A comparison of reinforcement learning algorithms for dynamic pricing in recommerce markets

Ein Vergleich von

Reinforcement-Learning-Algorithmen für die

dynamische Preisgestaltung auf Recommerce-Märkten

Jan Niklas Groeneveld

Universitätsbachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Science
(B. Sc.)

im Studiengang
IT Systems Engineering

eingereicht am 19. Juli 2022 am
Fachgebiet Enterprise Platform and Integration Concepts
der Digital-Engineering-Fakultät
der Universität Potsdam

Betreuer	Dr. Michael Perscheid
	Dr. Rainer Schlosser
	Johannes Huegle
	Alexander Kastius

Abstract

Trading second-hand items on online marketplaces is a rising business. Thereby, pricing is a major challenge. E-commerce is highly dynamic, the product range is large, and fast reactions to competitors are required. Thus, automation of pricing strategies is a necessary step for traders. This thesis is about dynamic pricing with Reinforcement Learning (RL), a machine learning technology. Today, thanks to intensive research in the past years, many different RL-algorithms exist. Five of these algorithms are compared to competitive rule-based strategies in monopoly, duopoly, and oligopoly markets. It is found that A2C, PPO, and SAC can succeed and outperform competitors, while DDPG and TD3 lack the necessary performance. Later, several problems that arise in real-world applications are tackled and solved. It is shown that missing some information about the competitor does not break the algorithms. The problem of not knowing the opposite's strategy can be successfully solved by self-play.

Zusammenfassung

Der Handel mit gebrauchten Artikeln auf Online-Marktplätzen ist ein wachsendes Geschäft. Dabei stellt gerade das Pricing eine große Herausforderung dar. Onlinehandel ist hochdynamisch, die Produktpalette ist groß, und es muss schnell auf Wettbewerber reagiert werden. Daher ist die Automatisierung von Preisstrategien ein notwendiger Schritt für Händler. Diese Arbeit befasst sich mit der dynamischen Preisgestaltung mit Reinforcement Learning (RL), einer Machine-Learning-Technologie. Dank intensiver Forschung in den letzten Jahren gibt es heute viele verschiedene RL-Algorithmen. Es werden fünf dieser Algorithmen auf Monopol-, Duopol- und Oligopolmärkten mit wettbewerbsorientierten regelbasierten Strategien verglichen. Es zeigt sich, dass A2C, PPO und SAC erfolgreich sind und die Konkurrenten übertreffen können, während DDPG und TD3 nicht die nötige Leistung erbringen. Später werden mehrere Probleme, die in realen Anwendungen auftreten, angegangen und gelöst. Es wird gezeigt, dass das Fehlen einiger Informationen über den Konkurrenten die Algorithmen nicht beeinträchtigt. Das Problem, die Konkurrenzstrategie nicht zu kennen, kann mit Self-Play erfolgreich gelöst werden.

Inhaltsverzeichnis

1 Einführung

1.1 Ein Überblick zu Recommerce und Preisstrategien

Nicht erst seitdem es Online-Marktplätze gibt, ist die Festlegung von Preisen eine zentrale Herausforderung im Geschäftsbetrieb. Durch Online-Marktplätze hat allerdings die Intensität des Wettbewerbs erheblich zugenommen, da Konkurrenzangebote leicht einsehbar sind und Preise in hochfrequentem Tempo gesetzt werden können. Zudem müssen alle Produkte des meist sehr breiten Angebots in Abhängigkeit der Marktsituation dynamisch eingepreist werden. Diese Gründe motivieren automatische Preisfindung, da menschliche Branchenexperten mit der Anpassung der Preise schlicht nicht mehr schritthalten können.

Eine weitere Schwierigkeit kommt im sogenannten Recommerce hinzu. Recommerce beschreibt laut [Arc05] den Rückkauf, die Aufwertung und den Weitervertrieb von gebrauchten Produkten im Internet. Gerade mit Blick auf die ressourcenintensive Modebranche verspricht Recommerce eine Steigerung der Nachhaltigkeit durch Einführung einer Kreislaufwirtschaft. Wer einmal ein Produkt gekauft hat und als Eigentümer es irgendwann nicht weiter besitzen möchte, bekommt die Möglichkeit, durch Rückverkauf des Produktes etwas Geld einzunehmen. Die gebrauchten und ggf. aufgewerteten Produkte werden dann zu niedrigeren Preisen weiterverkauft. Zur Herausforderung, neue Produkte einzupreisen, kommt also noch hinzu, Preise für gebrauchten Produkte und die Rückkaufpreise festzulegen. Dabei müssen auch Nebenbedingungen wie Lagerkosten beachtet werden, wodurch sich ein kompliziertes Problem ergibt. Abbildung 1.1 zeigt eine grafische Visualisierung des in dieser Arbeit untersuchten Duopolszenarios.

Die Herangehensweisen für das Dynamic Pricing, die heute in der Geschäftswelt genutzt werden, basieren vor allem auf regelbasierten Strategien. Dafür erstellen Marktexperten Heuristiken, die auf Erfahrungen und angestellten Überlegungen basieren. Auf diese Art können durchaus gute Preisstrategien erstellt werden. Allerdings werden dafür einerseits qualifizierte Marktkenner benötigt, und andererseits sind die Lösungen sehr speziell auf ein Marktszenario zugeschnitten. Verändern sich die Umstände oder soll die Lösung für einen neuen Markt aufgesetzt werden, so muss der gesamte manuelle Prozess erneut durchlaufen werden.

Reinforcement Learning (RL) ermöglicht, nur auf Grundlage eines Marktes durch Ausprobieren und Sammeln von Erfahrung eine Preisstrategie zu optimieren. Das Training geschieht dabei üblicherweise in einer Simulation, da auf echten Märkten durch das notwendige Ausprobieren ein erheblicher finanzieller Schaden entstünde.

Weil es sehr viele unterschiedliche RL-Algorithmen gibt, ist es eine interessante Frage, welche Vor- und Nachteile die einzelnen Algorithmen auf einem Recommerce-Markt haben. Einen solchen Vergleich herzustellen, ist Ziel dieser Arbeit. Zunächst wird das zu untersuchende Marktszenario in Abschnitt 1.2 als Markov-Entscheidungsprozess formalisiert und in Abschnitt 1.3 eine regelbasierte Strategie eingeführt.

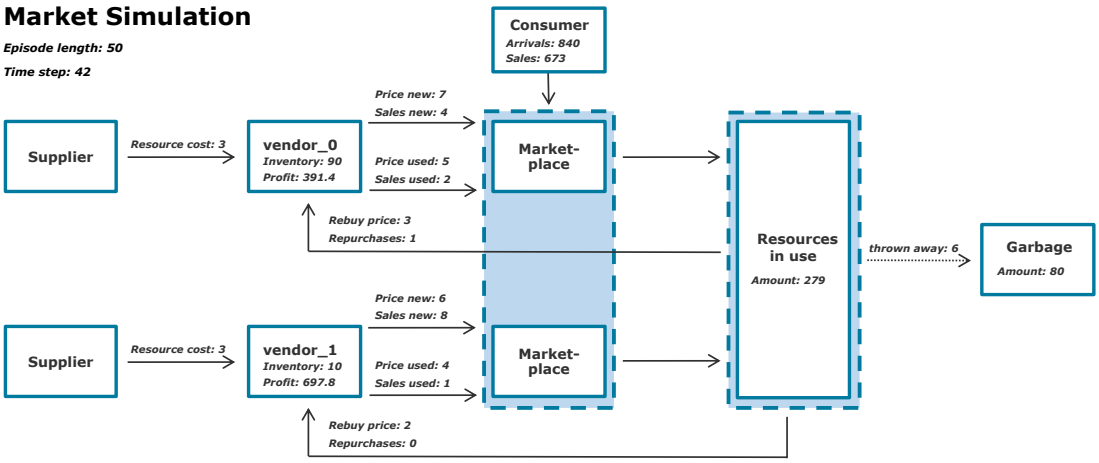

Abbildung 1.1: Zwei Konkurrenten (vendor_0 und vendor_1) bieten die gleiche Produktlinie an. Beide legen Preise für ihre neuen und gebrauchten Produkte fest. Die Kunden (Consumer) entscheiden dann, ob und bei wem sie kaufen. Neuprodukte werden zu einem festen Preis vom Zulieferer (Supplier) eingekauft. Die Eigentümer der im Gebrauch befindlicher Produkte (Resources in use) haben die Wahl, ob sie das Produkt an einen der Händler zurückverkaufen. Sie können ihr Produkt auch trotz der Rückkaufangebote wegwerfen, sobald sie es nicht länger besitzen wollen.

1.2 Der Markt als Markov-Entscheidungsprozess

Der Recommerce-Markt wird hier als Markov-Entscheidungsprozess modelliert. Es wird zunächst das Duopolszenario eingeführt. Dazu wird die Zeit in gleich lange Abschnitte diskretisiert. Vor jedem Schritt wird dem Agenten der Zustand des Marktes präsentiert. Daraufhin setzt der Agent seine Preise, die für den Rest des Schrittes gelten. In der Übertragung eines echten Marktes könnte ein Schritt zum Beispiel einer Stunde oder einer Minute entsprechen. Eigentlich soll die Optimierung für einen *infinite horizon* stattfinden. Das bedeutet, dass nicht auf einen festen Zeitraum hin optimiert wird. Stattdessen wird auf ein langfristig profitables Equilibrium hingearbeitet. Um allerdings das Problem für Training und Analyse handhabbar zu machen, werden Episoden betrachtet, die aus $n_{ep} \in \mathbb{N}$ Schritten bestehen. Dieses n_{ep} wurde in der Standardkonfiguration mit 500 so groß gewählt, dass der praktische Unterschied zu einer infinite-horizon-Betrachtung nur gering ist.

Das hier eingeführte Modell simuliert einen Recommerce-Markt mit zwei konkurrierenden Anbietern. Der Markt soll fair und symmetrisch sein, was bedeutet, dass beide Anbieter die gleichen Produkte verkaufen, gleiche Beliebtheit haben, abwechselnd und für die gleiche Länge die Preise setzen und die gleichen Kosten haben. Der erste Anbieter erneuert seine Preise stets zu Beginn jedes Schrittes, sein Wettbewerber nach der Hälfte des Schrittes. In beiden dieser Phasen kommen gleich viele Kunden. Damit können die Gewinne der beiden Anbieter direkt verglichen und auf die Güte der Preisstrategien geschlossen werden. Zum Markov-Entscheidungsprozess wird der Markt, wenn er aus der Sicht eines der beiden Anbieter betrachtet wird, während der andere Anbieter ein Teil der

Umgebung ist. Das Verhalten des Konkurrenten kann deterministisch oder stochastisch sein. Soll die Markov-Eigenschaft erfüllt werden, darf es sich aber während des Trainings und der Analyse nicht ändern.

Bei der folgenden Notation werden die Anbieter mit 1 und 2 indiziert. Dabei ist 1 der Anbieter, aus dessen Sicht das Marktgeschehen abläuft. Die in dieser Arbeit verwendeten Standardwerte für die Marktparameter sind im Anhang in der Tabelle 5.1 aufgeführt.

Für den Anbieter $i \in \{1, 2\}$ müssen in Abhängigkeit des Zustandes drei Preise gesetzt werden:

1. für gebrauchte Produkte der Produktlinie ($p_{i,gebraucht}$),
2. für diese Produkte in neu ($p_{i,neu}$) sowie
3. den Rückkaufpreis ($p_{i,re}$), für den von einem Eigentümer gebrauchte Produkte zurückgekauft werden.

Alle drei Preise bewegen sich zwischen 0 und dem maximal möglichen Preis $p_{max} \in \mathbb{N}$. Damit ist der (stetige) Aktionsraum als $\mathcal{A}_{\mathbb{R}} = [0,p_{max}]^3$ definiert. Alternativ lässt sich der Aktionsraum auch mit diskreten Preisstufen als $\mathcal{A}_{\mathbb{N}} = \{0,1,\ldots,p_{max}\}^3$ definieren. Die diskrete Formulierung schränkt deutlich stärker ein, ermöglicht aber (prinzipiell) exakte Lösungen, wie sie in Abschnitt 3.2 vorgestellt werden. Zudem unterscheiden sich die Reinforcement-Learning-Verfahren für diskrete und stetige Aktionsräume.

In jedem Schritt besucht eine gerade natürliche Zahl k an Kunden den Marktplatz. Dabei treffen immer $k/2$ Kunden auf den Marktzustand, in dem der erste Anbieter seine Preise aktualisiert hat und die alten Preise des zweiten Anbieters anliegen. Die anderen $k/2$ Kunden bekommen die unveränderten Preise des ersten Anbieters und die neuen Preise des zweiten Anbieters. Die Kunden bewerten für sich die Angebote und treffen eine von fünf möglichen Entscheidungen. Entweder kaufen sie kein Produkt, oder das gebrauchte oder neue bei einem der Anbieter. Das Kaufverhalten ist aber keinesfalls homogen. Es wird stark durch die Preise beeinflusst, aber unterschiedliche Kunden haben unterschiedliche Erwartungen, unterschiedliche Konsumgewohnheiten oder unterschiedliche Einstellungen zu Nachhaltigkeit. Deshalb wird eine Wahrscheinlichkeitsverteilung des Kaufverhaltens für die fünf Optionen erstellt, aus der dann für einzelne Kunden gesampelt wird. Das geschieht, indem für die einzelnen Optionen Präferenzen aufgestellt werden, aus denen dann per Softmax die Zähldichte der Wahrscheinlichkeitsverteilung berechnet wird. Dabei wird die Präferenz für das Nichtkaufen auf 1 festgelegt. Die Präferenz, das Neuprodukt bei Anbieter $i \in \{1, 2\}$ zu kaufen, lautet

$$\sigma_{kunde,neu,i} = \frac{10}{p_{i,neu} + 1} - \exp\left(p_{i,neu} + 1 - 0{,}8p_{max}\right). \tag{1.1}$$

Das Kaufen eines Gebrauchtproduktes bei Anbieter $i \in \{1, 2\}$ wird auf

$$\sigma_{kunde,gebraucht,i} = \frac{5{,}5}{p_{i,gebraucht} + 1} - \exp\left(p_{i,gebraucht} + 1 - 0{,}5p_{max}\right) \tag{1.2}$$

festgelegt. Die Addition der Preise mit eins verhindert Nulldivision.

Die Präferenz für eines der Angebote basiert also hauptsächlich auf dem Preis-Leistungs-Verhältnis. Gebrauchten Produkten wird dabei 55% der Wertschätzung im Vergleich zu Neuprodukten entgegengebracht. Zusätzlich sinkt die Präferenz deutlich, wenn der Preis $0{,}5p_{max}$ bei Gebraucht- und $0{,}8p_{max}$ bei Neuware überschreitet. Das modelliert die grundsätzlich beschränkte Zahlungsbereitschaft der Kunden für diese Produkte und verhindert einen Preiszyklus nach oben. Wie gefordert, werden die Produkte aller Anbieter als gleichwertig betrachtet. Fallen die Präferenzen für die Kaufoptionen niedrig aus, so wird durch die Softmax-Funktion eine hohe Wahrscheinlichkeit auf das Nichtkaufen entfallen.

