HAW
HAMBURG

MASTERTHESIS
Leon Fromm

Anwendung von Reinforcement Learning zur Simulation und Optimierung der Pulsformung in Kurzzeitlasern

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

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

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Leon Fromm

Anwendung von Reinforcement Learning zur Simulation und Optimierung der Pulsformung in Kurzzeitlasern

Masterthesis eingereicht im Rahmen der Masterprüfung
im Studiengang Informations- und Elektrotechnik
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

Abgegeben am 28. Oktober 2025

Leon Fromm

Thema der Masterthesis

Anwendung von Reinforcement Learning zur Simulation und Optimierung der Pulsformung in Kurzzeitlasern

Stichworte

Reinforcement Learning, Pulsformung, Kurzzeitlaser, Soft-Actor-Critic, Maschinelles Lernen, FROG

Kurzzusammenfassung

Diese Arbeit untersucht die Eignung von Reinforcement Learning zur Optimierung der Pulsformung in Kurzzeitlasern. Das zentrale Ziel ist die Minimierung der Pulsbreite durch gezielte Phasenmodulation. Es wird einerseits die grundsätzliche Eignung des RL-Ansatzes zur Pulsminimierung untersucht und andererseits analysiert, wie die Auswahl der Eingangsdaten den Lernerfolg beeinflusst, indem das Training mit Zeit-Intensitätsprofilen mit dem Training auf Basis von FROG-Spektrogrammen verglichen wird. Als Methode wird ein Soft-Actor-Critic Agent verwendet, welcher in einer Simulationsumgebung lernt. Dieser passt dabei eine Phasenmaske so an, dass der Laserpuls komprimiert wird. Die relative Verkürzung der Pulsbreite dient dabei als Belohnungssignal, um das Verhalten des Agenten zu steuern. Die Ergebnisse zeigen, dass der Agent in der Lage ist, die Pulsbreite für eine Vielzahl zufälliger Eingangspulse signifikant zu reduzieren. Mit einer Phasenmaske von 25 Stützstellen wurden die besten Ergebnisse erzielt. Dies führte zu einer mittleren Pulsverkürzung von über 57 Prozent. Konfigurationen mit 100 Stützstellen erwiesen sich hingegen als instabil und führten zu einer Pulsverbreiterung. Die Untersuchung zeigt, dass Reinforcement Learning ein vielversprechender Ansatz zur automatisierten Optimierung der Pulsformung ist und eine robuste Alternative zu klassischen Verfahren darstellt.

Leon Fromm

Title of the paper

Application of Reinforcement Learning for the Simulation and Optimization of Pulse Shaping in Ultrashort Lasers

Keywords

Reinforcement Learning, Pulse Shaping, Ultrashort Lasers, Soft-Actor-Critic, Machine Learning, FROG

Abstract

This work investigates the suitability of Reinforcement Learning for optimizing pulse shaping in short-pulse lasers. The central objective is the minimization of the pulse width through targeted phase modulation. On the one hand, the fundamental suitability of the RL approach for pulse minimization is examined, and on the other hand, it analyzes how the selection of input data influences learning success by comparing training with time-intensity profiles to training based on FROG spectrograms. A Soft-Actor-Critic agent is used as the method, which learns in a simulation environment. This agent adjusts a phase mask in such a way that the laser pulse is compressed. The relative shortening of the pulse width serves as the reward signal to control the agent's behavior. The results show that the agent is capable of significantly reducing the pulse width for a variety of random input pulses. The best results were achieved using a phase mask with 25 control points. This led to an average pulse shortening of over 57 percent. Configurations with 100 control points proved to be unstable and led to pulse broadening. The study shows that Reinforcement Learning is a promising approach for the automated optimization of pulse shaping and represents a robust alternative to classical methods.

Inhaltsverzeichnis

Inhaltsverzeichnis V

Abbildungsverzeichnis VIII

Tabellenverzeichnis X

Formelverzeichnis XI

Abkürzungsverzeichnis XII

1 Einleitung 1

1.1 Problemstellung und Zielsetzung 1

1.2 Vorgehensweise und Aufbau der Arbeit 1

2 Grundlagen 3

2.1 Fourier-Transformation 3

2.2 Light Amplification by Stimulated Emission of Radiation 4

2.2.1 Funktionsweise 4

2.2.2 Eigenschaften des Laserlichts 5

2.2.3 Fourier-Bandbreitenprodukt und Pulslimit 6

2.2.4 Dispersion und Chirp 6

2.2.5 Frequency Resolved Optical Gating 8

2.2.6 Kompression eines ultraschnellen Lasers 10

2.3 Neuronale Netze 11

2.3.1 Grundaufbau 11

2.3.2 Neuronen 13

2.3.3 Aktivierungsfunktionen 14

2.3.4 Arten des maschinellen Lernens 15

2.4 Reinforcement Learning 16

2.4.1 Markov-Entscheidungsprozess 18

2.4.2 Value und Policy 19

2.4.3 Q-Learning 19

2.4.4 Deep-Q-Networks 22

2.4.5 Soft-Actor-Critic..23

2.4.6 Vergleich Reinforcement-Algorithmen 24

2.5 Convolutional Neural Networks 26

2.6 Root Mean Square 28

3 Simulation 31

3.1 Erzeugen des Referenzpulses 31

3.2 Erzeugen des Referenz-FROG 33

4 Umsetzung von Reinforcement Learning 36

4.1 Aufbau...36

4.1.1 Ablauf Intensität 37

4.1.2 Ablauf FROG 40

4.2 Schichten 41

4.3 Belohnungsfunktion...42

4.4 Hyperparameter 44

5 Trainingsablauf...46

6 Training und Auswertung 48

6.1 Intensität 48

6.1.1 Fester Eingangspuls 48

6.1.2 Pulswechsel 49

6.1.3 Hyperparameter Tuning 51

6.1.4 Belohnungsfunktion 55

6.1.5 Anpassung der Netzgröße 57

6.1.6 Langlauf 58

6.2 FROG...60

6.3 Weiterführende Analysen..61

6.3.1 Anwendung anderer RL-Algorithmen..61

6.3.2 Vergleich des Pulsshapers bei 10, 25, 50 und 100 Stützstellen 62
6.3.3 Verkleinerung der einstellbaren Frequenz für den Pulsshaper 63
6.3.4 Überprüfung der enthaltenen Energie des Pulses 64
6.3.5 Test ohne Dispersion zweiter Ordnung oder dritter Ordnung...............64
6.3.6 Überprüfung des Fourier-Bandbreitenprodukt 66
7 Fazit und Ausblick 68
Literaturverzeichnis 70
Anhang 73
A 1 Code Referenzpuls 73
A 2 Code Referenz-FROG 77
Eidesstattliche Erklärung 81

Abbildungsverzeichnis

Abbildung 2.1 Schematische Darstellung eines Vier-Niveau-Lasers 4

Abbildung 2.2 Dispersion eines Pulses 8

Abbildung 2.3 Messaufbau Autokorrelation 9

Abbildung 2.4 FROG-Spektrogramm 9

Abbildung 2.5 Darstellung eines Gitter- und Prismenkompressor 10

Abbildung 2.6 Darstellung Chirp-Spiegel 11

Abbildung 2.7 Schematisches Perzeptron 12

Abbildung 2.8 Schematisches Neuronales Netz 13

Abbildung 2.9 Schematisches Neuronen 13

Abbildung 2.10 Unterschiedliche Aktivierungsfunktionen 15

Abbildung 2.11 Arten des maschinellen Lernens 15

Abbildung 2.12 Reinforcement Learning 17

Abbildung 2.13 Ablauf Q-Learning 20

Abbildung 2.14 Beispiel Gitter 2x2 21

Abbildung 2.15 Ablaufplan DQN 22

Abbildung 2.16 Beispielhafter Aufbau CNN 27

Abbildung 2.17 Filterkern 27

Abbildung 2.18 Pooling Schicht 28

Abbildung 3.1 Beispielhafter Puls 33

Abbildung 3.2 Beispielhafter FROG 35

Abbildung 4.1 Pulsshaper Ablaufplan 37

Abbildung 4.2 Flussdiagramm Training mittels Intensität 39

Abbildung 4.3 Flussdiagramm Training mittels FROG 40

Abbildung 4.4 Kleines Netz 42

Abbildung 4.5 Belohnungsfunktion eins 43

Abbildung 4.6 Belohnungsfunktion zwei 43

Abbildung 4.7 Belohnungsfunktion drei 44

Abbildung 6.1 Entwicklung der Ratio über 100.000 Trainingsschritte, fester Eingangspuls 48

Abbildung 6.2 Entwicklung der Ratio über 100.000 Trainingsschritte, unterschiedlicher Puls 50

Abbildung 6.3 Entwicklung der Ratio über 100.000 Trainingsschritte, gamma = 0,7 .52

Abbildung 6.4 Gekürzter Puls, Langlauf, 25 und 50 Stützstellen 59

Abbildung 6.5 Gekürzter FROG, 25 Stützstellen 61

Abbildung 6.6 Pulsshaper für 10, 25, 50 und 100 Stützstellen 62

Abbildung 6.7 Vergleich des Pulsshapers 63

Abbildung 6.8 Vergleich der Energie des Ein- und Ausgangspuls....................................64

Abbildung 6.9 Gekürzter Puls, ohne GDD/TOD, 25 Stützstellen 66

Tabellenverzeichnis

Tabelle 1 Q-Tabelle für 2x2 Gitter 21

Tabelle 2 Vergleich Reinforcement-Algorithmen 25

Tabelle 3 Auswertung von 100 Testpulsen, unterschiedlicher Puls....................51

Tabelle 4 Auswertung von 100 Testpulsen, gamma = 0,7....................53

Tabelle 5 Auswertung von 100 Testpulsen, buffer_size = 50.000 54

Tabelle 6 Auswertung von 100 Testpulsen, entropy = -0,5*S....................54

Tabelle 7 Auswertung von 100 Testpulsen, Episodenlänge = 40 55

Tabelle 8 Auswertung von 100 Testpulsen, zweite Belohnungsfunktion 56

Tabelle 9 Auswertung von 100 Testpulsen, dritte Belohnungsfunktion 57

Tabelle 10 Auswertung von 100 Testpulsen, unterschiedliche Netzgrößen 58

Tabelle 11 Auswertung von 100 Testpulsen, Langlauf 59

Tabelle 12 Auswertung von 100 Testpulsen, FROG 60

Tabelle 13 Vergleich SAC, PPO, TB3 62

Tabelle 14 Auswertung von 100 Testpulsen, ohne GDD/TOD 65

Tabelle 15 Fourier-Bandbreitenprodukt....................67

Formelverzeichnis

Formel 2.1 Diskrete Fouriertransformation 3

Formel 2.2 Energie-Zeit-Unschärferelation 5

Formel 2.3 Fourier-Bandbreitenprodukt 6

Formel 2.4 Brechungsindex eines Materials 6

Formel 2.5 Geschwindigkeit einer Frequenzkomponente 6

Formel 2.6 Autokorrelation 8

Formel 2.7 Gewichtete Summe der Neuronen 14

Formel 2.8 Markov Entscheidungsprozess 18

Formel 2.9 Belohnungsfunktion 18

Formel 2.10 Policy Q-Learning 19

Formel 2.11 Bellman-Gleichung 20

Formel 2.12 Zielwert Target-Netzwerk 23

Formel 2.13 Faltungsschicht 27

Formel 2.14 Diskrete Definition RMS 29

Formel 2.15 Verallgemeinertes RMS 29

Formel 2.16 RMS-Breite 29

Abkürzungsverzeichnis

FT	Fourier-Transformation
DFT	diskrete Fouriertransformation
FFT	Fast-Fourier-Transformation
Laser	Light Amplification by Stimulated Emission of Radiation
TBP	Fourier-Bandbreitenprodukt
GDD	Gruppengeschwindigkeitsdispersion
TOD	Dispersion dritter Ordnung
FROG	Frequency Resolved Optical Gating
ReLU	Rectified Linear Unit
MEP	Markov-Entscheidungsprozess
DQN	Deep Q-Networks
SAC	Soft-Actor-Critic
PPO	Proximal Policy Optimization
TD3	Twin Delayed Deep Deterministic Policy Gradient
CNN	Convolutional Neural Networks
RMS	Root Mean Square

1 Einleitung

1.1 Problemstellung und Zielsetzung

Die Steuerung und Optimierung von Kurzzeitlasern ist ein bedeutendes Feld in der modernen Physik und Technik. Eine zentrale Herausforderung dabei ist die Formung von Laserpulsen, um eine minimale Pulsbreite zu erreichen. Kürzere Pulse ermöglichen präzisere Anwendungen in Bereichen wie der Materialbearbeitung, der medizinischen Bildgebung oder der Grundlagenforschung. Traditionelle Methoden zur Pulsformung stoßen hierbei oft an ihre Grenzen, insbesondere wenn es um die Korrektur von Dispersionen höherer Ordnung geht.

In den letzten Jahren haben sich maschinelle Lernverfahren, insbesondere das Reinforcement Learning als leistungsfähige Werkzeuge zur Lösung komplexer Optimierungsprobleme etabliert. Im Gegensatz zu klassischen Algorithmen, lernt ein Reinforcement-Learning-Agent durch Interaktion mit einer Umgebung und die Maximierung einer Belohnung. Dieser Ansatz verspricht, auch in hochdimensionalen und nichtlinearen Systemen wie der Pulsformung effiziente Strategien zu finden.

Die zentrale Forschungsfrage dieser Arbeit lautet daher: „Inwiefern eignet sich Reinforcement Learning zur Minimierung der Pulsbreite durch Pulsformung in Kurzzeitlasern und welche Rolle spielt die Auswahl der Eingangsdaten für den Lernerfolg?“

Diese Arbeit untersucht, wie Reinforcement Learning die Pulsformung in Kurzzeitlasern zur Minimierung der Pulsbreite optimieren kann. Ein besonderer Fokus liegt dabei auf dem Einfluss der Eingangsdaten. Dabei wird der Lernerfolg bei der Verwendung von Zeit-Intensitäts-Profilen mit jenem bei der Nutzung von FROG-Spektrogrammen verglichen.

1.2 Vorgehensweise und Aufbau der Arbeit

Um diese Forschungsfrage zu beantworten, wird zunächst eine Simulationsumgebung für Laserpulse und deren Kompression mittels eines Puls-Shapers entwickelt. Auf dieser Grundlage wird ein Reinforcement-Learning-Agent, basierend auf dem Soft-Actor-Critic Algorithmus, implementiert und trainiert. Die Optimierung der Pulsbreite wird dabei als Ziel in der Belohnungsfunktion des Agenten verankert.

Die Arbeit ist wie folgt aufgebaut. Kapitel 2 legt die theoretischen Grundlagen in den Bereichen der Laserphysik, der Fourier-Transformation und verschiedener maschineller Lernverfahren dar. Ein besonderer Fokus liegt hierbei auf dem Reinforcement Learning und den spezifischen Algorithmen wie Q-Learning, Deep-Q-Networks und dem Soft-Actor-Critic. Kapitel 3 beschreibt die erstellte Simulationsumgebung zur Erzeugung von Laserpulsen und FROG-Spuren. In Kapitel 4 wird die konkrete Umsetzung des Reinforcement-Learning-Ansatzes für die Pulsformung erläutert, einschließlich des Aufbaus des Agenten, der Definition der Belohnungsfunktionen und der gewählten Hyperparameter. Kapitel 5 dokumentiert den Trainingsablauf sowie die methodische Vorgehensweise der Auswertung. Kapitel 6 präsentiert die systematische Anwendung und detaillierte Auswertung der trainierten Modelle. Dabei werden verschiedene Konfigurationen, wie die Anzahl der Stützstellen der Phasenmaske und die Netzgröße, untersucht und verglichen. Den Abschluss bildet Kapitel 7 mit einem Fazit und einem Ausblick auf mögliche weiterführende Arbeiten.

2 Grundlagen

2.1 Fourier-Transformation

Fourierreihen werden verwendet, um periodische Funktionen als Summe von Sinus- und Kosinusgliedern unterschiedlicher Frequenz darzustellen. Die Fourier-Transformation (FT) ist eine Erweiterung der Fourierreihen und ermöglicht die Darstellung von nicht periodischen Funktionen.

In der Zeitdomäne beschreibt ein Signal, wie sich eine Größe (z.B. Laserpuls), im zeitlichen Verlauf verändert. Die Frequenzdomäne hingegen gibt an, aus welchen Frequenzkomponenten das Signal besteht und mit welcher Amplitude und Phase diese vertreten sind. Die FT stellt das mathematische Verfahren dar, mit dem ein zeitabhängiges Signal in seine Frequenzbestandteile zerlegt werden kann.

Wichtig in dieser Arbeit ist die diskrete Fouriertransformation (DFT). Die DFT arbeitet mit endlichen und diskreten Daten, wie sie beispielsweise bei digitalisierten Signalen oder in numerischen Simulationen auftreten. Sie zerlegt eine endliche Folge von Messwerten in eine Summe von Sinus- und Kosinusfunktionen unterschiedlicher Frequenzen. Formal wird eine Liste von N Einträgen $x_0, \dots, x_{N-1}$ durch die DFT in eine neue Liste überführt, wobei jeder Wert X_k die Amplitude und Phase der k-ten Frequenzkomponente im Signal beschreibt.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kN} \qquad (2.1)$$

Ein effizienter Algorithmus zur Berechnung der DFT ist die Fast-Fourier-Transformation (FFT). Einer der bekanntesten Ansätze ist der Cooley-Tukey-Algorithmus. Dabei wird die Ausgangssequenz rekursiv in Teilfolgen mit geraden und ungeraden Indizes zerlegt, bis die DFTs auf sehr kleine Gruppen, meist mit zwei oder vier Elementen, angewendet werden können. Anschließend werden die berechneten Teilergebnisse mithilfe sogenannter Butterfly-Operationen zusammengeführt. Dieser rekursive Ansatz verringert den Rechenaufwand der DFT erheblich und ermöglicht damit die effiziente Analyse auch großer Datensätze.

2.2 Light Amplification by Stimulated Emission of Radiation

Light Amplification by Stimulated Emission of Radiation (Laser) (deutsch: Lichtverstärkung durch stimulierte Emission von Strahlung) bezeichnet Licht, das sich durch eine Reihe physikalischer Eigenschaften auszeichnet, die es von herkömmlichen Lichtquellen unterscheiden. Besonders auffällig ist die hohe räumliche Kohärenz. Das emittierte Licht ist stark gebündelt, weist eine minimale Divergenz auf und bleibt dadurch über weite Distanzen nahezu parallel. Ein weiteres zentrales Merkmal ist die hohe zeitliche Kohärenz. Laser erzeugen Licht mit einer definierten Wellenlänge und einer sehr schmalen spektralen Bandbreite, wodurch sie nahezu monochromatisch sind. Hinzu kommt die hohe Intensität des Laserlichts. Im Vergleich zu konventionellen Lichtquellen können mit Lasern sehr hohe Leistungsdichten erzielt werden. Außerdem ist Laserlicht stark gerichtet. Die Strahlung wird nahezu vollständig in eine Richtung ausgesandt. [1, pp. 1-3]

2.2.1 Funktionsweise

Die grundlegende Funktionsweise eines Lasers hängt vom Lasertyp ab. In diesem Beispiel wird ein Vier-Niveau-Laser betrachtet. Die Abbildung 2.1 zeigt den Zusammenhang der jeweiligen Stufen, auf die sich im nachfolgenden Text bezogen wird.

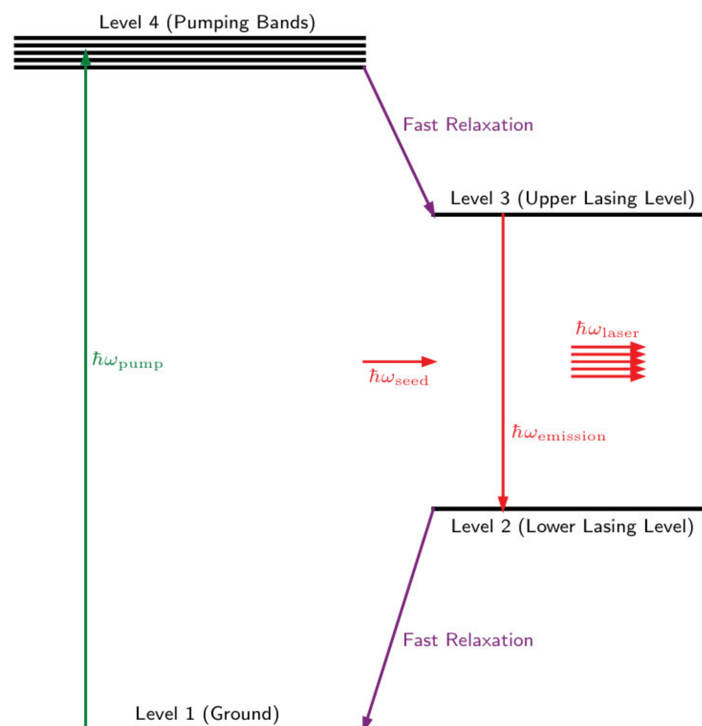


Abbildung 2.1 Schematische Darstellung eines Vier-Niveau-Lasers [2, p. 2.2]

In einem Lasersystem existieren vier Energieniveaus, wobei das erste Energieniveau das Grundniveau darstellt. Photonen werden durch beispielsweise optische Anregung vom Grundniveau auf das vierte Energieniveau gehoben. Dieser Prozess wird als optisches Pumpen bezeichnet. Vom vierten Energieniveau erfolgt ein schneller, automatischer Übergang auf das dritte Energieniveau, in dem eine möglichst große Anzahl von Teilchen vorliegen muss, während das zweite Energieniveau möglichst wenig besetzt sein sollte. Diese Verteilung wird als Besetzungsinversion bezeichnet und ist Voraussetzung für die Laseremission. Die eigentliche Laserstrahlung entsteht durch stimulierte Emission beim Übergang vom dritten auf das zweite Energieniveau. Der Energieunterschied zwischen diesen Niveaus bestimmt die Frequenz und damit die Farbe des emittierten Lichts. Die Anregung kann dabei entweder durch spontane Emission oder durch einen Seed-Laser mit der Photonenenergie $\omega\hbar_seed$ erfolgen. Der schnelle Übergang vom zweiten auf das erste Energieniveau sorgt für eine geringe Population im zweiten Energieniveau und unterstützt somit die Besetzungsinversion. [2, pp. 2-3]

2.2.2 Eigenschaften des Laserlichts

Bei Lasern wird grundsätzlich zwischen zwei Betriebsarten unterschieden, Dauerstrichlaser und Pulslaser. Dauerstrichlaser, die ein kontinuierliches Licht erzeugen, sind aus dem Alltag bekannt, weisen jedoch eine vergleichsweise geringe Ausgangsleistung auf. In Forschung und Industrie finden hingegen häufig Pulslaser Anwendung. Bei dieser Betriebsart erfolgt die Aussendung des Lichts in Form einzelner, zeitlich begrenzter Laserpulse. In dieser Arbeit werden Kurzzeitlasern genauer betrachtet. Diese sind definitionsgemäß stets Pulslaser. [3]

$$\Delta E \cdot \Delta t \geq \frac{\hbar}{2} \tag{2.2}$$

Die Energie-Zeit-Unschärferelation (Gleichung 2.2) besagt, dass ein Zustand mit sehr genau definierter Energie, also einem kleinen ΔE, über eine lange Zeit, also ein großes Δt, existieren muss und umgekehrt. Für einen Pulslaser, dessen Pulse nur für eine sehr kurze Zeit existieren, bedeutet dies, dass Δt klein ist. Daraus folgt, dass das Licht nicht mit nur einer einzelnen Wellenlänge ausgesendet wird, sondern eine entsprechend große spektrale Bandbreite aufweist. [2, p. 2.1]

2.2.3 Fourier-Bandbreitenprodukt und Pulslimit

Ein fundamentales Prinzip der Fouriertransformation besagt, dass ein Signal nicht gleichzeitig beliebig kurz in der Zeitdomäne und beliebig schmal in der Frequenzdomäne sein kann. Dieser Zusammenhang wird durch das Fourier-Bandbreitenprodukt (TBP) beschrieben. Es setzt die Dauer eines Pulses in ein direktes Verhältnis zu seiner spektralen Bandbreite. Das Produkt dieser beiden Größen kann einen minimalen Wert nicht unterschreiten:

$$\Delta t \cdot \Delta v \geq K \tag{2.3}$$

Ein Puls, der exakt dieses Minimum erreicht, wird als fourierlimitiert bezeichnet. Er besitzt die kürzt mögliche Dauer, die für sein gegebenes Spektrum physikalisch erreichbar ist. [4]

Werden beide Größen, wie in dieser Arbeit (siehe Abschnitt 2.6), als RMS-Breite definiert, gilt für die wichtige Referenzform eines Gauß-Pulses, $K = 0{,}5$.

Angenommen, ein simulierter Eingangspuls besitzt eine spektrale RMS-Bandbreite von $\Delta v = 4THz$. Die minimal mögliche, transformlimitierte RMS-Pulsdauer wäre dann:

$$\Delta t = \frac{K}{\Delta v} = \frac{0{,}5}{4 \times 10^{12} Hz} = 125 \times 10^{-15} s$$

Dies entspricht einer minimalen RMS-Pulsbreite von 125 fs.

2.2.4 Dispersion und Chirp

Im Vakuum ist die Lichtgeschwindigkeit für alle Wellenlängen gleich. In allen anderen Ausbreitungsmedien hängt die Ausbreitungsgeschwindigkeit des Lichts jedoch von der Wellenlänge ab. Diese Abhängigkeit wird als Dispersion bezeichnet.

$$n(\omega) = \frac{c_0}{c(\omega)} \tag{2.4}$$

$$c(\omega) = \frac{c_0}{n(\omega)} \tag{2.5}$$

Der Brechungsindex n eines Materials ist gegeben durch die Gleichung 2.4. c_0 beschreibt die Lichtgeschwindigkeit im Vakuum, c die Lichtgeschwindigkeit im Material. Durch Umstellung der Gleichung 2.4 nach $c(\omega)$, ergibt sich die Geschwindigkeit einer bestimmten Frequenzkomponente eines Pulses (Gleichung 2.5). Sie beschreibt nicht

die Gesamtgeschwindigkeit des Pulses, da die unterschiedlichen Frequenzen gemäß dieser Gleichung das Medium mit unterschiedlichen Geschwindigkeiten durchqueren.

Bei normaler Dispersion nimmt die Ausbreitungsgeschwindigkeit mit zunehmender Wellenlänge zu, während der Brechungsindex entsprechend abnimmt. Dies gilt beispielsweise für sichtbares Licht in Wasser oder Glas. Rotes Licht wird daher weniger stark abgelenkt als kurzwelliges blaues Licht. Wenn beispielsweise weißes Licht durch ein Prisma fällt, wird es in seine verschiedenen Wellenlängen aufgespalten und fächert sich entsprechend seiner Wellenlänge auf. Der entgegengesetzte Fall, bei dem höherfrequentes Licht schneller ist, wird abnormale Dispersion genannt. In beiden Fällen bewirkt die Dispersion jedoch, dass sich ein Puls beim Durchgang eines Mediums verbreitert. [5]

Bei nicht ultrakurzen Pulsen kann die Dispersion vernachlässigt werden, da die spektrale Bandbreite dieser Pulse sehr gering ist. Wie in Abschnitt 2.2.2 erläutert, weisen ultrakurze Pulse hingegen eine breite Bandbreite auf. Es ist also nicht die kurze Pulsdauer an sich, sondern die damit verbundene große Bandbreite, die dazu führt, dass ultrakurze Pulse besonders stark von Dispersion betroffen sind. Bei normaler Dispersion treffen die roten Frequenzkomponenten eines Pulses vor den blauen ein, wie in Abbildung 2.2 dargestellt. An diesem Punkt spricht man von einem sogenannten Chirp. Bei positiv gechirpten Pulsen nimmt die Frequenz während des Pulsverlaufs zu, da die niedrigfrequenten Anteile das Medium schneller durchqueren und somit früher eintreffen.

