nullen zu initialisieren. Dadurch werden deutlich mehr Zustände erreicht. Zudem ist es näher an der Realität, da die Dispersionseffekte je nach Eingangssignal nicht immer gleich sind.

Action Step Size

Die Action Step Size bestimmt, wie groß die Verschiebung einer Stützstelle pro Schritt sein soll. Dieser Parameter muss zum einen klein genug gewählt werden, damit der Puls genau genug in Form gebracht werden kann, zum anderen groß genug, damit es eine merkbare Veränderung des Zustands gibt. Dies beeinflusst auch die Belohnungsfunktion, da eine kleine Änderung des Zustands auch nur eine kleine Änderung der Belohnung zur Folge hat.

5.7.3 Hyperparameter des Trainingsprozesses

Discount

In den meisten RL-Aufgaben wird der Disount auf Werte im Bereich von >0.9 festgelegt. In dieser Aufgabenstellung führt jedoch häufig die richtige Verschiebung einer Stützstelle auch langfristig zu einer höheren Belohnung. Somit kann ausprobiert werden, ob ein niedrigerer Discount schneller zu einem guten Ergebnis führen kann und somit den Trainingsprozess beschleunigt. Der Trainingsprozess ist bei einem niedrigeren Discount schneller, da es weniger Durchläufe von ähnlichen Zuständen braucht, um die zeitliche Differenz nah an den Wert Null zu bringen.

Anzahl der Episoden

Die Anzahl der Episoden ist ein Hyperparameter, der in jedem Trainingsprozess eines neuronalen Netzes vorkommt. In der Regel soll lange genug trainiert werden, sodass die Genauigkeit möglichst hoch wird. Fängt die Lernkurve allerdings an zu stagnieren, sollte der Trainingsprozess abgebrochen werden, um nicht in den Bereich des Overfittings zu kommen. Beim RL muss die Anzahl der Episoden zu dem Epsilon decay passen. Nachdem Epsilon beim Minimum angekommen ist, muss noch für eine gewisse Anzahl von Episoden weiter trainiert werden.

Schritte pro Episode

Die Anzahl der Schritte pro Episode muss unter Berücksichtigung der Anzahl von Stützstellen und der Schrittweite, festgelegt werden. Aufgrund der zufälligen Initialisierung der Stützstellen sollte es auf jeden Fall möglich sein, die Stützstellen um jeweils 2 Pi bewegen zu können. Da das Netz besonders zu Beginn nicht nur gute Entscheidungen treffen wird, sollte ein entsprechender Puffer an Schritten eingeplant werden.

Epsilon Decay

Das Epsilon wird üblicherweise durch das Multiplizieren mit dem Faktor 0,99 in jeder Episode verringert.

Mini-Batch-Size

Die Wahl der Mini-Batch-Size beim RL hängt von Faktoren wie der Größe des Zustandsraums, der Größe des Aktionsraums und der Komplexität des Modells ab.
Eine zu kleine Mini-Batch-Size kann zu einer ineffizienten Nutzung der Ressourcen des Systems führen, da es mehr Schritte benötigt, um den gleichen Datensatz durchzuarbeiten. Eine zu große Mini-Batch-Size kann hingegen zu einer Überbeanspruchung des Speichers führen und die Rechenleistung beeinträchtigen.

Random Seed

Durch das Festlegen des gleichen Random Seeds für jeden Trainingsprozess werden immer die gleichen zufälligen Startwerte für die Stützstellen benutzt. Dadurch sind die Trainingsprozesse besser untereinander zu vergleichen. Für das Anwenden der trainierten Netze wird kein Random Seed festgelegt, um nicht nur die bereits vom Training bekannten Szenarien zu erhalten.

6 Trainingsversuche verschiedener Netzwerke

Im Folgenden werden jeweils die Trainingsprozesse der bisher besten Netzwerkmodelle und Hyperparameterkonfigurationen für vollverbundene, Faltungs- und Resdidual Netzwerke dargestellt. Die Anzahl der Stützstellen wird auf vier festgelegt. Das vereinfacht die Aufgabe weitestgehend, da es die Anzahl der Aktionen reduziert. Durch Tests mit dem evolutionären Optimierungsverfahren wurde sichergestellt, dass vier Stützstellen die Phasenfunktion noch genau genug abbilden können.
Wie zuvor erwähnt, sind alle Werte bezüglich der Pulse einheitenlos.

6.1 Voll verbundenes Netzwerk

In diesem Trainingsversuch wird ein voll verbundenes Netzwerk mit der folgenden Architektur benutzt:

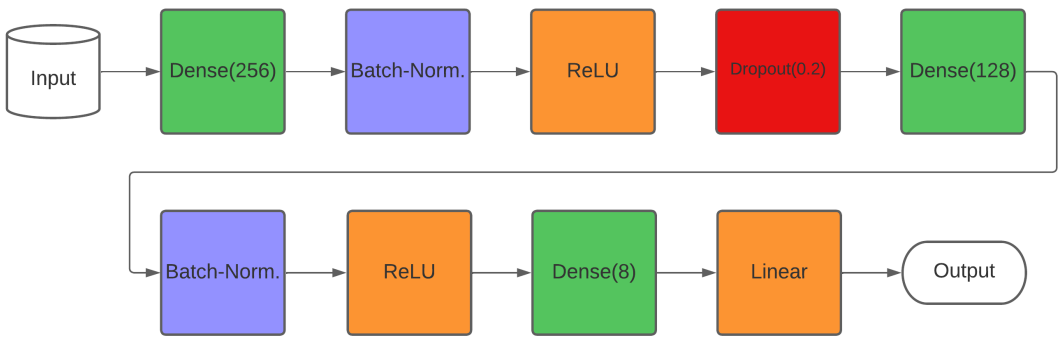

Abbildung 6.1: Architektur des verwendeten voll verbundenen Netzwerks

Die Daten des Spektrogramms werden auf den Wertebereich von null bis eins normiert. Die Stützstellen werden zu Beginn jeder Episode mit zufälligen Werten zwischen null und zwei Pi initialisiert. Die hier benutzten Hyperparameter wurden durch Variationen bei vorherigen Trainingsprozessen optimiert:

- Discount = 0.9
- Episoden = 15000
- Steps per Episode = 350
- Minibatch Size = 32
- Action Step Size = 0.1
- Epsilon Decay = 0.995
- Min Epsilon = 0.001
- Min Reward = 140
- Replay Memory Size = 50000
- Update Target Netzwerk alle 5 Episoden

Die folgende Belohnungsfunktion gilt für alle Trainingsversuche und deckt einen Wertebereich von ca. -70 bis 150 ab. Die Belohnung 150 entspricht etwa der Breite 0,71 und somit einem, in dieser Simulation, bandbreitenlimitiertem Puls:

$$reward = -200 + (width^{-1} \cdot 250) \tag{6.1}$$

Der Trainingsprozess verläuft, wie in Abbildung 6.2 dargestellt recht unstabil. Es ist oben das Delta der Pulsbreite sowie unten die Pulsbreite am Ende der Episode dargestellt. Im Bereich von Episode 12000 wird die Pulsbreite durchschnittlich um 0,59 auf einen Wert von 1,4 verringert.