Für jeden der beiden Anbieter enthält das Marktmodell einen Zähler für die Anzahl der im Lager befindlichen Gebrauchtprodukte, $n_{lager} \in \mathbb{N}$. Das Lager hat ein Fassungsvermögen von m_{lager} Produkten. Der initiale Lagerstand wird für beide Anbieter unabhängig und uniform zufällig zwischen 0 und m_{lager} gewählt. Wenn mehr Produkte zurücklaufen, werden diese verworfen und der Lagerstand bleibt beim Maximum. Kauft ein Kunde ein gebrauchtes Produkt, obwohl das Lager leer ist, so muss der Anbieter Entschädigung zahlen. Die Entschädigung ist auf $2p_{max}$ pro enttäuschtem Kunden festgelegt. Allerdings verursacht auch die Lagerhaltung Kosten: Am Ende eines jeden Schrittes werden Kosten von p_{lager} pro eingelagertem Element berechnet. Das wiederum treibt die Anbieter dazu, ihr Lager nicht voller als nötig zu halten. Es stellt sie vor die Herausforderung, mit Rückkauf- und Gebrauchtpreis nicht nur auf ihren Konkurrenten zu reagieren, sondern auch ihr Lager zu regulieren.

Weiterhin verwaltet der Markt einen Zähler für die Anzahl der Produkte, die sich in Zirkulation befinden (n_{zirk}). Er wird uniform zufällig zwischen 0 und $5m_{lager}$ initialisiert. Der Verkauf eines Neuproduktes erhöht diesen Zähler um eins. Im Gegensatz dazu werden bereits gebrauchte Produkte nicht noch einmal weiterverkauft, weshalb ihr Verkauf den Zähler nicht erhöht.

Je mehr Kunden im Besitz eines Produktes sind, desto mehr dieser Eigentümer erwägen, ihr Produkt nicht länger besitzen zu wollen. In der Kreislaufwirtschaft stehen sie dann vor vier Optionen: Sie können ihr Produkt weiter behalten, es wegwerfen oder an einen der beiden Anbieter zurückverkaufen. Die Möglichkeit des Rückkaufs ist nicht auf den Anbieter beschränkt, der ihnen das Produkt ursprünglich verkauft hat, sondern es findet Wettbewerb um den Rückkauf statt. Insgesamt stellen sich in jedem Schritt $c \cdot n_{zirk}$ Eigentümer dieser Entscheidung, wobei c ein Parameter ist, für den $0 < c \leq 1$ gilt. Nach dem gleichen Verfahren wie bei der Kaufentscheidung der Kunden wird das Eigentümerverhalten mit einer diskreten Wahrscheinlichkeitsverteilung modelliert, aus der dann für jeden der $c \cdot n_{zirk}$ Eigentümer gesampelt wird. Die Präferenz für das Halten eines Produktes wird auf 1 normiert. Die Wegwurfpräferenz

$$\sigma_{eigentuemer,wegwurf} = \min_{i \in \{1,2\}} \left(\min\left(p_{i,neu}, p_{i,gebraucht}\right)\right) - \max_i p_{i,re} \tag{1.3}$$

motiviert sich daher, dass die Eigentümer ihr Produkt wenig wertschätzen, wenn die Differenz zwischen dem niedrigsten Kaufpreis und dem höchsten Rückkaufpreis hoch ist. Die Präferenz für das Zurückverkaufen an den Anbieter $i \in \{1, 2\}$ ist direkt sein Rückkaufpreis, also

$$\sigma_{eigentuemer,rueckkauf,i} = p_{i,re}. \tag{1.4}$$

Anbieter, die mehr bieten, sind also beliebter. Die Zähldichte der Wahrscheinlichkeitsverteilung ergibt sich dann wieder als Softmax dieser Präferenzen.

Die Belohnung, die der Agent in einem Schritt erhält, ist die Differenz aus Einnahmen und Ausgaben des Agenten in diesem Schritt. Mit dem Verkauf neuer oder gebrauchter Ware erzielt der Agent Einnahmen. Ausgaben sind nicht nur der bereits erklärte Rückkaufpreis und die Lagerkosten und -strafen, sondern auch der Einkaufspreis für Neuware. Pro Neuverkauf zahlt der Agent $p_{einkauf}$. Damit muss für Gewinn auf dem Bereich der Neuware auf jeden Fall $p_{1,neu} > p_{einkauf}$ gelten.

Nun kann der Zustandsraum des Markov-Entscheidungsprozess $\mathcal{S}$ definiert werden. Er ist sechsdimensional und enthält Folgendes:

1. die Anzahl der gerade zirkulierenden Elemente,
2. die Anzahl der Elemente im eigenen Lager,
3. den Preis des gebrauchten Produktes des Konkurrenten ($p_{2,gebraucht}$),
4. den Neupreis des Konkurrenten ($p_{2,neu}$),
5. den Rückkaufpreis des Konkurrenten ($p_{2,re}$) und
6. den Lagerstand des Konkurrenten.

Diese Werte beeinflussen das Verhalten der Kunden und des Konkurrenten. Sie müssen gegeben sein, damit die sogenannte Markov-Eigenschaft erfüllt ist. Diese besagt, dass die Wahrscheinlichkeiten der Zustandsübergänge und der Belohnungen nur vom aktuellen Zustand und der gewählten Aktion abhängen und damit insbesondere stochastisch unabhängig von den zuvor besuchten Zuständen und Aktionen sind. Der Schrittzähler ist wegen der *infinite-horizon*-Optimierung nicht Teil des Zustandes.

Dass die für den Zustandsraum ausgewählten Dimensionen das künftige Verhalten des Systems beeinflussen, ist durch die Definition des Marktverhaltens klar. So werden Gebraucht- und Neupreis des Konkurrenten für die erste Hälfte dieses Schrittes direkt für das Kundenverhalten verwendet. Der Rückkaufpreis beeinflusst das Verhalten der Eigentümer, deren Anzahl wiederum durch die Anzahl der Elemente in Zirkulation gegeben ist. Der eigene Lagerstand erklärt durch Lagerkosten und -strafen einen Teil des eigenen Gewinnes, genau wie der Lagerstand des Konkurrenten einen Teil seines Verhaltens erklärt.

Obwohl die Anzahl der zirkulierenden Elemente und der Lagerstand des Konkurrenten für die Markov-Eigenschaft notwendig sind, bereitet die Messung dieser beiden Werte in der Praxis Probleme. Der Konkurrent lässt sich vermutlich nicht ins Lager schauen und die Anzahl der Elemente in Zirkulation kann allenfalls geschätzt werden. Deshalb wird in Abschnitt 4.7 die Frage diskutiert, ob die untersuchten Verfahren auch mit partiellen Beobachtungen zufriedenstellende Ergebnisse liefern.

Zuletzt muss noch beleuchtet werden, dass für den beschriebenen Markt keine weiteren Größen beobachtet werden müssen und somit die Markov-Eigenschaft tatsächlich erfüllt ist. Das Verhalten des Marktes ist vollständig definiert und kommt mit den genannten Größen aus. Auch der regelbasierte Wettbewerber in Abschnitt 1.3 verwendet für sein

Verhalten nur die aufgezählten Variablen. Messgrößen wie die vorher gesetzten Preise, die Anzahl der weggeworfenen Objekte oder die Anzahl der bisherigen Verkäufe könnte man im Zustand vermuten (sie sind ja schließlich auf der grafischen Visualisierung des Marktes), beeinflussen aber nicht das künftige Verhalten. Eine Gedächtnisfunktion der Kunden (und dann notwendigerweise auch der Agenten) über vergangenes Verhalten ist eine mögliche Erweiterung, die das Modell praxisnäher bringen würde. So könnten Kunden die vergangene Preispolitik in ihre Entscheidung mit einbeziehen und Anbietern ein höheres oder niedrigeres Renommee zubilligen. Das erfordert jedoch weitere Techniken, die nicht im Rahmen dieser Arbeit betrachtet werden.

1.3 Regelbasierte Strategien – Benchmark und Wettbewerber

Der heutige Industriestandard für das Dynamic Pricing ist die Festlegung von Regeln durch menschliche Experten. Sie erstellen Funktionen für die Einpreisung der verkauften Waren, denen prinzipiell die gleichen Argumente wie den RL-Agenten zur Verfügung stehen. Beim Design dieser Preisstrategien müssen die Herausforderungen des Marktes bereits erkannt sein und für verschiedene Situationen Lösungen vorweggenommen werden. Für den hier definierten Markt lassen sich einige Überlegungen anstellen:

- Die Kunden bevorzugen niedrigere Preise. Es erscheint also plausibel, bei Neu- und Rückkaufpreis die Konkurrenz zu unterbieten.
- Es sollte allerdings nur bis zu einem bestimmten Preis unterboten werden, weil ansonsten die niedrige Rendite trotz höherer Verkaufszahlen zu wenig Gewinn ermöglicht. Insbesondere muss der Verkaufspreis über dem Einkaufpreis liegen.
- Das Lager sollte möglichst leer gehalten werden, um dort Kosten zu sparen. Allerdings müssen genug Produkte vorrätig sein, um die Kundenwünsche zu erfüllen. Mit einem niedrigen Rückkaufpreis wird das Lager langsamer gefüllt, mit einem niedrigen Gebrauchtpreis schneller geleert. Umgekehrt wird ein höherer Rückkaufpreis und ein höherer Gebrauchtpreis gesetzt, falls das Lager gefüllt werden soll.
- Der Rückkaufpreis sollte stets so niedrig wie möglich gehalten werden.

Im Anhang ist eine regelbasierte Strategie definiert, die diesen Ideen folgt. Sie erreicht in Tests gegen sich selbst Ergebnisse von 1200 bis 1400 bei $n_{ep} = 500$. Es gelingt ihr, Gewinn zu erwirtschaften und den Lagerstand bei etwa 10 Produkten zu halten. Diese Ergebnisse fallen niedriger aus als die der RL-Agenten in der Analyse, und auch niedriger als die Ergebnisse dieser regelbasierten Strategie, wenn sie gegen andere Strategien spielt. Das liegt an ihrem permanenten Unterbieten, womit sie beim Spiel gegen sich selbst unverzüglich die Preisuntergrenze und damit eine schlechte Rendite erreicht.

Weiteres Tuning der regelbasierten Strategien ist möglich, aber mit sehr hohem Aufwand verbunden. Es erfordert weiterhin die sehr genaue Kenntnis eines speziellen Marktszenarios. Dazu gehört insbesondere auch die Preisstrategie des Wettbewerbers. Die Güte

einer Preisstrategie lässt sich nämlich nur in Bezug auf die Gegnerstrategie bewerten. So ist Unterbieten eine schlechte Strategie, wenn der Konkurrent ebenfalls konsequent unterbietet. Verhält sich der Konkurrent aber weniger aggressiv, so ist Unterbieten eine sehr leistungsstarke Strategie.

2 Related Work

Während Pricing ein altes Problem ist, war die Forschungsaktivität zu Pricing mit intensivem Wettbewerb und gerade auf Online-Marktplätzen bis vor einigen Jahren eher gering, wie die Übersichtsarbeit [GB22] aus dem Jahre 2022 zeigt. Demnach dominierten Monopolszenarien oder Szenarien mit starken, einschränkenden Annahmen.

Im Bereich der algorithmen- und datengetriebenen Preisoptimierung lässt sich die Schwierigkeit in zwei separate Herausforderungen unterteilen: Das Schätzen des Kundenverhaltens und das Optimieren des Pricings bei bekanntem oder erlerntem Kundenverhalten. In [SB18a] werden diese Probleme einzeln gelöst. Zunächst werden Regressionsverfahren angewandt, um das Kundenverhalten zu erlernen. Anschließend wird ein approximatives Dynamic-Programming-Verfahren vorgestellt, um trotz des »Fluchs der Dimension« Preisstrategien optimieren zu können. Die mit diesem Verfahren optimierten Preisstrategien konnten erfolgreich auf Amazon Marketplace angewandt werden.

RL für dynamisches Pricing wurde in den letzten Jahren für spezielle Märkte angewandt, etwa den Energiemarkt bei [Kim+16]. Dort bestand die Aufgabe darin, mittels dynamischer Preise die Nachfrage von Energieverbrauchern gegenüber den Produzenten zu regulieren. Es wurden tabellenbasierte Q-Learning-Verfahren angewandt und spezielle Anpassungen verglichen. Dabei wurde auch ein Multi-Agent-Setup vorgestellt, das erheblich bessere Ergebnisse als reines Q-Learning lieferte.

In [RO15] wurde RL verwendet, um Preisstrategien für einen Markt zu optimieren, bei dem verderbliche Ware innerhalb eines endlichen Horizonts verkauft werden muss. Dabei wurde nicht nur eine Produktlinie, sondern sehr viele betrachtet. Zwischen dem Verkauf unterschiedlicher Produktlinien gibt es komplexe Abhängigkeiten, die in klassischen Pricingansätzen nicht beachtet wurden, aber mit RL mit einbezogen werden können. Als Technologie wurde Tabular-Q-Learning mit Eligibility Traces verwendet.

Weiterhin wurde bei [Kra+19] RL für die Optimierung von Versicherungsprämien eingesetzt. Es wurde zwar ein domänenspezifisches Modell gewählt, das aber ebenfalls eine aktuelle Marktsituation als Zustand und Preise als Aktionen verwendet. Dabei wurden zwei Rewardfunktionen vorgeschlagen, wobei die eine nur den Gewinn bewertet, und die andere zusätzlich auch Kundenbindung. Das genutzte RL-Verfahren heißt VQQL. Es diskretisiert erst Zustands- und Aktionsraum, und wendet anschließend Tabular-Q-Learning an. Bei dieser Forschungsarbeit wurde mit einer großen spanischen Versicherungsgesellschaft zusammengearbeitet und deren Echtweltdaten verwendet.

Ebenfalls mit Tabular-Q-Learning wurde in der Arbeit [Con+22] zur Optimierung von Cloudpreisen gearbeitet. In diesem Anwendungsfall geht es darum, dass ein Cloudprovider Hardware beschaffen muss. Damit muss er Kunden Clouddienstleistungen anbieten, dabei so viel Geld wie möglich erzielen, aber auch die Kundenzufriedenheit aufrechterhalten. In der Auswertung ihrer Experimente konnten knapp 20% mehr Profit erzielt werden als mit state-of-the-art regelbasierten Systemen.

Mit einem sehr ähnlichen Marktmodell wie bei [SB18a] wurde in der Arbeit [KS22] RL

eingesetzt, um unter Wettbewerbsbedingungen eine Preisstrategie zu optimieren. Untersucht wurde eine *Linear Economy* im Duopol und Oligopol. Im Gegensatz zu den vorher gelisteten Arbeiten wurde hier allerdings Deep-Reinforcement-Learning eingesetzt, wie es ebenfalls Gegenstand dieser Arbeit ist. Als RL-Verfahren wurden Deep-Q-Learning (mit diskretem Aktionsraum) und das neue, beliebte Soft-Actor-Critic-Verfahren ausprobiert. Dabei schnitt Soft Actor Critic erheblich besser ab, aber die DQNs konnten die Aufgabe ebenfalls bewältigen. Es wurde weiterhin festgestellt, dass im Duopol gelegentlich automatisch Kartelle gebildet werden, wenn die Kaufbereitschaft der Kunden bei höheren Preisen nicht begrenzt wird.