Um diese Effekte quantitativ zu beschreiben, wird die spektrale Phase eines Pulses als Taylor-Reihe um die Mittenfrequenz entwickelt. Der Term zweiter Ordnung dieser Entwicklung, die Gruppengeschwindigkeitsdispersion (GDD), ist die primäre Ursache für die Pulsverbreiterung. Der Term dritter Ordnung, die Dispersion dritter Ordnung (TOD), führt zu einer asymmetrischen Verzerrung und wird vor allem bei sehr kurzen Pulsen relevant. [2]

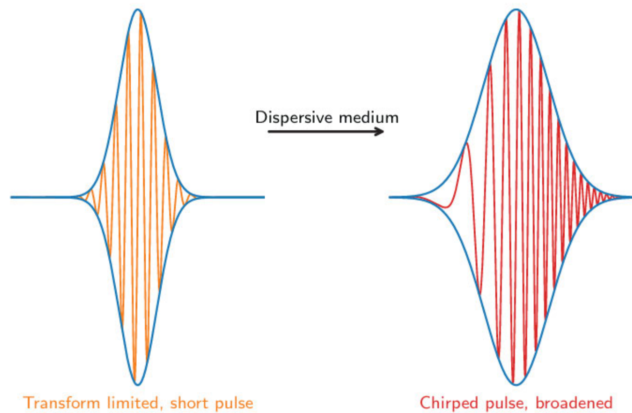


Abbildung 2.2 Dispersion eines Pulses [2, p. 3.4]

2.2.5 Frequency Resolved Optical Gating

Um zu bestimmen, wie kurz ein ultrakurzer Puls ist, ist es notwendig, ihn zu messen. Dabei spielen sowohl die Pulsdauer als auch die Ankunftszeiten der verschiedenen Frequenzkomponenten eine Rolle. Die extrem kurze Dauer der Pulse stellt hierbei eine messtechnische Herausforderung dar. Ein etabliertes Verfahren zur Charakterisierung ist die Autokorrelation. Hierbei wird der Puls in zwei Hälften geteilt, wobei eine Hälfte zeitlich verzögert wird. Anschließend werden beide Teile wieder überlagert. Je nach zeitlicher Überlappung entsteht ein Signal, dessen Intensität davon abhängt, wie ähnlich sich beide Pulsanteile sind. Aus diesem Signal lassen sich Informationen über die Pulsdauer gewinnen. Die Abbildung 2.3 zeigt einen schematischen Messaufbau.

Mathematisch lässt sich die Autokorrelation wie folgt ausdrücken:

$$A(\tau) = \int_{-\infty}^{\infty} f(t) \cdot f(t-\tau) dt \qquad (2.6)$$

Dabei bezeichnet $f(t)$ das Signal, τ die Zeitverschiebung und $A(\tau)$ den Autokorrelationswert für diesen Versatz. Für $\tau = 0$ überlagert sich das Signal vollständig mit sich selbst, was zu einem maximalen Wert führt. Für große τ überlappen sich die Signale nicht mehr, sodass der Wert entsprechend klein wird. Bei Zwischenwerten von τ ergibt sich eine teilweise Überlappung, die mittlere Korrelationswerte hervorruft.

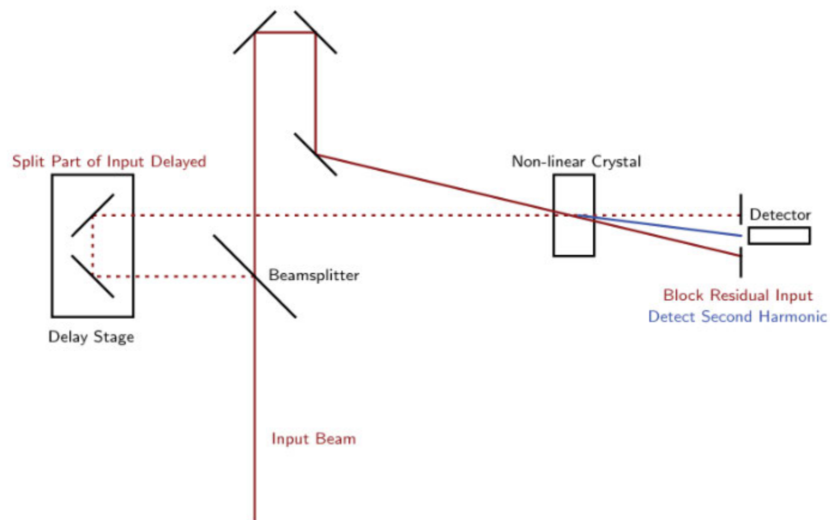


Abbildung 2.3 Messaufbau Autokorrelation [2, pp. 6-4]

Die klassische Autokorrelation liefert jedoch nur eingeschränkte Informationen. Zwar kann die zeitliche Dauer des Laserpulses bestimmt werden, nicht jedoch das vollständige elektrische Feld. Insbesondere die spektrale Phase bleibt unbekannt.

Das Frequency Resolved Optical Gating (FROG) (deutsch: frequenzaufgelöstes optisches Gating) erweitert dieses Verfahren, indem die Autokorrelation spektral aufgelöst wird. Anstelle einer einfachen Fotodiode wird dabei ein Spektrometer eingesetzt, sodass nicht nur die Intensität, sondern auch die spektrale Zusammensetzung des Pulses über die Zeit erfasst wird. Damit lassen sich sowohl die spektrale Amplitude als auch die Phase bestimmen. Dies ermöglicht die vollständige Rekonstruktion des elektrischen Feldes eines Laserpulses (siehe Abbildung 2.3).

Ein Spektrogramm ist eine Darstellungsform, die zeigt, wie sich die Frequenz eines Pulses im Verlauf der Zeit verändert. Es wird üblicherweise in Form einer Heatmap dargestellt, bei der die Intensität der verschiedenen Frequenzkomponenten als Funktion der Zeit sichtbar wird. Die Abbildung 2.4 zeigt beispielhaft solche Spektrogramme.

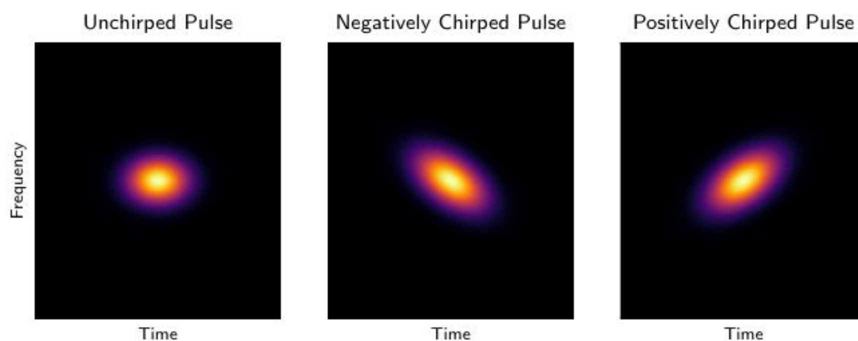


Abbildung 2.4 FROG-Spektrogramm [2, pp. 6-7]

2.2.6 Kompression eines ultraschnellen Lasers

Zur Kompression ultrakurzer Laserpulse können Prismen eingesetzt werden. Sie verkürzen die Pulsdauer, indem durch Dispersion und den wellenlängenabhängigen Brechungsindex bestimmte Frequenzanteile des Lichts unterschiedlich stark verzögert werden (siehe Abbildung 2.5). Durch die dabei entstehenden Wegdifferenzen können die verschiedenen Wellenlängen zeitlich wieder angenähert und der Puls somit komprimiert werden.

Neben Prismen werden auch Gitterkompressoren verwendet, die anstelle der Brechung die Beugung des Lichts ausnutzen, im Prinzip jedoch nach einem ähnlichen Funktionsprinzip arbeiten. Ein Nachteil von Gitter- und Prismenkompressoren ist, dass sie in der praktischen Anwendung vergleichsweise aufwendig und schwierig einzustellen sind. [2, pp. 5-8]

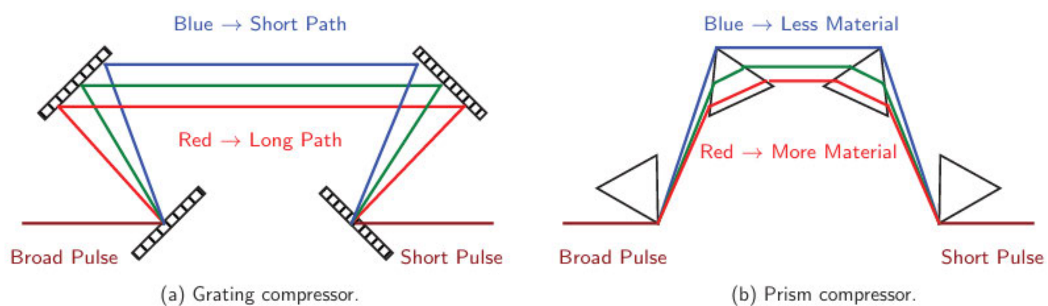


Abbildung 2.5 Darstellung eines Gitter- und Prismenkompressor [2, pp. 5-8]

Eine weitere Möglichkeit zur Pulsverkürzung stellen sogenannte Chirp-Spiegel dar (siehe Abbildung 2.6). Diese funktionieren, indem unterschiedliche Wellenlängen verschieden tief in die speziell beschichteten Spiegel eindringen. Ähnlich wie bei den zuvor beschriebenen Methoden kommt es dadurch zu einer zeitlichen Verzögerung bestimmter Wellenlängen gegenüber anderen. Auf diese Weise kann die Dispersion gezielt korrigiert und die Pulsdauer entsprechend komprimiert werden. [2, pp. 5-8]

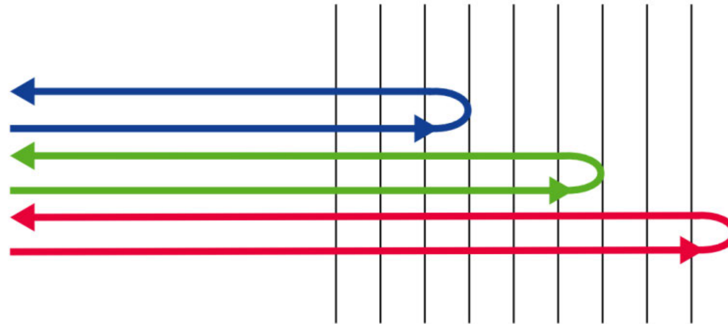


Abbildung 2.6 Darstellung Chirp-Spiegel [6]

Das Problem der zuvor beschriebenen Methoden besteht darin, dass sie Dispersion nur bis zur zweiten Ordnung kompensieren können. Für die Korrektur höherer Ordnungen, insbesondere der Dispersion dritter Ordnung, ist der Einsatz eines Pulse Shapers erforderlich. Mit diesem kann die Phase jeder einzelnen Wellenlänge gezielt und unabhängig voneinander gesteuert werden, wodurch eine präzisere Anpassung und Kompression ultrakurzer Pulse möglich wird. [2, pp. 5-8]

2.3 Neuronale Netze

Neuronale Netze, auch als künstliche neuronale Netze (KNN) bezeichnet, sind lernfähige Systeme, die mithilfe von miteinander verbundenen Knoten, den sogenannten Neuronen, Lernprozesse durchführen können. Ihre Struktur ist vom Aufbau biologischer neuronaler Netze inspiriert. Eingesetzt werden sie vor allem in Anwendungsgebieten, deren Komplexität eine Lösung durch klassische Programmiermethoden erschwert. [7, p. 19]

2.3.1 Grundaufbau

Die Grundlage für neuronale Netze wurde 1958 von Frank Rosenblatt mit seinem Modell des Perzeptrons geschaffen. Dieses ist in Abbildung 2.7 dargestellt. Das schematisch dargestellte Perzeptron verfügt über zwei binäre Eingänge, x_1 und x_2. Diese Eingänge werden jeweils mit Gewichten (w_1 und w_2) multipliziert und anschließend aufsummiert, sodass ein Gesamtwert entsteht. Übersteigt dieser Wert einen festgelegten Schwellenwert θ, so beträgt der Ausgang 1, andernfalls 0.

Das Perzeptron zeigte jedoch eine wesentliche Einschränkung. Es kann nicht-linear separierbare Probleme wie beispielsweise die XOR-Funktion nicht lösen. Diese Schwäche führte dazu, dass die Forschung an künstlichen neuronalen Netzen für

mehrere Jahre stagnierte. Erst die Entwicklung mehrschichtiger Netze und effizienter Lernverfahren ermöglichte spätere Fortschritte in diesem Bereich. [8, pp. 25-26]

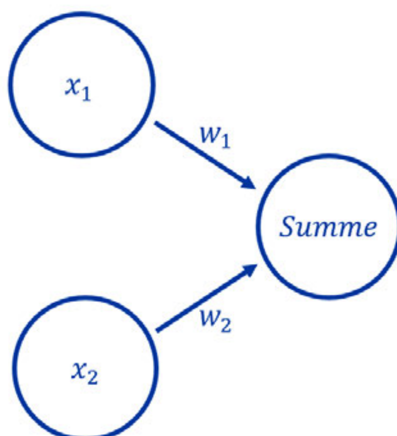


Abbildung 2.7 Schematisches Perzeptron, eigene Darstellung

Abbildung 2.8 zeigt ein beispielhaftes mehrschichtiges neuronales Netz. Es besteht aus einer beliebigen Anzahl von Neuronen und mindestens drei Schichten: einer Eingabeschicht, mindestens einer versteckten Schicht sowie einer Ausgabeschicht. Die Ausgänge einer Schicht dienen als Eingänge für die jeweils nachfolgende Schicht, mit Ausnahme der Eingabeschicht.

Die versteckte Schicht befindet sich zwischen Eingabe- und Ausgabeschicht. In ihr werden die von der Eingabeschicht empfangenen Informationen neu gewichtet und von Schicht zu Schicht bis zur Ausgabeschicht weitergereicht. Der genaue Prozess der Informationsverarbeitung innerhalb dieser Schicht ist von außen nicht einsehbar, weshalb sie auch als versteckte Schicht bezeichnet wird.

In der Eingabeschicht werden die Eingangswerte zugeführt, zum Beispiel Größe und Farbe bei einer Klassifikation zwischen Birne und Apfel. Die Ausgabeschicht liefert schließlich eine Entscheidung darüber, ob es sich um eine Birne oder einen Apfel handelt. Je nachdem, wie viele Ausgaben benötigt werden, variiert die Anzahl der Neuronen in der Ausgabeschicht. Im vorliegenden Beispiel sind es zwei, jeweils eines für Apfel und Birne.

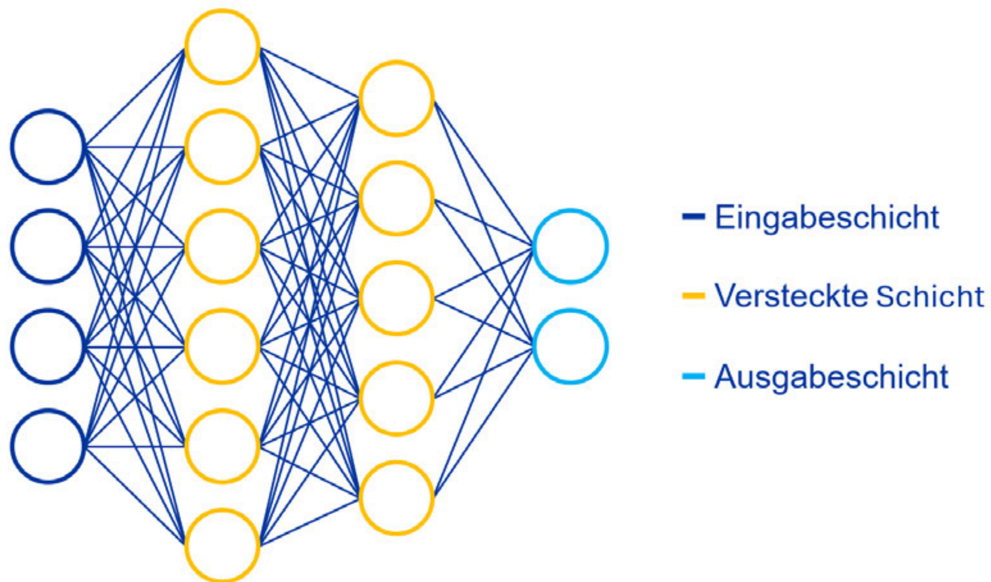


Abbildung 2.8 Schematisches Neuronales Netz, eigene Darstellung

2.3.2 Neuronen

Die Abbildung 2.9 zeigt drei Eingangsneuronen, die jeweils mit einem Neuron der versteckten Schicht verbunden sind. Die Verbindungen sind mit Gewichten w versehen, die mit den entsprechenden Eingabewerten multipliziert werden und während des Trainingsprozesses optimiert werden. In jeder Schicht, mit Ausnahme der Ausgabeschicht, ist zusätzlich ein Bias-Neuron enthalten, das stets den Wert 1 besitzt und ebenfalls mit den nachfolgenden Neuronen verbunden ist. Der Bias ermöglicht es, die Aktivierungsfunktion flexibel zu verschieben und so die Anpassungsfähigkeit des Modells zu erhöhen.

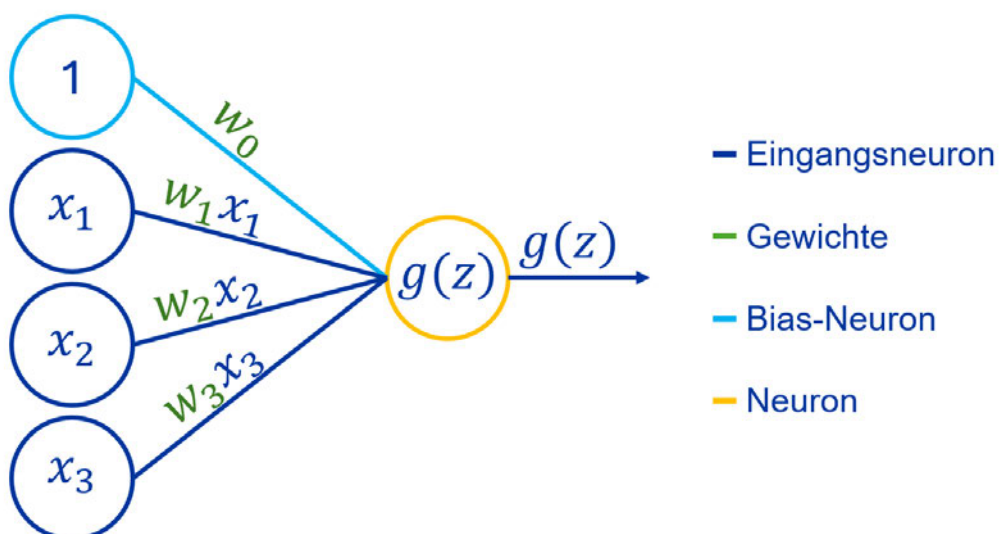


Abbildung 2.9 Schematisches Neuronen, eigene Darstellung

Im nachfolgenden Neuron die Aktivierungsfunktion $g(z)$ angewendet, auf die im nächsten Abschnitt näher eingegangen wird. Der Wert z ergibt sich als gewichtete Summe der Eingänge inklusive Bias (siehe Formel 2.7). Diese Berechnung wird für jedes Neuron im Netzwerk analog durchgeführt.

$$z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \quad (2.7)$$

2.3.3 Aktivierungsfunktionen

Für die Aktivierung eines Neurons existieren verschiedene Methoden. Im Folgenden werden zwei gebräuchliche Aktivierungsfunktionen näher betrachtet.

Ein erstes Beispiel ist die lineare Aktivierung. Sie ist durch die Gleichung $g(z) = z$ definiert, das heißt, der Eingabewert wird unverändert als Ausgabe übernommen. Lineare Aktivierungen finden häufig in der Ausgabeschicht von neuronalen Netzen Anwendung, insbesondere dann, wenn ein kontinuierlicher Wert vorhergesagt werden soll. Ein weiteres Beispiel ist die Rectified Linear Unit (ReLU), die durch die Gleichung $g(z) = max(0, z)$ definiert ist. Der Schwellenwert der Eingabe liegt bei 0, sodass bei negativen Werten eine 0 ausgegeben wird. Für Eingaben größer 0 verhält sich die ReLU wie eine lineare Funktion mit der Steigung 1. Ein wesentlicher Vorteil der ReLU-Aktivierungsfunktion besteht darin, dass sie sehr einfach zu berechnen ist und im Vergleich zu anderen Aktivierungsfunktionen weniger anfällig für das Problem des verschwindenden Gradienten ist. [8, pp. 29-33]

Das Problem des verschwindenden Gradienten entsteht vor allem durch die Verwendung von Aktivierungsfunktionen, die bei bestimmten Eingabewerten sehr kleine Ableitungen liefern und dadurch die Gradienten stark abschwächen. Werden die Gradienten zu klein, erhalten die tieferen Schichten eines neuronalen Netzes kaum noch nützliche Information für die Aktualisierung ihrer Gewichte, was das Lernen stark verlangsamt oder sogar verhindert. Bei der ReLU-Aktivierungsfunktion bleibt der Gradient für positive Eingabewerte konstant bei 1, sodass dieses Problem weitgehend vermieden wird.

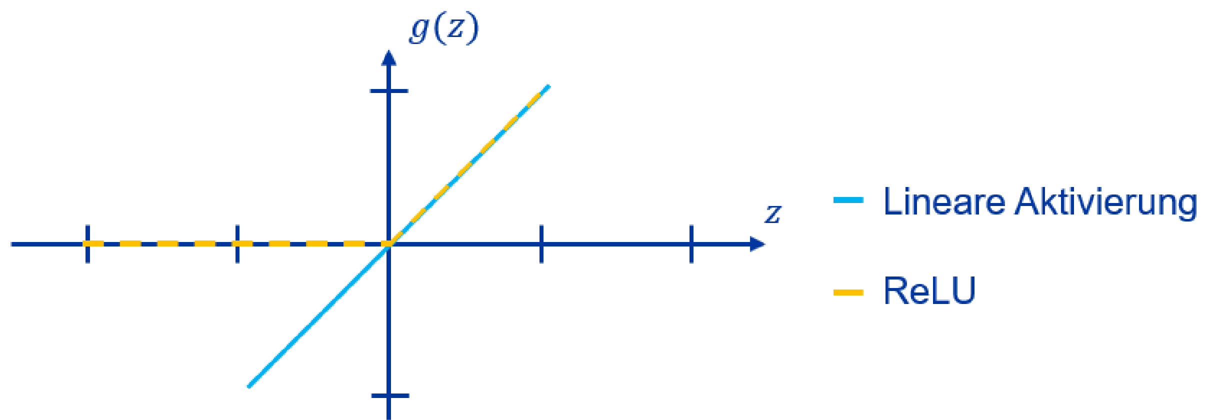


Abbildung 2.10 Unterschiedliche Aktivierungsfunktionen, eigene Darstellung

2.3.4 Arten des maschinellen Lernens

Grundsätzlich lassen sich drei Arten des maschinellen Lernens unterscheiden. Sie unterscheiden sich darin, auf welche Weise ein Modell trainiert wird und welche Informationen während des Lernprozesses zur Verfügung stehen (siehe Abbildung 2.11).

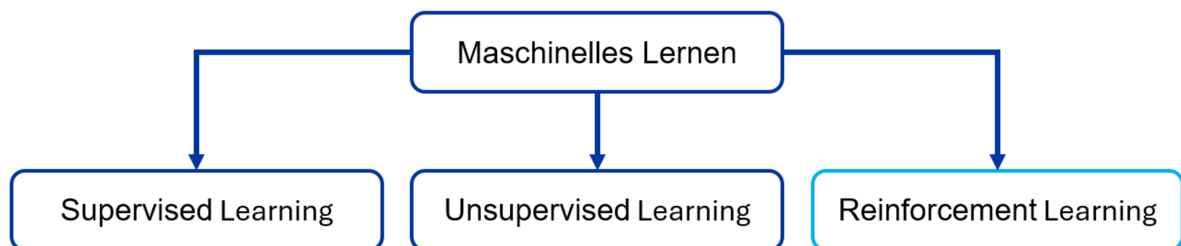


Abbildung 2.11 Arten des maschinellen Lernens, eigene Darstellung

Beim Supervised Learning (deutsch: überwachtes Lernen) wird ein Modell anhand eines gelabelten Datensatzes trainiert. Das bedeutet, dass die Trainingsdaten bereits mit den richtigen Antworten versehen sind. Das Modell lernt dabei, Zusammenhänge zwischen Eingaben und Ausgaben zu erkennen. Nach erfolgreichem Training kann es auch neue, unbekannte Daten richtig zuordnen. Stellt man dem Modell beispielsweise viele Bilder von Bananen und Äpfeln zur Verfügung, die jeweils korrekt beschriftet sind, so kann es die charakteristischen Merkmale beider Obstsorten erlernen und anschließend ein neues Bild korrekt als Banane oder Apfel klassifizieren.

Beim unsupervised Learning (deutsch: unüberwachtes Lernen) liegen dagegen keine gelabelten Daten vor. Das Modell versucht selbst, Muster und Strukturen in den Daten zu erkennen, indem es beispielsweise Gruppen bildet. Wird ein Datensatz mit Bildern von Bananen und Äpfeln ohne Beschriftungen verwendet, erkennt das Modell

dennoch, dass es zwei Gruppen gibt. Eine mit länglichen gelben Objekten und eine mit runden roten oder grünen. Zwar benennt es die Gruppen nicht direkt, kann aber Unterschiede herausarbeiten und so eine Struktur in den Daten erkennen.

Das Reinforcement Learning (deutsch: verstärkendes Lernen) unterscheidet sich von beiden Ansätzen. Hier lernt ein Agent durch Interaktion mit einer Umgebung eine Strategie, um eine langfristige Belohnung zu maximieren. Statt korrekter Antworten erhält er für seine Aktionen Rückmeldungen in Form von Belohnungen oder Bestrafungen. Ein typisches Beispiel ist ein Agent, der lernen soll, ein Labyrinth zu verlassen. Für jeden Schritt erhält er eine kleine negative Belohnung, um den Weg kurz zu halten, und für das Erreichen des Ziels eine große positive. Der Agent lernt so eine Abfolge von Aktionen (z. B. gehe nach links, dann geradeaus), die zur höchsten Gesamtbelohnung führt. In Abbildung 2.11 ist dieser Ansatz hervorgehoben, da sich die vorliegende Arbeit im weiteren Verlauf mit Reinforcement Learning befasst. [9]

2.4 Reinforcement Learning

Das Reinforcement Learning ist, wie in Abschnitt 2.3.4 bereits erwähnt, eine Art des maschinellen Lernens. Im Unterschied zu den anderen Ansätzen des maschinellen Lernens erfolgt das Lernen hier durch direkte Interaktion mit der Umgebung. Abbildung 2.12 zeigt den schematischen Aufbau eines Reinforcement-Learning-Frameworks.