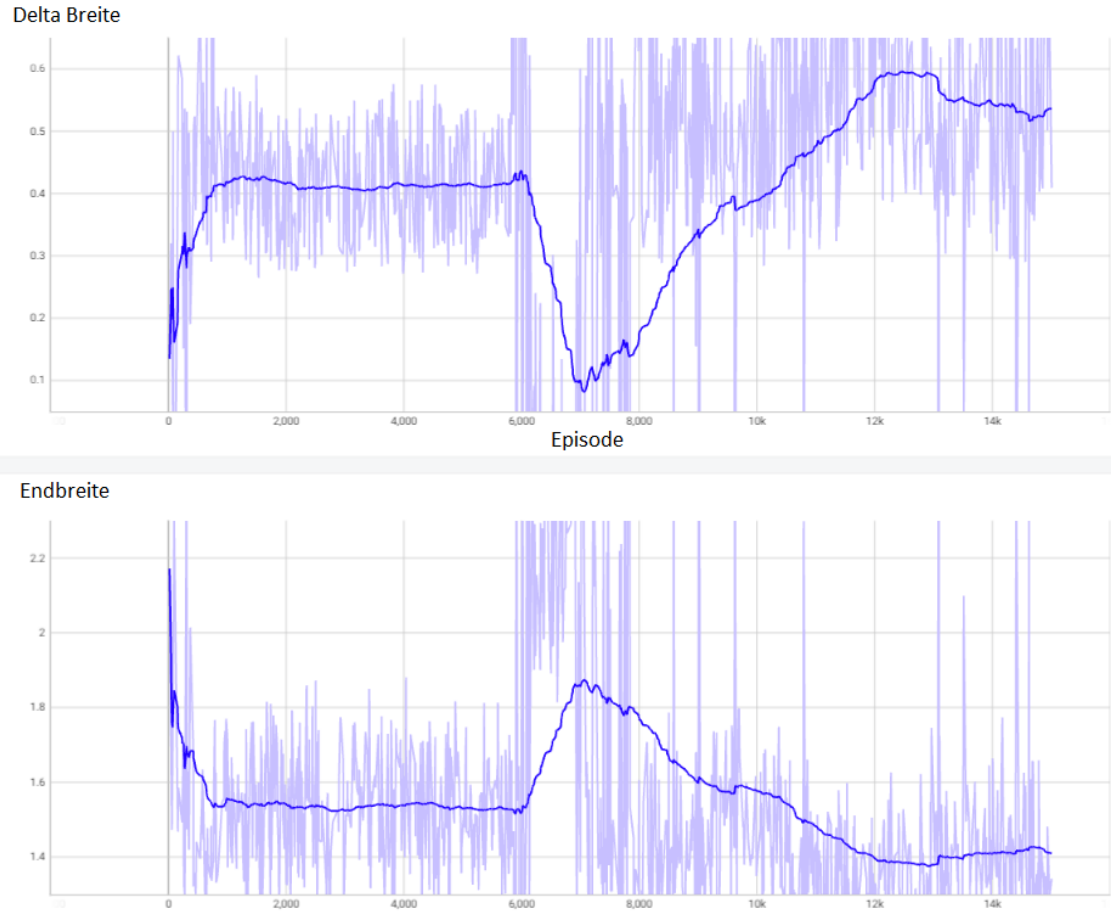

Abbildung 6.2: Trainingsversuch: Voll verbundenes Netzwerk

6.2 Faltungsnetzwerk

In diesem Trainingsversuch wird ein Faltungsnetzwerk, das nur einen Farbkanal hat, mit der folgenden Architektur benutzt:

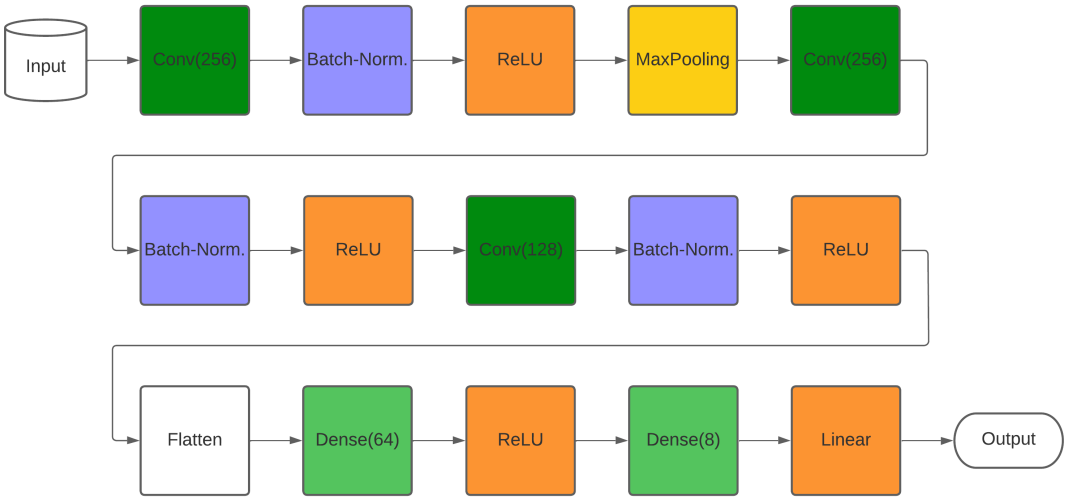

Abbildung 6.3: Architektur des Faltungsnetzes

Die Daten des Spektrogramms werden auf den Wertebereich von null bis eins normiert. Die Stützstellen werden zu Beginn jeder Episode mit zufälligen Werten zwischen null und zwei Pi initialisiert. Die hier benutzten Hyperparameter wurden durch Variationen bei vorherigen Trainingsprozessen optimiert:

- Discount = 0.7
- Episoden = 10000
- Steps per Episode = 200
- Minibatch Size = 32
- Action Step Size = 1
- Epsilon Decay = 0.999
- Min Epsilon = 0.001

- Min Reward = 140
- Replay Memory Size = 50000
- Update Target Netzwerk alle 5 Episoden

Abbildung 6.4 zeigt, dass der Trainingsprozess sehr unstabil verläuft. Die Pulsbreite am Ende der jeweiligen Episode schwankt sehr stark und liegt im Durchschnitt bei 1,5. Die Pulsbreite wird im Durchschnitt um 0,5 verringert.

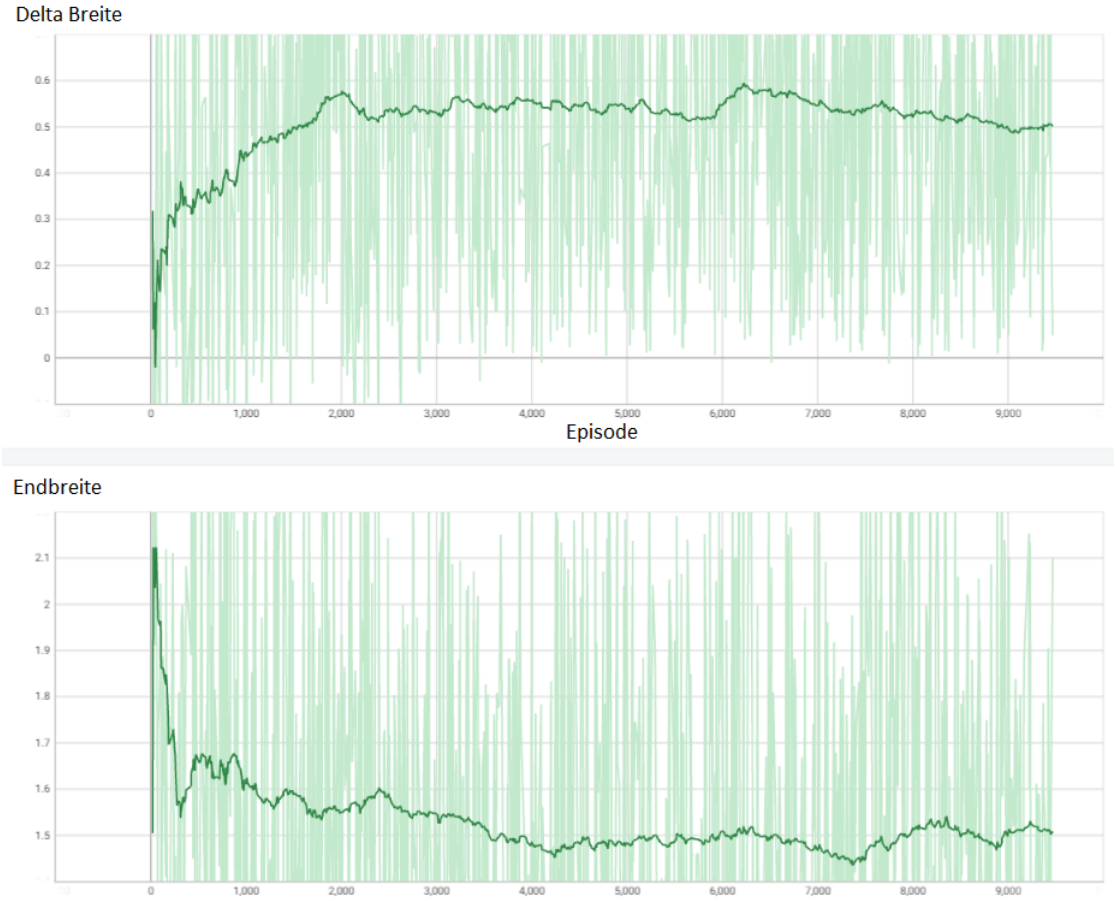

Abbildung 6.4: Trainingsversuch: Faltungsnetzwerk

6.3 Residual Network 50 Version 2

In diesem Trainingsversuch wird Transfer Learning mit dem Netzwerkmodell des Residual Netwerks 50 (ResNet50) Version 2 angewandt.
ResNets sind ein spezieller Typ von neuronalen Netzwerken, die das Problem des verschwindenden Gradienten lösen, das bei der Verwendung von sehr tiefen Netzen auftritt. ResNets verwenden sogenannte Residual-Blöcke, um die Informationsübertragung innerhalb des Netzes zu verbessern.