Auch für den Anwendungsfall der intelligenten Energieversorgung, der mit tabellenbasierten Methoden bereits bei [Kim+16] behandelt wurde, gibt es Ansätze mit Deep-Reinforcement-Learning. So haben [Moc+19] Deep-Q-Learning und ein Deep-Policy-Gradient-Verfahren (REINFORCE) auf dieses Problem angewandt. Die Resultate der Deep-Learning-basierten Verfahren wurden mit denen von Tabular Q-Learning verglichen. Dabei wurden die Vorteile der Deep-Learning-Verfahren, insbesondere Skalierbarkeit bei hochdimensionalem Beobachtungsraum, hervorgehoben.

Während es bisher kaum detaillierte Analysen der aktuelleren RL-Verfahren im Bereich des Dynamic Pricing gibt, so wurden die Algorithmen PPO und SAC auf anderen interessanten Problemstellungen verglichen. Bei [Lar+21] wurden PPO, DDPG, TD3 und SAC in der Stable-Baselines-Implementierung bei autonomer Schifffahrt verglichen. Die zu lösende Aufgabe war es, mit einem Schiff Routen zwischen Häfen zu fahren, ohne dabei mit Hindernissen zu kollidieren. Zur Analyse wurden verschiedene Situationen getestet, bei denen die Algorithmen jeweils unterschiedlich gut abschnitten. Nur Proximal Policy Optimization konnte jede Umgebung besser lösen als alle anderen Algorithmen und hat damit im Vergleich mit Abstand am besten abgeschnitten.

Im Bereich der unternehmensnahen Software wurden in [ASM21] die Algorithmen A2C, DDPG, TD3, PPO und SAC auf einem Problem verglichen, bei dem es darum geht, Lagerstände entlang einer Lieferkette zu optimieren. Diese Lieferkette ist von Unsicherheiten bei Bedarf- und Vorlaufzeiten geprägt. Im Ergebnis übertrafen SAC und PPO die anderen Algorithmen, während sie selbst ähnlich gut abschnitten.

Zu dem konkreten Szenario, Pricing in Recommerce-Märkten mit RL-Algorithmen zu optimieren, gibt es nach dem besten Wissen noch keine wissenschaftlichen Publikationen. Allerdings wird künstlicher Intelligenz und maschinellem Lernen eine hohe Bedeutung auf dem Weg zu einer nachhaltigeren Wirtschaft beigemessen. So wird künstliche Intelligenz in der Konzeptstudie [Jos+20] als »Enabler« für eine Kreislaufwirtschaft dargestellt. Die Kreislaufwirtschaft wird in den Kontext der großen globalen und politischen Herausforderungen gestellt, Ressourcen- und Energienutzung zur Bekämpfung der globalen Klimaerwärmung zu optimieren. Diese Kreislaufwirtschaft werde an den entscheidenden Stellen durch KI-Technologie betrieben werden.

3 Hintergrund

3.1 MDPs lösen: Zustands- und Aktionswerte

Bei der Einführung der Notation wird sich am Lehrbuch [SB18b] orientiert. Gegeben sei ein Markov-Entscheidungsprozess $(\mathcal{S}, \mathcal{A}, p)$ (kurz MDP). Dabei ist $\mathcal{S}$ der Zustands- und $\mathcal{A}$ der Aktionsraum. Wir nehmen an, dass alle Belohnungen (engl. Reward) aus einer (endlichen) Menge $\Lambda \subset \mathbb{R}$ stammen. Dann ist für einen Zustand $s \in \mathcal{S}$ und eine Aktion $a \in \mathcal{A}$ die Wahrscheinlichkeitsdichte, in einen Zustand $s' \in \mathcal{S}$ überzugehen und dabei die Belohnung $r \in \Lambda$ zu erhalten, als $p_{s,a}(s', r)$ gegeben.

Zentraler Begriff beim Lösen eines MDPs ist die sogenannte Policyfunktion. Sie kann deterministisch sein und als $\pi_{det} : \mathcal{S} \to \mathcal{A}$ formalisiert werden. Weiterhin können Policies stochastisch sein. Die Policyfunktion ist dann eine Zähl- oder Wahrscheinlichkeitsdichtefunktion der Form $\pi_{stoch} : \mathcal{S} \times \mathcal{A} \to [0, \infty)$. Mit der Policy, den Zustandsübergängen, Belohnungen und dem Startzustand S_0 ergibt sich dann eine Episode als $\tau = \left(S_0, A_0, R_0, S_1, A_1, R_1, \ldots, S_{n_{ep}-1}, A_{n_{ep}-1}, R_{n_{ep}-1}\right)$. Dabei sind für jedes $i \in \{0, \ldots, n_{ep} - 1\}$ stets S_i, A_i, R_i Zufallsvariablen, die für den i-ten Schritt Zustand, Aktion und Belohnung angeben. Mit einem optionalen Diskontierungsfaktor $\gamma \in [0, 1]$ werden die addierten Belohnungen

$$G_t := \sum_{i=t}^{n_{ep}-1} \gamma^{i-t} \cdot R_i \tag{3.5}$$

als Return (von Schritt t aus) bezeichnet. In einer infinite-horizon-Formulierung muss für eine endliche Summe notwendigerweise $\gamma < 1$ gewählt werden. Ziel ist, den Return der ganzen Episode, also G_0 zu maximieren.

Für einen Zustand $s \in \mathcal{S}$ unter einer Policy π gibt die Funktion

$$V_\pi(s) := \mathbb{E}_\pi[G_t | S_t = s] \tag{3.6}$$

den sogenannten Zustandswert an. Er drückt die (diskontierte) Summe aller Rewards aus, die nach dem Besuch des Zustands s noch erwartet werden, wenn die Aktionsauswahl mit der Policy π geschieht. Analog wird für den Zustand $s \in \mathcal{S}$ und eine Aktion $a \in \mathcal{S}$ unter einer Policy π der Aktionswert als

$$Q_\pi(s, a) := \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{3.7}$$

definiert, also die erwartete (diskontierte) Summe an Belohnungen nach dem Besuch von s und der Wahl von Aktion a, wenn danach der Policy π gefolgt wird. Zustands- und Aktionswerte sind zentrale Begriffe im Reinforcement Learning, die in vielen Algorithmen verwendet werden. Das Ziel, den Return G_0 zu maximieren, lässt sich auch äquivalent als Maximierung des Zustandswertes von S_0 formulieren.

3.2 Exakte Lösungen mittels Dynamic Programming

Ein klassisches Verfahren für das Lösen von Markov-Entscheidungsprozessen ist die dynamische Programmierung. Das setzt voraus, dass sowohl Zustands- als auch Aktionsraum endlich und die Übergangs- und Belohnungswahrscheinlichkeiten p bekannt sind. Anschließend wird eine angenäherte Funktion für die Zustandswerte V^* als Array gespeichert und mit beliebigen Werten initialisiert. Dann werden rundenweise für alle Zustände $s \in \mathcal{S}$ ihre Einträge in dieser Tabelle mithilfe der Bellman-Gleichung

$$V^*(s) \leftarrow \max_{a \in \mathcal{A}} \left(\sum_{s' \in \mathcal{S}, r \in \Lambda} p_{s,a}(s', r)(r + \gamma V^*(s')) \right) \tag{3.8}$$

als Berechnungsvorschrift aktualisiert. Unabhängig davon, ob diese Berechnung in-place oder mit zwei alternierenden Arrays ausgeführt wird, konvergiert sie zur Zustandswertefunktion einer optimalen Policy. Bei [SB18b] ist das Verfahren als Wertiteration zu finden. Es ist in analoger Weise mit Aktionswerten umsetzbar. Die optimale Policy ist dann die deterministische, sogenannte Greedy-Policy. Sie wählt in jedem Zustand das Argmax der Aktionen.

Dynamische Programmierung hat allerdings starke Forderungen, die in der Praxis schwer erfüllbar sind. Erstens müssen die Wahrscheinlichkeitsverteilungen der Belohnungen und Zustandsübergänge bekannt sein. Die sind aber unter anderem bei dem in dieser Arbeit untersuchten Markt nicht explizit vorhanden. Eine umfangreiche mathematische Analyse wäre erforderlich.

Dieses Problem lässt sich allerdings noch im Rahmen abgewandelter DP-Verfahren lösen. Grundidee dieser sogenannten Monte-Carlo-Verfahren ist es, eine Tabelle für die Schätzung der Aktionswerte zu speichern und damit eine Greedy-Policy zu erstellen. Es werden nach und nach Episoden generiert und mit den empirischen Daten die Schätzung der Aktionswerte verbessert. In jedem Schritt wird dabei mit einer Wahrscheinlichkeit von $1 - \varepsilon$ dieser Greedy-Policy gefolgt, und für die Exploration mit einer Wahrscheinlichkeit von ε eine zufällige Aktion ausgewählt. Weiteres dazu kann in [SB18b] nachgelesen werden.

3.3 Approximation als Ausweg bei zu großem Zustands- und Aktionsraum

Doch auch Monte-Carlo-Verfahren können auf den beschriebenen Markt nicht angewandt werden. In jedem Fall müsste die Formulierung mit dem diskreten Aktionsraum gewählt werden. Hindernis ist aber auch hier der große Zustands- und Aktionsraum: soll das Monte-Carlo-Verfahren angewandt werden, so ist dafür eine Tabelle der Größe $|\mathcal{S} \times \mathcal{A}|$ erforderlich. Bei den in Tabelle 5.1 genannten Standardwerten müsste die Tabelle $50 \cdot 500 \cdot 100 \cdot 10^3 \cdot 100 \cdot 10^3 = 2.5 \cdot 10^{14}$ Einträge haben. Obwohl dieser Wert bereits jeden verfügbaren Speicher sprengt, ist die Anzahl der unterschiedlichen Preisstufen mit 10 für realistische Verhältnisse recht niedrig angesetzt. Tatsächlich liegt der benötigte Speicher in $O(p_{max}^6)$, was eine Skalierung unmöglich macht. Weil dies sogar nur die

Betrachtung des Speichers war – bis Konvergenz erreicht ist, müsste jeder Zustand sehr oft durchlaufen werden – muss geschlussfolgert werden, dass eine exakte Lösung mit Tabellen nicht realisierbar ist.

Approximationen sind also unumgänglich. Sie beruhen auf der Einsicht, dass der Zustandsraum zwar sehr groß ist, aber viele Zustände sich sehr ähnlich sind. Dadurch können Modelle mit deutlich weniger Parametern als Zuständen diese gewünschten Funktionen mit für praktische Zwecke ausreichender Genauigkeit annähern. Beispiele und Erklärungen dazu sind in [Lap20] zu finden. Alle folgenden Verfahren nutzen sogenannte künstliche neuronale Netze, die am weitesten verbreitete Technologie, um komplexe nichtlineare Funktionen zu approximieren.

3.4 Q-Learning-Verfahren

Ein weit verbreitetes Verfahren zum Lösen von Markov-Entscheidungsprozessen ist das sogenannte Q-Learning. Die Idee ist hierbei, wie in dem in Abschnitt 3.2 beschriebenen Monte-Carlo-Verfahren, Aktionswerte einer optimalen Policy zu schätzen. Dazu wird als Berechnungsvorschrift die Bellman-Gleichung verwendet. Allerdings werden nicht unbedingt vollständige Episoden betrachtet, sondern für den Aktionswert nach dem Folgezustand wird die momentane Schätzfunktion verwendet. Diese Technik, zur Verbesserung von Schätzwerten rekursiv bisherige Schätzwerte zu verwenden, wird Bootstrapping genannt und wird laut [SB18b] als wichtige Technologie angesehen. Nach dem Training wird wieder die Greedy-Policy verwendet. Während des Trainings wird mit Wahrscheinlichkeit ε von ihr abgewichen, und für bessere Expolration eine zufällige Aktion gewählt. Das ε wird während des Trainings abgesenkt.

Dieses Verfahren lässt sich mit Tabellen oder neuronalen Netzwerken umsetzen. Mit neuronalen Netzen wird es *Deep-Q-Learning* genannt. *Deep-Q-Networks* (DQNs) sind für diskrete Aktionsräume entworfen worden. Die Schätzung der Aktionswertefunktion wird dabei so umgesetzt, dass an den Input des Netzes nur der Zustand angelegt wird und dann in einer Berechnung auf so viele Ausgabeneuronen abgebildet wird, wie es Aktionen gibt. So erhält man bei nur einem der aufwendigen Läufe durch das neuronale Netz bereits alle Aktionswerte auf einmal. Die Wahl des Maximums und des Argmax sind dann unmittelbar möglich. Das Training findet iterativ statt, indem die bisherigen Schätzwerte des Netzes mit Zustandsübergängen (s, a, r, s') aus der gesammelten Erfahrung $\mathcal{D}$ in Richtung des Zielwertes $r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$ bewegt werden. In den heutigen Deep-Learning-Frameworks wie PyTorch [Pas+19] wird üblicherweise mit einer Verlustfunktion gearbeitet, die dann automatisch differenziert wird. Mit einem Optimizer wie Stochastic Gradient Descent oder Adam [KB14] wird dieser Verlust dann mithilfe des Gradienten minimiert. Diese Verlustfunktion beim Deep-Q-Learning lautet für eine Schätzfunktion der Aktionswerte mit den Parametern ψ

$$L_{\mathcal{D}}(\psi) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_\psi(s, a) - \left(r + \gamma \max_{a' \in \mathcal{A}} Q_\psi(s', a') \right) \right)^2 \right]. \tag{3.9}$$

Erfolgreiche Q-Learning-Umsetzungen verwenden einen Experiencebuffer, in dem die Er-

fahrung $\mathcal{D}$ gespeichert wird. Weiterhin speichern sie zwei Netze: neben dem, das trainiert wird, auch ein sogenanntes Zielnetz, was in der Schätzung des Zielwertes verwendet wird. Das verhindert ungewollte Oszillationen und Instabilität, wie in [Mni+13] beschrieben. Zahlreiche Erweiterungen zu Deep-Q-Networks sind verbreitet und verbessern das Training erheblich. Die verbreitetesten von ihnen sind in [Hes+17] aufgeführt. Für die gleich folgenden Verfahren ist insbesondere die Erkenntnis wichtig, dass Q-Learning-Verfahren zur systematischen Überschätzung der Aktionswerte neigen. Näheres dazu steht in [HGS15].

Q-Learning-Verfahren sind Off-Policy-Algorithmen. Das bedeutet, dass ihr Training auch mit Zustandsübergängen stattfinden kann, die nicht von der aktuellen Policy erzeugt wurden. Der Grund dafür ist, dass bei einer perfekten Policy für **jedes** $s \in \mathcal{S}$ und $a \in \mathcal{A}$ die Bellman-Gleichung

$$Q(s,a) = \sum_{s' \in \mathcal{S}, r \in \Lambda} p_{s,a}(s', r) \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') \right) \tag{3.10}$$

erfüllt ist. Das ist zwar bei Approximationslösungen nicht erreichbar, die Minimierung des Erwartungswert über die quadrierten Differenzen zeigt aber die Entfernung von diesem Ziel.