Die Belohnung ist ein Wert, der sowohl groß als auch klein sowie positiv oder negativ sein kann. Sie dient dazu, dem Agenten Rückmeldung darüber zu geben, ob sein Verhalten in einer bestimmten Situation als gut oder schlecht bewertet wird. Der Zeitpunkt, zu dem eine Belohnung vergeben wird, kann flexibel definiert werden, beispielsweise nach einer bestimmten Zeitspanne, bei bestimmten Interaktionen oder am Ende einer Simulation. Die Belohnung verstärkt das Verhalten des Agenten, indem positive Belohnungen gewünschte Aktionen fördern und negative Belohnungen unerwünschte Handlungen hemmen. Obwohl die Belohnung sich auf die unmittelbar zuvor ausgeführte Aktion bezieht, ist das Ziel des Agenten, eine Strategie zu erlernen, die die Summe der zukünftigen Belohnungen maximiert. [10, p. 29]

Der Agent interagiert mit der Umgebung, indem er diese beobachtet, Aktionen ausführt und für diese Aktionen Belohnungen erhält.

Die Umgebung ist der Raum, in dem der Agent seine Aktionen ausführen kann. Die Kommunikation mit der Umgebung beschränkt sich auf die Aktionen des Agenten, die

Beobachtungen der Umgebung und die Rückmeldung in Form von Belohnungen. Nach jeder Aktion des Agenten passt die Umgebung ihren Zustand entsprechend an und gibt eine neue Beobachtung sowie eine Belohnung zurück.

Aktionen werden vom Agenten innerhalb der Umgebung ausgeführt. Grundsätzlich unterscheidet man zwischen zwei Arten von Aktionen, diskrete und stetige Aktionen. Diskrete Aktionen sind beispielsweise einmalige Bewegungen nach vorne oder hinten. Stetige Aktionen hingegen umfassen kontinuierliche Steuergrößen, wie etwa den Drehwinkel eines Lenkrads beim Steuern eines Autos. Im Fall stetiger Aktionen führen bereits kleine Änderungen der Eingabe nach kurzer Zeit zu unterschiedlichen Situationen in der Umgebung.

Die zweite Informationsquelle neben der Belohnung ist die Beobachtung. Beobachtungen geben Aufschluss darüber, was in der Umgebung aktuell geschieht, und unterstützen den Agenten dabei, die jeweilige Situation einzuschätzen. Sie können zudem Hinweise darauf liefern, unter welchen Bedingungen Belohnungen zu erwarten sind. Es ist jedoch zu unterscheiden zwischen Beobachtung und Zustand. Der Zustand umfasst die Gesamtheit aller relevanten Informationen der Umgebung, ist jedoch in vielen Fällen zu komplex, um vollständig erfasst zu werden. Die Beobachtung stellt lediglich einen Teil des Zustands dar und kann zudem ungenau oder unvollständig sein. [10, p. 32]

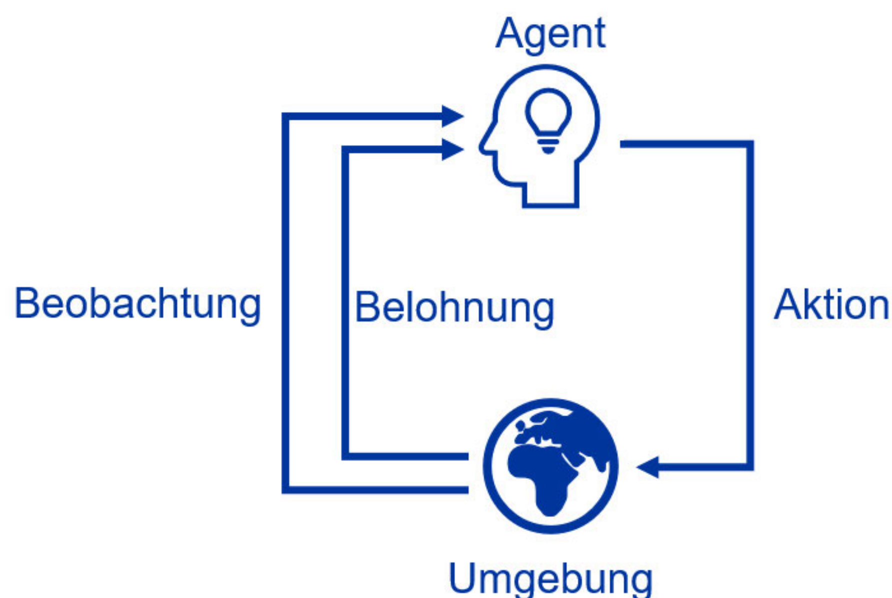


Abbildung 2.12 Reinforcement Learning, eigene Darstellung

2.4.1 Markov-Entscheidungsprozess

Der Markov-Entscheidungsprozess (MEP) ist ein Modell zur Beschreibung von Entscheidungsproblemen, bei denen ein Agent durch Interaktionen mit einer Umgebung ein Ziel erreichen soll. Die Gesamtheit aller möglichen Zustände eines Systems wird als Zustandsraum bezeichnet. Dieser kann sowohl endlich als auch kontinuierlich sein. Im Markov-Entscheidungsprozess ist diese Menge endlich. Die Formel 2.8 zeigt einen beispielhaften Ablauf. Ausgehend von einem Zustand s folgt eine Aktion a, auf die eine Belohnung r ausgegeben wird. Eine solche Abfolge wird als Markov-Prozess bezeichnet. Die Markov-Eigenschaft besagt, dass der nächste Zustand und die erhaltene Belohnung ausschließlich vom aktuellen Zustand und der gewählten Aktion abhängen, vorherige Zustände oder Aktionen sind dabei ohne Einfluss.

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2 \ . \ . \ . \tag{2.8}$$

Die Formel 2.9 beschreibt den erwarteten Ertrag G_t. Dieser setzt sich aus der Belohnung im aktuellen Zeitschritt t sowie den zukünftigen Belohnungen zusammen, die mit Potenzen eines Diskontfaktors γ gewichtet werden. Der Diskontfaktor γ legt fest, wie stark zukünftige Belohnungen im Vergleich zu unmittelbaren Belohnungen berücksichtigt werden. Ein Wert von γ nahe 1 führt dazu, dass zukünftige Belohnungen fast gleich stark wie aktuelle gewichtet werden, während ein Wert nahe 0 den Fokus stark auf kurzfristige Belohnungen legt.

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} \ . \ . \ . \tag{2.9}$$

Ein MEP lässt sich formal durch die folgenden Komponenten beschreiben. Die Zustandsmenge S umfasst alle möglichen Zustände, in denen sich die Umgebung befinden kann. Die Aktionsmenge A enthält die möglichen Aktionen, die der Agent in einem bestimmten Zustand ausführen kann. Die Übergangswahrscheinlichkeit $P(s'|s,a)$ beschreibt die Wahrscheinlichkeit, mit der nach Ausführung einer Aktion $a \in A$ im Zustand $s \in S$ der nächste Zustand $s' \in S$ erreicht wird. Die Belohnungsfunktion $R(s,a)$ definiert die erwartete Belohnung, die der Agent erhält, wenn er im Zustand s die Aktion a ausführt. Der MEP wird häufig als Tupel (S, A, P, R) dargestellt. [10, pp. 39-41]

Das Ziel bei der Arbeit mit einem Markov-Entscheidungsprozess ist es, eine optimale Policy zu finden. Eine Policy ist eine Strategie, die für jeden Zustand die beste Aktion vorschreibt, um den erwarteten Ertrag Gleichung 2.9 zu maximieren.

2.4.2 Value und Policy

Um in einem MEP eine optimale Strategie zu finden, benötigt der Agent eine Bewertungsfunktion für Zustände oder Aktionen, die sogenannten Value-Funktionen. Darauf aufbauend kann eine Policy entwickelt werden, die angibt, welche Aktion in einem bestimmten Zustand ausgeführt werden soll.

Während die Belohnung lediglich das unmittelbare Feedback der Umgebung nach einer einzelnen Aktion darstellt, betrachtet die Value-Funktion den erwarteten langfristigen Nutzen. Sie bewertet also nicht nur den aktuellen Schritt, sondern die Summe aller zukünftigen Belohnungen, die ausgehend von einem bestimmten Zustand oder einer bestimmten Aktion erzielt werden können. Man unterscheidet dabei zwischen der Zustandswertfunktion $V^{\pi}(s)$, die den erwarteten Ertrag bei Befolgung einer Policy π ab einem Zustand s angibt, und der Aktionswertfunktion $Q^{\pi}(s, a)$, die zusätzlich berücksichtigt, welche konkrete Aktion a in einem Zustand s ausgeführt wird.

Eine Policy beschreibt die Menge an Regeln, nach denen das Verhalten des Agenten gesteuert wird. Ziel des Agenten ist es, eine Policy zu finden, die den langfristig erwarteten Ertrag maximiert. Das Lernen im RL zielt daher darauf ab, eine solche optimale Policy zu finden, die den erwarteten Ertrag maximiert.

Formal wird eine Policy π definiert als eine Abbildung von Zuständen auf Wahrscheinlichkeitsverteilungen über Aktionen:

$$\pi(a|s) = P(A_t = a|S_t = s) \tag{2.10}$$

Die Policy gibt damit an, mit welcher Wahrscheinlichkeit der Agent in einem bestimmten Zustand s eine bestimmte Aktion a wählt. Es wird dabei unterschieden zwischen deterministischer Policy, wobei jedem Zustand genau eine Aktion fest zugeordnet wird und stochastischer Policy, bei denen Aktionen anhand einer Wahrscheinlichkeitsverteilung gewählt werden. [10, p. 44]

2.4.3 Q-Learning

Q-Learning ist eine Methode des bestärkenden Lernens, mit der eine optimale Strategie erlernt werden soll. Grundlage bildet das Optimalitätsprinzip nach Bellman, welches besagt, dass der Wert eines Zustands durch die unmittelbar erzielte Belohnung sowie die bestmögliche zukünftige Belohnung bestimmt wird. Ein schematischer Ablauf des Q-Learning-Prozesses ist in der Abbildung 2.13 dargestellt.

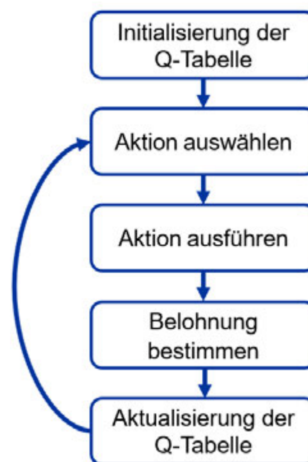


Abbildung 2.13 Ablauf Q-Learning, eigene Darstellung

Zu Beginn erfolgt die Initialisierung einer Q-Tabelle, in der die möglichen Zustände und die dazugehörigen Aktionen hinterlegt werden. Anschließend wird eine erste Aktion ausgewählt und in der Umgebung ausgeführt. Daraufhin wird die resultierende Belohnung durch die Umgebung zurückgemeldet. Auf Basis dieser Belohnung wird die Q-Tabelle mithilfe der Bellman-Gleichung aktualisiert (siehe Formel 2.11). $Q(s,a)$ beschreibt den erwarteten Gesamtertrag, wenn man im Zustand s eine Aktion a auswählt. r ist die unmittelbare Belohnung, die der Agent erhält, wenn eine Aktion ausgeführt wird. γ ist erneut der Diskontfaktor und gewichtet zukünftige Belohnungen ab. $max\ Q$ bezeichnet den optimalen Q-Wert im Folgezustand s'. Der Agent wählt in diesem Zustand das a', weil er denkt, dass dies die bestmögliche Aktion ist.

$$Q(s,a) = r(s,a) + \gamma \cdot max\ Q(s',a') \qquad (2.11)$$

Im nächsten Schritt wird erneut eine Aktion ausgewählt, wodurch sich der Ablauf fortlaufend wiederholt. Zur Veranschaulichung dient ein einfaches Beispiel. Die Abbildung 2.14 zeigt ein zweidimensionales Gitter mit einer Größe von 2×2. Der Agent startet in der linken unteren Ecke, während das Ziel in der rechten oberen Ecke liegt. Ziel ist es, den kürzesten Weg zum Ziel zu finden. Dem Agenten stehen dabei die vier Aktionen „oben“, „unten“, „links“ und „rechts“ zur Verfügung. Die Q-Tabelle wird initial mit dem Wert Null belegt. In den ersten Durchläufen werden die Aktionen zufällig ausgewählt. In späteren Durchläufen greift der Agent zunehmend auf die bereits aktualisierte Q-Tabelle zurück, um seine Entscheidungen zu treffen. Eine häufig eingesetzte Strategie ist hierbei die sogenannte Epsilon-greedy-Strategie, die ein Gleichgewicht zwischen Exploration (Erkundung neuer Aktionen) und Exploitation (Ausnutzung des bereits Gelernten) herstellt.

Abbildung 2.14 Beispiel Gitter 2x2, eigene Darstellung

Die Belohnungsstruktur wird wie folgt definiert: Jeder Schritt, der nicht zum Ziel führt, wird mit einer negativen Belohnung von -0,1 bestraft, um den Agenten zu motivieren, kurze Wege zu finden. Das Erreichen des Ziels im Zustand (1,1) wird hingegen mit +1 belohnt. Ungültige Züge, bei denen der Agent den Rand des Gitters überschreiten würde, führen dazu, dass er im selben Zustand bleibt und ebenfalls die Schrittstrafe von -0,1 erhält. Tabelle 1 zeigt eine beispielhafte Q-Tabelle nach einem erfolgreichen Lernprozess. Die Aktualisierung erfolgt nach jeder Aktion anhand der Bellman-Gleichung (Formel 2.11). Diese Berechnung berücksichtigt die unmittelbare Belohnung sowie den maximalen erwarteten zukünftigen Ertrag des Folgezustands.

Tabelle 1 Q-Tabelle für 2x2 Gitter

Zustand	**oben**	**oben**	**links**	**rechts**
0,0	0,70	0,46	0,46	0,70
0,1	1,00	0,70	0,46	0,70
1,0	0,70	0,46	0,70	1,00
1,1	0	0	0	0

Betrachtet man den Zustand $(0{,}0)$, so gibt es zwei grundsätzliche Handlungsoptionen. Wählt der Agent die Aktion „oben“, gelangt er in den Zustand $(0{,}1)$. Von dort aus kann er im nächsten Schritt optimal handeln und direkt das Ziel $(1{,}1)$ erreichen. Der Q-Wert berechnet sich damit folgendermaßen:

$$Q\big((0{,}0), oben\big) = -0{,}1 + 0{,}8 \cdot 1 = 0{,}7$$

Entscheidet sich der Agent hingegen für eine ungültige Aktion, wie zum Beispiel „unten", so bleibt er im Zustand $(0{,}0)$ stehen. Er erhält zunächst die Schritt-Kosten und kann anschließend wieder optimal weiterlaufen. In diesem Fall ergibt sich der Q-Wert zu:

$$Q\big((0{,}0), unten\big) = -0{,}1 + 0{,}8 \cdot V(0{,}0) = -0{,}1 + 0{,}8 \cdot 0{,}7 = 0{,}46$$

2.4.4 Deep-Q-Networks

Mit zunehmender Aufgabenkomplexität stößt tabellarisches Q-Learning an Grenzen. Es setzt endliche und handhabbare Zustands- und Aktionsmengen voraus. In realitätsnahen Szenarien sind Zustände oft kontinuierlich und mehrdimensional, wie etwa Positionen, Geschwindigkeiten, Beschleunigungen oder Winkel. Hier setzen Deep Q-Networks (DQN) an. Ein neuronales Netz approximiert die Q-Funktion und ermöglicht damit das Lernen in kontinuierlichen Zustandsräumen. Bekannt wurden DQN durch die 2015 publizierte Arbeit von DeepMind, in der aus reinen Bildpixeln stammende Zustände in diversen Atari-Spielen zu übermenschlicher Leistung führten. [11]

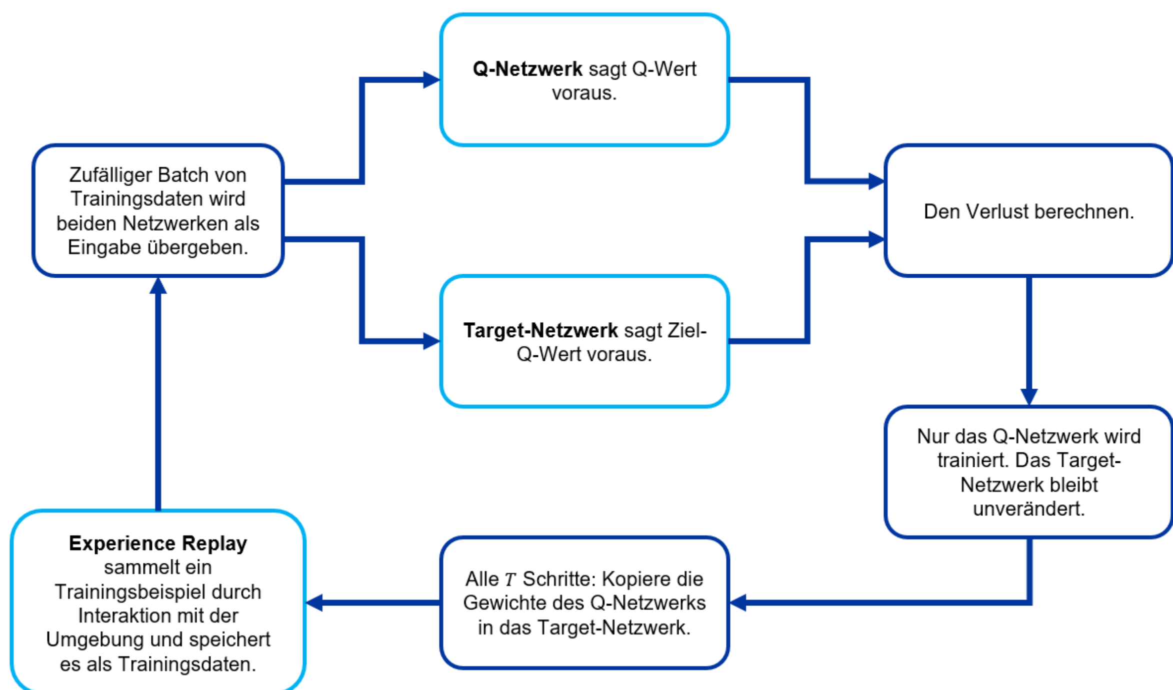


Abbildung 2.15 Ablaufplan DQN, eigene Darstellung

Ein DQN umfasst drei Kernelemente (siehe Abbildung 2.15). Das Q-Netzwerk, das Target-Netzwerk und den Experience-Replay-Puffer. Das Q-Netzwerk schätzt die Q-Werte und wird trainiert. Das Target-Netzwerk ist eine zeitverzögert aktualisierte

Kopie, die stabile Zielwerte liefert. Der Replay-Puffer speichert Übergänge für das spätere Training.

Der Trainingsablauf folgt der in Abbildung 2.15 dargestellten Schleife. Der Agent wählt eine Aktion, führt sie in der Umgebung aus und erhält Belohnung r sowie den Folgezustand s'. Der Übergang (s, a, r, s') wird im Replay-Puffer abgelegt. Für das Lernen wird ein zufälliges Mini-Batch aus dem Puffer gezogen. Das Q-Netzwerk berechnet daraus die vorhergesagten Q-Werte, das Target-Netzwerk liefert die Zielwerte (siehe Formel 2.12).

$$y = r + \gamma max Q(s', a'; \theta-) \quad (2.12)$$

Anschließend wird der Verlust gebildet und nur das Q-Netzwerk aktualisiert. In regelmäßigen Abständen werden die Gewichte synchronisiert, damit die Zielwerte mit dem Lernfortschritt Schritt halten, ohne das Training zu destabilisieren. [11]

2.4.5 Soft-Actor-Critic

Eine Erweiterung der bisher betrachteten Methoden stellt der sogenannte Actor-Critic-Ansatz dar. Während bei Q-Learning oder DQN die Policy nur implizit aus der erlernten Bewertungsfunktion (Critic) abgeleitet wird, ergänzt Actor-Critic dies um eine zweite Komponente, den Actor, der die Policy explizit repräsentiert. Der Actor repräsentiert eine Policy, nach der der Agent in einem bestimmten Zustand eine Aktion auswählt. Der Critic hingegen bewertet die vom Actor getroffene Entscheidung, indem er mithilfe einer Q-Funktion deren langfristigen Nutzen einschätzt.

Ein Beispiel hierfür ist ein Tennisspiel. Beim Q-Learning war es bisher so, dass der Agent erst am Ende des Spiels die Rückmeldung erhält, ob die Partie gewonnen oder verloren wurde. Es gibt jedoch keine Information darüber, welche einzelnen Entscheidungen während des Spiels gut oder schlecht waren. Im Actor-Critic-Ansatz ist die Rückmeldung differenzierter. Der Actor repräsentiert den Tennisspieler, der die Schläge ausführt, während der Critic die Rolle des Trainers übernimmt. Nach jedem Schlag bewertet der Critic, ob die gewählte Aktion sinnvoll war. Der Actor erhält dadurch nicht nur das Gesamtergebnis am Ende des Spiels, sondern detailliertes Feedback darüber, welche Aktionen vorteilhaft waren und welche vermieden werden sollten.

Darauf aufbauend stellt der Soft Actor-Critic (SAC) Algorithmus eine Weiterentwicklung dar. Während klassische Actor-Critic-Verfahren nur darauf ausgelegt sind, möglichst viel Belohnung zu sammeln, verfolgt SAC ein erweitertes Ziel. Neben der Belohnung wird auch berücksichtigt, dass die Strategie des Agenten eine gewisse Zufälligkeit beibehält. Dadurch wird vermieden, dass der Agent zu schnell immer wieder dieselben Handlungen ausführt und sich damit auf eine schlechte Lösung festlegt. Stattdessen probiert er weiterhin unterschiedliche Aktionen aus. Formal wird dies umgesetzt, indem SAC die Summe aus erwarteter Belohnung und einem gewichteten Entropieterm maximiert.

Um verlässliche Rückmeldungen zu erhalten, verwendet SAC zwei getrennte Critics. Auf diese Weise können übertriebene Einschätzungen einzelner Aktionen ausgeglichen werden. Zusätzlich greift SAC auf Stabilisierungstechniken zurück, die bereits beim DQN eingeführt wurden, wie etwa das Zwischenspeichern von Erfahrungen in einem Replay-Puffer und ein Zielnetzwerk. Durch diese Kombination eignet sich SAC besonders für Probleme mit vielen kontinuierlichen Stellgrößen, wie sie in dieser Arbeit auftreten. [12]

2.4.6 Vergleich Reinforcement-Algorithmen

Proximal Policy Optimization (PPO) ist eine On-Policy Policy-Gradient-Methode, die für einen stabilen Lernprozess konzipiert ist. Der Algorithmus lernt ausschließlich mit Daten der aktuellen Policy, die nach jedem Lernschritt verworfen werden. Dies führt zu geringer Dateneffizienz, aber hoher Zuverlässigkeit. PPO nutzt eine Actor-Critic-Architektur und implementiert einen Clipping-Mechanismus, der abrupte und destabilisierende Policy-Änderungen verhindert. Dieses Vorgehen gewährleistet ein inkrementelles Lernen, bei dem Stabilität über Lerngeschwindigkeit priorisiert wird. [13]

Twin Delayed Deep Deterministic Policy Gradient (TD3) ist ein dateneffizienter Off-Policy Actor-Critic-Algorithmus, der für kontinuierliche Aktionsräume entwickelt wurde. Er korrigiert die systematische Überschätzung von Q-Werten, die bei seinem Vorgänger DDPG auftritt. Dazu nutzt TD3 drei kombinierte Techniken. Clipped Double-Q Learning verwendet den Minimalwert zweier Q-Funktionen, um die Wertschätzung zu reduzieren. Verzögerte Policy-Updates aktualisieren den Actor seltener als den Critic, was die Stabilität erhöht. Target Policy Smoothing fügt der Zielaktion Rauschen hinzu und glättet so die Lernlandschaft. Der Algorithmus lernt eine deterministische Policy,

deren Exploration durch externes Rauschen während des Trainings sichergestellt wird. [14]

Tabelle 2 Vergleich Reinforcement-Algorithmen

Kriterium	SAC	TD3	PPO
Lernparadigma	Off-Policy	Off-Policy	On-Policy
Sample-Effizienz	sehr hoch	hoch	gering
Replay-Buffer	ja	ja	nein
Exploration	Entropie-Regelung	Action-Noise	Stoch. Policy
Policy	stochastisch	deterministisch	stochastisch

Die Wahl des passenden Reinforcement-Learning-Algorithmus ist entscheidend für den Erfolg in einer spezifischen Anwendung. Die in Tabelle 2 betrachteten Algorithmen, verfolgen unterschiedliche Ansätze. Im folgenden werden diese miteinander verglichen.

Ein grundlegendes Unterscheidungsmerkmal ist das Lernparadigma. Sowohl SAC als auch TD3 sind Off-Policy-Algorithmen. Das bedeutet, sie nutzen einen sogenannten Replay-Buffer, um vergangene Erfahrungen wiederholt für das Training zu verwenden. Dies führt zu einer sehr hohen Sample-Effizienz, da aus jeder einzelnen Interaktion mit der Umgebung ein maximaler Lernfortschritt erzielt wird. Im Gegensatz dazu steht PPO als On-Policy-Algorithmus, der ausschließlich mit den Daten der aktuellen Policy lernt und diese danach verwirft. Diese Vorgehensweise macht ihn zwar sehr stabil, aber auch datenineffizient. Für eine Anwendung wie das Puls-Shaping, bei der jede Interaktion einem Laser-Schuss entspricht, ist eine hohe Sample-Effizienz ein wichtiges Kriterium. Aus diesem Grund sind SAC und TD3 prinzipiell besser geeignet als PPO. [13]

Ein weiterer zentraler Aspekt ist die Art der erlernten Policy und die damit verbundene Exploration. TD3 lernt eine deterministische Policy, die für einen gegebenen Zustand immer dieselbe Aktion auswählt. Die notwendige Erkundung des Lösungsraums wird hier extern durch das Hinzufügen von Rauschen zur Aktion erzwungen. SAC und PPO hingegen lernen eine stochastische Policy, bei der Aktionen aus einer

Wahrscheinlichkeitsverteilung gezogen werden. Die Exploration ist somit ein fester Bestandteil der Strategie. SAC geht hierbei noch einen Schritt weiter, indem er die Entropie der Policy maximiert. Dies belohnt den Agenten explizit dafür, möglichst vielfältige Aktionen auszuprobieren. Diese Exploration macht SAC besonders robust gegen das Festfahren in nicht optimalen Lösungen, was in komplexen Umgebungen wie dem Puls-Shaping von großem Vorteil ist. [14]

Für die Anwendung als Puls-Shaper kristallisiert sich der SAC als der am wahrscheinlich besten geeignete Algorithmus heraus. Seine entscheidenden Vorteile sind die hohe Sample-Effizienz als Off-Policy-Ansatz und eine überlegene Explorationsstrategie. Die Maximierung der Entropie sorgt für eine systematische und robuste Suche nach dem Optimum, was der auf simplem Rauschen basierenden Exploration von TD3 deutlich überlegen ist.

2.5 Convolutional Neural Networks

In den vorangegangenen Abschnitten wurden Neuronale Netze und Reinforcement Learning eingeführt. Bisher lag der Fokus auf Architekturen, die einen eindimensionalen Merkmalsvektor von der Eingabe bis zur Ausgabe verarbeiten. Mit wachsender Zahl der Ein- und Ausgangsparameter steigen Größe und Parameterzahl solcher Netze sehr schnell. Im nächsten Schritt dient ein FROG-Bild als Eingang. Bei einer Auflösung von 256×256 Pixeln handelt es sich um ein Einkanalbild, bei dem jeder Pixelwert einer Intensität entspricht. Daraus ergeben sich $256 \cdot 256 \cdot 1 = 65.536$ Eingabewerte. Vollständig verbundene Schichten müssten dafür entsprechend breit sein und die Zahl der Gewichte würde explodieren. Das führt zu langen Trainingszeiten, hohem Rechenbedarf und großem Datenbedarf. Eine passende Lösung sind Convolutional Neural Networks (CNN).