Anstatt die Ausgabe eines Blocks direkt zur nächsten Schicht weiterzuleiten, wird die Ausgabe über eine "Shortcut-Verbindung", wie in Abbildung 6.5 dargestellt, zur übernächsten Schicht geleitet. Diese Verbindung ermöglicht es dem Netzwerk, Informationen zu speichern und weiterzugeben, die in den vorherigen Schichten verloren gehen würden.

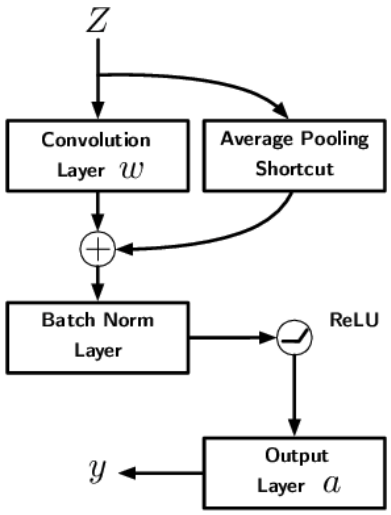

Abbildung 6.5: Shortcut eines Residual Netzwerks [20]

Die Shortcut-Verbindung ermöglicht es dem Netzwerk zu erlernen, welche Schichten übersprungen werden sollen, um die bestmögliche Genauigkeit zu erreichen. Dadurch können ResNets deutlich tiefer trainiert werden als herkömmliche Netzwerke, was zu höheren Genauigkeiten und besserer Generalisierungsfähigkeit führt.

ResNets haben in vielen Bereichen der Bildverarbeitung und des ML sehr gute Ergebnisse erzielt und gehören zu den leistungsfähigsten Architekturen für die Bildklassifizierung, Objekterkennung und Segmentierung. [20]

Die Architektur des ResNet50 V2 ist in Abbildung 6.6 dargestellt:

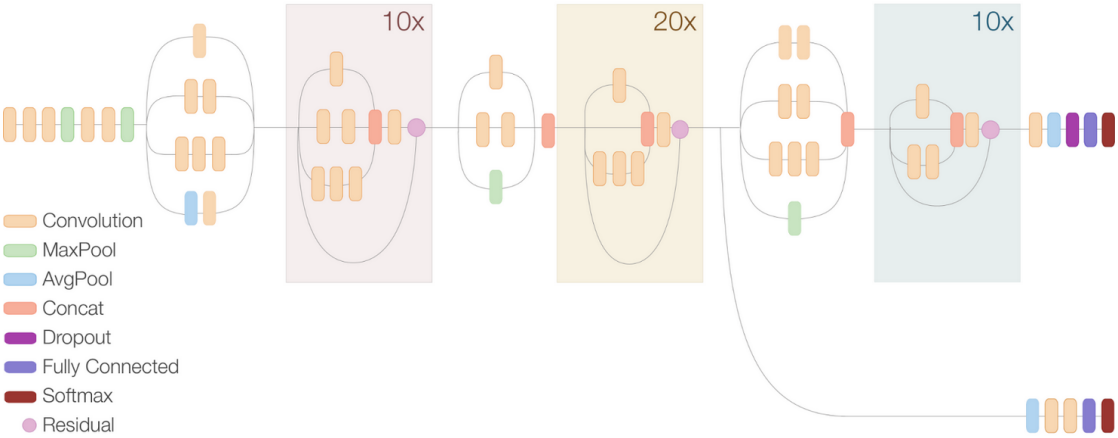

Abbildung 6.6: Netzwerk Architektur von ResNet50 V2 [12]

Um das Spektrogramm in die Form des erwarteten Inputs zu bringen, wird der eine Farbkanal des Spektrogramms auf alle 3 Farbkanäle kopiert. Das Farbbild wird durch die ResNet50 V2 preprocessing Funktion so aufbereitet, wie die Bilder mit denen das Netz trainiert wurde.

Die hier benutzten Hyperparameter wurden durch Variationen bei vorherigen Trainingsprozessen optimiert:

- Discount = 0.9
- Episoden = 5000
- Steps per Episode = 350
- Minibatch Size = 32
- Action Step Size = 0.1
- Epsilon Decay = 0.995
- Min Epsilon = 0.001
- Min Reward = 140
- Replay Memory Size = 50000
- Update Target Netzwerk alle 5 Episoden

Abbildung 6.7 zeigt, dass der Trainingsprozess im Vergleich zu den anderen recht stabil verläuft. Die Pulsbreite am Ende der jeweiligen Episode wird im Durchschnitt um 1,15 auf 0,79 verringert.