3.5 DDPG und TD3

Die Verallgemeinerung auf stetige Aktionsräume ist bei Deep-Q-Networks nicht ohne grundlegende Designänderungen möglich. Allerdings gibt es ein RL-Verfahren, Deep Deterministic Policy Gradient (DDPG), das in [Sil+14] publiziert wurde und als Verallgemeinerung von Q-Learning auf Umgebungen mit stetigen Aktionsräumen angesehen werden kann. Es handelt sich dabei ebenfalls um ein Off-Policy-Verfahren. Ein neuronales Netz mit den Parametern ψ berechnet die Aktionswerteschätzfunktion $Q_\psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (jetzt muss die Aktion mit in die Eingabe). Um an die Policy zu gelangen, ist es dabei aber kaum möglich, das tatsächliche Maximum dieser Funktion Q_ψ zu bestimmen. Für neuronale Netze ist keine analytische Lösung möglich und numerische Verfahren sind viel zu aufwendig für die vielen benötigten Trainingsschritte. Die Lösung besteht in einem weiteren neuronalen Netz mit den Parametern ϕ, das explizit die deterministische Policyfunktion $\pi_\phi : \mathcal{S} \to \mathcal{A}$ berechnet, ein Actor-Netz. Es gibt wieder einen Experiencebuffer für die gesammelte Erfahrung $\mathcal{D}$. Das Training des Q-Netzes findet wie beim DQN statt, außer dass für den Zielwert die Policyfunktion verwendet wird. Die dazugehörige Verlustfunktion lautet sehr ähnlich zu der von Deep-Q-Learning

$$L_\mathcal{D}(\psi) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_\psi(s,a) - \left(r + \gamma Q_{\psi'}(s', \pi_\phi(s')) \right) \right)^2 \right]. \tag{3.11}$$

Dabei wird bei DDPG ein Zielnetz mit Parametern ψ' für die Schätzung der Policyfunktion verwendet, was nach jedem Optimierungsschritt mit dem Hyperparameter $\rho \in [0,1]$ und der Polyak-Mittelung $\psi' \leftarrow (1-\rho)\psi' + \rho\psi$ aktualisiert wird. Zur Optimierung der Policyfunktion in dem Sinne, dass sie für einen Zustand möglichst das

Argmax berechnet, wird Gradientenanstieg verwendet. Das wird dadurch möglich, dass beide neuronalen Netze differenzierbar sind und der Aktionsraum stetig ist. So kann der Gradient $\nabla_\phi \mathbb{E}_{s\sim\mathcal{D}}\left[Q_\psi(s, \pi_\phi(s))\right]$ durch die automatische Differenzierung des Deep-Learning-Frameworks berechnet und ein Optimierungsschritt in die Richtung des größten Anstiegs vorgenommen werden.

Nachfolgende Analysen haben ergeben, dass die systematische Überschätzung der Funktionswerte bei DDPG sich besonders stark auf die Leistung auswirkt. In der Arbeit [FHM18] wurden deshalb drei Verbesserungen für den DDPG-Algorithmus vorgeschlagen:

1. Es werden zwei separate Netze für die Aktionswertefunktion gespeichert. Für die Berechnung des Zielwertes wird immer das Minimum von beiden genommen.

2. Weil die gelernten Q-Funktionen teils lokal starke Unterschiede haben, die vom Gradientenanstieg ausgenutzt werden, wird bei der Berechnung des Zielwertes der Aktionsauswahl durch das Policynetz normalverteiltes Rauschen angefügt. Die Aktionen werden dabei selbstverständlich auf den Aktionsraum geclippt.

3. Im Gegensatz zu vielen DDPG-Implementierungen wird das Policy-Netz nur alle zwei Q-Netz-Updates trainiert.

Der entstehende Algorithmus wird als TD3 bezeichnet und übertrifft DDPG bei den dort aufgeführten Standardbenchmarks.

3.6 Policy-Gradient-Verfahren

Eine alternative Herangehensweise zu den wertebasierten Verfahren sind die Policy-Gradient-Verfahren. Während wertebasierte Verfahren zunächst Aktions- oder Zustandswerte schätzen, und daraus ihre Policy herleiten, berechnen Policy-Gradient-Verfahren direkt eine stochastische Policyfunktion π_ϕ mit Parametern ϕ, häufig ein neuronales Netz. Policy-Gradient-Verfahren verfolgen das gleiche in Abschnitt 3.1 formulierte Ziel, nämlich eine Policy zu finden, die den Zustandswert des Startzustandes und damit den erwarteten Return einer Episode maximiert. Das ursprüngliche Verfahren dieser Klasse heißt REINFORCE [Wil92] und ist aus dem Jahr 1992. Danach wurden zahlreiche Verbesserungen vorgenommen, die zu A2C und PPO führen. Die Grundidee bei diesen Algorithmen ist, direkt Gradientenanstieg für die Anpassung der Policy zu nutzen, sodass der Zustandswert des Startzustandes maximiert wird. Dazu gibt es ein bedeutendes theoretisches Resultat, das eine Möglichkeit aufzeigt, den Gradienten aus gesampelten Episoden leicht zu schätzen, das sogenannte Policy-Gradient-Theorem. Es lässt sich für den REINFORCE-Algorithmus als

$$\nabla_\phi V_{\pi_\phi}(S_0) \propto \mathbb{E}_{S_t,A_t}\left[G_t \nabla \log \pi_\phi(S_t, A_t)\right] \tag{3.12}$$

formulieren, wobei S_t und A_t als Zufallsvariablen den Zustand und die Aktion im t-ten Schritt angeben. G_t ist der Return vom t-ten Schritt aus. Der REINFORCE-Algorithmus sampelt nun Episoden unter Verwendung einer Policy und nutzt eine Schätzung dieses

Gradienten für Gradientenanstieg. Die Updateregel ist intuitiv: Aktionen, auf die ein positiver Return folgt, werden wahrscheinlicher gemacht und Aktionen mit negativem Return unwahrscheinlicher.

Doch REINFORCE hat für die praktische Anwendung einen offensichtlichen Nachteil: Die Bewertung, ob eine Aktion »gut« oder »schlecht« ist, richtet sich immer unmittelbar nach dem gemessenen Return. In einer Umgebung, die nur negative Rewards vergibt, wird der Gradient immer in die absteigende Richtung zeigen, auch für Aktionen, die in der Situation deutlich besser sind als andere. Abhilfe schafft hier die sogenannte Baseline, eine Funktion $b : \mathcal{S} \to \mathbb{R}$, die einen Vergleichswert für Zustände angibt. Sie ist sinnvollerweise eine Schätzung des Zustandswertes. Mit ihr wird bewertet, ob ein erreichter Return für einen Zustand gut oder schlecht ist. Anstelle des Returns wird dann für den Gradienten der Advantagewert $\hat{A}_t := G_t - b(S_t)$ verwendet. Dieser Algorithmus wird Advantage-Actor-Critic (A2C) genannt. Das ist eine Metapher für die zwei Netze Actor (berechnet die Policy) und Critic (schätzt den Zustandswert und ermöglicht damit die »Kritik« einer Aktion). In manchen Implementierungen teilen sich die beiden Netze vordere Schichten.

Um aus den Trainingsdaten den Advantagewert zu schätzen, gibt es mehrere Herangehensweisen. Eine Möglichkeit ist es, die Episoden bis zum Ende zu sampeln und die dabei gemessenen Werte zu verwenden. Das ist ein Schätzer ohne Bias, aber mit großer Varianz. Alternativ kann Bootstrapping verwendet werden, was die Varianz reduziert aber Bias einführt. Um einen Kompromiss zwischen Bias und Varianz zu finden, wurde der Generalized Advantage Estimator [Sch+15b] vorgeschlagen, der auch in den hier verwendeten Implementierungen verwendet wird.

Im Gegensatz zu den Q-Learning-Verfahren aus dem Abschnitt 3.4 handelt es sich bei dieser Gruppe um On-Policy-Algorithmen. Das bedeutet, dass für das Training nur Erfahrung verwendet werden kann, die mit der neuesten Policy gewonnen wurde. Training mit älteren Daten ist nicht möglich.

Ein Problem bei Actor-Critic-Methoden ist, dass bei der Änderungen der Policy oft zu große Schritte gemacht werden. Das kann die Besuchswahrscheinlichkeiten der Zustände radikal verändern. Dann stimmen nämlich die Zustandswertschätzungen nicht mehr, ein Grund für Instabilität. Um das Problem zu lindern, soll beim Lernen die Veränderung der Wahrscheinlichkeiten der Policy gering gehalten werden. Ein wichtiger Beitrag dazu war TRPO [Sch+15a], das die Kullback-Leibler-Divergenz zwischen alten und neuen Wahrscheinlichkeiten begrenzt. Sein Nachfolger PPO [Sch+17] verfolgt das gleiche Ziel, jedoch auf mathematisch deutlich einfachere Weise. Dazu wird der Gradientenanstieg nach dem Sammeln von Erfahrung in kleinere Schritte zerlegt und die alten Parameter ϕ_{old} gespeichert. Das geschieht, um für eine Aktion $a \in \mathcal{A}$ in einem Zustand $s \in \mathcal{S}$ das Verhältnis aus neuer und alter Wahrscheinlichkeit $r(t, \phi, \phi_{old}) := \pi_\phi(S_t, A_t)/\pi_{\phi_{old}}(S_t, A_t)$ berechnen zu können. Damit wird mit dem Hyperparameter $\varepsilon \in [0, 1]$ die Zielfunktion

$$L_{\phi_{old}}(\phi) = \mathbb{E}_{S_t, A_t}\left[\min\left(\hat{A}_t \text{clip}(1-\varepsilon, r(t,\phi,\phi_{old}), 1+\varepsilon), \hat{A}_t r(t,\phi,\phi_{old})\right)\right] \tag{3.13}$$

definiert, um sie zu maximieren. Idee dahinter ist, die Wahrscheinlichkeit guter Aktionen – egal wie gut sie waren – um höchstens den Faktor ε zu verbessern. Die Absenkung der Wahrscheinlichkeit für schlechte Aktionen soll auf die gleiche Art begrenzt werden. Hat

sich die Wahrscheinlichkeit einer guten Aktion bereits um mehr als den Faktor ε erhöht (bzw. der einer schlechten Aktion verringert), so fällt der Gradient auf null. Dass nicht nur geclippt, sondern auch minimiert wird, behandelt den Fall, dass die Wahrscheinlichkeit einer guten Aktion gesenkt bzw. die einer schlechten Aktion erhöht wurde. Gerät diese Wahrscheinlichkeit aus dem Vertrauensbereich (in die falsche Richtung) heraus, so soll dies weiterhin korrigiert werden.

3.7 Soft Actor Critic

Eines der neuesten Verfahren ist Soft Actor Critic (SAC) [Haa+18a]. Es kann als Verallgemeinerung von TD3 mit stochastischer Policy angesehen werden. Eine zentrale Änderung gegenüber den anderen Verfahren ist die Umformulierung des RL-Ziels, den erwarteten Return einer Episode zu maximieren. Es wird darum ergänzt, dass bei jedem Schritt die Entropie $\mathcal{H}$ des Actors hoch sein soll. Das bedeutet, dass eine Policy π gefunden werden soll, die möglichst zufällig arbeitet und dennoch sehr gute Ergebnisse liefert. Mit dem Parameter α wird die Balance aus Entropie und Belohnungen eingestellt. Die Formulierung dieses »soften« Returns lautet:

$$G_{soft,t} := \sum_{i=t}^{n_{ep}-1} \gamma^{i-t}(R_i + \alpha\mathcal{H}(\pi(\cdot|S_i))) \tag{3.14}$$

Das Training der Aktionswertefunktion findet genau wie bei TD3 statt. Die Verlustfunktion zur Optimierung der Policy schreibt sich mit folgendem Ausdruck:

$$L_{Q_\theta}(\phi) = \mathbb{E}_{S_t\sim\mathcal{D}}\left[D_{KL}\left(\pi_\phi(\cdot \mid S_t)|\frac{\exp(Q_\theta(S_t,\cdot))}{Z_\theta(S_t)}\right)\right] \tag{3.15}$$

Es wird dabei aus den Aktionswerten eine Wahrscheinlichkeitsverteilung gebaut, indem sie exponenziert und durch einen Normierungsfaktor $Z_\theta(S_t)$ geteilt werden. Dieser Normierungsfaktor wird nur mathematisch dafür benötigt, dass das Integral über den gesamten Aktionsraum 1 ergibt. Für die Gradientenberechnung ist er aber eine Konstante, da er nicht von φ abhängt. Deshalb wird er nicht berechnet und stattdessen die Lernrate angepasst. Diese konstruierte Wahrscheinlichkeitsverteilung gewichtet wie erwartet die Aktionen mit hohem Aktionswert höher. In diese Richtung soll dann die Policy verändert werden. Im Programm muss aber diese komplizierte Verlustfunktion nicht abgeleitet werden, sondern lässt sich mit einigen Umformungen deutlich leichter formulieren. Der Entropieparameter α kann entweder manuell eingestellt oder wie in [Haa+18b] erklärt während des Trainings automatisch erlernt werden.

Soft Actor Critic ist angetreten, um bei der höheren Sample-Effizienz eines Off-Policy-Algorithmus weiterhin die Stabilität der On-Policy-Algorithmen zu haben. Mit Soft Actor Critic konnten schwierige Aufgaben in verschiedenen Bereichen gelöst werden. Nach dem aktuellen Stand der Forschung ist Soft Actor Critic einer der beliebtesten RL-Algorithmen für den kontinuierlichen Aktionsraum.

4 Vergleich der Algorithmen

4.1 Die Algorithmen auf diesem Markt

Der Vergleich der RL-Algorithmen findet auf dem in 1.2 definierten Markt statt. Dieser Markt wird auf der Testplattform simuliert, die im Rahmen des Bachelorprojektes entwickelt wurde. Mit den Agenten wird über die Gym-Schnittstelle [Bro+16] kommuniziert. Für die zu vergleichenden Algorithmen werden die Implementierungen der Bibliothek *Stable Baselines* [Hil+18] verwendet. Stable Baselines ist eine weit verbreitete Open-Source RL-Bibliothek, die in Python geschrieben ist. Sie baut auf PyTorch [Pas+19] auf, einem der beliebtesten Deep-Learning-Frameworks, das in der Forschung weite Verbreitung findet. Die in Stable Baselines implementierten Algorithmen entsprechen in den meisten Fällen unmittelbar den vorgeschlagenen Algorithmen der Originalpaper und zeichnen sich durch eine hohe Lesbarkeit des Codes aus. [1] Alle Hyperparameter sind konfigurierbar.

In dieser Untersuchung werden nur Algorithmen verglichen, die mit stetigem Aktionsraum arbeiten. Der Hauptgrund dafür ist, dass die neuronalen Netze für eine diskrete Formulierung für jede einzelne Aktion ein Ausgabeneuron benötigen. So müssen Deep-Q-Networks mit diskreten Aktionen für jede einzelne Aktionen aus $\mathcal{A}_{\mathbb{N}}$ einen Aktionswert schätzen. A2C und PPO in der diskreten Variante verwenden ebenfalls für jede Aktion ein Ausgabeneuron, um deren Wahrscheinlichkeit anzugeben. Das sind bei der Konfiguration mit $p_{max} = 10$ bereits 1000 Ausgabeneuronen, und das Wachstumsverhalten ist kubisch in der Anzahl der Preisstufen. Außerdem haben die Aktionen in einer diskreten Formulierung im Gegensatz zu der gewählten stetigen keinerlei Ordnung. Sie spiegeln auch nicht wider, dass die Preise reelle Vektoren sind, deren (euklidische, Manhattan-, …) Abstände eine sinnvolle Bedeutung haben. Das dürfte das Problem für die Agenten unnötig schwieriger machen. So wurde etwa in [KS22] festgestellt, dass Soft Actor Critic bei ihren Pricing-Benchmarks besser abschnitt als Deep-Q-Learning.