Es existieren CNNs für eindimensionale und zweidimensionale Daten. Da hier mit Bilddaten gearbeitet wird, wird im Folgenden nur der Aufbau eines zweidimensionalen CNN betrachtet. Die Abbildung 2.16 zeigt einen beispielhaften Aufbau.

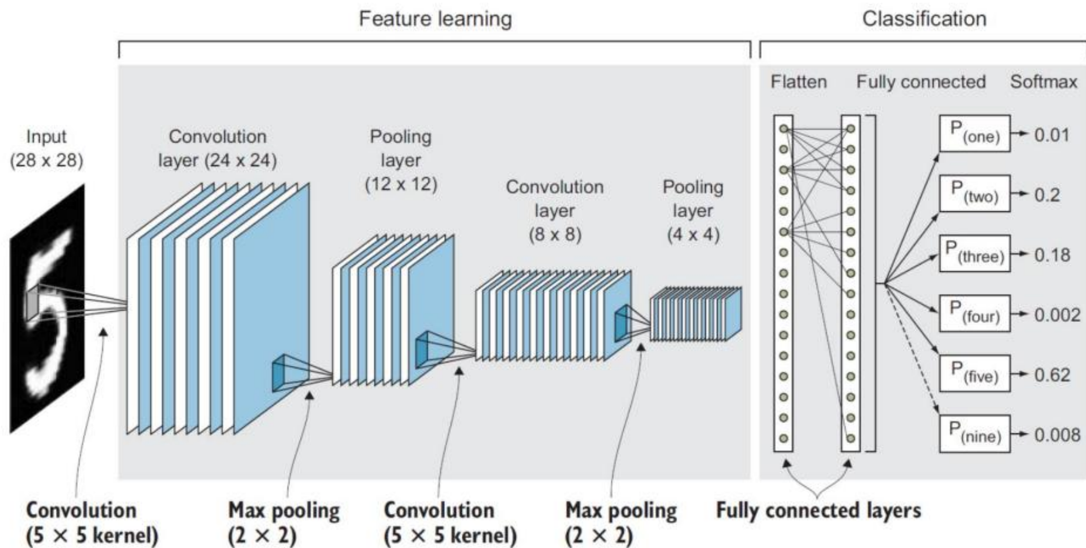


Abbildung 2.16 Beispielhafter Aufbau CNN [15]

Ein CNN besteht aus Faltungsschichten, Pooling-Schichten und vollständig verbundenen Schichten. Im Folgenden wird auf die einzelnen Schichten genauer eingegangen.

In einer Faltungsschicht wird ein Filterkern über das Eingabebild geschoben. Abbildung 2.17 veranschaulicht dieses Vorgehen. Dies wird mit unterschiedlichen Filterkernen wiederholt.

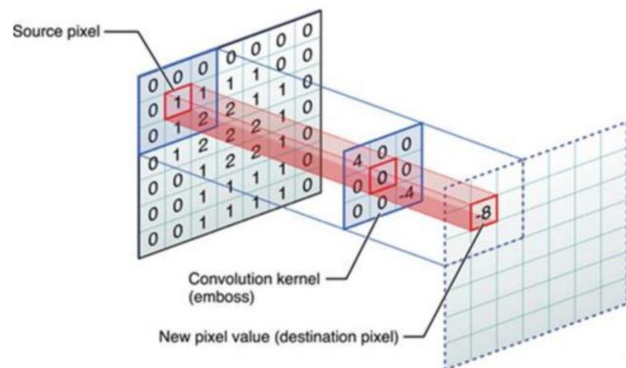


Abbildung 2.17 Filterkern [15]

Die Aktivierung an Position i, j entsteht als inneres Produkt des Filterkerns mit dem aktuell betrachteten Bildausschnitt in x- und y-Richtung. Eine mögliche Schreibweise ist:

$$y_{i,j}^{k} = \sum_{u=0}^{h} \sum_{v=0}^{w} K_{u,v}^{k} \, X_{i+u,j+v} \tag{2.13}$$

Dabei ist X das Eingabebild, K der Filterkern und h, w die Höhe und Breite des Kerns. Faltungsschichten erkennen und extrahieren Merkmale im Bild. Dazu zählen Kanten, Linien, Texturen und einfache Formen. Für jeden Filter entsteht eine eigene Feature-Map. [16]

Die Pooling-Schicht reduziert die Auflösung der Merkmale. Dabei werden überflüssige Informationen verworfen. Ein Beispiel ist Max-Pooling. Dabei wird aus einem $2 x 2$ Quadrat der höchste Wert übernommen. Der Rest wird verworfen. Die Abbildung 2.18 zeigt dies. Im linken oberen Quadrat ist die 4 der höchste Wert, deswegen wird sie behalten und der Rest verworfen. Pooling-Schichten sorgen für eine Datenreduktion, in diesem Beispiel um 75 Prozent. Pooling-Schichten besitzen keine eigenen Gewichte, daher entsteht kein zusätzlicher Trainingsaufwand. Die Reduzierung der Bildgröße der Feature Maps verringert den Rechenaufwand deutlich. In der Regel wird jede Feature Map separat behandelt, damit bleibt die Anzahl der Feature Maps unverändert. [16]

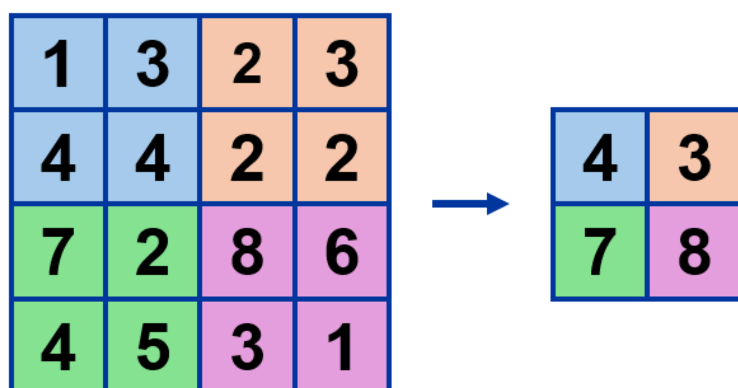


Abbildung 2.18 Pooling Schicht, eigene Darstellung

Nachdem mehrfach Convolutional- und Pooling-Layer im neuronalen Netz eingesetzt wurden, bilden vollständig verknüpfte Schichten den Abschluss. Alle Elemente der vorhergehenden Schicht sind mit jedem Ausgabemerkmal verknüpft. Die Anzahl der Neuronen ist hier abhängig von der Anzahl der Ausgaben. [16]

2.6 Root Mean Square

Der Root Mean Square (RMS) (deutsch: quadratischer Mittelwert) wird berechnet, indem man die Werte quadriert, den Mittelwert dieser Quadrate bildet und die Quadratwurzel zieht. RMS ist ein gutes Maß für Größen, die sich über die Zeit ändern. Es fasst die Größe einer variierenden Menge zu einem einzelnen Wert zusammen. RMS ist nie

negativ und gegenüber Vorzeichenwechseln invariant. Große Beträge wiegen stärker, weil sie beim Quadrieren schneller wachsen. [17]

Die diskrete Definition lautet:

$$RMS = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x_n^2} \qquad (2.14)$$

Verallgemeinert mit Gewichten $w_n > 0$:

$$RMS = \sqrt{\frac{\sum_n w_n x_n^2}{\sum_n w_n}} \qquad (2.15)$$

In dieser Arbeit misst die RMS-Breite die zeitliche Streuung der Pulsenergie. Das Breitenmaß soll unabhängig von der absoluten Lage auf der Zeitachse sein. Deshalb wird auf den Intensitätsschwerpunkt μ zentriert und die RMS auf die Abstände $t_n - \mu$ angewendet. Die Intensitäten I_n dienen als Gewichte, damit energiehaltige Anteile stärker in die Breite eingehen. Damit ergibt sich die intensitätsgewichtete RMS-Breite wie folgt:

$$RMS_{Breite} = \sqrt{\frac{\sum_n I_n (t_n - \mu)^2}{\sum_n I_n}} \quad mit \quad \mu = \frac{\sum_n t_n I_n}{\sum_n I_n} \qquad (2.16)$$

Als einfache Beispielrechnung dienen die Intensitätswerte $I_n = \{1,2,3,2,1\}$ an den Zeitpunkten $t_n = \{1,2,3,2,1\}$.

$$RMS = \sqrt{\frac{\sum_{n=1}^{5} w_n x_n^2}{\sum_{n=1}^{5} w_n}} = \sqrt{\frac{1 \cdot 1^2 + 1 \cdot 2^2 + 1 \cdot 3^2 + 1 \cdot 2^2 + 1 \cdot 1^2}{1 + 1 + 1 + 1 + 1}} = \sqrt{\frac{19}{5}} \approx 1{,}949$$

Dies beschreibt die effektive Höhe der gegebenen Wertefolge. Um nun die Streuung in der Zeit (das Breitenmaß) zu erhalten, wird zunächst der Schwerpunkt berechnet:

$$\mu = \frac{\sum_n t_n I_n}{\sum_n I_n} = \frac{1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 + 4 \cdot 2 + 5 \cdot 1}{1 + 2 + 3 + 2 + 1} = \frac{27}{9} = 3$$

Der Schwerpunkt liegt also beim dritten Zeitschritt. Damit ergibt sich die RMS-Breite zu:

$$RMS_{Breite} = \sqrt{\frac{\sum_n I_n (t_n - \mu)^2}{\sum_n I_n}}$$

$$= \sqrt{\frac{1(1-3)^2 + 2(2-3)^2 + 3(3-3)^2 + 2(4-3)^2 + 1(5-3)^2}{9}}$$

$$= \sqrt{\frac{1 \cdot 4 + 2 \cdot 1 + 3 \cdot 0 + 2 \cdot 1 + 1 \cdot 4}{9}} = \sqrt{\frac{12}{9}} \approx 1{,}1547$$

3 Simulation

Im vorherigen Kapitel wurden die theoretischen Grundlagen zu Laserpulsen, deren Messverfahren sowie den eingesetzten Optimierungsverfahren erläutert. Aufbauend darauf widmet sich das folgende Kapitel der praktischen Umsetzung in Form von Simulationsdaten, die als Grundlage für das Training der neuronalen Netze dienen.

Hierbei werden zwei Ansätze verfolgt. Ein neuronales Netz wird mit Pulsen trainiert, während ein zweites auf Basis von FROG-Daten trainiert wird. In den nachfolgenden Unterkapiteln wird die jeweilige Erzeugung der Simulationsdaten beschrieben.

3.1 Erzeugen des Referenzpulses

Der vollständige Code zur Pulsgenerierung befindet sich in Anhang A 1.

Zu Beginn (Zeile 1 bis 5) werden die wichtigsten Bibliotheken eingebunden. $NumPy$ wird für numerische Berechnungen benötigt, $Matplotlib$ für die Darstellung von Ergebnissen. Aus $SciPy$ werden Funktionen für Interpolation sowie später für Optimierungsverfahren genutzt. Zusätzlich wird das Modul $random$ verwendet, um Zufallszahlen zu erzeugen. Anschließend werden grundlegende Konstanten definiert (Zeile 7 bis 9). Mit $femto = 1e - 15$ wird eine Einheit für Femtosekunden festgelegt. Darüber hinaus wird mit $angFreqTHz = 2 * np.pi * 1e12$ eine Referenz-Winkelkreisfrequenz angegeben.

Im nächsten Schritt (Zeile 11 bis 20) werden die Zeit- und Frequenzachsen definiert, auf denen alle weiteren Berechnungen stattfinden. Dazu wird zuerst die Anzahl der Stützstellen N und das zeitliche Abtastintervall Ts festgelegt. Aus beiden ergibt sich die Gesamtdauer des Fensters. Auf dieser Basis entstehen die Zeitachse tt sowie eine zentrierte Variante ttc, bei der $t = 0$ in der Fenstermitte liegt. Für eine bessere Darstellung und um später leicht die RMS-Breite auszuwerten. Für den Frequenzbereich wird eine zur Abtastung passende Kreisfrequenz-Obergrenze ws verwendet. Daraus ergibt sich die Schrittweite $w0$ und die diskrete Frequenzachse ww.

Damit die Phase nicht überall im Spektrum verändert wird, wird ein Frequenzbereich festgelegt, in dem der „Pulsshaper“ aktiv sein darf (Zeile 22 und 23). Dieses Fenster begrenzt die Manipulation auf den Teil des Spektrums, der für den Puls auch wirklich relevant ist. Außerhalb des Fensters bleibt die Phase unverändert. Im Code liegt das Shaper-Fenster zwischen $\omega = 0$ und $ws/3$.

Als Nächstes wird ein breitbandiges Referenzspektrum für die Amplitude definiert (Zeile 25 bis 29). Damit erhält der spätere Zeitpuls eine realistische spektrale Breite, wie sie für ultrakurze Pulse nötig ist. Im Code wird das Spektrum als Summe glatter Anteile um eine Kreisfrequenz wp aufgebaut und anschließend auf 1 normiert. So bleibt die Skalierung in allen folgenden Schritten konsistent.

Als Nächstes wird die Referenzphase aufgebaut (Zeile 30 bis 43). Dazu werden zunächst eine feste Anzahl Stützstellen innerhalb des definierten Shaper-Fensters gewählt. Auf diesen Punkten wird eine Phase vorgegeben, die sich aus einem quadratischen Anteil und einem kubischen Anteil um die Mittenfrequenz zusammensetzt. Ein leicht gewichteter, gaussförmiger Zufallsterm kommt hinzu. Dieser sorgt für etwas Varianz. Anschließend werden die Phasenwerte per kubischer Spline-Interpolation auf die vollständige Frequenzachse übertragen. Mit einer Maske wird sichergestellt, dass die Phase nur innerhalb des Shaper-Fensters in die Referenzphase eingetragen wird, außerhalb bleibt sie Null. So entsteht eine gechirpte Referenzphase, die GDD/TOD berücksichtigt und als Basis für die nachfolgende Pulserzeugung dient.

Nachdem Amplitude und Phase des Referenzspektrums festgelegt sind, wird daraus der Zeitpuls berechnet (Zeile 45 bis 47). Aus Amplitude und Phase wird ein komplexes Spektrum gebildet. Die inverse FFT überführt dieses Spektrum in den Zeitbereich. Mit $fftshift$ wird der Puls so verschoben, dass $t = 0$ in der Fenstermitte liegt. Anschließend erfolgt eine Normierung auf die maximale Betragsamplitude.

Die Funktion $pulseShaper$ bildet die Wirkung eines Phasenshapers nach (Zeile 53 bis 74). Der Eingabepuls $ytin$ wird in den Frequenzbereich transformiert. Die Phase wird über Stützstellen $(wStuetz, phStuetz)$ per kubischer Spline-Interpolation auf die lokale Frequenzachse übertragen, jedoch nur bis zur oberen Grenze $wStuetz$. Außerhalb bleibt die Phase null. Anschließend wird die Phase als Faktor angewendet. Zum Schluss erfolgt die Rücktransformation in den Zeitbereich und eine Normierung auf die maximale Betragsamplitude. Die Amplitude des Spektrums wird nicht verändert.

Die Funktion $calcRMSWidth$ (Zeile 75 bis 93) berechnet die zeitliche RMS-Breite des Signals. Als Eingabe werden der komplexe Puls und die Zeitachse tt übergeben. Innerhalb der Funktion wird daraus zunächst die Intensität berechnet. Negative Werte werden abgeschnitten, anschließend werden Schwerpunkt und das zweite Moment gebildet. Daraus folgt die Varianz und die RMS-Breite.

Die Funktion $make_random_input_pulse$ (Zeile 99 bis 155) fasst die zuvor eingeführten Bausteine zu einem Generator für Trainingsdaten zusammen, welcher leicht per Funktionsaufruf aufgerufen werden kann. Zunächst wird eine spektrale Amplitude aus mehreren zufällig gesetzten Gauß-Peaks aufgebaut und auf Eins normiert. Anschließend wird eine spektrale Phase erzeugt, die sich aus zufälligen GDD-/TOD-Anteilen sowie einem über Stützstellen definierten und per Spline interpolierten Zusatzterm zusammensetzt. Die Phase wird nur innerhalb des Shaper-Fensters angewandt. Aus Amplitude und Phase entsteht ein komplexes Spektrum, das per inverser FFT in den Zeitbereich überführt und auf die maximale Betragsamplitude normiert wird. Das Ergebnis ist ein normalisierter, komplexer Zeitpuls, der sich als Eingabe für das schnelle Erzeugen von Beispielpulsen im Training des neuronalen Netzes eignet. Abbildung 3.1 zeigt einen solchen Puls.

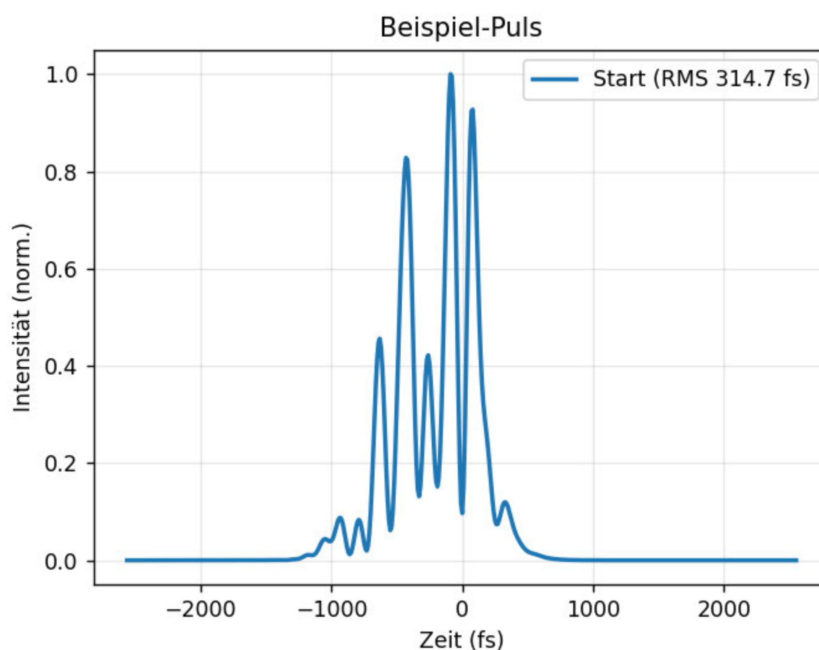


Abbildung 3.1 Beispielhafter Puls

3.2 Erzeugen des Referenz-FROG

Der vollständige Code zur FROG-Simulation ist dem Anhang A 2 zu entnehmen.

Die Datei $frogOpt.py$ ergänzt die Pulsgenerierung um Berechnung und Darstellung von FROG-Traces. Der folgende Abschnitt beschreibt den dafür benötigten Code. Viele Funktionen sind mit $pulseOpt.py$ identisch oder werden daraus aufgerufen. Im Folgenden werden Abweichungen und zusätzliche Schritte erläutert.

Am Anfang werden die Importe ausgeführt (Zeilen 1 bis 9). $NumPy$ wird für numerische Operationen verwendet und $Matplotlib$ für die Darstellung. Aus $pulseOpt$ werden Pulserzeugung, RMS-Berechnung und Hilfsfunktionen übernommen. Zusätzlich werden die Konstanten $femto$ und N (Stützstellen) importiert, damit Datengrundlage und Auflösung mit pulseOpt identisch sind.

Danach werden Hilfsfunktionen und Metriken definiert (Zeile 15 bis 33). Die Funktion $pulse_intensity$ berechnet die Pulsintensität als Funktion der Zeit. Die Funktion $pulse_metrics$ bestimmt drei Auswertungsgrößen des Pulses, das sind RMS-Breite, FWHM und Energie. Die Funktion $circshift$ erzeugt eine zeitlich verschobene Kopie des Pulses, die für die FROG-Berechnung benötigt wird.

Im Anschluss folgt der FROG-Algorithmus (Zeilen 34 bis 96). Die Funktion $create_shg_ampl$ erzeugt die komplexe SHG-Amplitude aus dem Zeitpuls $E(t)$, der aus pulseOpt stammt. Der Algorithmus bildet daraus eine Matrix A mit N Zeilen und N Spalten, wobei die Spalten die Verzögerungen τ und die Zeilen die Frequenzen ω repräsentieren. Für jede Spalte wird eine Kopie des Pulses um τ verschoben, punktweise mit $E(t)$ multipliziert und in den Frequenzbereich transformiert. Das Ergebnis wird als Spalte in A abgelegt. Dieser Ablauf wird für alle Verzögerungen wiederholt, bis alle Spalten gefüllt sind. $frog_from_pulse$ berechnet aus yt und Ts die SHG-FROG-Amplitude, bildet das Intensitätsbild auf das Maximum normiert. Zurückgegeben werden das FROG-Bild in der Größe $N \times N$ und die τ-Achse. $frog_to_obs$ normiert das FROG-Bild auf sein Maximum, skaliert die Werte in den Bereich [0, 10] und gibt je nach Bedarf einen flachen 1D-Vektor für RL-Observations oder ein 2D-Bild aus. Ein kleines eps verhindert Division durch Null. $frog_and_pulse$ bündelt die Schritte. Als Eingabe dienen yt und Ts. Die Funktion gibt mehrere Werte zurück: den FROG-Trace ($Ishg$ mit zugehöriger τ-Achse), die zeitliche Intensität des Pulses (It mit zugehöriger t-Achse) sowie die Metriken RMS und Energie.

Zum Schluss enthält der Code eine Demoausgabe (Zeilen 102 bis 136). Sie wird beim Direktstart der Datei ausgeführt. Zunächst wird ein zufälliger Zeitpuls yt erzeugt. Danach wird das FROG-Bild berechnet und es werden RMS und Energie im Terminal ausgegeben. Anschließend werden FROG-Bild und zeitliche Intensität dargestellt. Die Abbildung 3.2 zeigt ein beispielhaftes Ausgabebild.

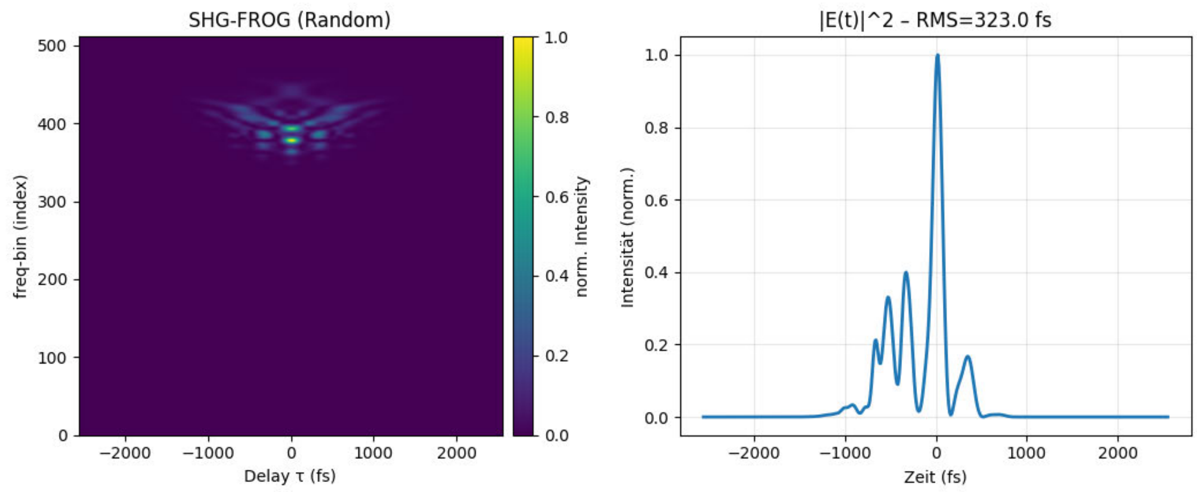


Abbildung 3.2 Beispielhafter FROG

4 Umsetzung von Reinforcement Learning

4.1 Aufbau

Die Abbildung 2.12 zeigt die grundlegende Struktur des Reinforcement-Learning-Zyklus und wurde in Unterkapitel 2.4 behandelt. Abbildung 4.1 zeigt dies übertragen für die Pulsformung mit Reinforcement Learning. Der Kreislauf verbindet die Beobachtung des Ausgangspulses mit dem Agenten, dessen Aktion als Phasenverschiebung an den Pulsshaper geht. Die Umgebung bewertet die resultierende Pulsverkürzung und liefert die Belohnung zurück.

Die Beobachtung ist ein Vektor, der sich je nach Konfiguration aus bis zu vier Teilen zusammensetzt: der auf obs_size Punkte heruntergesampelten Zeit-Intensität des Ausgangspulses, der optionalen spektralen Amplitude, der aktuellen Phasenmaske und dem RMS-Verhältnis. Diese Komponenten werden zu einem Vektor zusammengeführt und als $spaces.Box$ bereitgestellt. Alternativ lässt sich ein FROG-Trace erzeugen und zu einem Beobachtungsvektor skalieren.

Der Agent ist ein SAC. Das Trainingsskript instanziiert den Agenten und steuert Lernen, Evaluation und Checkpoints. Zentrale Hyperparameter kommen aus der Konfiguration, etwa Netzarchitektur, Entropieregulierung und Lernrate.

Die Aktion ist ein kontinuierlicher Vektor der Länge S. Intern wird er mit $action_scale$ in reale Phasenschritte abgebildet. Die Phasenwerte werden inkrementell addiert und anschließend auf $\pm\pi$ begrenzt.

Die Umgebung wendet die Phasenmaske auf das Eingangssignal an. Dazu werden die Stützstellen über den spektralen Bereich interpoliert und als Phase $\varphi(\omega)$ auf die komplexe Amplitude aufgebracht. Eine inverse Fourier-Transformation liefert das Zeitsignal $y(t)$. Aus $y(t)^2$ werden Intensität und RMS-Breite berechnet. Diese Kennzahlen fließen in die Beobachtung ein, bestimmen die Belohnung und werden für die Auswertung gespeichert.

Die Belohnung misst primär die relative Verbesserung der aktuellen RMS-Breite im Vergleich zur Start-Breite. Ausführlich wird dies in Unterkapitel 4.3 erklärt.

Jede Episode startet mit einem Reset der Phasenmaske und einer definierten Start-RMS. Eingangspulse können pro Reset randomisiert oder über mehrere Episoden gehalten werden. Das Episodenende wird durch ein Zeitlimit vorgegeben.