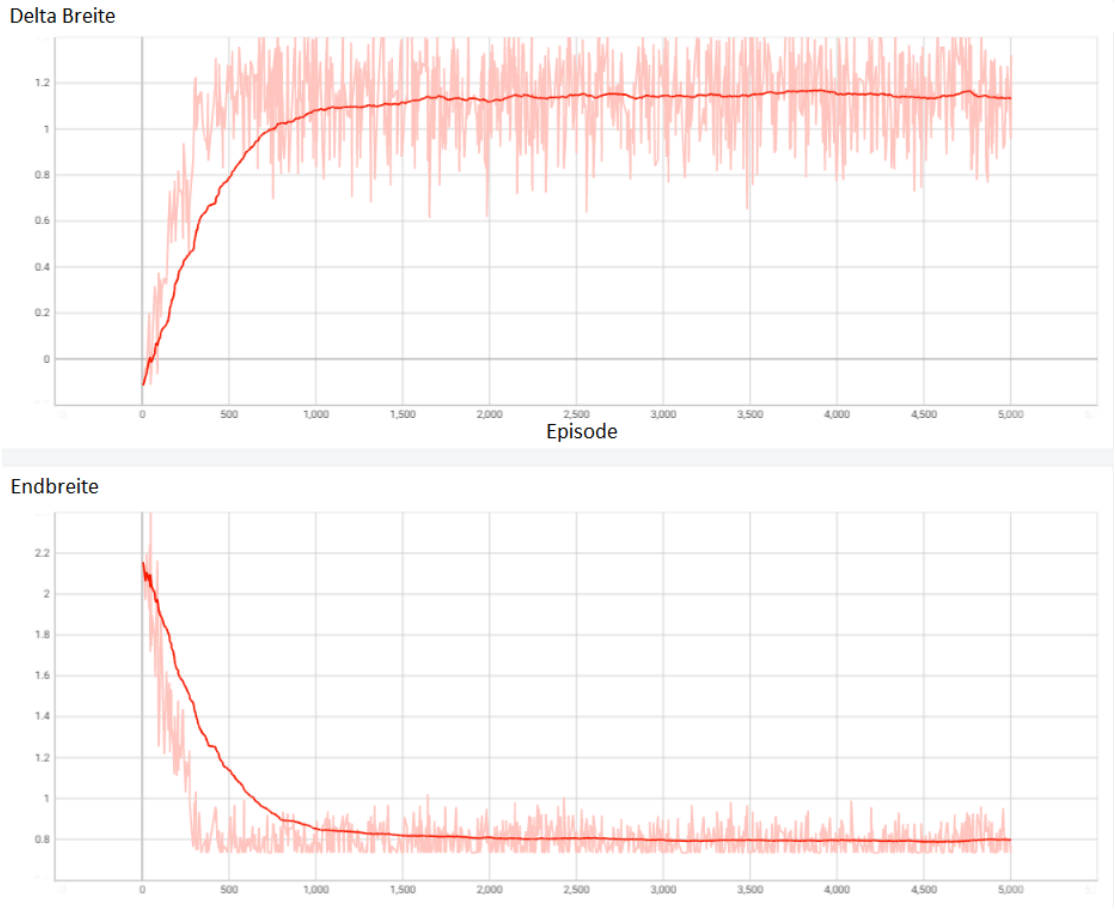

Abbildung 6.7: Trainingsversuch: ResNet50 V2

Das ResNet scheint die Aufgabe sehr gut zu lösen und hat dabei den kürzesten Trainingsprozess.
Schaut man sich mithilfe der *render*-Funktion eine Episode der Pulsformung an, fällt aber auf, dass lediglich die hinterste Stützstelle so weit nach oben verschoben wird, bis entweder die gewünschte Pulsbreite erreicht ist oder die maximale Anzahl der Schritte erreicht wurde.
Das ist nicht effizient und gleicht einem systematischen Ausprobieren, ohne dass die Informationen des Spektrums genutzt werden.

Um diese Vorgehensweise einzuschränken, werden Limitierungen für die Stützstellen festgelegt. Diese liegen bei -2 Pi und 4 Pi. In Abbildung 6.8 ist der Trainingsprozess des ResNets mit der sonst selben Konfiguration dargestellt. Die Pulsbreite wird im Durchschnitt um 0,5 auf 1,43 verringert.

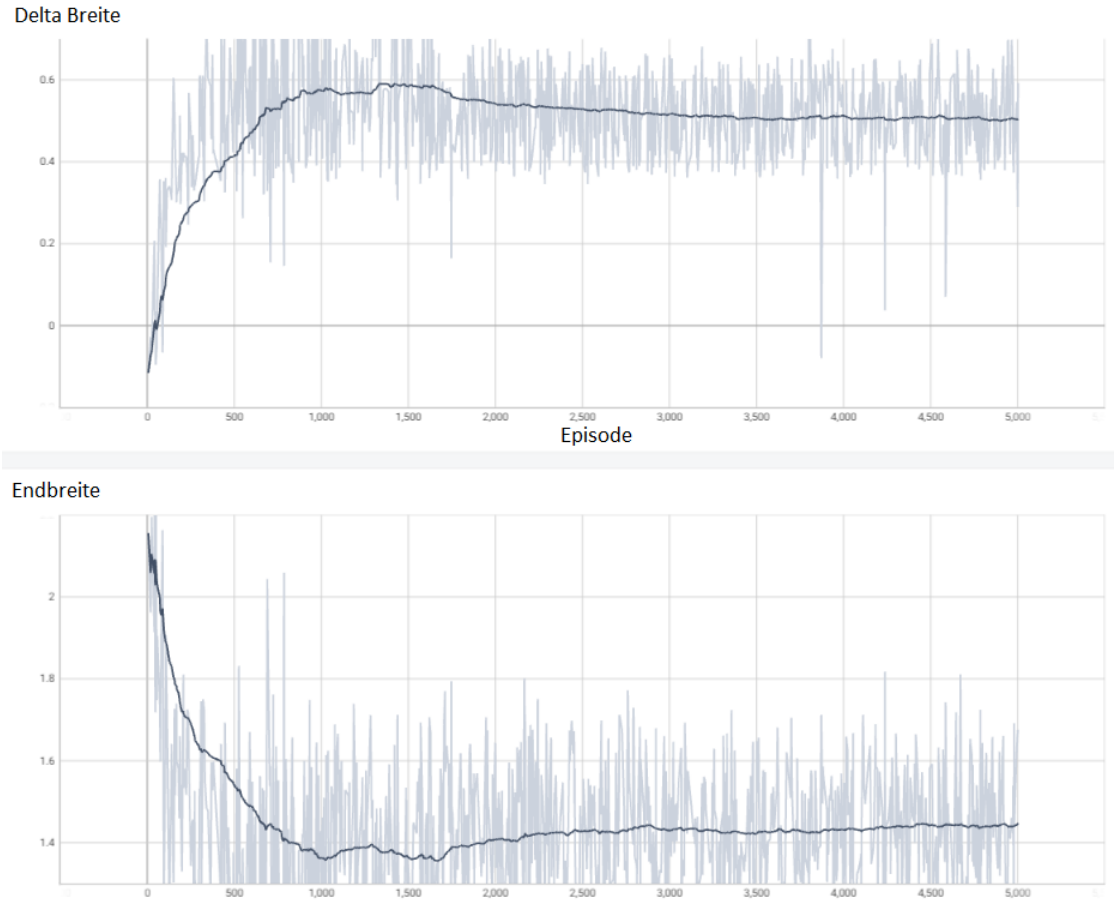

Abbildung 6.8: Trainingsversuch: ResNet50 V2 mit Limits

7 Anwenden der trainierten Netzwerke

Um die Performance der RL Agenten einschätzen zu können, werden die trainierten Agenten mit dem evolutionären Optimierungsverfahren, das ein bisher gängiges Verfahren zur Pulsformung ist, verglichen. Hierbei werden das voll verbundene Netzwerk (Dense) und die ResNets jeweils mit und ohne Grenzen berücksichtigt. Das Faltungsnetzwerk war im Trainingsprozess zu unstabil als dass es in Frage kommen würde. Faltungsnetzwerke sind dadurch aber nicht zwangsläufig ungeeignet für diese Aufgabe.
Es soll die Pulsbreite bei 100 Pulsen, mit jeweils unterschiedlichen Startwerten für die einzustellende Phasenfunktion, minimiert werden.
Verglichen werden hierbei die durchschnittliche Pulsbreite und die durchschnittliche Anzahl der benötigten Schritte.

7.1 Anwenden auf Gauss-Pulse aus dem Training

Zunächst wird die Performance bei einfachen Gauss-Pulsen, wie sie auch im Training vorkamen, in Tabelle 7.1 verglichen. Hierbei ist zu beachten, dass beim Training die Abbruchbedingung für die Episoden jeweils bei einer Pulsbreite von 0,74 lag. Somit ist eine Pulsbreite von 0,74 das bestmögliche Ergebnis der RL Agenten. Die durchschnittliche Startbreite der Pulse liegt bei 1,93.

Die gewünschte Pulsform ist in Abbildung 7.1 in blau dargestellt und die Pulsform mit Dispersionseffekten in orange.

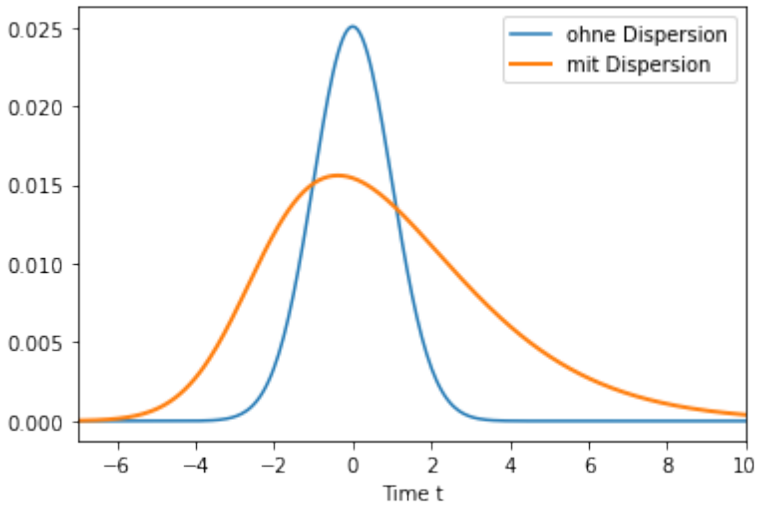

Abbildung 7.1: Dispersionseffekte auf einen einfachen Gauss-Puls

Tabelle 7.1: Vergleich zwischen evolutionärem Optimierungsverfahren und RL Agenten bei Pulsen aus dem Training

	ResNet ohne Grenzen	ResNet mit Grenzen	Dense	Evolutionär
∅ Pulsbreite	0,809	1,46	1,46	0,707
∅ Schrittzahl	336	350	350	147

Der Agent des ResNets mit Grenzen sowie der Agent des voll verbundenen Netzwerks zeigen die gleiche Performance bezüglich der Pulsbreite. Die Schrittzahl von 350 kommt dadurch zustande, dass die gewünschte Pulsbreite nie erreicht wird und somit die maximal mögliche Anzahl von Schritten ausgeführt wird.
Der Agent des ResNets ohne Grenzen kommt im Vergleich zum evolutionären Optimierungsverfahren nicht ganz auf den gewünschten bandbreitenlimitierten Puls und benötigt dafür mehr als doppelt so viele Schritte.