Die Algorithmen mit stochastischer Policy (A2C, PPO und SAC) verwenden Normalverteilungen als parametrisierte Wahrscheinlichkeitsverteilung, für die Mittelwert und Standardabweichung geschätzt werden. Die einzelnen Verteilungen für Gebrauchtpreis, Neupreis und Rückkaufpreis sind stochastisch unabhängig und damit insbesondere auch die Kovarianzen null.

Weil kaum theoretische Erkenntnisse über die Ermittlung optimaler Hyperparameter vorliegen und diese stets problemspezifisch sind, wäre eine erschöpfende Optimierung der Hyperparameter nur mit erheblichem experimentellem Aufwand möglich. Das ist nicht im Umfang dieser Arbeit. In dieser Untersuchung der Algorithmen wird deshalb von den Hyperparametern der Originalpaper ausgegangen. Diese sind im Anhang in den Tabellen 5.2, 5.3, 5.4 und 5.5 zu finden. Dennoch wurden an einigen Stellen bessere

1 Die SAC-Implementierung verwendet nicht den im Originalpaper vorgeschlagenen Reparameterization Trick und schätzt nur Aktionswerte, keine Zustandswerte.

Hyperparameter gefunden und Aussagen über die Algorithmen über mehrere Hyperparameterkombinationen abgesichert.

Die Netze aller Algorithmen haben stets zwei Schichten und die gleiche Architektur für Actor und Critic. Bei A2C und PPO haben die Netze in beiden Schichten 64 Neuronen. Bei DDPG und TD3 sind es in der ersten Schicht 400 und in der zweiten Schicht 300 Neuronen. Das Netz bei SAC hat in beiden Schichten 256 Neuronen. Die Algorithmen mit diesen Netzgrößen zu vergleichen, ist dennoch angemessen, weil es sich dabei um die Netzgrößen handelt, die in den Originalapern zu PPO, SAC und TD3 für die Algorithmen vorgeschlagen, und auch für die Standardbenchmarks verwendet wurden. [2] Es ist daher davon auszugehen, dass die Netzgrößen seitens der Autoren wohlüberlegt gewählt wurden. Um die Aussagen abzusichern, wurde PPO mit unterschiedlichen Netzgrößen getestet, wobei nur ein geringer Einfluss auf die Performance festgestellt wurde. Lernkurven dazu sind in Abbildung 5.1 abgedruckt.

Zur Analyse der Trainingsläufe wird mit Lernkurven als Diagrammen gearbeitet. Sie dienen dazu, die Leistung der Agenten in Abhängigkeit der Anzahl der bisher trainierten Episoden zu visualisieren. Weil die Returns der einzelnen Episoden stärkeren Schwankungen unterworfen sind, zeigen die Lernkurven den Mittelwert der letzten 100 Episoden, um einen glatten Trainingsverlauf darstellen zu können. Dass sich das Modell während dieser 100 Episoden ändert und damit veraltete Daten für die Mittelung herangezogen werden, muss bei der Verwendung der Lernkurven stets bedacht werden. Deshalb werden die Modelle dort, wo genaue Leistung für den Vergleich benötigt wird, dediziert analysiert.

Alle Algorithmen werden zunächst im Duopol gegen den regelbasierten, unterbietenden Wettbewerber trainiert und evaluiert. Anschließend werden Trainingsdurchläufe gegen die eigene Policy, bei unvollständiger Information und mit einer angepasste Rewardfunktion getestet. Diese Tests dienen dazu, Herausforderungen anzugehen, die bei RL in einer realen Anwendung auftreten können. Zuletzt werden Monopol- sowie Oligopolszenarien untersucht.

4.2 DDPG und TD3

Die Lernkurven der beiden sehr ähnlichen Verfahren DDPG und TD3 sind in Abbildung 4.1 dargestellt. Für jeden der beiden Algorithmen wurden acht Trainingsläufe durchgeführt, die mit unterschiedlichen Lernraten parametrisiert wurden. Der Standardwert liegt für beide Algorithmen bei 10^{-3}. Es wurden Trainingsläufe mit diesen Lernraten sowie kleineren und größeren durchgeführt. Die beim Training beobachteten Muster gleichen sich sowohl bei den beiden Algorithmen als auch den unterschiedlichen Parametrisierungen. Zu Beginn des Trainings ist bei einigen Läufen eine Verbesserung der gemittelten Returns, die primäre Leistungskennziffer, zu beobachten. Allerdings kommt diese Verbesserung bei allen Algorithmen zum Erliegen. In keinem der Durchläufe konnten Gewinne erwirtschaftet werden.

Die schlechten Ergebnisse kommen deshalb zustande, weil die Agenten über das gesamte Training hinweg nicht lernen, sinnvolle Preise zu setzen. Bei vielen Durchläufen

2 A2C bekommt für die Vergleichbarkeit die gleiche Netzarchitektur wie PPO, da die Algorithmen sehr ähnlich sind, genauso wie DDPG und TD3.

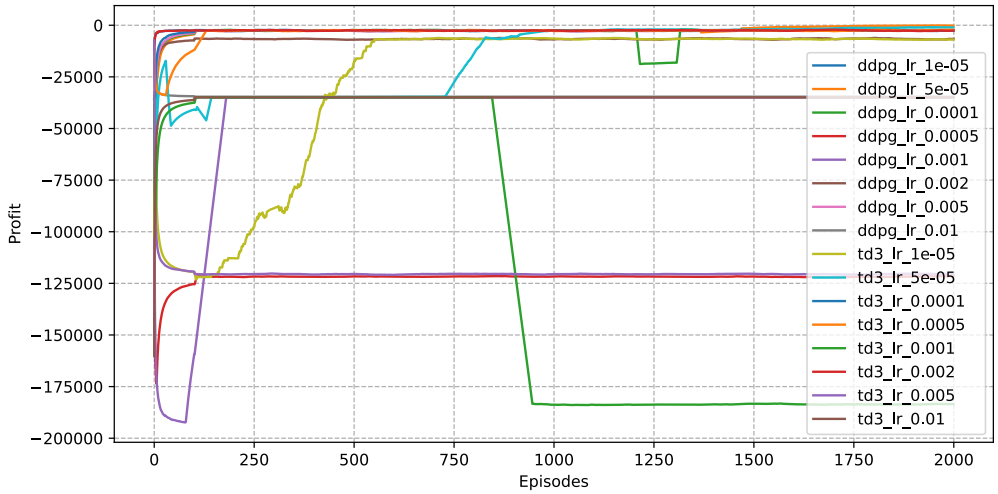

Abbildung 4.1: Lernkurven von DDPG und TD3 mit unterschiedlichen Lernraten

bleiben die Neupreise ohne Anpassung auf die aktuelle Situation beim Maximum 10 hängen, oft sogar auch die Gebrauchtpreise. Die Tatsache, dass die Agenten lange auf einer Stelle bleiben und auch die gleichen verbesserungswürdigen Aktionen immer wieder ausführen, könnte auf unzureichende Exploration schließen lassen. Allerdings konnte auch eine verrauschte Aktionswahl zur besseren Exploration keine Verbesserung der Leistung bewirken.

Weiterhin sieht man in den Trainingsdurchläufen, dass sich die Leistung oft ruckartig verändert. Der dabei in den Lernkurven zu erkennende lineare Auf- oder Abstieg ist der Durchschnittsbildung über die letzten hundert Episoden geschuldet. Tatsächlich findet die Änderung sprunghaft statt.

Zahlreiche der Trainingsdurchläufe sind von starken Einbrüchen der Leistung geprägt. Obwohl Instabilität bei DDPG und TD3 als Probleme bekannt sind, enttäuscht, dass trotz verschiedenster ausprobierter Werte in der Lernrate und Techniken zur verstärkten Exploration die Beherrschung dieses Marktes nicht erlernt werden konnte. Deshalb werden in den folgenden Abschnitten DDPG und TD3 nicht weiter betrachtet.

4.3 On-Policy-Learning – A2C und PPO

Abbildung 4.2 stellt die Lernkurven der Algorithmen dar. Dabei wurde PPO mit zwei Hyperparametrisierungen verwendet, die sich im Parameter ϵ unterscheiden. Das eine Mal wird 0,2 wie im Originalpaper verwendet, das andere Mal 0,3. Jedes dieser Experimente wurde vier Mal unabhängig für eine Million Schritte (zweitausend Episoden) durchgeführt. Die Lernkurven verwenden wieder die laufenden Durchschnitte der Episodenreturns. Der Bereich zwischen maximalem und minimalem Episodenreturns dieser vier Agenten ist eingefärbt. Die Linie stellt ihren Durchschnitt dar. In dieser Grafik wird der Verlustbereich ausgeblendet, um einen genaueren Vergleich im oberen Leistungsbereich zu ermöglichen.

Es fällt bei A2C ein sprunghafter Anstieg bei 100 Episoden auf. Grund dafür ist der

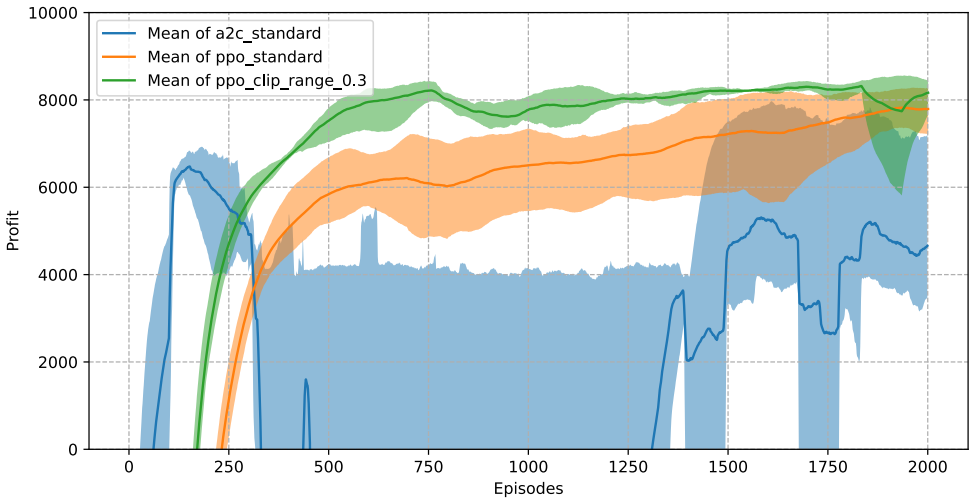

Abbildung 4.2: Lernkurven von vier A2C- und vier PPO-Durchläufen auf dem Duopol mit unterbietendem Wettbewerber

laufende Durchschnitt, bei dem nach 100 Episoden die sehr schlechten Ergebnisse vom Anfang des Trainings aus der Mittelung herausfallen. Dieser Effekt tritt auch bei anderen Lernkurven-Diagrammen auf und liegt nur an diesem Diagrammtyp.

Aus der Grafik geht unmittelbar hervor, dass alle diese Agenten die Gewinnzone erreichen. Sie erreichen alle mit Ergebnissen über 6000 pro Episode gute Ergebnisse. Auf Grundlage dieser Grafik kann man feststellen, dass die Parametrisierung mit $\varepsilon = 0{,}3$ bessere Performance einbringt als die Standardparametrisierung.

Für belastbare Aussagen über die Spitzenperformance wurde bei jedem Trainingsdurchlauf alle 100 Episoden das nach dem Rolling Average beste Modell aus den vorigen 100 Episoden gespeichert. Nach dem Training wurden diese Modelle auf je 25 unabhängigen Episoden getestet, um eine genauere Schätzung ihrer Performance zu erhalten. Die Leistungsmaxima der drei A2C-Trainingsdurchläufe lagen bei 8160, 8520, 7150 und 8510. Bei den PPO-Durchläufen mit $\varepsilon = 0{,}2$ waren die Maxima 7380, 7780, 8330 und 8120. Die PPO-Durchläufe mit $\varepsilon = 0{,}3$ erreichten 8300, 8680, 8360 und 8350. Dass die Spitzen der A2C-Durchläufe in der Lernkurve niedriger sind als die hier aufgeführten Zahlen, liegt an den Rolling Averages und der erheblich geringeren Trainingsstabilität der A2C-Agenten. Sie halten das hohe Leistungsniveau nur sehr kurz, weshalb die Mittelwertbildung mit anderen Ergebnissen während des Trainings niedrigere Werte ergibt. Sichert man jedoch die Parameter der A2C-Agenten in dem Moment, in dem sie gerade gute Ergebnisse liefern, erzielen auch sie wettbewerbsfähige Leistung.

Die Advantage-Actor-Critic-Agenten erreichen das hohe Leistungsniveau nach deutlich weniger Episoden als die PPO-Agenten. Ihr Durchschnitt überschreitet die Schwelle zum Gewinn nach etwa 60 Episoden. Die Grafik 4.3 stellt Details eines einzelnen A2C-Durchlaufes dar. Es handelt sich dabei um einen typischen Durchlauf mit mittlerer Peak-Performance. Man sieht die Instabilität nicht nur an den Ergebnissen, sondern auch an den starken Schwankungen, denen die Aktionsauswahl während des Trainings

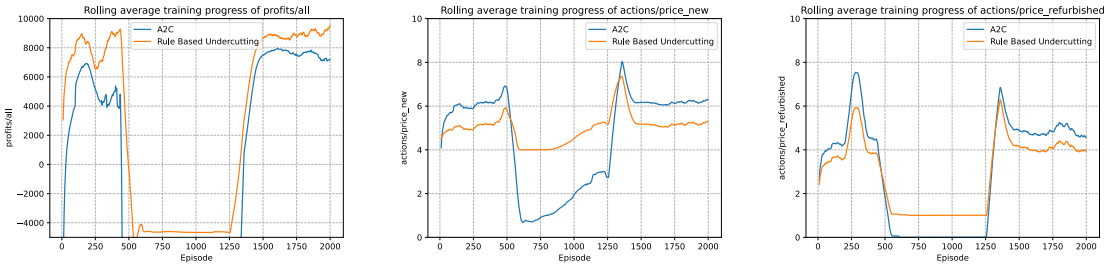

Abbildung 4.3: Detaillierte Betrachtung eines A2C-Trainingsdurchlaufes zur Visualisierung der Instabilität: (links) Lernkurve mit schnellem initialem Anstieg, danach Abstürze und Erholungen; (mittig) durchschnittliche Auswahl der Neupreise, zeigt erhebliche Schwankungen; (rechts) durchschnittliche Auswahl der Gebrauchtpreise, ebenfalls instabil

unterworfen ist. Wie auch bei den anderen A2C-Agenten stürzt seine Leistung mitunter deutlich ab und fällt weit in die Verlustzone. Nicht immer erholen sich die Agenten davon, oft erhalten sie weiterhin schlechte Ergebnisse. Diese heftigen Abstürze führen dazu, dass die Mittelwertlinie der A2C-Agenten in Abbildung 4.2 wieder unter 0 fällt, obwohl einige der Agenten weiterhin akzeptable Ergebnisse liefern.