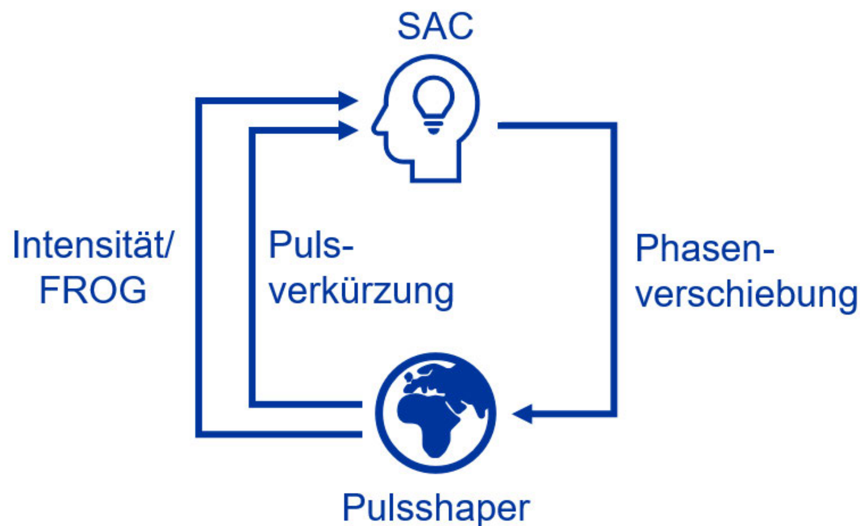


Abbildung 4.1 Pulsshaper Ablaufplan, eigene Darstellung

4.1.1 Ablauf Intensität

Die Umsetzung des Trainingsablaufs basiert auf mehreren Python-Skripten. Die Abbildung 4.2 visualisiert das Zusammenspiel dieser Komponenten. Die nachfolgenden Abschnitte erläutern die spezifischen Aufgaben der einzelnen Dateien $config.py$, $train_sac.py$, $pulseOpt.py$ und $pulse_rl_env.py$ innerhalb dieses Prozesses.

config.py

Die $config.py$ legt die Einstellungen für die Umgebung und das Training fest und bündelt die zentralen Laufzeitparameter. Sie setzt die Episodenlänge über MAX_EP_LEN und die Gesamtzahl der Trainingsschritte mit $TOTAL_STEPS$. Für Protokolle und Speicherstände legt sie $LOGDIR$ sowie die Intervalle $SAVE_EVERY$ und $EVAL_EVERY$ fest. Diese Konstanten werden im Trainingsskript direkt importiert und bestimmen dort Checkpoints und Evaluationen.

Über das ENV_KW konfiguriert die Datei die Gym-Umgebung. Hinterlegt sind die Anzahl der Phasenstützstellen S und die Länge des heruntergesampelten Intensitätsvektors obs_size. Aktionen sind inkrementell definiert und werden mit dem Faktor $action_scale$ skaliert, welcher als Quotient aus pi und MAX_EP_LEN berechnet wird. Die Phase wird durch phi_max normiert. Beim Reset sind Zufallspulse aktiv und $hold_pulse_episodes$ steuert die Dauer der Wiederverwendung eines Eingangspulses. Diese Werte werden in $PulseShaperEnv$ übergeben.

$TRAIN_KW$ liefert die Hyperparameter. Darin stehen Lernrate, Discountfaktor, Puffer- und Batch-Größe, Trainingsfrequenz, Anzahl der Gradienten-Schritte, Warm-up-

Phase, tau sowie die Netzarchitektur und die Aktivierungsfunktion. Das Trainingsskript liest die $TRAIN_KW$ ein und überschreibt damit seine Standardeinstellungen.

pulseOpt.py

Die $pulseOpt.py$ wurde in Abschnitt 3.1 bereits ausführlich erklärt.

train_sac.py

Die Datei $train_sac.py$ setzt das Training des SAC-Agents auf. Sie lädt ENV_KW und $LOGDIR$ aus der config und übernimmt $TRAIN_KW$, um Standardwerte zu überschreiben.

Die Umgebung wird zweimal erzeugt. Die erste Instanz heißt $train_env$ und dient ausschließlich dem Lernen. Die zweite Instanz heißt $eval_env$ und dient ausschließlich der regelmäßigen Auswertung. Beide Umgebungen werden identisch konfiguriert, jeweils mit $PulseShaperEnv$ und den Parametern aus ENV_KW. Die maximale Episodenlänge wird für beide Umgebungen durch $TimeLimit$ festgelegt.

Die Auswertungen laufen über den $EvalAndLogCallback$. Er greift nur auf $eval_env$ zu und beeinflusst das Training nicht. So kann der Agent lernen, während die Leistung in festen Abständen separat gemessen und protokolliert wird.

Für den Ablauf sind mehrere Callbacks aktiv. $CheckpointCallback$ speichert Modelle in festen Schrittintervallen. Der $ProgressBarCallback$ dient zur Visualisierung des Trainingsfortschritts in der Konsole. $RewardComponentsPerEpisodeCallback$ schreibt Metriken pro Episode in ein Logverzeichnis. $EvalAndLogCallback$ bewertet in regelmäßigen Abständen auf der separaten Evaluationsumgebung und protokolliert die Ergebnisse. Zum Schluss wird das trainierte Modell gespeichert.

pulse_rl_env.py

Die Pulse-Umgebung organisiert den Ablauf von Reset, Schritt, Beobachtung und Phasenmaske. Beim Reset setzt sie die Phase auf null, berechnet aus dem aktuellen Eingangspuls die Startintensität und legt die Start-RMS als Referenz ab. Ein neuer Eingangspuls wird nur in festem Abstand gezogen, damit der Agent eine Strategie erlernt, unterschiedliche Eingangspulse zu generalisieren. Die erste Rückgabe enthält die normalisierte Beobachtung und eine Info mit der Startbreite in Femtosekunden.

Im nächsten Schritt verarbeitet die Umgebung einen Vektor der Länge S als Aktion. In der Standardeinstellung werden die Werte inkrementell auf die Phasenmaske addiert,

alternativ ersetzt die Aktion die Maske vollständig. Danach wird die Phase nicht geklippt, sondern numerisch stabil auf den Bereich $\pm\pi$ abgebildet. Diese Abbildung vermeidet Sprünge an den Grenzen und hält benachbarte Phasenwerte auch numerisch nah. Anschließend wird die Maske auf den Eingangspuls angewandt und die Intensität im Zeitbereich berechnet. Daraus wird die RMS berechnet.

Der Reward wird in Abschnitt 4.3 näher erläutert. Das Episodenende setzt die Umgebung nicht selbst, die Begrenzung der Schrittlänge übernimmt $TimeLimit$ aus der $config$.

Die Beobachtung ist der Eingabevektor für das neuronale Netz. Basierend auf der Implementierung wird dieser Vektor aus mehreren Komponenten zusammengesetzt, um dem Agenten ein vollständiges Bild der aktuellen Situation zu geben. Er enthält die auf obs_size Punkte heruntergesampelte und normierte Zeit-Intensität, welche den zeitlichen Pulsverlauf abbildet. Ebenso ist eine heruntergesampelte Spektral-Amplitude des Eingangspulses enthalten. Ein weiterer Teil des Vektors ist die intern geführte Phasenmaske, die nach jedem Schritt stabil auf den Bereich von $-\pi$ bis π abgebildet. Als viertes Element hängt die Umgebung ein einzelnes Merkmal an. Es ist das Verhältnis der aktuellen RMS zur Start RMS und wird auf null bis eins begrenzt. Damit sieht das Netz in jeder Eingabe den relativen Fortschritt der Verkürzung.

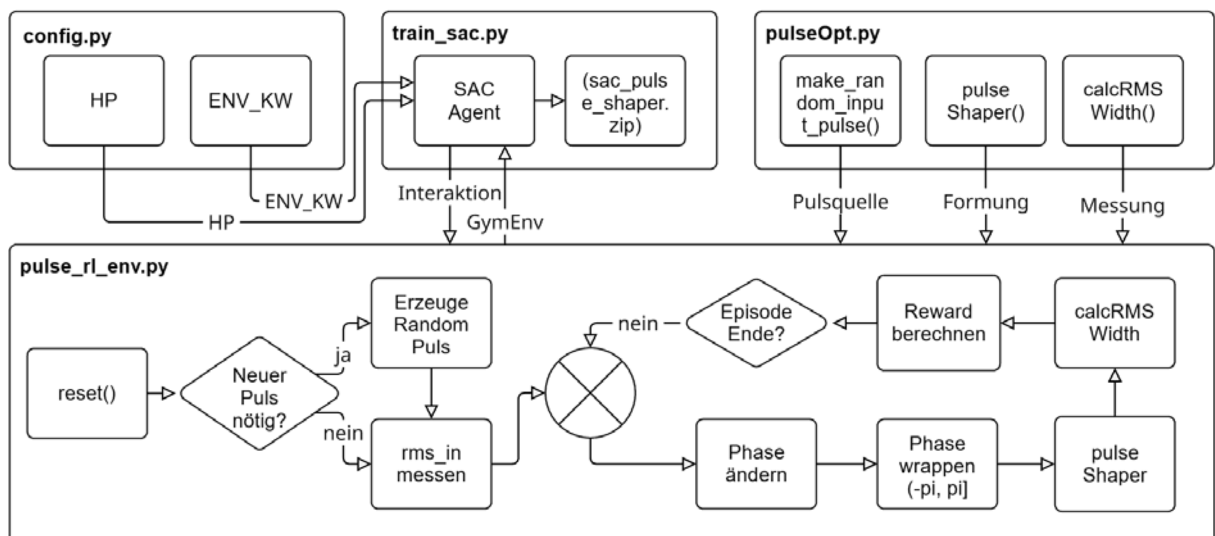


Abbildung 4.2 Flussdiagramm Training mittels Intensität

4.1.2 Ablauf FROG

Der Grundaufbau des Trainings mittels FROG ähnelt dem Vorgehen bei der Intensitätsmessung und folgt dem Schema in Abbildung 4.3. Bei diesem Ansatz dient jedoch ausschließlich ein Bild als Netzeingang. Die Umgebung berechnet pro Schritt ein FROG-Bild aus dem aktuellen Zeitfeld. Das Bild wird auf Eins normiert, auf $256x256$ skaliert und in Ganzzahlen abgelegt. Die Observation besteht nur aus diesem Bild. Weitere Vektorteile entfallen. Dadurch sieht das Netz ausschließlich die zeit-frequente Struktur des Pulses und keine zusätzlichen Kurvenmerkmale.

Der anfängliche Zeitpuls wird mithilfe der in $pulseOpt.py$ definierten Funktionen erzeugt und als normiertes Zeitfeld bereitgestellt. Aus genau diesem Feld berechnet die FROG-Umgebung ein FROG-Bild. Das Netz gibt einen Phasenvektor mit S Stützstellen zurück. Die Umgebung skaliert und wrappt diese Phase und wendet sie mit pulseShaper auf den Zeitpuls an. Aus der resultierenden Intensität wird die RMS-Breite berechnet und gegen die beim Reset gespeicherte Start-RMS ausgewertet.

Das Trainingsskript verwendet dafür eine reine Bild-Policy. Anstelle einer Multi-Input-Policy kommt ein CNN-fähiger Policy-Typ zum Einsatz. Die restliche Lernlogik bleibt gleich. Zwei identisch konfigurierte Umgebungen, externe Episodenbegrenzung und Checkpoints bleiben bestehen. Speicherparameter können angepasst werden, weil Bilddaten den Replay-Puffer stärker belasten.

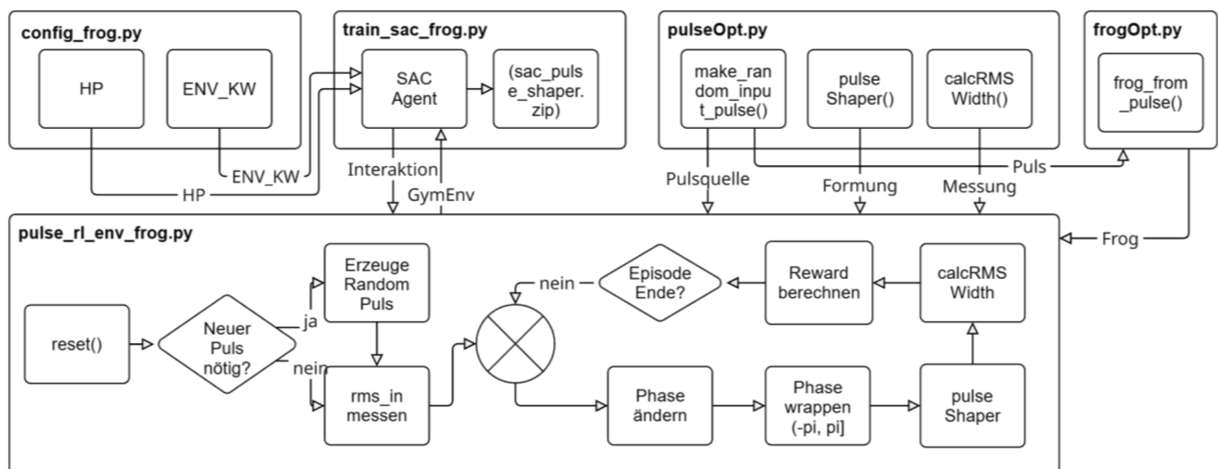


Abbildung 4.3 Flussdiagramm Training mittels FROG

4.2 Schichten

Für diese Arbeit wurden drei unterschiedlich große Netze implementiert, um den Einfluss der Modellgröße auf die Ergebnisse zu untersuchen. Die verwendete SAC-Implementierung wird aus Stable Baselines 3 importiert. Diese besteht aus einem Feature Extractor zur Merkmalsextraktion und separaten Policy- (Actor) und Value-Netzwerken (Critic).

Bei der Verarbeitung des eindimensionalen Intensitätsvektors fungiert der Feature Extractor als einfacher Durchgang. Die in der Konfiguration angegebene net_arch mit beispielsweise $(128, 256, 128)$ definiert die Architektur der nachfolgenden Hidden Layer. Diese Schichten werden von Actor und Critic gemeinsam genutzt, bevor sich das Netz in zwei separate Ausgabeköpfe aufteilt.

Für die Verarbeitung der zweidimensionalen FROG-Bilder wird ein CNN als Feature Extractor vorgeschaltet. Stable Baselines 3 nutzt die Standardarchitektur. Diese besteht aus drei Faltungsschichten mit anschließender ReLU-Aktivierungsfunktion, einer Flatten-Schicht zur Umwandlung in einen Vektor und einer vollständig verknüpften Schicht. Die Ausgabe dieses CNNs, ein Merkmalsvektor, dient als Eingabe für das nachgeschaltete Netz, die durch die net_arch definiert ist. Die Abbildung 4.4 zeigt zur Verständlichkeit den schematischen Aufbau des nachgeschalteten Netzes.

Der Satz der universellen Approximation besagt, dass ein mehrschichtiges, hinreichend großes Netz beliebige Funktionen mit beliebiger Genauigkeit approximieren kann, sofern genügend Neuronen vorhanden sind. Größere Modelle können daher komplexere Zusammenhänge erfassen und in diesem Fall Strategien erlernen, um die Pulsbreite zu reduzieren. [18]

Gleichzeitig beschreibt das Bias-Variance-Tradeoff den Zielkonflikt zwischen Modellgröße und Generalisierungsfähigkeit. Kleine Netze neigen zu Unteranpassung mit hohem Bias, große Netze zu Überanpassung mit hoher Varianz. Es ist daher ein ausgewogenes Größenmaß zu wählen. [19]

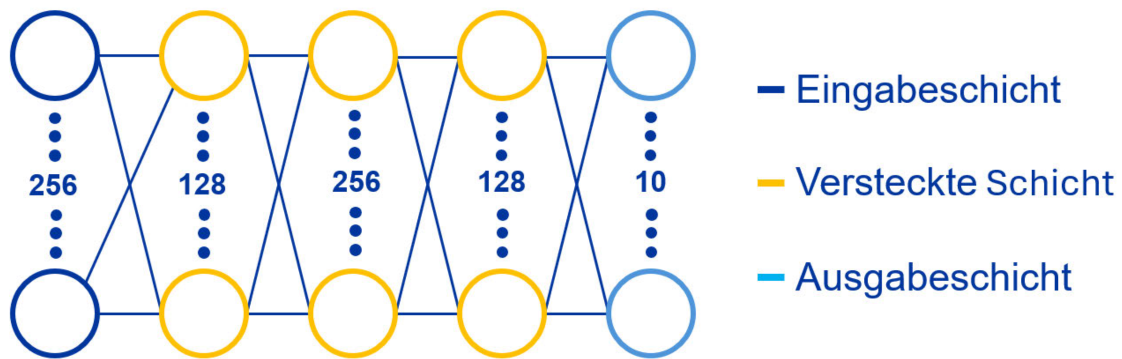


Abbildung 4.4 Kleines Netz

Als Orientierung folgt eine Beispielrechnung für die Parameterzahl des kleinen Netzes.

$$(256 \cdot 128) + 128 + (128 \cdot 256) + 256 + \ (256 \cdot 128) + 128 + (128 \cdot 10) + 10$$

$$32.896 + 33.024 + 32.896 + 1.290 = 100.106$$

Die Eingabeschicht hat 256 Werte, da der Intensitätsvektor auf 256 Punkte festgelegt wird. Die Ausgabeschicht hat 10 Neuronen, da 10 Stützstellen der Phasenmaske beispielhaft eingestellt werden. Die Schicht $256 \rightarrow 128$ hat $256 \cdot 128$ Gewichte und 128 Bias, zusammen 32.896 Parameter. Die Schicht $128 \rightarrow 256$ hat $128 \cdot 256$ Gewichte und 256 Bias, zusammen 33.024 Parameter. Die Schicht $256 \rightarrow 128$ hat erneut 32.896 Parameter. Die Ausgabeschicht $128 \rightarrow 10$ hat $128 \cdot 10$ Gewichte und 10 Bias, zusammen 1.290 Parameter. Die Gesamtsumme beträgt 100.106. Die Parameterzahlen für das mittlere Netz in dieser Beispielrechnung betragen 331.274 und 1.186.826 für das große Netz.

4.3 Belohnungsfunktion

Im Reinforcement Learning legt die Belohnungsfunktion fest, was der Agent erreichen soll. Sie weist jeder Zustands-Aktions-Kombination einen Wert zu. Positive Werte verstärken erwünschtes Verhalten, negative Werte dämpfen unerwünschtes Verhalten. Über dieses Signal werden die Netzwerkgewichte aktualisiert und die Policy weiterentwickelt. Ist die Belohnung unpassend definiert, optimiert der Agent in die falsche Richtung. Zu schwache Signale bremsen das Lernen und begünstigen Instabilitäten. Eine unpassend gewählte Belohnungsfunktion führt zum Nichterfüllen der Aufgabe.

Dichte Belohnungen liefern nach jedem Schritt Feedback und beschleunigen oft den Fortschritt, erhöhen aber das Risiko falscher Anreize. Spärliche Belohnungen melden sich erst am Episodenende. Sie spiegeln das Ziel besser, erschweren jedoch die

Kreditzuweisung über viele Schritte. Positive Belohnungen fördern eher Exploration, negative Belohnungen führen zu vorsichtigem Verhalten. Für die Aufgabe des Pulsshapers wurde drei unterschiedliche Belohnungsfunktionen getestet, welche im Folgenden erläutert werden.

Erste Belohnungsfunktion

```
1  # Variablen
2  eps = 1e-20
3
4  # RMS-Vergleich
5  reward = (self._rms_in_s - rms_s) / (self._rms_in_s + eps)
```

Abbildung 4.5 Belohnungsfunktion eins

Die erste Belohnungsfunktion aus Abbildung 4.5 misst ausschließlich die relative Verbesserung der Pulsbreite. Ein kleiner Wert $eps = 1e - 20$ verhindert Division durch Null. Positive Werte bedeuten, der Puls nach dem Pulsshaper ist kürzer als der Startpuls, negative länger. Die Normierung macht den Maßstab unabhängig von der absoluten Startbreite.

Zweite Belohnungsfunktion

```
1  # Variablen
2  eps = 1e-20
3  phi = femto * 1000 / (self._rms_in_s + eps)
4
5  # relative Verbesserung ggü. Start
6  r_ratio = 1.0 - float(rms_s / (self._rms_in_s + eps))
7
8  # Kleiner Bonus NUR bei neuem Bestwert
9  best_gain = (self._best_rms_s - rms_s) / (self._rms_in_s + eps)
10 r_best = 0.5 * float(max(best_gain, 0.0))
11
12 # Gesamt-Reward
13 reward = r_ratio * phi + r_best * phi
14
15 # State-Updates
16 if rms_s < self._best_rms_s:
17     self._best_rms_s = rms_s
18 self._prev_rms_s = rms_s
```

Abbildung 4.6 Belohnungsfunktion zwei

Die zweite Belohnungsfunktion Abbildung 4.6 orientiert sich primär am Startzustand. Der Term r_ratio misst die relative Verbesserung der aktuellen Pulsbreite gegenüber der Startbreite. Positive Werte bedeuten kürzer als zu Beginn, negative länger. Zusätzlich gibt es einen Bestwert-Bonus. Nur wenn rms_s den bisherigen Bestwert $best_rms_s$ unterbietet, entsteht ein Zusatz r_best. Das belohnt Fortschritte. Außerdem wird eine skalierende Normierung phi ergänzt. phi skaliert die Beiträge relativ zur Startbreite und macht den Reward zwischen Episoden mit unterschiedlich großen Startpulsen vergleichbarer. Die Gesamt-Belohnung lautet $reward = r_{ratio} * phi + r_{best} * phi$. Nach jedem Schritt werden die Zustände aktualisiert. Ist rms_s besser als $best_rms_s$, wird der Bestwert ersetzt und $prev_rms_s$ wird auf den aktuellen Wert gesetzt.

Dritte Belohnungsfunktion

```
1  # RMS-Vergleich
2  reward = 1 / rms_s
```

Abbildung 4.7 Belohnungsfunktion drei

Die in Abbildung 4.7 verwendete Belohnungsfunktion bewertet die Pulsqualität über die absolute RMS-Breite des Ausgangspulses. Der Reward ist der Kehrwert der RMS-Breite. Kürzere Pulse erhöhen den Wert. Im Unterschied zur normierten Differenz erfasst sie keine relative Verbesserung gegenüber dem Eingang, sondern minimiert die Breite absolut. Damit entfällt die Abhängigkeit von der Startbreite.

4.4 Hyperparameter

Hyperparameter sind Einstellungen, die vor dem Training festgelegt werden und den Lernverlauf steuern. Im Unterschied zu Modellparametern, die während des Trainings aus Daten gelernt werden, bleiben Hyperparameter unverändert. Die systematische Anpassung dieser Größen heißt Hyperparametertuning und dient dem gezielten Modelltraining. Im Folgenden werden die im Modell verwendeten Hyperparameter aufgeführt und ihr Nutzen für das Trainingsverhalten erläutert.

Die Lernrate steuert die Größe der Gewichtsaktualisierung pro Optimierungsschritt. Übliche Werte liegen zwischen $1e - 5$ und $1e - 3$. Eine hohe Lernrate beschleunigt das Lernen, kann aber zu Überschreiten des Optimums, Oszillation und Divergenz führen. Eine zu niedrige Lernrate bewirkt sehr langsame Verbesserungen, begünstigt das Verharren in lokalen Minima und erfordert viele Epochen. Optimierer wie RMSprop

und Adam passen die effektive Schrittweite dynamisch an und mildern diese Effekte, die Wahl einer geeigneten Grund-Lernrate bleibt jedoch entscheidend. [19]

Gamma (γ) ist der Diskontfaktor und bestimmt das Verhältnis von zukünftigen zu unmittelbaren Belohnungen. Kleine Werte fokussieren kurzfristige Erträge, Werte nahe 1 betonen langfristige Erträge. Die theoretischen Grundlagen wurden in Abschnitt 2.4.3 behandelt.

Der Replay-Buffer speichert vergangene Erfahrungen des Agenten, also Zustand, Aktion, Belohnung und Folgezustand. So lernt der Agent nicht nur aus den letzten Schritten, sondern aus einem breiten Spektrum an Situationen. Die Größe $buffer_size$ legt fest, wie viele Erfahrungen vorgehalten werden. Ein großer Buffer erhöht die Vielfalt und senkt Overfitting, kann aber veraltete Daten sammeln und die Anpassung verlangsamen. Ein kleiner Puffer ist näher an der aktuellen Policy und reagiert schneller, bietet jedoch weniger Diversität. [20]

Die $batch_size$ legt fest, wie viele Trainingsbeispiele in einem Schritt zur Aktualisierung des neuronalen Netzes verwendet werden. Im RL entspricht sie der Anzahl an Erfahrungen, die aus dem Replay-Buffer gezogen werden. Das Modell wird nicht nach einer einzelnen Erfahrung, sondern nach einem Bündel von Erfahrungen aktualisiert. Kleinere Batches führen zu häufigeren Updates mit kleineren Schritten. Übliche Werte liegen zwischen 32 und 256. [19]

Der Parameter ent_coef ist der Entropiekoeffizient und gewichtet den Entropieterm der Policy. Entropie misst die Zufälligkeit der Aktionswahl. Ein höherer Wert erhöht die Stochastik und fördert Exploration. Ein zu niedriger Wert macht die Policy zu deterministisch und begünstigt verfrühte Ausbeutung. Der Parameter $target_entropy$ legt die gewünschte Entropie der Policy fest. Liegt die beobachtete Entropie unter dem Ziel, wird ent_coef angehoben. Liegt sie darüber, wird ent_coef gesenkt. In Verfahren wie SAC wird ent_coef oft automatisch so angepasst, dass die Policy-Entropie gegen $target_entropy$ konvergiert. [21]

τ ist der Soft-Update-Parameter des Zielnetzwerks in SAC. Statt die Zielnetz-Gewichte komplett durch die des Hauptnetzwerks zu ersetzen, werden sie schrittweise angenähert. Pro Schritt gilt $\theta_target \leftarrow (1-\tau) \cdot \theta_target + \tau \cdot \theta_online$. Kleine τ-Werte stabilisieren das Training, große τ-Werte lassen das Zielnetz schneller folgen und erhöhen die Varianz. [22]

5 Trainingsablauf

Der folgende, mehrstufige Trainingsablauf wird zur Optimierung der Konfiguration für den Zeitpuls verwendet. Für das FROG-Setup wird hingegen direkt ein Langlauf mit den besten ermittelten Einstellungen durchgeführt.

Zunächst wird ein Baseline-Szenario mit festem Eingangspuls definiert. Die Umgebung behält den Puls über alle Episoden unverändert. Die Stützstellen werden auf 10 gestellt. Das Netz ist zunächst klein (siehe 4.2). Die restlichen Einstellungen bleiben auf dem Standard. Jeder Lauf umfasst einhunderttausend Schritte. Ziel dieser Phase ist es, die grundsätzliche Lernfähigkeit des Modells in einem einfachen Szenario zu validieren und eine saubere Referenz für die folgenden, komplexeren Versuche zu schaffen.

Im zweiten Schritt wird der Puls nach jeder Episode gewechselt. Die Phasenauflösung wird variiert. Es werden 10, 25, 50 und 100 Stützstellen getestet. Das Netz bleibt klein und die restlichen Einstellungen bleiben auf dem Standard. Auch hier werden einhunderttausend Schritte trainiert.

Im dritten Schritt werden die Hyperparameter gezielt abgestimmt. Variiert werden Lernrate, Replay-Puffer, Entropieeinstellung und Discountfaktor. Pro Konfiguration wird ein kurzer Lauf mit konstantem Budget durchgeführt. Gewählt wird die Variante mit dem besten Lernverlauf unter gleichen Bedingungen.