Die geformten Pulse, die den Durchschnittswerten entsprechen, sind in Abbildung 7.2 dargestellt. Der Puls des evolutionären Optimierungsverfahrens ist aus Übersichtlichkeitsgründen nicht mit abgebildet. Der Puls würde sich stark mit dem Puls ohne Dispersionseffekte überschneiden.

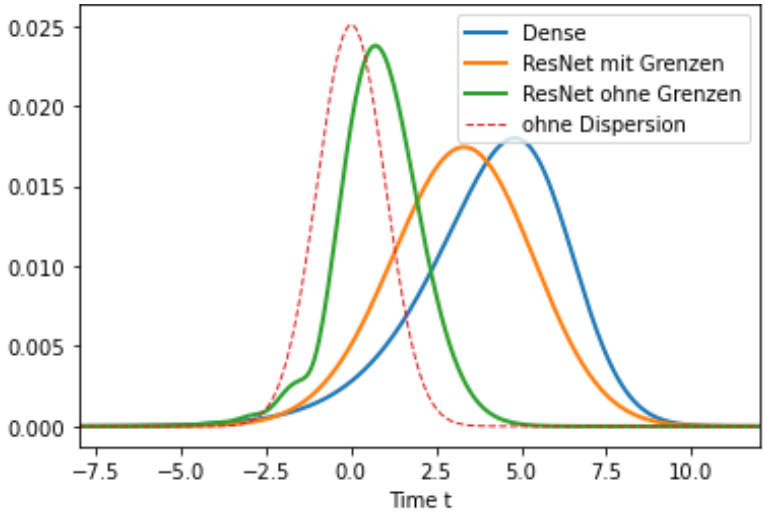

Abbildung 7.2: Geformte einfache Gauss-Pulse

7.2 Anwenden auf überlagerte Gauss-Pulse

Um die Generalisierungsfähigkeit der RL-Agenten zu testen, wird die Performance auf Pulsen, die aus zwei bzw. drei überlagerten Gauss-Pulsen bestehen, getestet.

7.2.1 Zwei überlagerte Gauss-Pulse

Die gewünschte Pulsform ist in Abbildung 7.3 in blau dargestellt und die Pulsform mit Dispersionseffekten in orange. Die Pulsbreite des Pulses ohne Dispersionseffekte beträgt 0,69. Die durchschnittliche Startbreite der Pulse liegt bei 2,54.

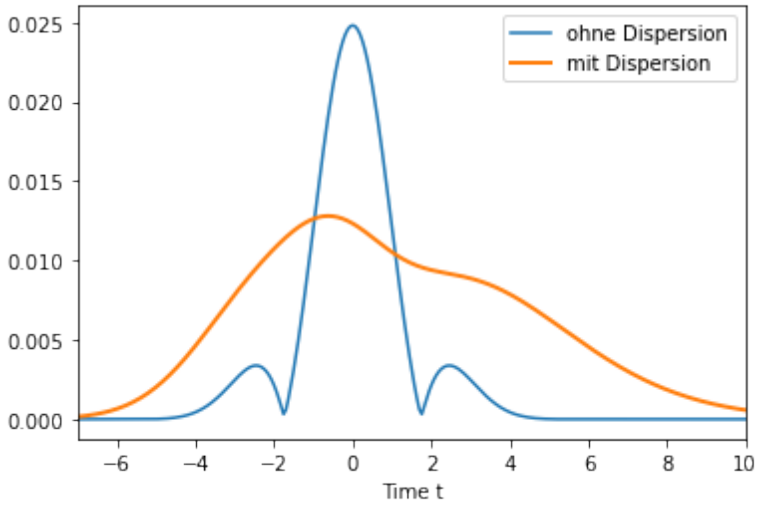

Abbildung 7.3: Dispersionseffekte auf einen Puls, bestehend aus zwei überlagerten Gauss-Pulsen

Die Performance der RL Agenten und des evolutionären Optimierungsverfahrens wird in Tabelle 7.2 verglichen.

Tabelle 7.2: Vergleich zwischen evolutionärem Optimierungsverfahren und RL Agenten bei zwei überlagerten Gauss-Pulsen

	ResNet ohne Grenzen	ResNet mit Grenzen	Dense	Evolutionär
∅ Pulsbreite	0,863	1,859	1,756	0,692
∅ Schrittzahl	339	350	350	144

Die Agenten des ResNets mit Grenzen sowie des voll verbundenen Netzwerks zeigen auch hier eine ähnliche Performance bezüglich der Pulsbreite. Der Agent des ResNets ohne Grenzen kommt auf einen ca. 7% breiteren Puls als bei dem einfachen Gauss-Puls. Das evolutionäre Optimierungsverfahren schafft es hingegen, die Dispersionseffekte vollkommen auszugleichen.

Die geformten Pulse, die den Durchschnittswerten entsprechen, sind in Abbildung 7.4 dargestellt. Der Puls des evolutionären Optimierungsverfahrens ist aus Übersichtlichkeitsgründen nicht mit abgebildet. Der Puls würde sich stark mit dem Puls ohne Dispersionseffekte überschneiden.

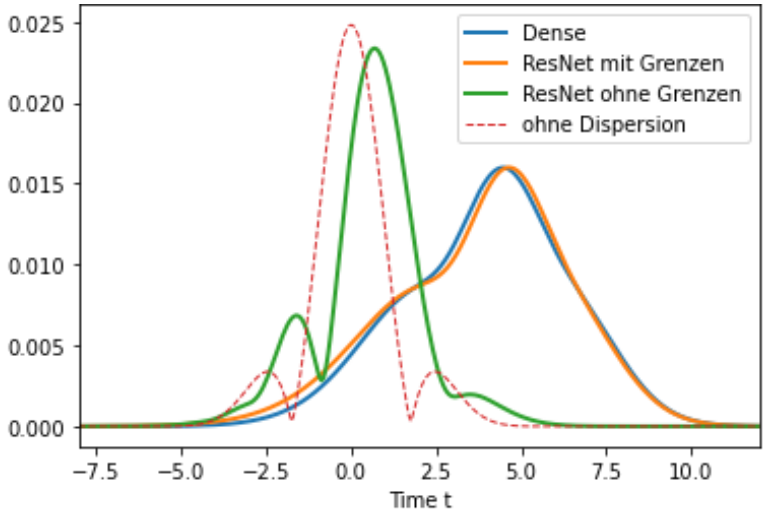

Abbildung 7.4: Geformte Pulse bei zwei überlagerten Gauss-Pulsen

7.2.2 Drei überlagerte Gauss-Pulse

Die gewünschte Pulsform ist in Abbildung 7.5 in blau dargestellt und die Pulsform mit Dispersionseffekten in orange. Die Pulsbreite des Pulses ohne Dispersionseffekte beträgt 1,11. Die durchschnittliche Startbreite der Pulse liegt bei 2,57.