Im Gegenzug dazu ist die Trainingsstabilität bei den PPO-Varianten deutlich höher. Zwischen den vier Trainingsdurchläufen der PPO-Agenten gibt es nur geringe Unterschiede. Die Leistungsentwicklung liegt in einem schmalen Band und Abstürze finden nicht statt. Allerdings benötigt PPO in der Standardkonfiguration knapp 250 Episoden und in der Version mit $\varepsilon = 0{,}3$ ungefähr 180 Episoden, um die Gewinnzone zu erreichen. Auch danach ist die Leistungssteigerung geringer als bei den anderen Algorithmen. In

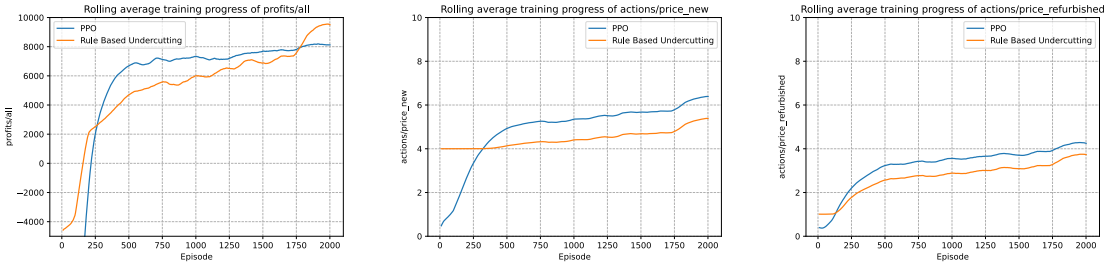

Abbildung 4.4: Detaillierte Betrachtung eines PPO-Trainingsdurchlaufes (Variante mit $\varepsilon = 0{,}2$) analog zu Abbildung 4.3

Abbildung 4.4 ist ein detaillierter Blick in einen typischen PPO-Trainingsdurchlauf zu sehen. Dessen durchschnittliche Performance steigt bis zum Ende auf 8160, und weist bei Return und der Aktionsauswahl eine durchweg stabile Entwicklung auf.

Der Unterschied in der Lerngeschwindigkeit und -stabilität zwischen PPO zu A2C überrascht nicht, er existiert *by design*. Diese Experimente zeigen, dass PPO in seiner Intention, die Trainingsstabilität zu erhöhen, erfolgreich ist. Diese Erhöhung der Trainingsstabilität findet dadurch statt, dass die Änderung der stochastischen Policy bei den Trainingsschritten begrenzt wird. Aus dieser Begrenzung der Policyänderung ist die geringere Lerngeschwindigkeit eine logische Schlussfolgerung.

So lässt sich auch erklären, warum der Durchlauf mit $\varepsilon = 0{,}3$ schneller trainiert. Mit größerem ε wird pro Sample eine größere Policyänderung erlaubt, was zu schnellerem Training führt. Allerdings steigt damit wieder das Risiko von Instabilität, weshalb die Parametrisierung des PPO-Algorithmus einen Trade-off zwischen Trainingsgeschwindigkeit und Stabilität eröffnet. Die Wahl des Standardparameters 0,2 mag in diesen Experimenten konservativ erscheinen, 0,3 ist ebenfalls noch recht stabil, weist aber bereits kleinere Abstürze auf. Das Experiment zu Abbildung 5.2 im Anhang zeigt, wie sich PPO bei größerem ε verhält. Es bestätigt 0,3 als eine gute Wahl.

Eine Beobachtung in Abbildung 4.4 verdient besondere Beachtung, weil sie zunächst paradox erscheint: Obwohl der RL-Agent den regelbasierten Agenten nach etwas Training übertrifft, wird er später wieder überholt. Das erweckt den Eindruck, als würde der Agent im Verlaufe des Trainings schlechter. Jedoch ist das Gegenteil der Fall. Der PPO-Agent erlernt zunächst, die Preise für Neu- und Gebrauchtware höher zu setzen. Dieses Training dauert aus den diskutierten Gründen im Vergleich lange, aber bei Neuverkaufspreisen, die größer als vier sind, macht er die Erfahrung, dass der regelbasierte Wettbewerber ihn immer um 1 unterbietet. Durch wechselseitiges Unterbieten führt eine Preisabwärtsspirale dazu, dass sich der Preis nur knapp oberhalb des Einkaufspreises einpegelt. Die dabei äußerst niedrige Rendite ermöglicht jedoch nur niedrige Gewinne, sodass der Agent in seinem weiteren Training die Erfahrung macht, dass er seinen Gewinn steigern kann, indem er seine Preise höher setzt. Er wird dann weiterhin unterboten, allerdings nur um den Wert eins. Dabei nimmt er in Kauf, dass mehr Kunden wegen des niedrigen Preises beim Konkurrenten kaufen und dieser bei steigenden Preisen auch an jedem Kunden mehr verdient. Mit den Kunden, die der RL-Agent aber noch bekommt, kann er seine Gewinne im Vergleich zum niedrigpreisigen Markt dennoch steigern. Der Effekt besteht also darin, dass der RL-Agent in Reaktion auf den ihn immer unterbietenden Konkurrenten diesem einen überproportionalen Gewinnanstieg erlaubt, um selbst mehr Gewinne machen zu können. Die Existenz dieses Effekts ist der Tatsache geschuldet, dass das einzige Optimierungskriterium für die Agenten der eigene Gewinn ist. Eine Rewardformulierung, die ein Übertreffen des Konkurrenten als Ziel mit einbezieht, wird in Abschnitt 4.5 behandelt.

4.4 Soft Actor Critic

Auf dem gleichen Duopolmarkt wurde auch SAC ausprobiert. Dabei wurde die SAC-Variante genommen, bei der der Entropiekoeffizient α automatisch trainiert wird. Dieser hat sich bei allen Trainingsdurchläufen bei etwa 1,5 eingependelt. Die Grafik 5.4 im Anhang zeigt, dass unterschiedliche manuell eingestellte Entropiekoeffizienten ähnliche Trainingsergebnisse erreichen, wobei sich die Vermutung bestätigt, dass sehr niedrige Entropiekoeffizienten zu wenig Exploration bezwecken und Läufe mit deutlich höheren Entropiekoeffizienten wegen starker Regularisierung schlechtere Leistungsmaxima erreichen.

Grafik 4.5 zeigt die Lernkurve von vier Soft-Actor-Critic-Agenten bei einem Trainingsdurchlauf mit 2000 Episoden, genau so viele wie beim obigen Vergleich. Zum Vergleich sind vier Trainingsdurchläufe mit PPO angegeben, die den aus Abschnitt 4.3 bekannten

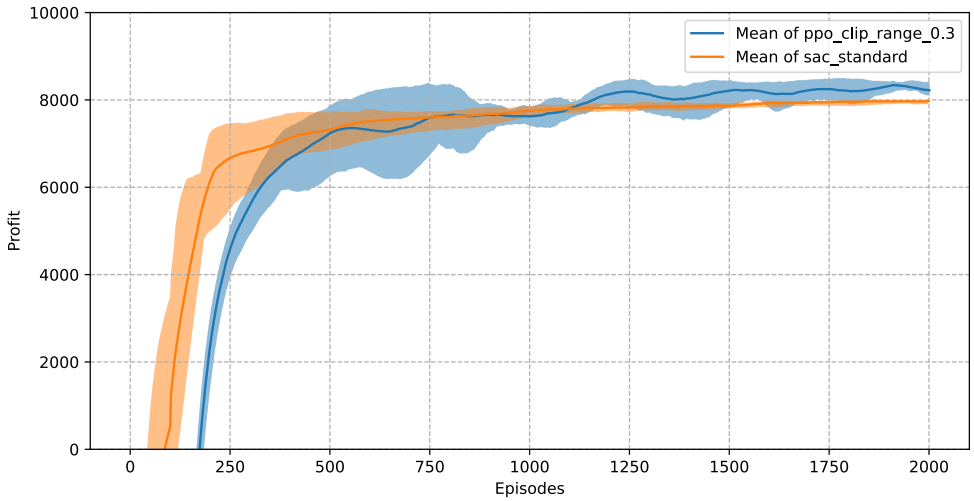

Abbildung 4.5: Lernkurven von vier SAC-Durchläufen im direkten Vergleich mit PPO

Verlauf aufweisen. Im Anhang in Abbildung 5.3 ist die Lernkurve von SAC im direkten Vergleich mit A2C abgedruckt.

Soft-Actor-Critic wurde als Off-Policy-Algorithmus neben dem Erreichen von wettbewerbsfähigen Ergebnissen für zwei Hauptziele entwickelt: Erstens soll es eine hohe Sample Efficiency haben, was bedeutet, dass es beim Training deutlich weniger Schritte benötigt. Zweitens soll SAC eine hohe Trainingsstabilität haben. Die Lernkurve bestätigt, dass Soft-Actor-Critic diese zwei Hauptziele erreicht. Profitabel arbeitet der Agent nach etwa 70 Episoden, was nur leicht hinter A2C liegt und erheblich besser als PPO ist. Ab etwa 250 bis 300 Episoden sind die Agenten nah an ihrer Peak-Performance. Das restliche Training bringt nur noch kleine Leistungszugewinne. Die hohe Sample-Efficiency liegt daran, dass SAC die gesammelte Erfahrung im Experiencebuffer speichert und sehr oft zum Training nutzt. Dadurch werden viel weniger Samples als bei PPO benötigt, das alte Erfahrung nicht mehr verwenden kann, sobald sich die Policy geändert hat. Alle vier Agenten bewegen sich für den Rest des Trainings in einem schmalen Bereich, wobei der Durchschnittsreturn knapp 8000 erreicht. Ob weiteres Training zusätzliche Verbesserungen bezwecken können, lässt sich nicht sicher sagen. Allerdings ändern sich die gemessenen Kenngrößen wie Aktionsauswahl und Loss im späteren Verlauf des Trainings kaum noch. Der Durchschnitt der vier SAC-Agenten liegt zwar dauerhaft über dem Durchschnitt der A2C-Agenten, aber gute A2C-Durchläufe erreichen bessere Performance in der Spitze als die SAC-Agenten. Die Maximalleistungen dieser vier SAC-Durchläufe liegen bei 8120, 7890, 8120 und 8160. Damit bleiben sie ebenfalls hinter denen der PPO-Agenten zurück, die meist über 8300 als Return erzielen.

Wie auch für die anderen Algorithmen wurden für SAC ein typischer Durchlauf und die gemittelten Aktionen in Abbildung 4.6 abgedruckt. Die durchschnittlichen Preise starten – offenbar durch eine andere Parameterinitialisierung – auf höherem Niveau als bei PPO. Sie sinken dann und enden bei 6,4 und 5,4, genau wie bei PPO.

Warum PPO in diesem Markt etwas besser abschneidet, ist eine äußerst interessante

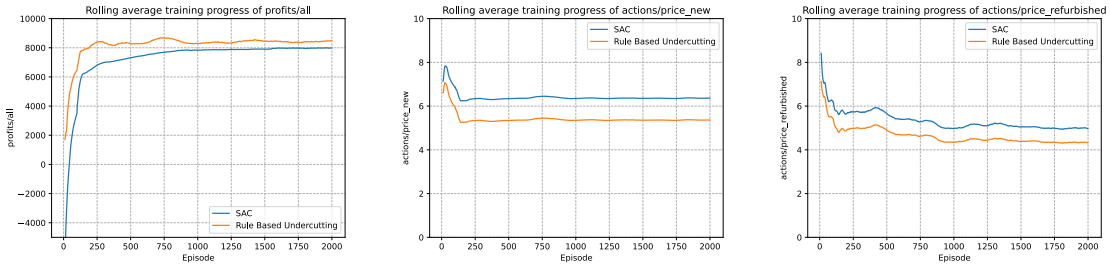

Abbildung 4.6: Detaillierte Betrachtung eines SAC-Trainingsdurchlaufes

Frage, deren Beantwortung man sich aber auf Grundlage dieser Experimente nur annähern kann. Eine plausible Erklärung wird in Abschnitt 4.7 auf Grundlage der dortigen Beobachtungen geliefert. Kurz ausgedrückt wird dort festgestellt, dass SAC in der Policy Zusammenhänge lernt, die nicht gelernt werden sollten. Eine andere mögliche Erklärung, dass die Policy der SAC-Agenten wegen der Entropieregularisierung größere Standardabweichungen schätzt und deshalb weniger genau als die von PPO ist, lässt sich durch die Daten aus dem Training nicht bestätigen. Bei beiden Trainingsalgorithmen haben die Policies zum Ende des Trainings Standardabweichungen von etwa 0,5, wobei SAC keine größeren Standardabweichungen hat.

Für die Frage »Wie lange dauert das Training?« gibt es zwei mögliche Kenngrößen. Einmal kann die Anzahl der benötigten Samples bewertet werden, die Sample Efficiency, oder es kann die tatsächliche Zeit auf der Uhr herangezogen werden. Letztere hängt stark von der Hardware sowie der Geschwindigkeit der Simulation im Vergleich zum Optimieren der Parameter ab. Bei der für diese Experimente verwendeten Implementierung und Hardware benötigt das Training pro Schritt mit SAC etwa 3,7 Mal so lang wie mit PPO. Das Sammeln der Beispiele aus dem Markt nimmt bei PPO etwa 25% der Trainingszeit ein, bei SAC nur etwa 6%. Die Geschwindigkeit des Trainings ist bei A2C und PPO sehr ähnlich. Rechnet man somit die tatsächliche Trainingszeit, relativiert sich der höhere Samplebedarf bei PPO. Insbesondere wird deutlich, dass A2C in reinem Zeitbedarf den anderen Algorithmen erheblich überlegen ist.

Weil insbesondere das SAC-Training so zeitaufwändig ist und der Lernfortschritt im späteren Verlauf so gering ist, wird bei einigen der folgenden Versuche mit SAC die Anzahl der Trainingsschritte reduziert. Gleiches geschieht bei A2C, da hier bereits sehr früh die maximale Performance erreicht werden.

4.5 Den Konkurrenten übertreffen – eine angepasste Rewardfunktion

In den bisher untersuchten Lernkurven erreichen die Agenten zwar sehr gute Profite, lassen sich aber vom regelbasierten Konkurrenten übertreffen. In Abschnitt 4.3 wurde bereits erklärt, dass dies nicht als Schwäche der Algorithmen zu deuten ist, sondern auf die Rewardfunktion zurückgeführt werden kann, die nur eigenen Profit bewertet. Nun ist die Maximierung des eigenen Profits keine ungeeignete Kenngröße, dennoch werden

Unternehmen ungern ihren Konkurrenten in einem symmetrischen Markt mehr Gewinn überlassen als sich selbst. Deshalb liegt es nahe, in der Belohnungsfunktion nicht nur die eigenen Profite zu bewerten, sondern auch, ob der Konkurrent übertroffen wird.

Eine Möglichkeit besteht darin, den Markt als *Zero-Sum-Spiel* zu formulieren, indem die Belohnung genau die Differenz aus eigenem Gewinn und dem des Konkurrenten ist. Diese Formulierung legt den klaren Fokus auf das Übertreffen des Konkurrenten und eröffnet weiterhin möglicherweise das Marktszenario für Erkenntnisse aus der Spieltheorie für Zero-Sum-Spiele. Allerdings ist diese Formulierung nicht praxistauglich, weil das Ziel der Profitmaximierung gar nicht bewertet wird. So kann es passieren, dass bei einer Optimierung der Differenz zwar der Konkurrent deutlich übertroffen wird, allerdings bei insgesamt niedrigen Gewinnen oder sogar in der Verlustzone.

Deshalb wurde für die folgende Versuchsreihe eine gemischte Rewardfunktion verwendet. Sie berechnet einfach die Summe aus Profit und Differenz zum Konkurrenten. Für diese Experimente wurden beide Summanden gleich gewichtet, es könnte aber ein Hyperparameter eingefügt werden, um die beiden Ziele Profitmaximierung und Übertreffen des Konkurrenten zu balancieren.