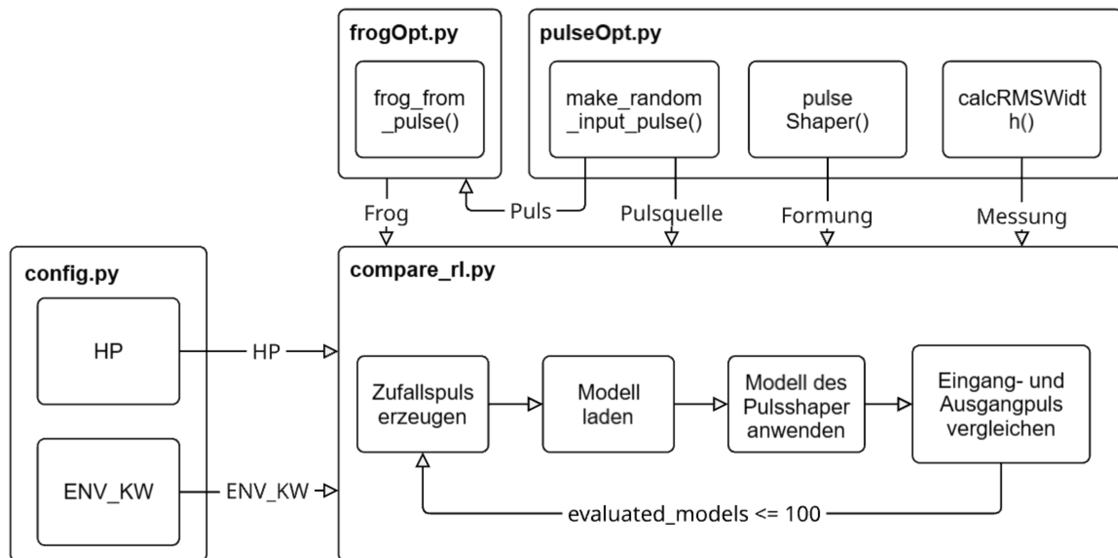
Im vierten Schritt werden unterschiedliche Reward-Varianten geprüft. Es stehen drei Formulierungen zur Auswahl. Alle übrigen Einstellungen bleiben identisch mit der besten Konfiguration aus dem vorherigen Schritt. Das Ziel ist es, die Reward-Form zu wählen, die zügige und monotone Verkürzung der Pulsbreite begünstigt und zugleich stabil bleibt.

Im fünften Schritt wird die Netzgröße skaliert. Es werden drei Architekturen getestet (siehe 4.2). Hierfür wird die Anzahl der Trainingsschritte auf 250.000 gehoben. Alle übrigen Parameter bleiben auf dem zuvor ermittelten Optimum. Aus dieser Matrix wird die Kombination aus Netzgröße und Budget ausgewählt, die das beste Verhältnis aus Leistung, Stabilität und Rechenaufwand liefert.

Zum Abschluss wird ein Langlauf mit der besten Gesamtkonfiguration durchgeführt. Das Netz wird über eine Million Schritte trainiert.

Aufbauend auf der so ermittelten, optimalen Konfiguration schließen sich weiterführende Analysen an. Zuerst werden die vom Agenten erzeugten Pulsprofile untersucht. Im Anschluss wird analysiert, wie das Modell die Pulskompression erreicht, wenn gezielt entweder die GDD oder die TOD deaktiviert wird.

Abschließend wird die anfängliche Wahl des SAC validiert. Hierfür wird seine Leistung unter den optimierten Bedingungen direkt mit den Ergebnissen der alternativen Algorithmen TD3 und PPO verglichen.



5.1 Flussdiagramm Auswertung

Die Datei $compare_rl.py$ wertet trainierte SAC-Modelle aus. Zu Beginn erzeugt sie für einen zufälligen Eingangspuls einen Referenzzustand, indem die Phase auf null gesetzt und die resultierende RMS-Breite als Ausgangswert gemessen wird. Dann lädt sie das Modell, passt die Umgebung an die Aktionsgröße des Modells an und führt einen deterministischen Rollout bis zum Episodenende. Anschließend werden Intensität und Phase aus der Umgebung geholt, die RMS-Breiten von Referenz und Ergebnis berechnet und ein Plot mit Intensität über der Zeit und Phase über der Frequenz gespeichert. Das Skript trainiert nichts und ändert keine Gewichte.

Im zweiten Schritt prüft das Skript die Leistung über hundert unterschiedliche Startpulse. Für jeden Seed wird zuerst die RMS des Eingangspulses bei Nullphase gemessen. Anschließend wird für denselben Eingangspuls das trainierte Modell angewendet und die RMS-Breite des resultierenden Ausgangspulses bestimmt. Am Ende gibt das Skript den Mittelwert der Eingangsbreiten aus sowie den Mittelwert der Ausgangsbreiten und das Verhältnis beider Größen. Zusätzlich meldet es die durchschnittliche absolute und prozentuale Verkürzung.

6 Training und Auswertung

In diesem Kapitel werden die optimalen Parameter des neuronalen Netzes ermittelt und der Ablauf folgt dem in Kapitel 5 beschriebenen Schema. Zu Beginn läuft ein kleines Netz mit den Schichtgrößen 128, 256 und 128. Die Lernrate beträgt 3e-4 und der Diskontfaktor gamma liegt bei 0,99. Der Replay-Puffer umfasst 5.000 Übergänge und die Batch-Größe beträgt 128. Der Entropieterm wird automatisch eingestellt. Pro Interaktion mit der Umgebung wird genau ein Trainingsschritt (ein Gradienten-Update) durchgeführt. Das Lernen beginnt nach 5.000 Schritten. Der Polyak-Faktor tau ist 0,005. Als Aktivierungsfunktion kommt ReLU zum Einsatz. Das Training läuft über CUDA, dementsprechend auf der GPU. Diese Startkonfiguration dient als Referenz für alle folgenden Varianten.

6.1 Intensität

6.1.1 Fester Eingangspuls

Zunächst wird ein fester Eingangspuls getestet, um zu prüfen, ob das Netz ihn verkürzen kann. Die Abbildung 6.1 zeigt den Verlauf der Ratio $R = RMS_{IN}/RMS_{OUT}$ über einhunderttausend Trainingsschritte. Nach fünftausend Schritten liegt R bei 0,992 und damit ohne Verkürzung. Danach steigt R rasch an und erreicht bei fünfundzwanzigtausend Schritten den Spitzenwert 1,78. Im Anschluss sinkt R langsam und stabilisiert sich zwischen dreißigtausend und neunzigtausend Schritten im Bereich von etwa 1,64 bis 1,70. Am Ende liegt R bei 1,654. Das Muster zeigt eine frühe Lernphase mit schnellem Gewinn und eine anschließende Plateauphase ohne nachhaltige weitere Verbesserung.

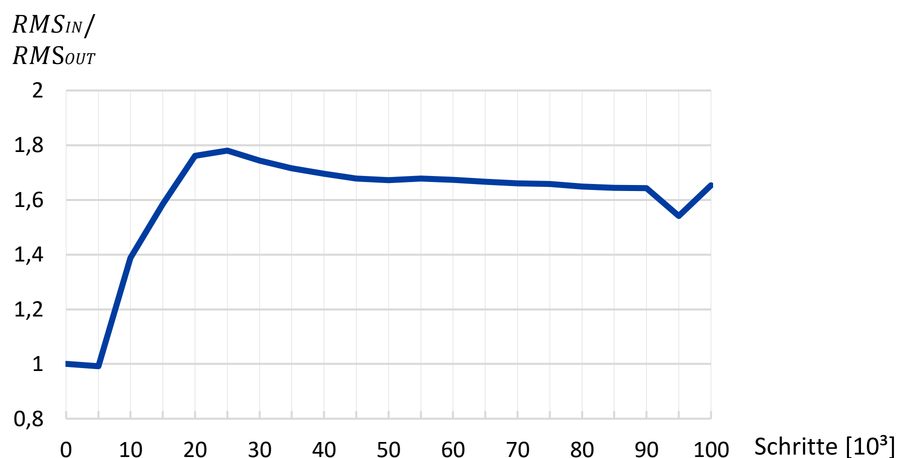


Abbildung 6.1 Entwicklung der Ratio über 100.000 Trainingsschritte, fester Eingangspuls

Die frühe Spitze lässt sich durch starke Exploration erklären. Bei SAC mit automatischer Entropie ist die Politik zu Beginn bewusst stochastisch und probiert weite Bereiche des Aktionsraums aus. Zusammen mit der groben Aktionsskalierung und der geringen Zahl von zehn Stützstellen entstehen große, wirksame Schritte, die weit weg vom Optimum schnelle Verbesserungen ermöglichen. Das spätere Absinken und das Plateau deuten auf eine Verengung der Politik hin. Die Entropie nimmt im Verlauf ab, die Politik wird deterministischer. Zusätzlich verdrängen neuere, leicht schlechtere Erfahrungen ältere gute Beispiele aus dem Replay-Buffer und der Kritiker gewichtet diese stärker. In der Nähe des Optimums wirken die Aktionsschritte zudem zu grob, wodurch kleine Verschlechterungen verbleibende Verbesserungen überdecken. Diese Punkte werden in den folgenden Abschnitten gezielt überprüft.

6.1.2 Pulswechsel

Im zweiten Versuch wird der Eingangspuls nach jeder Episode neu erzeugt. Die Abbildung 6.2 bündelt die Verläufe der Ratio für zehn, fünfundzwanzig, fünfzig und einhundert Stützstellen. Die Bewertung orientiert sich an den gleichen Kriterien wie im ersten Test und vergleicht jeweils RMS vor und nach dem Shaper.

Für zehn Stützstellen zeigt R durchgängig Werte oberhalb von eins. Nach einem frühen Anstieg bis etwa Schritt 15.000 mit einem Maximum von rund 1,33 pendelt sich die Ratio auf einem erhöhten Niveau ein. Zwischen 70.000 und 90.000 bleibt R überwiegend bei rund 1,20. Am Ende liegt R bei 1,13. Das Netz verkürzt den jeweils neuen Puls damit zuverlässig.

Für 25 Stützstellen steigt R zunächst rasch an und erreicht früh einen Spitzenwert von etwa 1,30. Im weiteren Verlauf verliert das Netz an Leistung. Ab etwa 60.000 fällt R teils deutlich unter eins und erreicht ein Minimum von etwa 0,70. Gegen Ende erholt sich die Ratio nicht und endet bei 0,87. Die Verkürzung ist damit nicht mehr stabil. Das Verhalten deutet auf eine zunehmende Unsicherheit bei wechselnden Eingangspulsen hin.

Für 50 Stützstellen liegen die Werte überwiegend unter eins. Nach einem frühen Rückgang bewegt sich R lange zwischen 0,70 und 0,95 und endet bei 0,79. Das Netz tendiert damit zur Verbreiterung des Pulses.

Für 100 Stützstellen bricht R früh stark ein und bleibt über das gesamte Training meist deutlich unter eins. Die Ratio fällt bis auf etwa 0,37 und schließt bei 0,52. Das Netz generalisiert bei wechselnden Pulsen nicht und verschlechtert den Puls.

In der Gesamtsicht zeigt die Variation des Eingangspulses eine klare Abhängigkeit von der Anzahl an Stützstellen. Mit 10 Stützstellen gelingt eine robuste Verkürzung über viele Episoden. Mit wachsender Stützstellenzahl nimmt die Stabilität ab und die Leistung kippt in Richtung Verbreiterung. Sinnvoll erscheint eine Anpassung der Hyperparameter, um das Training auch mit mehreren Stützstellen zu stabilisieren.

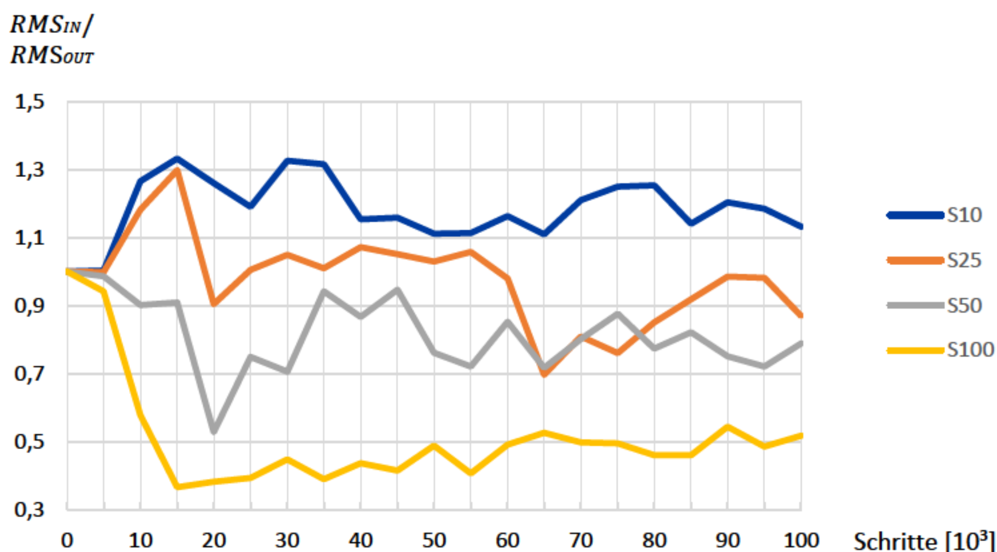


Abbildung 6.2 Entwicklung der Ratio über 100.000 Trainingsschritte, unterschiedlicher Puls

Die Tabelle 3 fasst Mittelwerte über 100 feste Eingangspulse zusammen. Pro Puls werden der RMS nach dem Shaper, die relative Verkürzung in Prozent und die absolute Änderung in Femtosekunden ausgewertet. Da in allen Versuchen dieselben hundert Eingänge verwendet werden, ist der mittlere RMS des Eingangs konstant mit 372,71 fs und wird nicht gesondert aufgeführt.

Die Testergebnisse spiegeln das Training wider. Bei 10 Stützstellen bleibt R im Training über weite Strecken oberhalb von eins und die Tabelle bestätigt eine mittlere Verkürzung um 16,6 % auf 310,79 fs. Bei 25 Stützstellen zeigt der Trainingsverlauf nur einen kurzen Anstieg und fällt danach unter eins, entsprechend ergibt sich im Mittel keine Verbesserung mit minus 0,5 %. Bei 50 Stützstellen liegt R überwiegend unter eins und die Auswertung weist eine deutliche Verbreiterung um 158,3 % mit 962,84 fs aus. Bei 100 Stützstellen bricht R früh ein und bleibt niedrig, konsistent dazu wächst der mittlere RMS stark auf 1433,59 fs mit minus 284,6 %.

Tabelle 3 Auswertung von 100 Testpulsen, unterschiedlicher Puls

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
10	310,79	16,6	61,93
25	374,44	-0,5	-1,72
50	962,84	-158,3	-590,12
100	1433,59	-284,6	-1060,88

6.1.3 Hyperparameter Tuning

In den folgenden Abschnitten werden zentrale Hyperparameter systematisch untersucht. Variiert werden gamma, buffer_size, das Entropieziel und die Episodenlänge. Die übrige Konfiguration bleibt konstant. Die Bewertung erfolgt einheitlich über das Verhältnis der RMS-Breite vor und nach dem Shaper sowie über die resultierenden Mittelwerte über feste Testpulse.

gamma

Die folgenden Ergebnisse wurden mit gamma gleich 0,7 erzielt und gegen eine Baseline mit 0,99 verglichen. Der Wert 0,7 erwies sich in mehreren Vorversuchen als konsistent beste Einstellung.

Die neuen Trainingsläufe mit einem angepassten gamma zeigen eine deutliche Verbesserung. Bei 10 Stützstellen bleibt R immer oberhalb von 1 und endet bei 1,40. Damit hält das Netz die zuverlässige Verkürzung. Bei 25 Stützstellen steigt R nun kontinuierlich an, überschreitet ab 30.000 deutlich die Marke von 1,5, erreicht Maximalwerte über 2,18 und schließt stabil bei 2,10. Die Verkürzung ist hier dauerhaft und ausgeprägt. Bei 50 Stützstellen kippt das Verhalten gegenüber dem vorherigen Versuch in den positiven Bereich. Nach anfänglicher Schwäche steigt R ab 40.000 wiederholt über 1 und endet bei 1,43. Das Netz verkürzt damit auch bei höherer Auflösung zuverlässig. Bei 100 Stützstellen bleibt R zwar unter 1, erholt sich leicht und schließt bei 0,72.

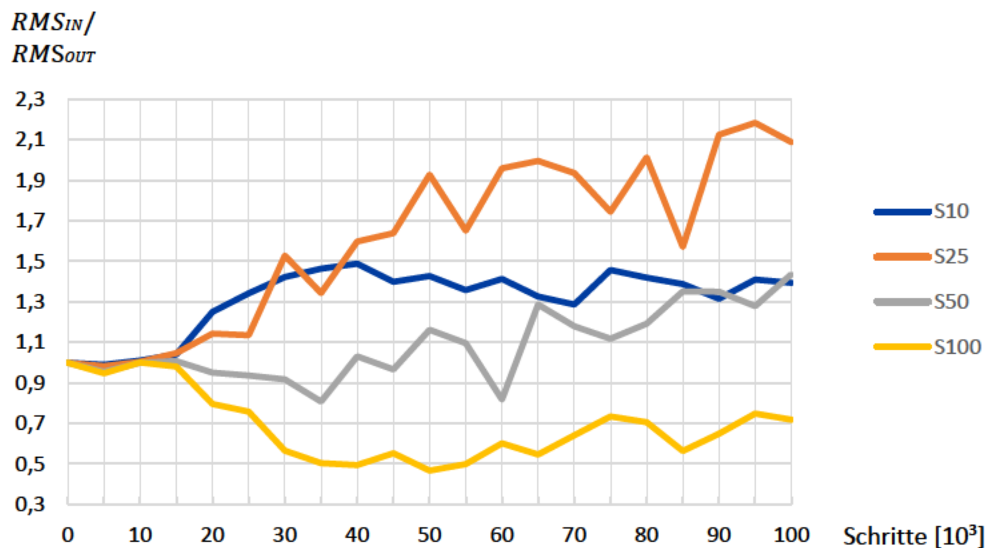


Abbildung 6.3 Entwicklung der Ratio über 100.000 Trainingsschritte, gamma = 0,7

Die Tabelle 4 mit angepasstem gamma bestätigt die Lernkurven und zeigt durchgängig bessere Mittelwerte. Bei 10 Stützstellen sinkt der Ausgangs-RMS von 310,79 fs auf 281,04 fs. Die relative Verkürzung steigt von 16,6 % auf 24,6 % und die absolute Verkürzung wächst von 61,93 fs auf 91,68 fs. Bei 25 Stützstellen dreht sich das Ergebnis deutlich ins Positive. Der Ausgangs-RMS sinkt von 374,44 fs auf 211,37 fs. Die relative Verkürzung steigt von −0,5 % auf 43,3 % und die absolute Verkürzung von −1,72 fs auf 161,35 fs. Bei 50 Stützstellen wechselt das Verhalten von einer starken Verbreiterung zu einer stabilen Verkürzung. Der Ausgangs-RMS sinkt von 962,84 fs auf 291,01 fs. Die relative Verkürzung steigt von −158,3 % auf 21,9 % und die absolute Verkürzung von −590,12 fs auf 81,70 fs. Bei 100 Stützstellen bleibt eine Verbreiterung bestehen, fällt jedoch deutlich geringer aus. Der Ausgangs-RMS sinkt von 1433,59 fs auf 576,56 fs. Die relative Änderung verbessert sich von −284,6 % auf −54,7 % und die absolute von −1060,88 fs auf −203,85 fs. Es ist jedoch noch stark negativ. Das angepasste gamma wird beibehalten.

Pro Schritt erfolgt eine kleine Anpassung der Phasenmaske und es wird eine Belohnung erfasst, bis das Episodenende nach 20 Schritten erreicht ist. Gamma steuert als Diskontfaktor das Verhältnis von zukünftigen zu aktuellen Belohnungen und damit die Empfindlichkeit des Lernens gegenüber Veränderungen der Eingänge (siehe Formel 2.9). Ein kleinerer Wert für Gamma macht den Agenten kurzsichtiger. Anstatt zu versuchen, die Belohnung über die gesamte Episodenlänge von 20 Schritten zu maximieren, konzentriert er sich stärker auf die unmittelbaren Verbesserungen in den nächsten wenigen Schritten. Dies vereinfacht das Problem führt in diesem Anwendungsfall, bei

dem sich der Eingangspuls pro Episode ändert, zu einem stabileren und effektiveren Lernprozess.

Tabelle 4 Auswertung von 100 Testpulsen, gamma = 0,7

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
10	281,04	24,6	91,68
25	211,37	43,3	161,35
50	291,01	21,9	81,7
100	576,56	-54,7	-203,85

buffer_size

Im Anschluss an die Gamma-Anpassung wird die Puffergröße untersucht. Ab hier werden keine Trainingskurven mehr gezeigt, sondern ausschließlich die Auswertung der 100 Testpulse in der Tabelle. Die Anpassung der buffer_size zielt darauf ab, die Stabilität des Lernens zu erhöhen, indem das Verhältnis von alten zu neuen Erfahrungen im Replay-Buffer verändert wird.

Die Ergebnisse zeigen Zugewinne gegenüber der reinen gamma-Anpassung für 10, 25 und 50 Stützstellen. Bei 10 Stützstellen sinkt der Ausgangs-RMS leicht. Es kam zu einer Verkürzung von 25,0 %, das sind 0,4 % mehr im Vergleich zu vorher. Für 25 Stützstellen kam es ebenfalls zu einer zusätzlichen Verkürzung. Erreicht wurden 46,9 %, das sind plus 3,6 Prozentpunkte gegenüber 43,3 %. Für 50 Stützstellen kam es zu einer klaren zusätzlichen Verkürzung. Erreicht wurden 35,9 %, das sind plus 14,0 % gegenüber 21,9 %. Bei 100 Stützstellen verschlechtert sich das Ergebnis jedoch. Der Ausgangs-RMS steigt im Vergleich zur reinen gamma-Anpassung. Die relative Verkürzung fällt auf -71,3 %, was einer Verschlechterung um 16,6 Prozentpunkte entspricht.

Damit stabilisiert eine größere buffer_size die Verkürzung für 10, 25 und 50 Stützstellen spürbar. Für 100 Stützstellen bleibt das Ergebnis negativ und reagiert empfindlich auf die höhere Aktionsauflösung. Als nächster Schritt bietet sich eine Anpassung des Entropieziels an, um die Exploration feiner zu steuern und eventuell die Robustheit bei 100 Stützstellen zu erhöhen. Die angepasste buffer_size wird beibehalten.

Tabelle 5 Auswertung von 100 Testpulsen, buffer_size = 50.000

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
10	279,60	25,0	93,11
25	197,84	46,9	174,87
50	238,90	35,9	133,81
100	638,35	-71,3	-265,64

Entropie

Im Anschluss an die Anpassung der buffer_size wird die Entropie untersucht. Die Entropie steuert die Exploration. Standard ist ein automatisches Entropieziel nahe −s mit s als Aktionsdimension, also der Anzahl der Stützstellen. In diesen Läufen wurde das Ziel auf $-0{,}5 \cdot s$ gesetzt, um die Streuung der Aktionen zu reduzieren.

Mit 10 Stützstellen ergibt sich keine Änderung und die Verkürzung bleibt bei 25,0 %. Mit 25 Stützstellen zeigt sich eine leichte Verschlechterung um 2,7 Prozentpunkte auf 44,2 %. Mit 50 Stützstellen verbessert sich die Verkürzung um 2,0 Prozentpunkte auf 37,9 %. Mit 100 Stützstellen stabilisiert das Training nicht und der Puls wird weiter verbreitert auf −80,8 %.

Da die Anpassung bei 10 Stützstellen keine Verbesserung bewirkt, die Ergebnisse bei 25 Stützstellen verschlechtert, bei 50 nur geringfügig verbessert und bei 100 Stützstellen keine Abhilfe schafft, wird sie verworfen. Für die weiteren Experimente werden die bisherigen Einstellungen mit gamma und angepasster buffer_size beibehalten.

*Tabelle 6 Auswertung von 100 Testpulsen, entropy = -0,5*S*

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
10	279,62	25,0	93,10
25	208,04	44,2	164,67
50	231,63	37,9	141,08
100	673,78	-80,8	-301,06

Episodenlänge

Im Anschluss an die Entropie wird die Episodenlänge untersucht. Anstelle von 20 Schritten pro Episode werden 40 Schritte verwendet. Die Episodenlänge legt fest, wie viele Schritte pro Episode ausgeführt werden und wie lange die Phasenmaske innerhalb einer Episode angepasst wird, bevor ein Reset erfolgt. Eine längere Episode bedeutet mehr aufeinanderfolgende Aktionen auf denselben Eingangspuls und verschiebt das Lernsignal stärker in die Zukunft.

In den vorliegenden Läufen mit 40 Schritten kommt es durchweg zu längeren Pulsen im Vergleich zur Einstellung mit 20 Schritten. Die Verlängerung auf 40 Schritte bringt somit keine Vorteile. Die Leistungsfähigkeit bleibt bei 10 Stützstellen unverändert, während sie bei 25 und 50 Stützstellen deutlich abnimmt und sich bei 100 Stützstellen stark verschlechtert. Die Episodenlänge wird wieder auf 20 gesetzt. Die zuvor gewählten Einstellungen mit gamma 0,7 und angepasster buffer_size werden beibehalten.

Tabelle 7 Auswertung von 100 Testpulsen, Episodenlänge = 40

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
10	280,21	24,8	92,51
25	240,21	35,6	132,50
50	283,71	23,9	89,01
100	794,39	-113,10	-421,67

6.1.4 Belohnungsfunktion

Die bisherigen Ergebnisse basieren auf der ersten Belohnungsfunktion (siehe Abschnitt 4.3). Im nächsten Schritt werden die beiden alternativen Belohnungsfunktionen untersucht. Die Bewertung erfolgt wie zuvor konsistent über das Verhältnis der RMS-Breite vor und nach dem Shaper. Für den direkten Vergleich wird die Tabelle 5 herangezogen, die die Auswertung der Testpulse zusammenfasst.

Zweite Belohnungsfunktion

Bei 10 Stützstellen ergibt sich eine marginale Verbesserung der relativen Verkürzung auf 25,1 %. Dies entspricht einer Steigerung von 0,1 Prozentpunkten. Eine deutliche Steigerung wird bei 25 Stützstellen erzielt. Hier verbessert sich die Verkürzung um 4,4 Prozentpunkte auf 51,3 %. Auch für 50 Stützstellen wird eine Verbesserung erreicht. Die relative Verkürzung steigt um 2,1 Prozentpunkte auf 38,0 %. Das Ergebnis bei 100 Stützstellen verschlechtert sich hingegen. Die Pulsverbreiterung nimmt zu und die relative Verkürzung fällt auf -81,5 %.

Die zweite Belohnungsfunktion führt somit zu besseren Ergebnissen für 25 und 50 Stützstellen, löst jedoch nicht das Problem bei hoher Auflösung. Das Verhalten bei 100 Stützstellen wird sogar noch instabiler.

Tabelle 8 Auswertung von 100 Testpulsen, zweite Belohnungsfunktion

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
10	279,00	25,1	93,71
25	181,69	51,3	191,02
50	231,17	38,0	141,54
100	676,38	-81,5	-303,66

Dritte Belohnungsfunktion

Für 10 Stützstellen bleibt die relative Verkürzung mit 25,0 % unverändert. Bei 25 Stützstellen wird eine Verbesserung auf 47,9 % erreicht. Dieses Ergebnis übertrifft die erste Funktion, bleibt jedoch hinter der Leistung der zweiten zurück. Eine ähnliche Tendenz zeigt sich bei 50 Stützstellen, wo die Verkürzung auf 36,9 % steigt. Auch hier wird die erste Funktion übertroffen, die zweite aber nicht erreicht. Bei 100 Stützstellen fällt die Pulsverbreiterung mit -77,5 % geringer aus als bei der zweiten Funktion, das Ergebnis ist jedoch schlechter als bei der ersten.

Im direkten Vergleich der drei Ansätze bietet die zweite Belohnungsfunktion die besten Ergebnisse, insbesondere bei 25 und 50 Stützstellen. Obwohl keine der Funktionen

das Problem bei 100 Stützstellen löst, zeigt die zweite Funktion das größte Potenzial zur Pulsverkürzung. Sie wird daher für die weiteren Experimente verwendet.