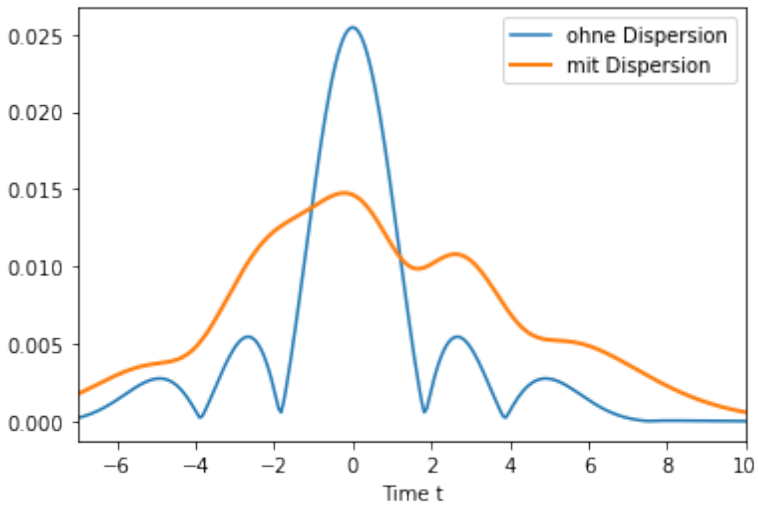

Abbildung 7.5: Dispersionseffekte auf einen Puls, bestehend aus drei überlagerten Gauss-Pulsen

Die Performance der RL Agenten und des evolutionären Optimierungsverfahrens wird in Tabelle 7.3 verglichen.

Tabelle 7.3: Vergleich zwischen evolutionärem Optimierungsverfahren und RL Agenten bei drei überlagerten Gauss-Pulsen

	ResNet ohne Grenzen	ResNet mit Grenzen	Dense	Evolutionär
∅ Pulsbreite	1,19	2,009	1,94	1,11
∅ Schrittzahl	350	350	350	106

Die Agenten des ResNets mit Grenzen sowie des voll verbundenen Netzwerks zeigen auch hier eine ähnliche Performance bezüglich der Pulsbreite. Der Agent des ResNets ohne Grenzen verfehlt die gewünschte Pulsbreite um ca. 7%. Das evolutionäre Optimierungsverfahren schafft es hingegen, die Dispersionseffekte vollkommen auszugleichen. Auffällig ist hier, dass das Optimierungsverfahren dazu nur 106 Schritte braucht, während das ResNet ohne Grenzen die gewünschte Pulsform scheinbar nie erreicht und somit alle 350 Schritte ausführt. Das liegt daran, dass die Abbruchbedingung bei einer Pulsbreite von 0,74 liegt. Eine Pulsbreite von 0,74 ist bei dieser Pulsform jedoch nicht erreichbar.

Die geformten Pulse, die den Durchschnittswerten entsprechen, sind in Abbildung 7.6 dargestellt. Der Puls des evolutionären Optimierungsverfahrens ist aus Übersichtlichkeitsgründen nicht mit abgebildet. Der Puls würde sich stark mit dem Puls ohne Dispersionseffekte überschneiden.

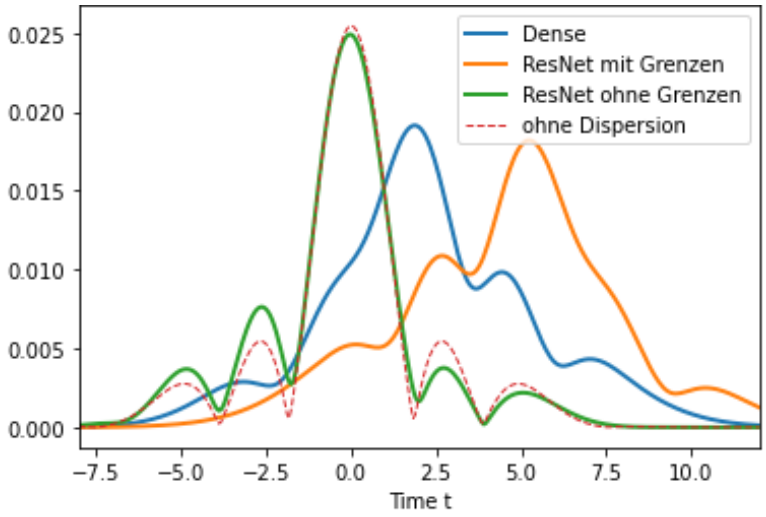

Abbildung 7.6: Geformte Pulse bei drei überlagerten Gauss-Pulsen

8 Ergebnis

Die in dieser Arbeit trainierten RL-Agenten schaffen es im Gegensatz zum evolutionären Optimierungsverfahren nicht, die Dispersionseffekte zufriedenstellend auszugleichen. Lediglich der Agent des ResNets ohne Begrenzung für die Stützstellen, schafft es, die Pulse der gewünschten Form nahezubringen. Das gelingt ihm aber nur durch systematisches Ausprobieren, indem er eine Stützstelle so lange weiter in eine Richtung verschiebt, bis die gewünschte Pulsbreite oder die maximale Anzahl an Schritten erreicht ist.

Dieses Vorgehen entspricht nicht der Vorstellung die Informationen des Spektrogramms zu nutzen um die unterschiedlichen Stützstellen jeweils so zu verschieben, dass der gewünschte Puls möglichst schnell erreicht wird.

Die RL-Agenten des voll verbundenen Netzwerks und des ResNets mit Stützstellenbegrenzung schaffen es zwar die Pulsbreite zu verringern, bleiben dabei aber weit von dem gewünschten Ergebnis entfernt.

Die Pulsformer Aufgabe lässt sich in eine Reinforcement Learning Aufgabe übersetzen und durch weitere Trainingsprozesse mit anderen Netzwerken und besserer Feinabstimmung der Hyperparameter, können die Ergebnisse sicherlich noch verbessert werden.

In dem folgenden Kapitel werden Vorschläge zur Verbesserung der RL-Lösung vorgestellt.

Wenn es gelingt die RL-Lösung weiter zu verbessern, soll diese auch für alternative Anwendungen beim Formen von Pulsen, bei denen es nicht darum geht die Pulsbreite zu minimieren, eingesetzt werden. Dafür wäre es notwendig, dass es eine messbare Größe, wie in diesem Fall die Pulsbreite gibt, die in die Belohnungsfunktion mit einfließen kann.

9 Ausblick auf weitere Arbeiten

Auf Grundlage dieser Arbeit können weitere Maßnahmen, zur Verbesserung der RL-Lösung und der Simulation, getroffen werden. Hier werden einige dieser Ansätze vorgestellt.

Verbesserung der Simulation:

Für eine bessere Generalisierbarkeit könnte es sinnvoll sein, verschiedene Pulsformen, wie die überlagerten Gauss-Pulse mit variierenden Werten für GDD und TOD, bereits im Training zu benutzen. Durch verschiedene Pulsformen würden weitere Formen und Merkmale in den Spektrogrammen entstehen, mit denen letztendlich gearbeitet wird.
Auf die simulierten Daten könnte ein Messrauschen gelegt werden, um realistischere Bedingungen für die Simulation zu schaffen.

Verbesserung der RL-Lösung:

Um die RL-Lösung zu verbessern, müssen weitere Agenten mit anderen Hyperparameter-Konfigurationen trainiert werden.
Jedoch sollten zuvor ein paar grundlegende Dinge überdacht werden. Eine Aktion des Agenten bewegt in diesem Fall nur eine Stützstelle um einen festgelegten Wert. Effizienter wäre es, wenn bei jeder Aktion alle Stützstellen gleichzeitig bewegt werden. Diese sollten dabei nicht um einen festgelegten Wert verschoben werden, sondern um einen individuellen Wert. Dementsprechend bräuchte es für jede Stützstelle ein Ausgabeneuron, aus dem sich der Wert der Verschiebung ableiten lässt.

Desweiteren sollten andere Netzwerkmodelle getestet werden. Für diese Anwendung würde sich ein auf Audiosignale trainiertes Netzwerk anbieten, da diese auch mit Spektrogrammen arbeiten. Hierfür muss die Eingabeschicht des Netzwerks so angepasst werden, dass keine Audiodatei, sondern direkt das Spektrogramm als Input genommen werden kann. Dabei ist darauf zu achten, dass sowohl die Größe des Spektrogramms, als auch der Wertebereich zu den Trainingsdaten des Netzwerks passt.