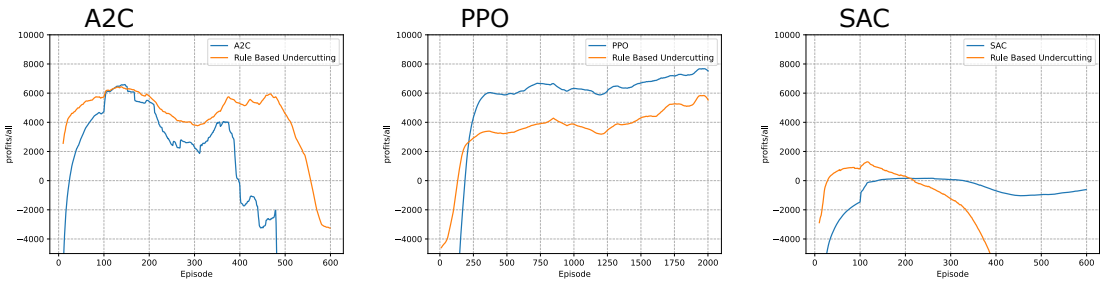

Abbildung 4.7: Repräsentative Durchläufe der Algorithmen bei gemischter Rewardfunktion

Im Anhang sind die Lernkurven der Algorithmen mit gemischter Rewardfunktion im Vergleich zur Standardrewardfunktion abgedruckt (Abbildungen 5.5, 5.6 und 5.7). Wie erwartet liegen die Gewinne der Agenten niedriger als bei der anderen Rewardfunktion (schließlich sind Gewinne nicht mehr das alleinige Optimierungskriterium), allerdings fällt dieser Unterschied bei den Algorithmen unterschiedlich aus. Weiterhin erhöht sich durch den Wechsel der Rewardfunktion die Instabilität bei allen Algorithmen. Ob das Ziel, neben guten Gewinnen die Konkurrenz zu überholen, erreicht wird, zeigt Abbildung 4.7.

Obwohl der A2C-Durchlauf wieder Probleme mit Instabilität zeigt, erreicht er ein durchschnittliches Maximum von 7900, der Konkurrent nur 7200. Er überholt also bei der veränderten Rewardfunktion den regelbasierten Konkurrenten, und erhält gleichzeitig gute Gewinne.

Beim PPO-Durchlauf ist genau das zu sehen, was Ziel der Anpassung der Rewardfunktion war: Der PPO-Agent bleibt nach dem ersten Übertreffen des Konkurrenten diesem über das gesamte Training hinweg voraus und erreicht ein Maximum von 8000 im Vergleich zu 6100 des regelbasierten Konkurrenten.

Der Soft-Actor-Critic-Durchlauf erreicht nach der gemischten Rewardfunktion sogar die besten Ergebnisse, obwohl der Agent selbst in die Verlustzone rutscht. Er hat eine Policy gefunden, bei der der regelbasierte Konkurrent ständig Strafen zahlen muss, weil

er keine gebrauchten Produkte liefern kann. Dadurch macht der regelbasierte Konkurrent einen deutlich größeren Verlust Die Details zu diesem Durchlauf sind interessant und in Abbildung 5.8 im Anhang erläutert. Aus Sicht der Algorithmenanalyse ist es ein bemerkenswertes Resultat, dass der SAC-Agent dazu fähig war, eine solche Strategie zu explorieren und zu entwickeln. Für die praktische Verwendung ist diese Policy allerdings nicht geeignet. Der Profit des Agenten sollte beim Design der Rewardfunktion höher gewichtet werden.

4.6 Training gegen die eigene Policy

In den Durchläufen aus den Abschnitten 4.2, 4.3 und 4.4 trainierten die RL-Agenten alle gegen den unterbietenden, regelbasierten Wettbewerber. Das erfüllt die theoretischen Anforderungen eines Markov-Entscheidungsprozesses, wirft jedoch für praktische Anwendungen eine Reihe von Problemen auf. Erstens muss dafür die Policy des Konkurrenten bekannt sein. Weil jedoch davon ausgegangen werden kann, dass der Konkurrent seine Preisstrategie nicht verraten wird, müsste sie unter Inkaufnahme von Ungenauigkeiten aus historischen Daten geschätzt werden. Zweitens ist die mittels RL trainierte Strategie nur gegen diese bestimmte regelbasierte Strategie gerichtet. Ändert der Wettbewerber seine Preisstrategie plötzlich, schwächt das die Performance der RL-Strategie und macht neues Training erforderlich.

Deshalb wünscht man sich eine Strategie, die gegen möglichst viele Wettbewerberstrategien bestehen kann, und nicht auf eine spezielle überangepasst ist. DeepMind hatte eine ähnliche Herausforderung beim Training von Go- und Schachstrategien, das durch Self-Play gelöst wurde. Das kann in [Sil+17a; Sil+17b] nachgelesen werden.

Für diesen Markt wurde eine Self-Play Variante entwickelt, bei der ein RL-Agent weiterhin auf einem Duopol-Markt trainiert, aber die Policy des Wettbewerbers die eigene Policy ist. In der programmiertechnischen Umsetzung wird für den Gegner ein Zeiger auf den RL-Agenten übergeben, der schließlich auch die Policyfunktion implementiert. Dadurch, dass der Agent stets gegen sich selbst spielt, wird die Markov-Eigenschaft verletzt, dennoch zeigen die Ergebnisse Trainingserfolg. Die Motivation hinter Self-Play ist, dass der Agent einerseits die ganze Zeit einem ebenbürtigen Gegner gegenübersteht, aber andererseits auch, dass er ständig Strategien gegen die eigene entwickelt, die dann wiederum herausgefordert werden. Dadurch soll der Agent auf eine Vielzahl von Policies vorbereitet werden.

Experimente mit Self-Play wurden mit den Algorithmen durchgeführt, die sich in der vorangegangenen Analyse als grundsätzlich geeignet für diesen Markt erwiesen haben: A2C, PPO und SAC. Dabei wurde die gesamte Versuchsreihe mit der normalen Rewardfunktion und der gemischten Rewardfunktion aus Abschnitt 4.5 durchgeführt.

In der Abbildung 4.8 sind die Lernkurven der drei Algorithmen beim Training gegen sich selbst abgedruckt. Die Lernkurven bei gemischter Rewardfunktion sehen ähnlich aus und sind im Anhang in Abbildung 5.9 zu finden. Diese Kurven zeigen zunächst, dass Lernerfolg bei allen dieser drei Agenten erreicht wird. Sie alleine können allerdings nicht aussagen, ob die Agenten sich tatsächlich gegen die Konkurrenz behaupten können.

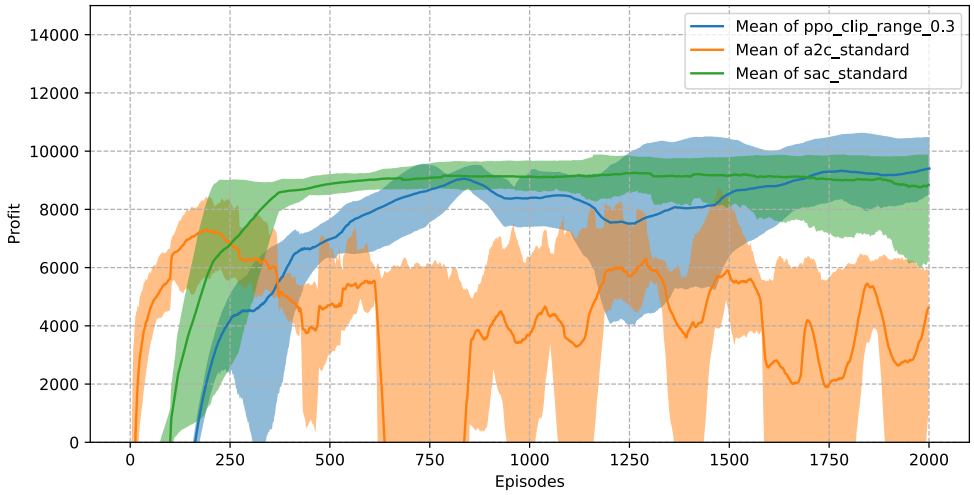

Abbildung 4.8: Lernkurve von vier A2C-, PPO- und SAC-Durchläufen beim Self-Play; die Algorithmen wurden über 2000 Episoden trainiert

Deshalb dürfen auch die Returns dieser Lernkurven nicht mit denen aus den vorigen Abschnitten verglichen werden.

Um Vergleichbarkeit herstellen zu können, wurde während des Self-Plays alle 50 Episoden ein Modell gespeichert und jedes dieser Modelle anschließend für 25 Episoden getestet. Damit wurden akkurate Lernkurven erstellt, die das beim Self-Play trainierte Modell mit dem regelbasierten Agenten vergleichen. Für die drei Algorithmen und die zwei Rewardfunktionen wurde ein mittlerer Durchlauf ausgewählt und in der Abbildung 4.9 veranschaulicht.

Bei beiden Rewardfunktionen haben die Agenten den Markt erfolgreich erlernt und können sich auch in dem Benchmark gegen den regelbasierten Konkurrenten behaupten. So liegen die Peaks aller Agenten im Benchmark bei 8000 oder knapp darunter, allerdings mit starken Schwankungen bei A2C. Von den A2C-Agenten erreichen einige kein Niveau von 5000. Die Auswirkungen der gemischten Rewardfunktion sind bei den Trainingsdurchläufen zu erkennen. Dass während des Trainings die Differenz zwischen eigenem und gegnerischem Profit mit als Optimierungskriterium verfolgt wird, führt tatsächlich dazu, dass der durch Self-Play trainierte Agent auch im Vergleich mit dem regelbasierten Konkurrenten sich zumindest nicht übertreffen lässt (A2C), den Konkurrenten im Gegensatz zur normalen Rewardfunktion dauerhaft abhängt (PPO) oder den Abstand zum Konkurrenten vergrößert (SAC).

Obwohl die Benchmarkergebnisse erwartungsgemäß nicht ganz an die Ergebnisse des Trainings direkt gegen den regelbasierten Agenten heranreichen, so sind sie nahe daran. Das ist bemerkenswert, da diese Erfolge im Benchmark erreicht wurden, ohne den regelbasierten Wettbewerber je vorher gesehen zu haben. Damit können diese Experimente als erfolgreich gewertet werden. Die Fähigkeiten vom Training gegen die eigene Policy werden damit auch für ein Duopol bestätigt.

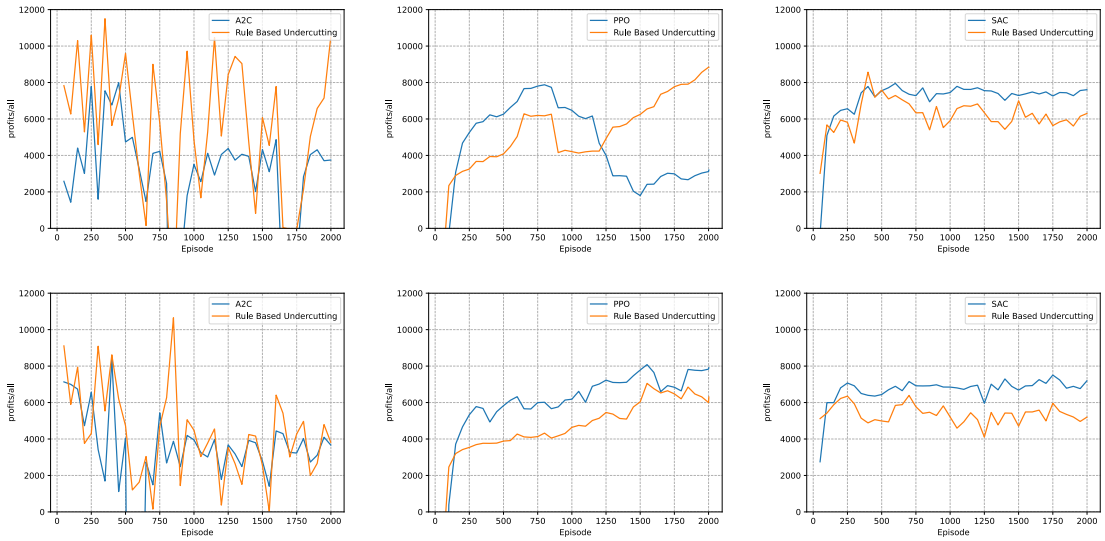

Abbildung 4.9: Repräsentative Self-Play Durchläufe; spaltenweise nach Agenten: (links) A2C, (mittig) PPO, (rechts) SAC; obere Zeile mit normaler Rewardfunktion, untere Zeile mit gemischter Rewardfunktion

4.7 Partielle Beobachtungen

Für die Markov-Eigenschaft ist ein sechsdimensionaler Aktionsraum erforderlich. Jedoch ist die Beobachtung mancher der verwendeten Größen in der Praxis nicht machbar. So wird der Konkurrent sich nicht ins Lager schauen lassen, und die Anzahl der Produkte in Zirkulation ist schwer zu schätzen. Deshalb drängt sich die Frage auf, ob die Algorithmen auch ohne diese beiden Informationen funktionieren.

In Abbildung 4.10 sind die Lernkurven der drei Algorithmen beim Training gegen die unterbietende, regelbasierte Strategie dargestellt. Das Ergebnis fällt dabei äußerst überraschend aus. Die Erwartung, dass weniger Informationen zu schlechteren Ergebnissen führen müssten, erfüllt sich nicht. Bis auf eine leicht verschlechterte Stabilität ist die Performance bei PPO ähnlich. Dieses geringfügige Absinken der Stabilität lässt sich vermutlich direkt durch das Fehlen von Informationen erklären. Bei A2C verbessert sich ohne diese Informationen die Performance, und bei SAC sogar deutlich. Während die SAC-Agenten, denen nur der Lagerstand des Konkurrenten vorenthalten wird, recht ähnliche, leicht verbesserte Ergebnisse wie die mit vollständiger Information erzielen, sind die Agenten, bei denen zusätzlich die Information über die Anzahl der Produkte in Zirkulation fehlt, deutlich besser. Es senkt sich nicht nur die Zeit bis zum Erreichen der Leistungsspitze auf unter 100 Episoden, sondern es verbessert sich sogar die endgültige Performance. Damit reicht die Leistung näher, aber nicht ganz an die von PPO heran. Das ist eine wichtige Erkenntnis mit Blick auf die praktische Anwendbarkeit, und dennoch stellt sich die Frage, wie sich die unerwartet gute Leistung mit unvollständiger Information erklären lässt.