Tabelle 9 Auswertung von 100 Testpulsen, dritte Belohnungsfunktion

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
10	279,41	25,0	93,30
25	194,14	47,9	178,58
50	235,01	36,9	137,70
100	661,52	-77,5	-288,81

6.1.5 Anpassung der Netzgröße

Anknüpfend an die Auswahl der Belohnungsfunktion wird der Einfluss der Netzgröße untersucht. Die Untersuchung beschränkt sich auf 25 und 50 Stützstellen. 100 Stützstellen werden nicht weiterverfolgt, da sie sich in den bisherigen Experimenten als nicht stabilisierbar erwiesen haben. 10 Stützstellen werden ebenfalls ausgeschlossen, da hier auch mit einem größeren Netz keine weitere Leistungssteigerung zu erwarten ist. Zusätzlich wurde die Anzahl der Trainingsschritte auf 250.000 erhöht, um auch die großen Netze vollständig zu trainieren.

Bei der Konfiguration mit 25 Stützstellen erzielt das mittlere Netz mit einer relativen Verkürzung von 54,2 % das beste Ergebnis. Das kleine und das große Netz erreichen mit 53,0 % und 53,7 % geringfügig niedrigere Werte. Die Leistungsunterschiede zwischen den Netzgrößen sind hier minimal.

Ein ähnliches Bild zeigt sich bei 50 Stützstellen. Hier schneidet das kleine Netz mit einer Verkürzung von 48,1 % am besten ab, dicht gefolgt vom großen Netz mit 47,8 % und dem mittleren Netz mit 47,3 %. Auch in diesem Fall liegen die Ergebnisse der drei Netzarchitekturen sehr nah beieinander.

Obwohl das mittlere und das kleine Netz die jeweiligen besten Ergebnisse erzielen, sind die Leistungsunterschiede sehr klein. Es wird angenommen, dass das große Netz durch die höhere Parameteranzahl eine größere Kapazität besitzt, um komplexere Zusammenhänge zu erlernen. Dieses Potenzial zur Generalisierung und Robustheit wird

für einen langen Trainingslauf als vorteilhaft erachtet. Daher wird für das nachfolgende Langzeittraining das große Netz verwendet.

Tabelle 10 Auswertung von 100 Testpulsen, unterschiedliche Netzgrößen

Stützstellen	Netzgröße	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]
25	klein	175,05	53,0
25	mittel	170,81	54,2
25	groß	172,56	53,7
50	klein	193,62	48,1
50	mittel	196,31	47,3
50	groß	194,70	47,8

6.1.6 Langlauf

Abschließend wurde der Langzeitlauf mit dem großen Netz und einer Trainingsdauer von einer Million Schritten durchgeführt. Die Ergebnisse bestätigen die Annahme, dass eine längere Trainingszeit zu einer weiteren Verbesserung der Pulsverkürzung führt.

Für die Konfiguration mit 25 Stützstellen wurde die relative Verkürzung auf 57,3 % gesteigert. Dies entspricht einer weiteren Verbesserung von 3,1 Prozentpunkten im Vergleich zum besten vorherigen Ergebnis mit dem mittleren Netz.

Bei 50 Stützstellen fällt die Verbesserung noch deutlicher aus. Hier wurde eine relative Verkürzung von 54,9 % erzielt. Das ist eine Steigerung um 6,8 Prozentpunkte gegenüber dem bisherigen Bestwert des kleinen Netzes.

Der Langzeitlauf war somit erfolgreich und führte in beiden Konfigurationen zu einer signifikanten Leistungssteigerung. Insbesondere bei 50 Stützstellen konnte das Netzwerk sein Potenzial durch die längere Trainingsphase voll ausschöpfen und das Ergebnis erheblich verbessern.

Tabelle 11 Auswertung von 100 Testpulsen, Langlauf

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
25	159,23	57,3	213,48
50	168,00	54,9	204,72

Die Abbildung 6.4 visualisiert die Ergebnisse der Langzeitläufe für 25 und 50 Stützstellen. Der obere Graph zeigt die zeitliche Intensität. Die ursprüngliche Pulsform wird durch beide Modelle deutlich verkürzt. Die Energie wird in einem zentralen Peak gebündelt, was die starke Verkürzung aus Tabelle 11 visuell bestätigt. Der untere Graph stellt die vom neuronalen Netz erzeugten Phasenmasken dar. Es ist erkennbar, dass beide Modelle für 25 und 50 Stützstellen eine sehr ähnliche Phasenstruktur im Frequenzbereich gelernt haben. Eine detaillierte Erläuterung der Phasenmasken erfolgt in Abschnitt 6.3.2.

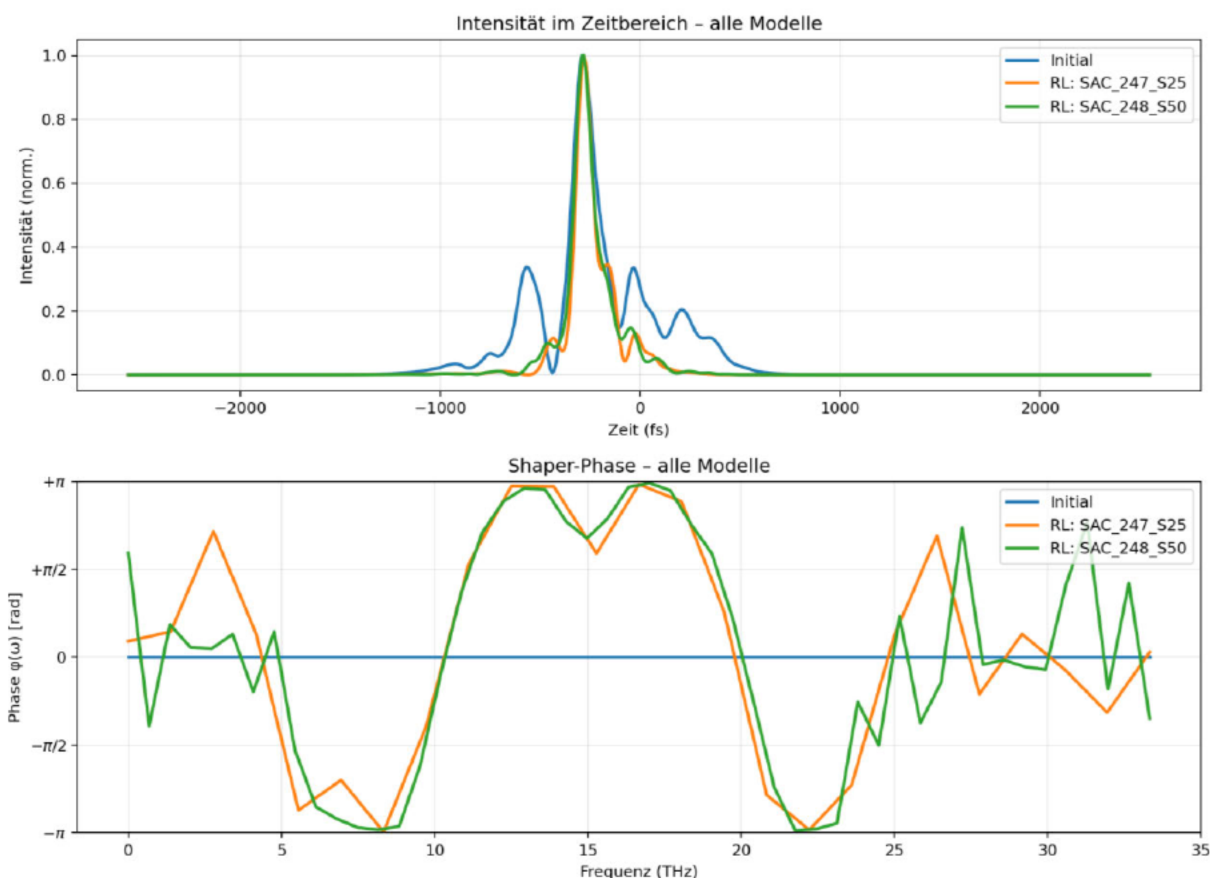


Abbildung 6.4 Gekürzter Puls, Langlauf, 25 und 50 Stützstellen

6.2 FROG

Aufbauend auf dem vorherigen Langzeitlauf wurde ein identisches Experiment mit FROG-Bildern als Netzwerkeingabe durchgeführt. Das Training umfasste ebenfalls eine Million Schritte.

Die Ergebnisse zeigen, dass auch mit FROG-Bildern eine effektive Pulsverkürzung möglich ist.

Bei 25 Stützstellen wurde eine relative Verkürzung von 54,1 % erreicht. Mit 50 Stützstellen wurde eine Verkürzung von 49,5 % erzielt.

Im direkten Vergleich zum Langlauf mit dem Zeitpuls als Eingabe sind die Resultate jedoch schwächer. Bei 25 Stützstellen fällt die Verkürzung um 3,2 Prozentpunkte geringer aus (54,1 % gegenüber 57,3 %). Bei 50 Stützstellen ist der Leistungsunterschied mit 5,4 Prozentpunkten noch deutlicher (49,5 % gegenüber 54,9 %).

Tabelle 12 Auswertung von 100 Testpulsen, FROG

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
25	171,13	54,1	201,59
50	188,32	49,5	184,40

Die Abbildung 6.5 visualisiert das Ergebnis für den Fall mit 25 Stützstellen. Das FROG-Bild des Eingangspulses weist eine komplexe, in der Zeit ausgedehnte Struktur auf. Nach der Kompensation ist das FROG-Bild des Ausgangspulses signifikant kompakter und auf die Zeitverzögerung Null zentriert. Dies bestätigt die erfolgreiche Kompression des Pulses.

Obwohl die Verwendung von FROG-Spuren als Eingabe zu einer deutlichen Pulsverkürzung führt, erreicht sie nicht die Spitzenwerte des Trainings mit dem Intensitätspuls.

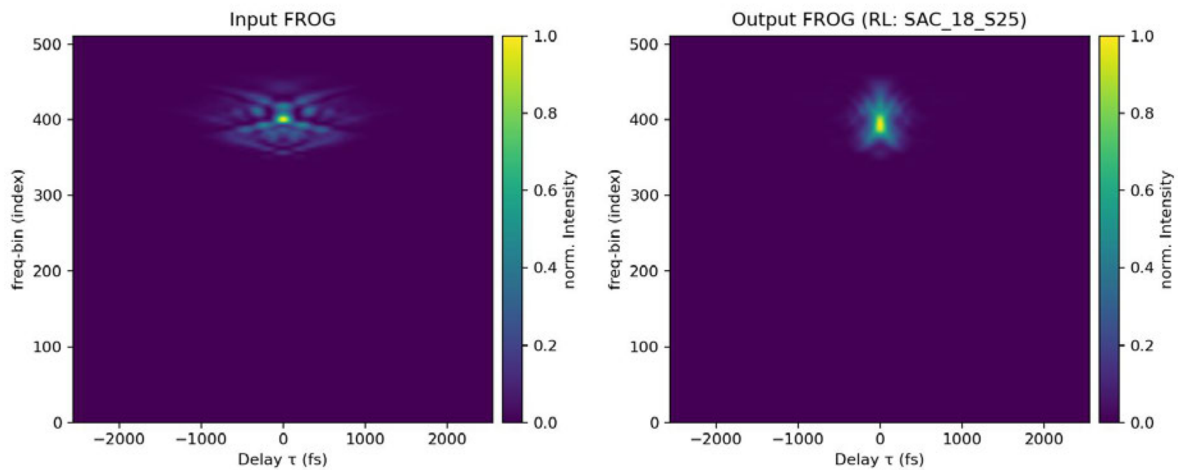


Abbildung 6.5 Gekürzter FROG, 25 Stützstellen

6.3 Weiterführende Analysen

6.3.1 Anwendung anderer RL-Algorithmen

In den Grundlagen wurde der SAC-Algorithmus als der am besten geeignete Ansatz für die Pulsformung eingestuft. Alle bisherigen Experimente und Optimierungen basierten ausschließlich auf SAC. Um diese ursprüngliche Auswahl zu validieren, wird die Leistung nun mit den alternativen Algorithmen PPO und TD3 verglichen. Die alternativen Modelle werden dabei mit den identischen Hyperparametern trainiert, die zuvor für den SAC als optimal ermittelt wurden. Dies umfasst das große Netz, einen Diskontfaktor von 0,7, eine Puffergröße von 50.000 und die zweite Belohnungsfunktion. Die Untersuchung erfolgt beispielhaft mit 25 Stützstellen.

Die Auswertung der Ergebnisse bestätigt die Eignung des SAC-Algorithmus. Das SAC-Modell erreichte eine mittlere Pulsverkürzung von 57,3 Prozent. Die alternativen Ansätze blieben hinter diesem Wert zurück. Das PPO-Modell erzielte eine Verkürzung von 32,2 Prozent und das TD3-Modell 28,0 Prozent. Die Resultate untermauern die in den Grundlagen getroffene Annahme, dass SAC für das vorliegende Optimierungsproblem die leistungsfähigste Methode ist. Es ist anzumerken, dass die verwendeten Hyperparameter gezielt für den SAC optimiert wurden. Diese Einstellungen sind nicht zwangsläufig ideal für PPO oder TD3, was die geringere Leistung dieser Algorithmen miterklären könnte.

Tabelle 13 Vergleich SAC, PPO, TB3

Stützstellen	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]	Absolute Verkürzung [fs]
SAC	159,23	57,3	213,48
PPO	252,63	32,2	120,09
TD3	268,37	28,0	104,35

6.3.2 Vergleich des Pulsshapers bei 10, 25, 50 und 100 Stützstellen

Die Abbildung 6.6 zeigt die vom Agenten erlernten Phasenmasken für die Konfigurationen mit 10, 25, 50 und 100 Stützstellen. Diese Modelle entsprechen den optimierten Läufen unter Verwendung der zweiten Belohnungsfunktion.

Es ist erkennbar, dass die Modelle mit 25 (grün) und 50 (rot) Stützstellen ein ähnliches Phasenprofil aufweisen. Sie erzeugen eine relativ glatte Kurve im zentralen Frequenzbereich zwischen 10 und 20 THz. Dieses Verhalten deutet darauf hin, dass beide Modelle eine vergleichbare Strategie zur Kompensation der dominanten Phasenfehler des Eingangspulses gelernt haben.

Das Modell mit 10 (orange) Stützstellen zeigt eine gröbere, stärker schwankende Kurve, die aber dennoch zu einer Verkürzung führte. Im Gegensatz dazu steht das Modell mit 100 (violett) Stützstellen. Die zugehörige Phasenkurve ist stark oszillierend. Dieses Muster bestätigt die Ergebnisse aus Tabelle 8, wonach der Agent bei dieser hohen Anzahl an Stellgrößen instabil wurde und keine sinnvolle Kompensationsstrategie mehr erlernen konnte.

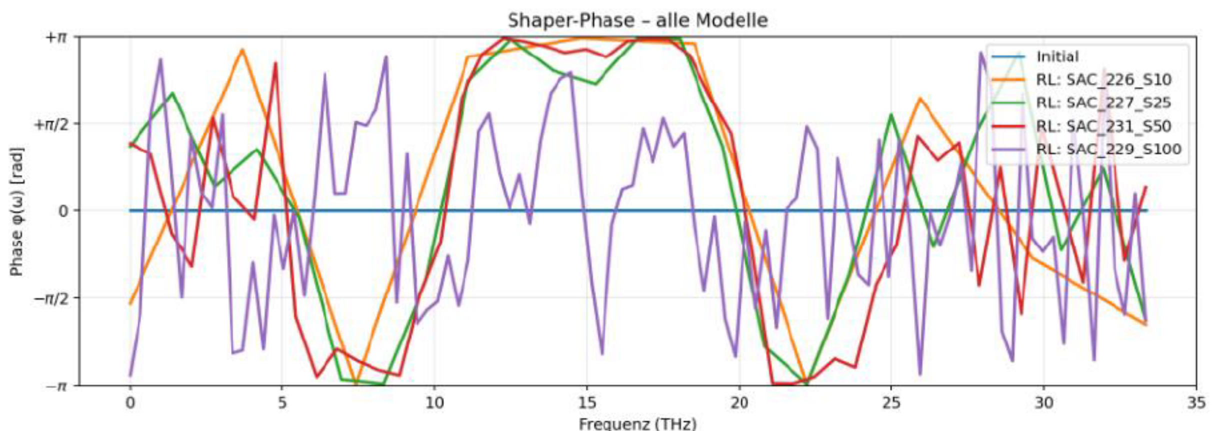


Abbildung 6.6 Pulsshaper für 10, 25, 50 und 100 Stützstellen

6.3.3 Verkleinerung der einstellbaren Frequenz für den Pulsshaper

Im nächsten Schritt wurde der Einfluss des Frequenzbereichs untersucht. Dafür wurde ein Modell mit 25 Stützstellen trainiert, dessen Aktionsraum auf den Frequenzbereich zwischen 10 und 20 THz beschränkt wurde. Die Möglichkeit, den Frequenzbereich einzugrenzen, ist in der Umgebung implementiert. Außerhalb dieses Fensters blieb die Phase unverändert.

Die Abbildung 6.7 stellt die erlernte Phasenmaske dieses Modells (unten) dem Referenzmodell aus dem Langzeitlauf gegenüber (oben), das über den vollen Frequenzbereich verfügte.

Ein direkter Vergleich des relevanten Frequenzabschnitts von 10 bis 20 THz zeigt, dass das limitierte Modell in dem engen Aktionsbereich eine Phasenstruktur erlernt hat, die der Struktur des Referenzmodells in demselben Frequenzabschnitt stark ähnelt. Es ist bemerkenswert, dass beide Agenten dieselbe Lösungsstrategie für diesen Bereich finden, obwohl ein Modell nur diesen kleinen Ausschnitt verändern konnte, während das andere Modell die gesamte Phase anpasste. Dies deutet darauf hin, dass die gefundene Phasenanpassung in diesem Frequenzfenster eine robuste und möglicherweise die effektivste Strategie zur Kompensation der dortigen Phasenfehler darstellt.

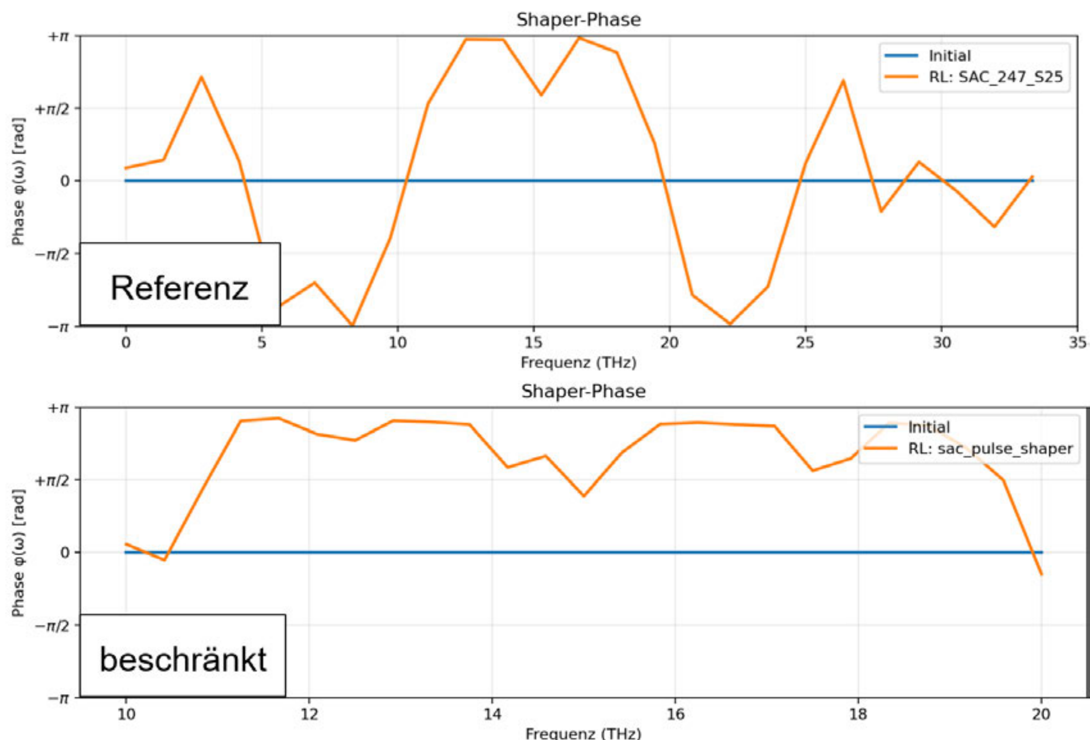


Abbildung 6.7 Vergleich des Pulsshapers

6.3.4 Überprüfung der enthaltenen Energie des Pulses

Die Abbildung 6.8 zeigt den optimierten Puls im Verhältnis zum Eingangspuls. Im Gegensatz zu den vorherigen Darstellungen wird hier nur der Eingangspuls auf eine Spitzenintensität von 1 normiert und der Ausgangspuls relativ dazu dargestellt.

Die Simulation des Pulsshapers moduliert ausschließlich die spektrale Phase und verändert nicht die spektrale Amplitude. Theoretisch muss die Gesamtenergie des Pulses, die das Integral unter der Intensitätskurve darstellt, dabei erhalten bleiben.

Die Abbildung bestätigt diese Annahme. Die berechneten Werte sind für den Eingangspuls und den Ausgangspuls identisch. Die erfolgreiche Pulsverkürzung führt zu einer Umverteilung dieser Energie. Der optimierte Puls bündelt die Energie in einem deutlich kürzeren Zeitfenster, was zu einer Spitzenintensität führt, die doppelt so hoch ist wie die des ursprünglichen Pulses.

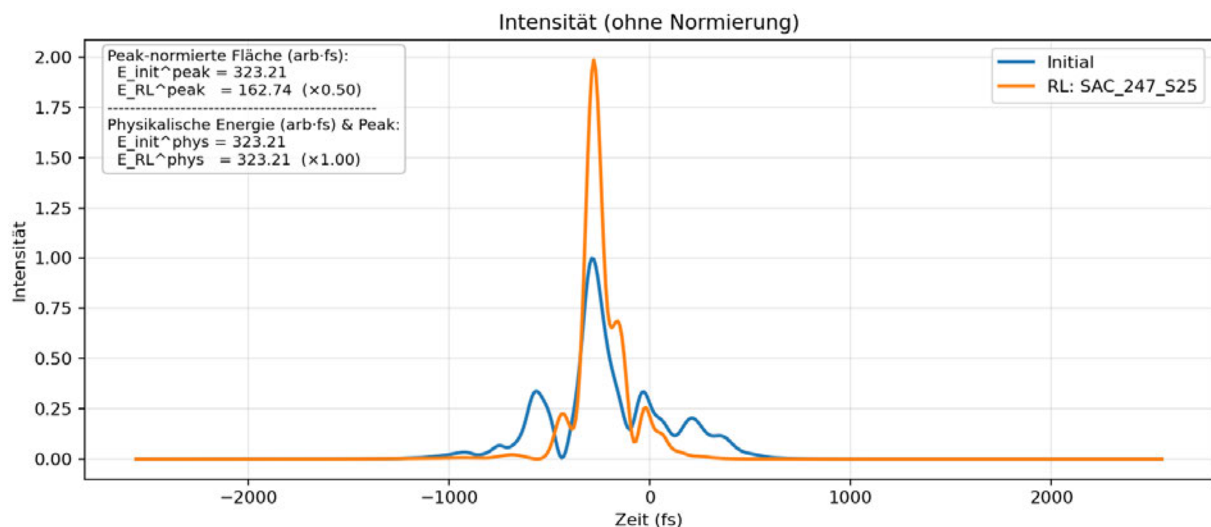


Abbildung 6.8 Vergleich der Energie des Ein- und Ausgangspuls

6.3.5 Test ohne Dispersion zweiter Ordnung oder dritter Ordnung

Zuletzt wurden zwei weitere Langzeitläufe durchgeführt, um den Einfluss von Dispersion zweiter und dritter Ordnung isoliert zu untersuchen. Diese Tests wurden mit der optimierten Konfiguration für 25 Stützstellen durchgeführt. Es sollte beobachtet werden, wie stark GDD und TOD die Pulsverkürzung limitieren.

Im ersten Testlauf wurden die Eingangspulse ohne GDD verwendet. Dies führte zu einer relativen Verkürzung von 73,3 %, wobei die RMS-Breite von 289,36 fs auf 77,28 fs reduziert wurde.

Im zweiten Lauf wurden die Pulse ohne TOD eingesetzt. Hier wurde mit 73,8 % eine fast identische, relative Verkürzung erreicht. Die Pulsbreite sank von 295,43 fs auf 77,55 fs.

Beide Ergebnisse stellen eine massive Verbesserung gegenüber dem bisherigen Bestwert von 57,3 % dar, bei dem beide Dispersionsordnungen vorhanden waren. Die Resultate zeigen, dass sowohl GDD als auch TOD wesentliche limitierende Faktoren für die erreichbare Pulsverkürzung waren. Sobald eine dieser dominanten Phasenstörungen entfernt wird, kann das neuronale Netz die verbleibenden Aberrationen deutlich effektiver kompensieren und eine signifikant stärkere Kompression erzielen.

Tabelle 14 Auswertung von 100 Testpulsen, ohne GDD/TOD

Dispersion	RMS-Eingang [fs]	RMS-Ausgang [fs]	Verhältnis der Verkürzung [%]
ohne GDD	289,36	77,28	73,3
ohne TOD	295,43	77,55	73,8

Die Abbildung 6.6 visualisiert die Resultate für beide Fälle. Die Graphen für die Pulse ohne GDD (oben) und ohne TOD (unten) sind sich visuell sehr ähnlich. In beiden Fällen wird der anfänglich breite Puls mit ausgeprägten Vorpulsen in einen einzelnen, intensiven Hauptpeak transformiert. Dies demonstriert die hohe Effizienz der Energiebündelung nach Entfernung der jeweiligen dominanten Dispersionsordnung.

Zusätzlich wurde untersucht, ob eine sequenzielle Anwendung der beiden spezialisierten Netzwerke zu einer weiteren Verbesserung führen könnte. Dazu wurde das auf Pulse ohne GDD trainierte Netz und das auf Pulse ohne TOD trainierte Netz jeweils mit Testpulsen konfrontiert, bei denen beide Dispersionsordnungen aktiv waren. Dieser Ansatz führte jedoch nicht zum Erfolg. Beide Netzwerke bewirkten eine Verlängerung der Pulse, weshalb eine Hintereinanderschaltung nicht zielführend ist.