Um das systematische Ausprobieren des Agenten wie beim ResNet zu unterbinden, sollten sinnvolle Grenzen für die Stützstellen gefunden werden. Die aktuellen Werte der Stützstellen könnten auch über einen parallelen Pfad ins Netzwerk mit einfließen, um die Information über die Stützstellen für den Agenten verfügbar zu machen.

Um Entscheidungen bezüglich weiterer Trainings-Konfigurationen treffen zu können, ist im Anhang A.2 die Dateistruktur des angehängten Datenträgers erklärt. Die Dateien enthalten den Code für die Simulation und alle Trainingsprozesse inklusive Log-Dateien. Der Datenträger zur Arbeit befindet sich beim Erstgutachter und kann eingesehen werden.

Literaturverzeichnis

[1] : *Artificial Intelligence Magazine*. – URL https://becominghuman.ai/. – Zugriffsdatum: 05.04.2023

[2] : *Deque*. – URL https://docs.python.org/3/library/collections.html#collections.deque. – Zugriffsdatum: 14.04.2023

[3] : *Imagesc*. – URL https://pypi.org/project/imagesc/. – Zugriffsdatum: 14.04.2023

[4] : *Keras*. – URL https://keras.io/about/. – Zugriffsdatum: 14.04.2023

[5] : *Matplotlib*. – URL https://matplotlib.org/stable/index.html. – Zugriffsdatum: 14.04.2023

[6] : *Max Pooling*. – URL https://paperswithcode.com/method/max-pooling. – Zugriffsdatum: 27.04.2023

[7] : *Numpy*. – URL https://numpy.org/doc/stable/. – Zugriffsdatum: 14.04.2023

[8] : *Scipy*. – URL https://docs.scipy.org/doc/scipy/. – Zugriffsdatum: 14.04.2023

[9] : *Tensorboard*. – URL https://www.tensorflow.org/tensorboard. – Zugriffsdatum: 14.04.2023

[10] : *Tensorflow*. – URL https://www.tensorflow.org/learn. – Zugriffsdatum: 14.04.2023

[11] : *Tqdm*. – URL https://tqdm.github.io/. – Zugriffsdatum: 14.04.2023

[12] Alemi, Alex: *Improving Inception and Image Classification in TensorFlow*. – URL https://ai.googleblog.com/2016/08/improving-inception-and-image.html. – Zugriffsdatum: 27.04.2023

[13] Dahlkemper, Jörg: *Skript: Deep Learning in der Bildverarbeitung, HAW Hamburg.* 2022

[14] Degrave, J. ; Felici, F. ; Buchli, J. ; al. et: Magnetic control of tokamak plasmas through deep reinforcement learning. In: *Nature* 602 (2022), S. 414–434

[15] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning : das umfassende Handbuch : Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze.* mitp Verlag, 2018. – ISBN 9783958457010

[16] Gridin, Ivan: *Practical Deep Reinforcement Learning with Python.* BPB Online, 2022. – ISBN 9789355512062

[17] Jünemann, Klaus: *Skript: Maschinelles Lernen und Neuronale Netze, HAW Hamburg.* 2022

[18] Kanade, Vijay: *What Is Reinforcement Learning? Working, Algorithms, and Uses.* 2022. – URL https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-reinforcement-learning/. – Zugriffsdatum: 06.03.2023

[19] Lapan, Maxim: *Deep Reinforcement Learning : das umfassende Praxis-Handbuch.* mitp Verlag, 2020. – ISBN 9783747500385

[20] Liu, Tianyi ; Chen, Minshuo ; Zhou, Mo ; Du, Simon S. ; Zhou, Enlu ; Zhao, Tuo: Towards Understanding the Importance of Shortcut Connections in Residual Networks. In: Wallach, H. (Hrsg.) ; Larochelle, H. (Hrsg.) ; Beygelzimer, A. (Hrsg.) ; Fox, E. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 32, Curran Associates, Inc., 2019. – URL https://proceedings.neurips.cc/paper_files/paper/2019/file/7716d0fc31636914783865d34f6cdfd5-Paper.pdf

[21] Monmayrant, Antoine ; Weber, Sébastien J. ; Chatel, Béatrice: A newcomer's guide to ultrashort pulse shaping and characterization. In: *Journal of Physics* 43 (2010)

[22] Pickering, James D.: *Ultrafast Lasers and Optics for Experimentalists.* IOP Publishing Ltd, 2021. – ISBN 9780750336598

[23] Pozza, Dalla ; Buffoni, N. ; Martina, L. ; al. et: Quantum reinforcement learning: the maze problem. In: *Quantum Machine Intelligence* 11 (2022)

[24] St. John, Jason ; Herwig, Christian ; Kafkes, Diana ; Mitrevski, Jovan ; Pellico, William A. ; Perdue, Gabriel N. ; Quintero-Parra, Andres ; Schupbach, Brian A. ; Seiya, Kiyomi ; Tran, Nhan ; Schram, Malachi ; Duarte, Javier M. ; Huang, Yunzhi ; Keller, Rachael: Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster. In: *Phys. Rev. Accel. Beams* 24 (2021), Oct, S. 104601. – URL https://link.aps.org/doi/10.1103/PhysRevAccelBeams.24.104601

[25] Trebino, Rick: *Ultra Fast Optics Group*. – URL https://www.swampoptics.com/frog.html. – Zugriffsdatum: 06.03.2023

[26] Trebino, Rick ; DeLong, Kenneth W. ; Fittinghoff, David N. ; Sweetser, John N. ; Krumbügel, Marco A. ; Richman, Bruce A.: Measuring ultrashort laser pulses in the time-frequency domain using frequency-resolved optical gating. In: *Rev. Sci. Instrum* 68 (1997)

[27] Vogt, Patrick: *Skript: Spezielle Gebiete zum Software Engineering, FH Bielefeld*. 2018. – URL https://sgse18.github.io/book/ai-ml/nn/

[28] Wasylczyk, Piotr: *Ultrafast Optics*. – URL https://www.fuw.edu.pl/~zopt/photonics/UFO_PW/UFO_07_Pulse_shaping.pdf. – Zugriffsdatum: 11.05.2023

[29] Yamashita, Yoshiharu: *Convolutional Neural Networks - Basics*. 2017. – URL https://mlnotebook.github.io/post/CNN1/. – Zugriffsdatum: 17.04.2023

A Anhang

A.1 Ablaufdiagramm Trainingsprozess

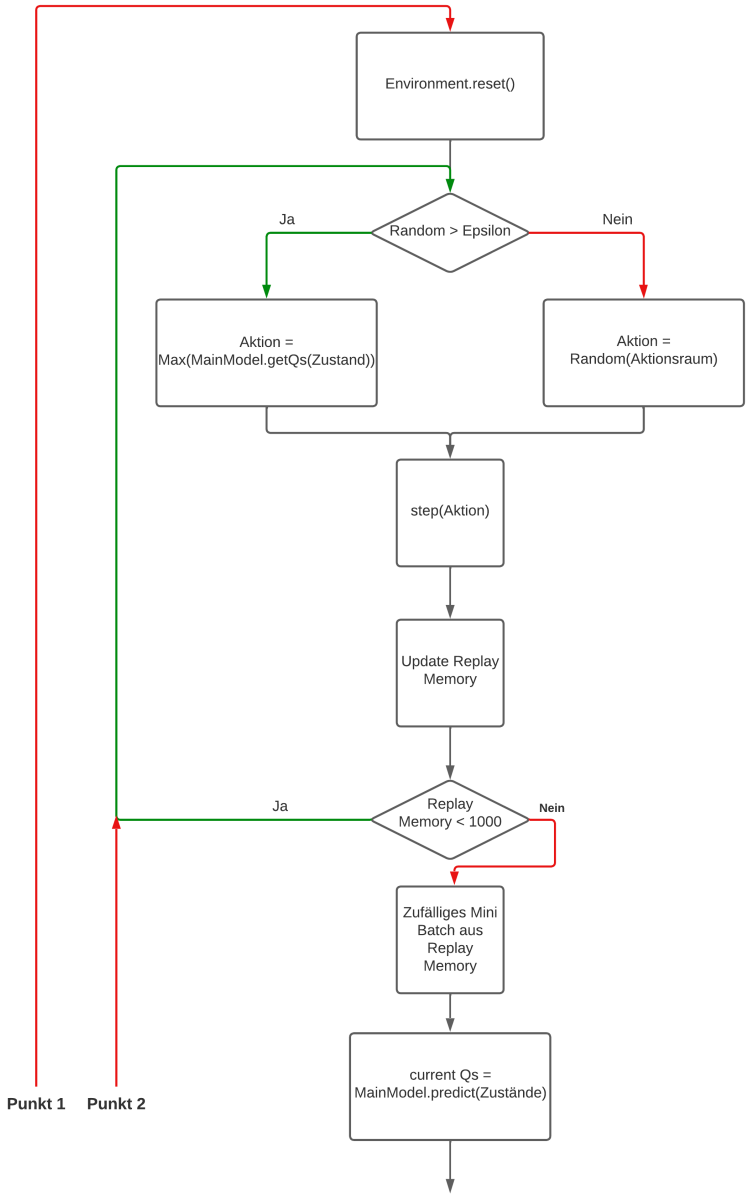

Abbildung A.1: Ablaufdiagramm des Trainingsprozesses Teil 1

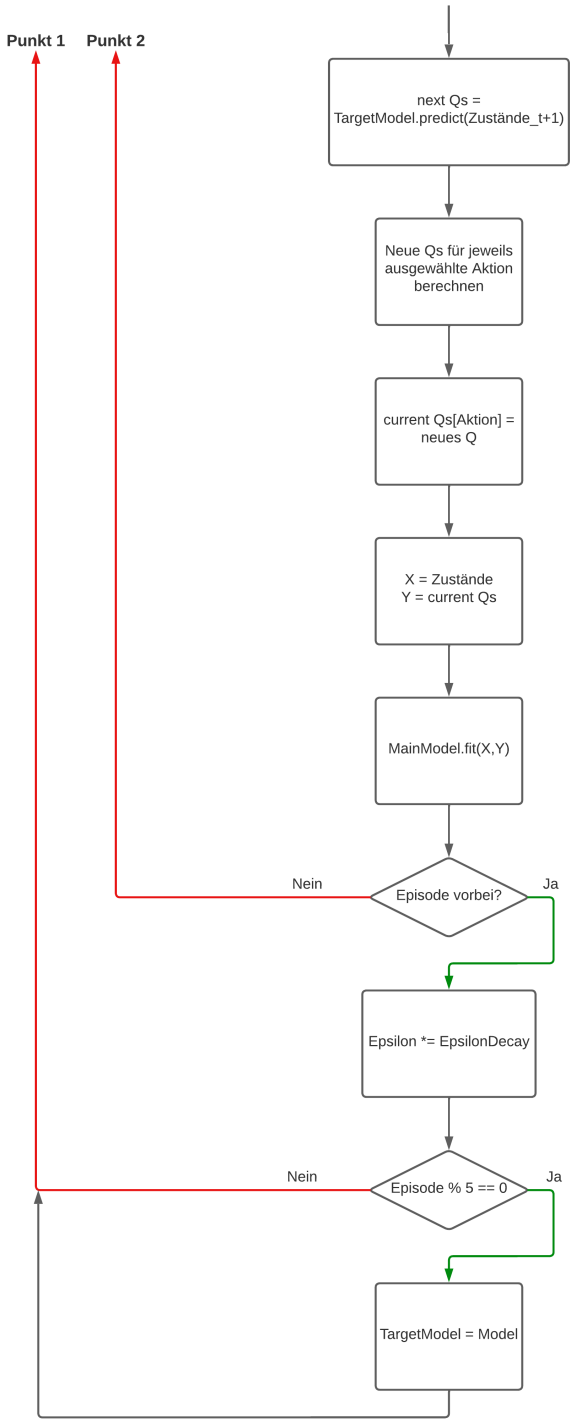

Abbildung A.2: Ablaufdiagramm des Trainingsprozesses Teil 2

A.2 Code Verzeichnis

Alle Programmcodes sowie gespeicherte Netzwerkmodelle und die Log-Dateien sind in der folgenden Ordnerstruktur abgelegt:

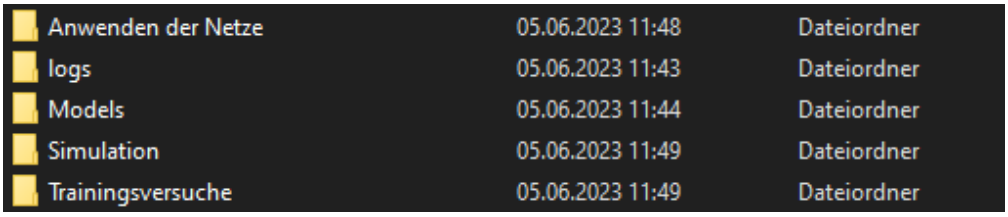

Abbildung A.3: Dateiverzeichnis

Der Ordner “Simulation“ enthält den Code für den Pulsformer, das Erstellen von Spektrogrammen und zum Ausrechnen der Pulsbreite. Diese Dateien müssen über den entsprechenden Dateipfad importiert werden.

Der Ordner “Trainingsversuche“ enthält alle bisher ausprobierten Konfigurationen. Hier können die Parameter-Konfigurationen entnommen und mit den Log-Dateien im Ornder “logs“ mithilfe von Tensorboard bewertet werden.

Der Ordner “Anwenden der Netze“ enthält den Code zum importieren gespeicherter Modelle aus dem Ordner “Models“. Diese Modelle werden für beliebige Pulsformen angewendet und über die *render*-Funktion visualisiert.

Glossar

Agent Der Agent ist der "Spieler" in diesem Szenario, der angelernt wird, möglichst gute Aktionen auszuführen.

Aktion Vom Agenten (Netzwerk) gewählte Aktion (engl. action) aus dem Aktionsraum, der alle für dieses Szenario möglichen Aktionen enthält.

Belohnung Nach jeder Aktion wird die Belohnung (engl. reward) berechnet und dem Agenten mitgeteilt. Sie beschreibt die Güte der Aktion.

Discount Der Disocunt-Faktor (γ) legt fest, wie stark zukünftige Belohnungen im Vergleich zu der unmittelbaren Belohnung gewichtet werden sollen. Dieser Faktor kann zwischen Null und Eins liegen und ist üblicherweise im Bereich >0,9, da das Ziel ist, nach einer Reihe von Aktionen die Belohnung zu maximieren.

Epsilon Das Epsilon (ϵ) lässt zufällige Aktionen zu, auch wenn laut Q-Wert eine andere Aktion ausgeführt werden soll. Dadurch werden größere Teile des Zustandsraums während des Trainings erkundet. Das Epsilon wird jede Episode durch die Multiplikation mit dem Hyperparameter Epsilon-decay verringert.

Overfitting Overfitting tritt auf, wenn ein Modell zu viele Parameter, im Verhältnis zu den verfügbaren Trainingsdaten hat. Das Modell passt sich dann zu sehr an die Trainingsdaten an und kann nicht gut verallgemeinern. Es ist wie das Auswendiglernen von Antworten auf Fragen für eine Prüfung, ohne zu verstehen, warum die Antworten richtig sind.

Q-Wert Die Q-Werte (engl. Q-Value) repräsentieren jeweils die Güte einer Aktion. Die Aktion, zugehörig zum höchsten Q-Wert, wird ausgeführt.

Umgebung Die Umgebung (engl. environment) ist die Welt in der der Agent agiert. Sie enthält die Regeln und gibt Zustände und Belohnungen zurück.

Zustand Der Zustand (engl. state), in der sich die Umgebung befindet. Z.B. Position von Spielfiguren.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original