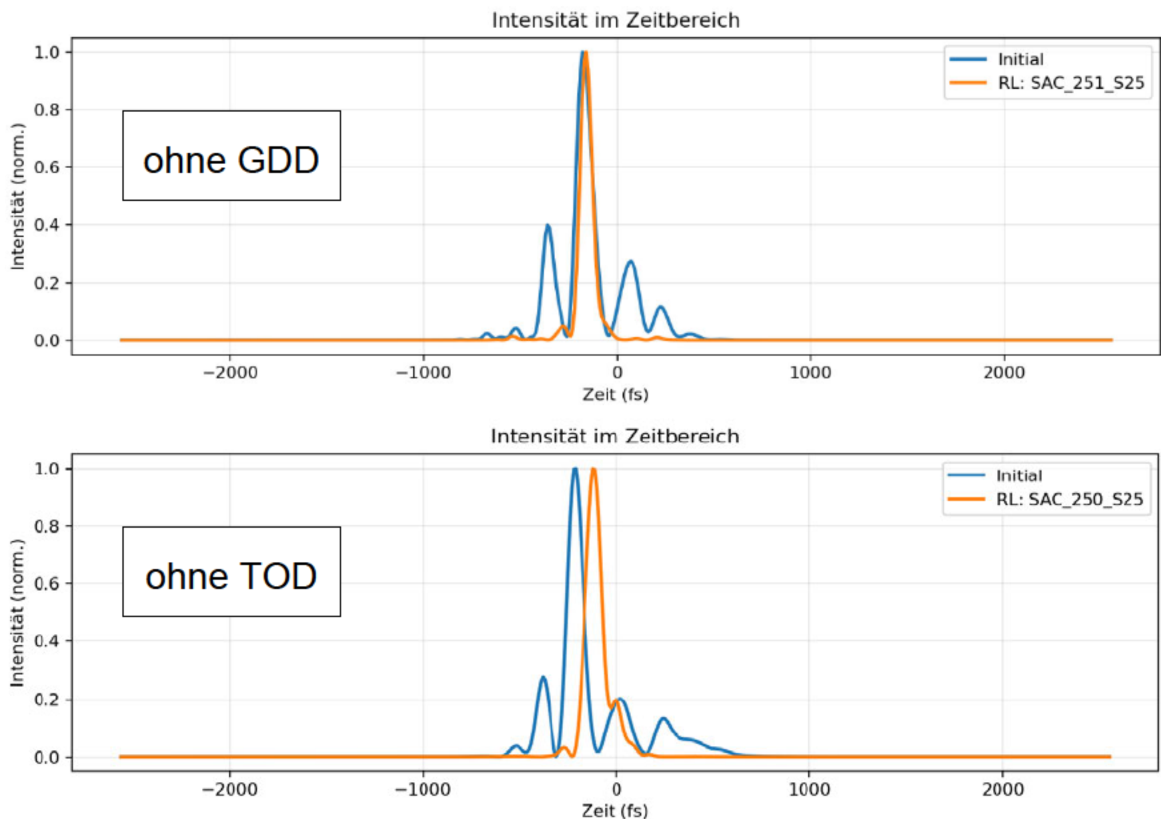


Abbildung 6.9 Gekürzter Puls, ohne GDD/TOD, 25 Stützstellen

6.3.6 Überprüfung des Fourier-Bandbreitenprodukt

Die Analyse des Fourier-Bandbreitenprodukts zeigt, wie nah die jeweiligen Pulse am theoretisch möglichen Limit sind. Wie in Abschnitt 2.2.3 erläutert, liegt das minimale TBP für einen Gauß-Puls bei 0,5. Der simulierte Eingangspuls weist mit einem TBP von 7,30 einen sehr hohen Wert auf. Dies bestätigt, dass der Puls stark phasenverzerrt ist und weit von seiner optimalen Kompression entfernt liegt. Die optimierten Modelle mit 25 und 50 Stützstellen verbessern diese Situation deutlich. Sie senken das TBP auf Werte von 3,12 und 3,29. Dies zeigt, dass der Agent die Phasenfehler erfolgreich kompensiert und den Puls näher an das Fourier-Limit bringt. Die Testläufe unter den Bedingungen ohne GDD oder TOD zeigen erwartungsgemäß noch niedrigere TBP-Werte um 1,5. Die TBP-Auswertung zeigt, dass das TBP durch die optimierten Modelle halbiert werden kann, die Pulse aber aufgrund der komplexen Phasenfehler das theoretische Optimum von 0,5 nicht erreichen.

Tabelle 15 Fourier-Bandbreitenprodukt

Puls	RMS-Ausgang [fs]	TBP
Eingang	372,71	7,30
25 Stützstellen	159,23	3,12
50 Stützstellen	168,00	3,29
ohne GDD	77,28	1,51
ohne TOD	77,55	1,52

7 Fazit und Ausblick

Die Forschungsfrage dieser Arbeit lautete: „Inwiefern eignet sich Reinforcement Learning zur Minimierung der Pulsbreite durch Pulsformung in Kurzzeitlasern und welche Rolle spielt die Auswahl der Eingangsdaten für den Lernerfolg?“

Die Ergebnisse zeigen, dass der Ansatz grundsätzlich ein leistungsfähiges Werkzeug zur Minimierung der Pulsbreite ist. Ein trainierter SAC-Agent konnte eine mittlere Pulsverkürzung von über 57 Prozent für eine Vielzahl zufälliger Eingangspulse erreichen. Der Erfolg war jedoch maßgeblich an eine systematische Optimierung der Hyperparameter geknüpft.

Ein Standardmodell ohne gezielte Anpassungen scheiterte in den Versuchen bei höherer Komplexität und führte sogar zu einer Verbreiterung der Pulse. Der entscheidende Faktor für den Erfolg war die systematische Optimierung der Hyperparameter. Dabei erwiesen sich besonders zwei Änderungen als wesentlich. Zum einen führte die Reduzierung des Diskontfaktors gamma auf 0,7 und zum anderen die Vergrößerung des buffer_size zu stabilen und leistungsfähigen Ergebnissen. Dies legt nahe, dass der Agent am besten lernt, wenn er sich auf kurzfristige Erfolge konzentriert und aus einem großen Pool an vergangenen Versuchen schöpfen kann. Für ein Problem, bei dem sich der Eingangspuls ständig ändert, erweist sich diese kurzsichtige und erfahrungsbasierte Strategie als am stabilsten.

Hinsichtlich der zweiten Hälfte der Forschungsfrage, die sich mit der Rolle der Eingangsdaten befasst, ist das Ergebnis eindeutig. Beide Ansätze, sowohl mit dem direkten Intensitätspuls als auch mit dem FROG-Bild als Eingabe, haben zuverlässig funktioniert und zu einer deutlichen Pulsverkürzung geführt. Der Intensitätspuls hatte zwar einen leichten Vorteil, was vermutlich daran liegt, dass er dem Netz die relevanten Informationen direkter präsentiert. Da aber auch das Training mit den komplexeren FROG-Bildern erfolgreich war, zeigt die grundsätzliche Robustheit des Ansatzes.

Eine interessante Erkenntnis der Arbeit ergibt sich aus den Experimenten zur Entfernung von GDD oder TOD, die eine enorme Leistungssteigerung zeigten. Sobald einer dieser dominanten Phasenfehler fehlte, verbesserte sich die Pulsverkürzung auf über 73 Prozent. Dies lässt den Schluss zu, dass die eigentliche Stärke des RL-Ansatzes in der Korrektur der verbleibenden, komplexen Phasenstörungen liegt. Solange die großen und systematischen Fehler durch GDD und TOD das Problem dominieren, hat

das Netz Schwierigkeiten. Fällt einer dieser Hauptstörfaktoren weg, kann das Netz seine Fähigkeiten nutzen, um den Puls sehr effizient zu komprimieren.

Zusammenfassend lässt sich die Forschungsfrage wie folgt beantworten: Reinforcement Learning ist ein sehr fähiges Werkzeug zur Pulsformung, wenn der Lernprozess sorgfältig konfiguriert wird. Die Auswahl der Eingangsdaten spielt dabei eine geringere Rolle. Die Methode zeigt ihre Stärken besonders bei der Korrektur von komplexen Phasenfehlern. Sie stößt jedoch an ihre Grenzen, wenn dominante und breitbandige Fehler wie GDD und TOD gleichzeitig das Problem bestimmen oder der Lösungsraum durch zu viele Stützstellen zu groß wird.

Basierend auf diesen Erkenntnissen ergeben sich mehrere vielversprechende Ansätze für zukünftige Arbeiten.

Ein logischer nächster Schritt wäre die Erweiterung des Optimierungsziels. Die reine Minimierung der Pulsbreite stellt nur einen spezifischen Anwendungsfall dar. Künftige Arbeiten könnten den Agenten darauf trainieren, gezielt komplexe Pulsformen zu erzeugen. Dies könnte beispielsweise die Generierung von Doppelpulsen mit definierbarem zeitlichen Abstand oder Pulsfolgen mit einer bestimmten temporalen Struktur.

Des Weiteren wäre die Untersuchung der Skalierbarkeit des Aktionsraums sinnvoll. Die Instabilität des SAC-Agenten bei 100 Stützstellen deutet darauf hin, dass der Algorithmus an seine Grenzen stößt, wenn der Aktionsraum sehr groß wird. Ein Ansatz zur Lösung dieses Problems wäre ein gestuftes Training. Der Agent könnte zunächst mit einer geringen Anzahl an Stützstellen trainiert werden. Sobald er dort eine stabile Strategie zur Kompensation erlernt hat, könnte dieses vortrainierte Modell als Startpunkt für ein anschließendes Training mit einer höheren Anzahl von Stützstellen genutzt werden. Diese Skalierung des Aktionsraums könnte dem Agenten helfen, die feinere Phasenmodulation zu erlernen, ohne die im Training beobachtete Instabilität zu entwickeln.

Der wichtigste Schritt für die Zukunft ist jedoch die Übertragung des Konzepts auf ein reales Experiment. Die vorliegende Arbeit basiert vollständig auf einer simulierten Umgebung. Die Validierung des Ansatzes an einem physikalischen Kurzzeitlaser-System wäre ein wichtiger Beweis für die Praxistauglichkeit. Ein solcher Aufbau würde zeigen, ob der Agent in der Lage ist, mit realen Messdaten und Rauschen umzugehen und den Laserpuls in Echtzeit zu optimieren.

Literaturverzeichnis

[1] M. W. Sigrist, Laser: Theorie, Typen und Anwendungen, Springer Berlin Heidelberg.

[2] J. D. Pickering, Ultrafast Lasers and Optics for Experimentalists, IOP Publishing.

[3] 07 2025. [Online]. Available: https://www.leifiphysik.de/atomphysik/laser/ausblick/laser-typen.

[4] „RP-Photonics," 10 2025. [Online]. Available: https://www.rp-photonics.com/time_bandwidth_product.html#:~:text=In%20ultrafast%20laser%20physics%2C%20it,are%20generating%20substantially%20chirped%20pulses..

[5] „Duden," 07 2025. [Online]. Available: https://learnattack.de/schuelerlexikon/physik/dispersion.

[6] „layertec," 08 2025. [Online]. Available: https://www.layertec.de/de/coatings/ultrafast-laser-coatings/ultrafast-laser-coatings-introduction/.

[7] R. S. Joachim Steinwendner, Neuronale Netze programmieren, Rheinwerk Computing.

[8] D. Sonnet, Neuronale Netze kompakt, Springer Vieweg.

[9] A. Jung, Maschinelles Lernen, Die Grundlagen, Springer.

[10] M. Lapan, Deep Reinforcement Leraning.

[11] „TensorFlow," 08 2025. [Online]. Available: https://www.tensorflow.org/agents/tutorials/0_intro_rl.

[12] „OpenAI," 08 2025. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/sac.html#soft-actor-critic.

[13] „OpenAI," 10 2025. [Online]. Available: https://openai.com/index/openai-baselines-ppo/.

[14] „OpenAI,“ [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/td3.html. [Zugriff am 10 2025].

[15] P. D. I. J. Dahlkemper, „Vorlesungsfolie Bildverarbeitung Kapitel 9“.

[16] S. J. Prince, Understanding Deep Learning.

[17] „wraycastle,“ 09 2025. [Online]. Available: https://wraycastle.com/blogs/knowledge-base/how-do-you-calculate-rms-1.

[18] „deepmind,“ 09 2025. [Online]. Available: https://www.deep-mind.org/2023/03/26/the-universal-approximation-theorem/.

[19] „ibm,“ 09 2025. [Online]. Available: https://www.ibm.com/de-de/think/topics/learning-rate.

[20] „tensorflow,“ 09 2025. [Online]. Available: https://www.tensorflow.org/agents/tutorials/5_replay_buffers_tutorial.

[21] „milvus,“ 09 2025. [Online]. Available: https://milvus.io/ai-quick-reference/how-does-entropy-regularization-improve-exploration.

[22] „sable-baselines3,“ 09 2025. [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/modules/sac.html.

[23] A. A. Awan, „datacamp,“ [Online]. Available: https://www.datacamp.com/de/tutorial/introduction-q-learning-beginner-tutorial.

[24] T. W. A. Kugi, 08 2025. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.acin.tuwien.ac.at/fileadmin/cds/lehre/opt/WS2015/Kapitel2.pdf.

[25] „edmundoptics,“ 08 2025. [Online]. Available: https://www.edmundoptics.de/knowledge-center/application-notes/optics/basics-of-ultrafast-lasers/.

[26] „ibm.com,“ 09 2025. [Online]. Available: https://www.ibm.com/de-de/think/topics/hyperparameter-tuning.

[27] „ibm,“ 09 2025. [Online]. Available: https://www.ibm.com/think/topics/bias-variance-tradeoff.

[28] „studysmarter,“ 09 2025. [Online]. Available: https://www.studysmarter.de/schule/informatik/algorithmen-und-datenstrukturen/evolutionaere-algorithmen/.

Anhang

A 1 Code Referenzpuls

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import scipy.interpolate as intp
4  import scipy.optimize as opt   # für Nelder-Mead
5  import random
6
7  # ---------- Grundkonstanten ----------
8  femto = 1e-15
9  angFregTHz = 2 * np.pi * 1e12  # Winkelkreisfrequenz in rad/s pro THz
10
11 # ---------- Gitter / Achsen ----------
12 N = 512
13 Ts = 10 * femto
14 T0 = N * Ts
15 idxVec = np.arange(0, N)
16 tt = idxVec * Ts
17 ttc = tt - T0 / 2
18 ws = 2 * np.pi / Ts           # Nyquist-Kreisfrequenz
19 w0 = ws / N
20 ww = idxVec * w0
21
22 # ---------- Shaper-Grenzen ----------
23 shaperLims = (0, ws / 3)       # in diesem Bereich darf die Phase
   verändert werden
24
25 # ---------- Referenzspektrum (Amplitude) ----------
26 wp = ws / 8
27 alpha = 15 * angFregTHz ** 2
28 YwrefAmpl = np.exp(-1 / alpha * (ww - wp) ** 2) + 0.6 * np.exp(-2 / (1
   * alpha) * (ww - 1.5 * wp) ** 2)
29 YwrefAmpl = YwrefAmpl / max(YwrefAmpl)
```

```
30 # ---------- Referenzphase ----------
31 Sref = 25
32 wStuetz = np.linspace(shaperLims[0], shaperLims[1], Sref)
33 random.seed(1234)
34 gdd = 0.3 / angFregTHz ** 2
35 tod = 0.1 / angFregTHz ** 3
36 phStuetz = 0.5 * gdd * (wStuetz - wp) ** 2 + (1.0 / 6.0) * tod *
   (wStuetz - wp) ** 3
37 phStuetzRd = np.pi * (2.0 * np.random.rand(Sref) - 1.0) * np.exp(-0.5 /
   alpha * (wStuetz - wp) ** 2)
38 phStuetz += phStuetzRd
39 phIntp = intp.CubicSpline(wStuetz, phStuetz)
40
41 phaseRef = np.zeros(N)
42 shaperMask = ww < shaperLims[1]
43 phaseRef[shaperMask] = phIntp(ww[shaperMask])
44
45 # ---------- Referenzpuls im Zeitbereich ----------
46 ytref = np.fft.fftshift(np.fft.ifft(YwrefAmpl * np.exp(1j * phaseRef)))
47 ytref /= max(np.abs(ytref))
48
49 # ====================================================================
50 #                           Nützliche Funktionen
51 # ====================================================================
52
53 def pulseShaper(ytin, Ts, phStuetz, wStuetz):
54     """
55     Simuliert den Phasenshaper:
56     - Interpoliert die phasen-Stützstellen auf die volle Frequenzachse
   (außerhalb des Shaper-Fensters = 0)
57     - Wendet die Phase im Spektralbereich an
58     - IFFT -> Zeitbereich, normiert
59     """
60     Nloc = len(ytin)
61     ws_loc = 2 * np.pi / Ts
62     w0_loc = ws_loc / Nloc
63     ww_loc = np.arange(0, Nloc) * w0_loc
64
65     Yw = np.fft.fft(np.fft.fftshift(ytin))
66
67     phIntp_loc = intp.CubicSpline(wStuetz, phStuetz)
68     ph = np.zeros(Nloc)
69     mask = ww_loc < wStuetz[-1]
70     ph[mask] = phIntp_loc(ww_loc[mask])
71
72     Yw *= np.exp(1j * ph)
73     ytout = np.fft.fftshift(np.fft.ifft(Yw))
74     return ytout / (np.max(np.abs(ytout)) + 1e-20)
```

```
75 def calcRMSWidth(yd, tt):
76     """
77     RMS-Breite (σ) eines nichtnegativen Signals 'yd' auf Stützstellen
   'tt'.
78     Erwartet typischerweise Intensität (|y|^2).
79     """
80     Nloc = len(yd)
81     if len(tt) != Nloc:
82         raise ValueError("yd and tt must have the same length")
83
84     inten = np.asarray(yd, dtype=float)
85     inten = np.maximum(inten, 0.0)
86
87     norm_fac = np.sum(inten) + 1e-20
88     mom1 = float(np.sum(tt * inten) / norm_fac)
89     mom2 = float(np.sum((tt ** 2) * inten) / norm_fac)
90     md = mom2 - mom1 ** 2
91     if md < 0:
92         return -1.0
93     return float(np.sqrt(md))
94
95 # ====================================================================
96 #        NEU: Zufälligen Eingangspuls (Amplitude + Phase) erzeugen
97 # ====================================================================
98
99 def make_random_input_pulse(
100     seed=None,
101     n_peaks_range=(1, 3),          # Anzahl spektraler Gauss-Peaks
102     amp_range=(0.5, 1.0),          # Peakamplituden
103     center_rel_range=(0.08, 0.45), # Peakzentren relativ zu Nyquist
   (0..0.5)
104     width_rel_range=(0.005, 0.05), # Peakbreiten relativ zu Nyquist
105     poly_phase_std=(0.2, 0.05),    # std(GDD), std(TOD) in 1/THz^2,
   1/THz^3
106     spline_jitter_std=0.5,         # Zusatzrauschen auf Phasen-
   Stützstellen (rad)
107     Sref_local=25                  # Stützstellen für die Phasen-Spline
108):
109     Erzeugt einen *neuen* Eingangspuls (yt) mit zufälliger spektraler
   Amplitude und Phase.
110     Rückgabe: zeitlicher Puls yt (komplex), normiert auf max|yt| = 1.
111     """
112     rng = np.random.default_rng(seed)
```

```
113     # --- Spektrale Amplitude: Summe von zufälligen Gauss-Peaks ---
114     n_peaks = rng.integers(n_peaks_range[0], n_peaks_range[1] + 1)
115     Yw_ampl = np.zeros_like(ww, dtype=float)
116     for _ in range(n_peaks):
117         a = rng.uniform(*amp_range)
118         c_rel = rng.uniform(*center_rel_range)  # 0..0.5
119         c = c_rel * ws
120         w_rel = rng.uniform(*width_rel_range)   # 0..0.5
121         sig = max(w_rel * ws, 1e-20)
122         Yw_ampl += a * np.exp(-0.5 * ((ww - c) / sig) ** 2)
123
124     Yw_ampl /= (np.max(Yw_ampl) + 1e-20)
125
126     # --- Spektrale Phase: (Zufalls-GDD/TOD) + glatte Spline + Jitter -
    --
127     Sref_loc = int(Sref_local)
128     wStuetz_loc = np.linspace(shaperLims[0], shaperLims[1], Sref_loc)
129
130     gdd_std, tod_std = poly_phase_std
131     gdd_rnd = rng.normal(0.0, gdd_std) / (angFregTHz ** 2)  # s^2
132     tod_rnd = rng.normal(0.0, tod_std) / (angFregTHz ** 3)  # s^3
133
134     # Zentrum der Phase: falls 'wp' existiert, sonst Maximum der
    Amplitude
135     try:
136         w_center = wp
137     except NameError:
138         w_center = ww[np.argmax(Yw_ampl)]
139
140     ph_stuetz = 0.5 * gdd_rnd * (wStuetz_loc - w_center) ** 2 + (1.0 /
    6.0) * tod_rnd * (wStuetz_loc - w_center) ** 3
141     # Glatter, gaussförmig gewichteter Jitter über dem Shaper-Fenster
142     ph_stuetz += rng.normal(0.0, spline_jitter_std, size=Sref_loc) *
    np.exp(
143         -0.5 * ((wStuetz_loc - w_center) / (0.25 * ws)) ** 2
144     )
145
146     phIntp_loc = intp.CubicSpline(wStuetz_loc, ph_stuetz)
147     phase_loc = np.zeros_like(ww)
148     mask = ww < wStuetz_loc[-1]
149     phase_loc[mask] = phIntp_loc(ww[mask])
150
151     # --- Zeitpuls erzeugen ---
152     Yw = Yw_ampl * np.exp(1j * phase_loc)
153     yt = np.fft.fftshift(np.fft.ifft(Yw))
154     yt /= (np.max(np.abs(yt)) + 1e-20)
155     return yt
```

A 2 Code Referenz-FROG

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # --- Import aus deiner (entrümpelten) pulseOpt.py ---
5  from pulseOpt import (
6      N, Ts, ttc, femto,
7      make_random_input_pulse,
8      calcRMSWidth,
9  )
10
11 #
   ========================================================================
   ======
12 #                       Hilfsfunktionen & Metriken
13 #
   ========================================================================
   ======
14
15 def pulse_intensity(yt, normalize=True):
16     """|E(t)|^2 und Zeitachse ttc zurückgeben."""
17     I = np.abs(yt) ** 2
18     if normalize:
19         I = I / (np.max(I) + 1e-20)
20     return I.astype(np.float32), ttc
21
22 def pulse_metrics(yt):
23     """RMS (fs), FWHM (fs), Energie (arb.) der Zeit-Intensität
   |E(t)|^2."""
24     I = np.abs(yt) ** 2
25     rms_s = calcRMSWidth(I, ttc)
26     rms_fs = rms_s / femto
27     fwhm_fs = 2.355 * rms_fs
28     energy = float(np.trapz(I, x=ttc))
29     return dict(rms_fs=rms_fs, fwhm_fs=fwhm_fs, energy=energy)
30
31 def _circshift(x, shift):
32     """Zirkularer Shift."""
33     return np.roll(x, int(shift) % len(x))
```

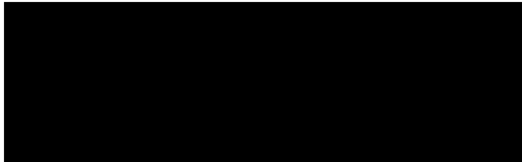
```
34 ======================================================================
======
35 #                      SHG-FROG (Amplitude & Intensität)
36 #
======================================================================
======
37
38 def create_shg_ampl(E_t, Ts):
39     """
40     Komplexe SHG-FROG-Amplitude A(ω, τ) (Form: N x N).
41     Spalte = fester Delay τ, Zeile = Frequenz-Bin (fft-Bin).
42     """
43     E = np.asarray(E_t, dtype=np.complex128)
44     Nloc = E.size
45     assert Nloc == N, "E_t muss Länge N haben (gleiches Grid wie
pulseOpt)."
46
47     # Delays: -N/2 .. N/2-1
48     delay_idx = np.arange(-Nloc // 2, Nloc // 2, dtype=int)
49     tau_axis = delay_idx * Ts
50
51     # Phasenkorrektur für SHG (Frequenzverdopplung)
52     shift_factor = np.exp(-1j * 2.0 * tau_axis)
53
54     A = np.zeros((Nloc, Nloc), dtype=np.complex128)
55     for col, d_idx in enumerate(delay_idx):
56         E_shifted = _circshift(E, d_idx)
57         argument = E * E_shifted * shift_factor[col]
58         A[:, col] =
np.fft.fftshift(np.fft.fft(np.fft.fftshift(argument)))
59
60     return A
61
62 def frog_from_pulse(yt, Ts, normalize=True):
63     """
64     Aus einem Zeitpuls yt das SHG-FROG-Intensity-Trace berechnen.
65     Rückgabe:
66       Ishg: (N x N) Intensität (optional auf max=1 normiert)
67       tau_axis: (N,) Verzögerungsachse in s
68     """
69     A = create_shg_ampl(yt, Ts)
70     Ishg = np.abs(A) ** 2
71     if normalize:
72         Ishg = Ishg / (np.max(Ishg) + 1e-20)
73
74     delay_idx = np.arange(-len(yt)//2, len(yt)//2)
75     tau_axis = delay_idx * Ts
76     return Ishg.astype(np.float32), tau_axis
```

```
77 def frog_to_obs(Ishg, scale_to=(0.0, 10.0), flatten=True, eps=1e-20):
78     """
79     Skaliert ein FROG-Bild auf [a,b] und gibt optional einen flachen
   Vektor zurück.
80     Default: [0,10] – kompatibel zu RL-Observations.
81     """
82     a, b = scale_to
83     X = np.asarray(Ishg, dtype=np.float32)
84     X = X / (np.max(X) + eps)
85     X = a + (b - a) * X
86     return X.flatten() if flatten else X
87
88 def frog_and_pulse(yt, Ts, normalize=True):
89     """
90     Praktischer Wrapper: FROG + Zeit-Intensität + Metriken.
91     Rückgabe: Ishg, tau_axis, I_t, t_axis, metrics
92     """
93     Ishg, tau_axis = frog_from_pulse(yt, Ts,  normalize=normalize)
94     I_t, t_axis = pulse_intensity(yt, normalize=True)
95     metrics = pulse_metrics(yt)
96     return Ishg, tau_axis, I_t, t_axis, metrics
97
98 #
   ========================================================================
   ======
99 #                               Demo (Direktrun)
100#
   ========================================================================
   ======
101
102if __name__ == "__main__":
103     print("[frogOpt] Demo: Random-Puls -> FROG + |E(t)|^2 +
   RMS/FWHM/Energie")
104
105     # === Random-Puls erzeugen (jeder Lauf neu) ===
106     yt = make_random_input_pulse(seed=None)
107
108     # === FROG + Zeit-Intensität + Metriken berechnen ===
109     Ishg, tau_axis, I_t, t_axis, met = frog_and_pulse(yt, Ts)
110
111     print(f"RMS  = {met['rms_fs']:.2f} fs")
112     print(f"FWHM = {met['fwhm_fs']:.2f} fs")
113     print(f"Ener. = {met['energy']:.3g} (arb.)")
114
115     # === Plot: links FROG, rechts Zeit-Intensität ===
116     fig, axs = plt.subplots(1, 2, figsize=(10.5, 4.4))
```

```
117     im = axs[0].imshow(
118         Ishg, aspect='auto', origin='lower',
119         extent=[tau_axis[0]/femto, tau_axis[-1]/femto, 0, N-1]
120     )
121     axs[0].set_xlabel("Delay τ (fs)")
122     axs[0].set_ylabel("freq-bin (index)")
123     axs[0].set_title("SHG-FROG (Random)")
124     cbar = plt.colorbar(im, ax=axs[0], pad=0.02)
125     cbar.set_label("norm. Intensity")
126 
127     axs[1].plot(t_axis / femto, I_t, lw=2)
128     axs[1].set_xlabel("Zeit (fs)")
129     axs[1].set_ylabel("Intensität (norm.)")
130     axs[1].set_title(f"|E(t)|^2 – RMS={met['rms_fs']:.1f} fs")
131     axs[1].grid(alpha=0.3)
132 
133     plt.tight_layout()
134     plt.show()
135 
136     print("[frogOpt] Demo fertig.")
```

Eidesstattliche Erklärung

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



Hamburg, den 28. Oktober 2025