Zur Erklärung ist zunächst heranzuführen, dass die ausgelassenen Informationen nur eine nachgeordnete Rolle spielen. Sie erklären zwar zu einem Teil die künftige

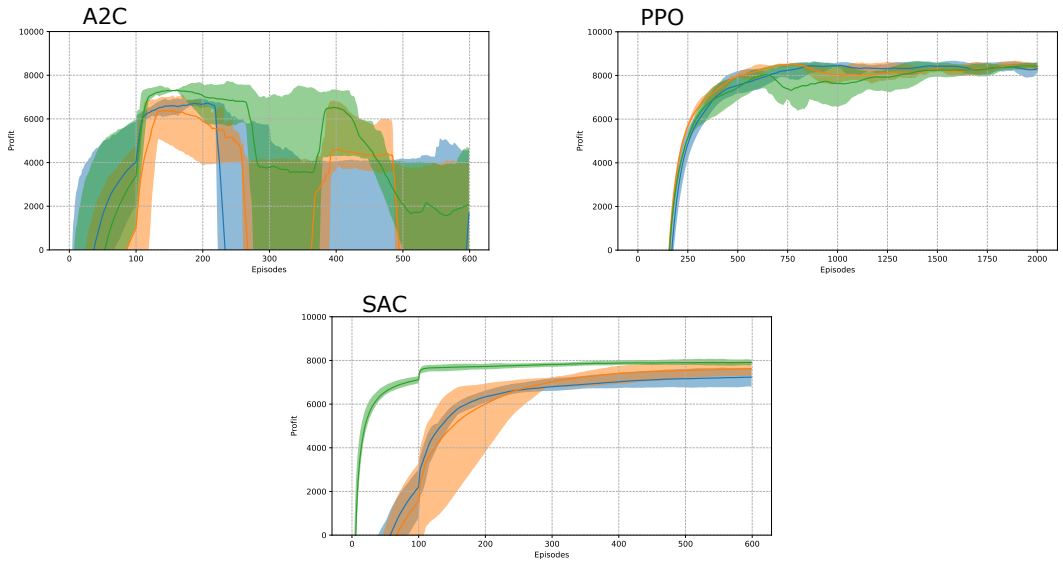

Abbildung 4.10: Lernkurven von A2C, PPO und SAC bei vollständiger im Vergleich zu partieller Beobachtung; (blau) vollständige Beobachtung, (orange) ohne Lagerstand des Konkurrenten und (grün) ohne Lagerstand des Konkurrenten und Anzahl der Produkte

Aktion des Konkurrenten und die Anzahl der verkaufswilligen Eigentümer, allerdings sind die Auswirkungen so indirekt, dass sie schwer für die Verbesserung einer Policy zu verwerten sind. [3] Ein höherdimensionaler Beobachtungsraum stellt zudem grundsätzlich für Machine-Learning-Verfahren eine Herausforderung dar. Es steigt dann der Bedarf an Samples, Muster in den Daten sind schwerer zu erkennen.

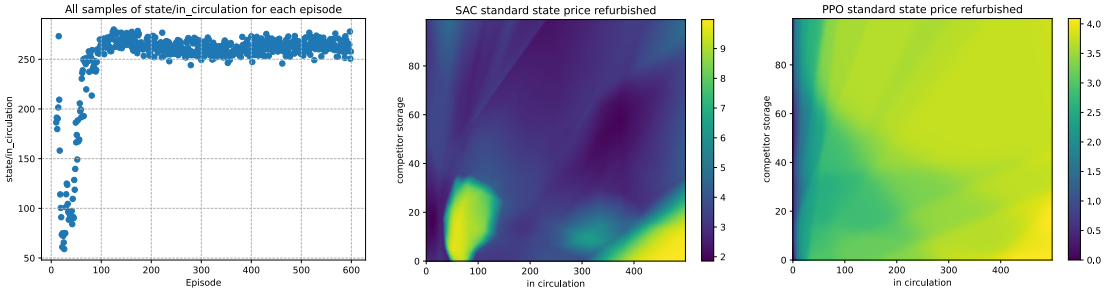

Abbildung 4.11: Betrachtung der Agenten mit vollständiger Information: (links) Verlauf des *in-circulation*-Zählers im SAC-Training, (mittig und rechts) Gebrauchtpreis in Abhängigkeit von *in-circulation* und Lagerstand des Konkurrenten bei SAC und PPO; andere Argumente sind dabei fest auf einen Wert gewählt, der typisch bei einem Trainingsdurchlauf ist (eigener Lagerstand: 25, Neupreis Konkurrent: 6, Gebrauchtpreis Konkurrent: 4, Rückkaufpreis Konkurrent: 0)

Für die Erklärung des speziellen Phänomens bei SAC genügt die höhere Dimension allerdings nicht. Ein weiterer Erklärungsansatz drängt sich auf. Zunächst kann aus Abbildung 4.11 (Mitte) gelesen werden, dass ein SAC-Agent starke Abhängigkeiten von den beiden Argumenten (Zähler der Anzahl in Zirkulation und Lagerstand des Konkurrenten)

3 Die regelbasierten Konkurrenten verwenden diese Information auch nicht.

erlernt. So unterscheidet sich der Mittelwert der SAC-Policy für Gebrauchtpreise bei ähnlichen Situationen zwischen 2 und 10, ein Sprung durch den gesamten Aktionsraum. Dass sich eine nahezu optimale Policy so verhalten sollte, ist angesichts der geringen Bedeutung der beiden Argumente auszuschließen, was die Policy des deutlich erfolgreicheren PPO-Agenten bestätigt. Wie man es erwarten würde, hängt sie kaum von diesen beiden Argumenten ab. Die niedrigeren Leistungsmaxima von SAC lassen sich vermutlich durch derartige Schwächen der Policy erklären.

Für die von SAC erlernte Policy lässt sich eine Erklärung in der Funktionsweise von Soft Actor Critic finden. Sie basiert darauf, dass bei weiterentwickelten Policies im Markt mehr Produkte in Zirkulation sind. Das ist in Abbildung 4.11 (links) gezeigt und liegt daran, dass wenig trainierte Policies im Gegensatz zu den weiterentwickelten oft zu viel zurückkaufen (und dabei zu viel Geld ausgeben). Das bedeutet, dass gerade am Anfang der Experiencebuffer nur mit Zustandsübergängen mit niedrigen *in-circulation*-Werten gefüllt wird. Diese Samples bleiben im Experiencebuffer, sind aber für das spätere Training nicht mehr hilfreich und können die Anomalien in der Policy erklären. So liegt eine starke Anomalie in der Policy im Bereich zwischen 50 und 150 Produkten in Zirkulation, genau der Bereich, aus dem die frühen Samples stammen. Das Entfallen des *in-circulation*-Eintrages verbessert also die Performance des SAC wohl deshalb, weil dieser dann nicht mehr die Möglichkeit hat, mit missweisenden Samples tatsächlich nicht existente Zusammenhänge zu erlernen. Es ist damit als Schwäche von SAC gegenüber dem On-Policy-Verfahren PPO zu verzeichnen, dass es weniger gut in der Lage ist, die geringe Relevanz dieses Attributes selbstständig zu erlernen.

4.8 Durch die Policy induziertes Marktgeschehen

Die vorigen Untersuchungen bewerteten die Leistung der Policy vorrangig anhand der Leistung, die ein Agent über den Trainingsverlauf hinweg erzielt. Es lohnt sich jedoch ebenfalls, eine einzelne Episode genauer zu betrachten. Dabei interessiert man sich dafür, ob situationsangemessene Preise gewählt werden, ob das Lager erfolgreich reguliert wird und ob sich Gleichgewichte einstellen. Es muss jedoch beachtet werden, dass es sich um exemplarische Durchläufe handelt. Bei Verallgemeinerungen ist deshalb große Vorsicht geboten.

Im ersten Durchlauf einer Episode wurde ein PPO-Agent gegen den unterbietenden regelbasierten Wettbewerber getestet. Der PPO-Agenten hat die Modellparameter, die bei dem Self-Play-Durchlauf mit gemischter Belohnungsfunktion trainiert wurden, dessen Trainingsverlauf in Abbildung 4.9 abgedruckt ist. Details aus dieser Episode sind im Anhang in Abbildung 5.10 abgedruckt. Dabei wurden die Lagerstände der beiden Agenten über die vollen 500 Schritte der Episode abgedruckt. Von den gesetzten Preisen wurde ein 25-Schritte-Intervall zum Ende der Episode gezeigt. Zunächst ist zu erkennen, dass beide Agenten einen Startzustand mit unerwünscht hohem Lagerstand bekommen. Bei PPO ist die Initialisierung fast beim Maximum. Beide Agenten leeren nun ihre Lager, um Lagerkosten zu sparen. Nach weniger als 50 Schritten pendelt sich das Lager bei beiden Agenten um 10 Produkte herum ein. Während dieser Wert beim regelbasierten Agenten fest einprogrammiert ist, hat sich der PPO-Agent ein sehr ähnliches Niveau beim

Training selbst beigebracht. Die Lagerschwankungen, die sich eingestellt haben, sind beim regelbasierten Agenten kleiner als beim PPO-Agenten. Eine naheliegende Erklärung dafür ist, dass der regelbasierte Agent den vorgegebenen Lagerstand sehr aggressiv regelt. So ist bei Rückkauf- und Gebrauchtpreis zu sehen, dass beinahe bei jedem Schritt zwischen einem sehr niedrigem und einem sehr hohen Preisniveau gewechselt wird. Die Unterschiede zwischen benachbarten Preisen fallen bei dem RL-Agenten deutlich niedriger aus. Man kann auf Basis dieser Diagramme vorschlagen, dass der regelbasierte Agent bei der Regulierung des Lagers größere Toleranz walten lassen könnte, um die Preissprünge zwischen aufeinanderfolgenden Schritten zu reduzieren. Bei den Neupreisen sieht man ebenfalls die einprogrammierte Logik im Preisverlauf: Der regelbasierte Agent setzt hier keine eigenen Impulse und unterbietet den PPO-Agenten stets um eins. Dadurch ist seine Kurve die um einen halben Schritt nach rechts und eine Preisstufe nach unten versetzte Kurve des PPO-Agenten. Am Ende der Episode stehen als Profite 7994 für den PPO-Agenten gegen 5844 des regelbasierten Agenten.

Neben der Gegenüberstellung mit dem regelbasierten Agenten wurde eine Episode des PPO-Agenten gegen einen ebenfalls mit gemischter Rewardfunktion und Self-Play trainierten SAC-Agenten untersucht. Dabei wurden die gleichen Diagramme erstellt. Sie sind im Anhang in Abbildung 5.11 zu finden. Die Initialisierung geschah wie im obigen Verlauf mit deutlich mehr Produkten im Lager als gewünscht, die zuerst abverkauft werden, bis sich der Lagerstand beider Agenten unter 20 einpegelt. Das Preisniveau für Neuprodukte ist bei beiden Agenten auf ähnlichem und stabilen Niveau. Es bewegt sich zwischen 5 und 6. Während die anderen Preise sich ebenfalls auf relativ stabilem Niveau bewegen, sticht der Gebrauchtpreis des SAC-Agenten hervor. Er ist stärkeren Schwankungen zwischen 2 und 9 unterworfen. Am Ende der Episode liegt der PPO-Agent mit 9920 zu 9319 vor dem SAC-Agenten.

4.9 Vergleich im Monopol

In den vorigen Abschnitten wurden die Algorithmen in einem Recommerce-Duopol mit einem regelbasierten, unterbietenden Konkurrenten getestet. Für reale Anwendungen sind aber viele verschiedene Szenarien zu meistern. Es ist daher von Interesse, wie die Algorithmen auf verschiedenen Marktszenarien abschneiden, die in der Praxis auftreten können. Ein Szenario, das leichter ist sein dürfte, ist das Monopol. Hier ist der RL-Agent der einzige Anbieter, während weiterhin die gleiche Anzahl von Kunden den Markt besucht. Die anderen Dynamiken des Marktes sind unverändert. Als Rewardfunktion wird der Gewinn des Agenten gewählt – Vergleiche mit Konkurrenten erübrigen sich im Monopol.

Abbildung 4.12 zeigt die Lernkurven der drei Agenten auf dem Monopolmarkt mit vollständiger Information. Dabei stechen sofort die Parallelen zum Duopol-Markt ins Auge:

1. Die relative Reihenfolge der Agenten bei der Lernkurve ist die gleiche: PPO schneidet am besten ab, dann SAC vor A2C.

2. A2C erreicht als erstes die Gewinnzone, danach folgt SAC und zuletzt PPO.

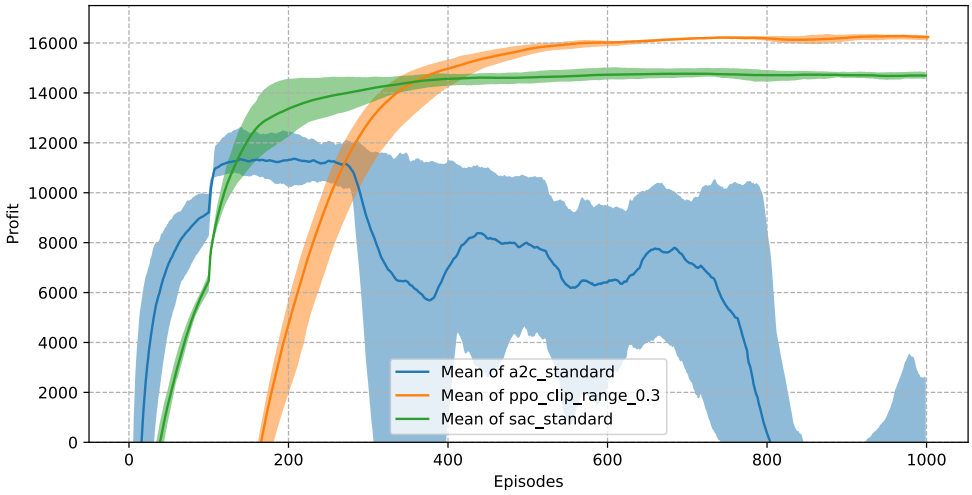

Abbildung 4.12: Lernkurven von vier A2C-, PPO- und SAC-Durchläufen in einem Monopolszenario über 1000 Episoden

3. Das Training von PPO und SAC ist sehr stabil, A2C-Training ist von Abstürzen geprägt.

Auch wenn der Monopol-Markt einfacher ist, so benötigt das Training nicht erheblich weniger Samples. Vergleicht man den Punkt, an dem die Algorithmen keine deutlich besseren Ergebnisse mehr erreichen, liegt A2C in beiden Setups bei 100 bis 200 Episoden, SAC bei etwa 300 Episoden und PPO bei etwa 600 Episoden.

Im Monopol erzielen diese drei Algorithmen etwa doppelt so hohe Gewinne wie im Duopol. Auf der einen Seite ist das plausibel, schließlich müssen sie sich den Markt nicht mehr aufteilen. Auf der anderen Seite könnte man aber erwarten, dass die Gewinne sogar höher ausfallen, indem die Monopolsituation durch Verlangen höherer Preise ausgenutzt wird. Dass dies nicht geschieht, liegt an der begrenzten Kaufbereitschaft der Kunden, gerade bei höheren Preisen. Im Anhang in Abbildung 5.12 wird dazu ein PPO-Durchlauf im Monopol mit einem im Duopol mit Blick auf Verkaufszahlen verglichen.

4.10 Vergleich im Oligopol

Als weiteres Szenario wird das Oligopol untersucht. Dabei tritt der RL-Agent gegen die folgenden Agenten an:

1. eine Verallgemeinerung des bekannten regelbasierten Agenten (unterbietet das Minimum der anderen um eins),
2. einen regelbasierten Agenten, der mithilfe der Preise zwar sein Lager reguliert, aber nicht auf die anderen Agenten reagiert,
3. einen Agenten, der feste Preise hat (6 als Neupreis, 3 als Gebrauchtpreis und 2 als Rückkaufpreis) und

4. einen Agenten, der speziell für das Oligopolszenario erstellt wurde. Als Neupreis unterbietet er den Median der anderen Anbieter, und reguliert sein Lager so, dass es etwa 7 Produkte enthält.

Jeder Schritt wird hier in Fünftel zerlegt, die Anbieter setzen ihre Preise reihum. In jedem dieser Fünftel kommen vier Kunden zum Marktplatz, insgesamt also weiterhin 20. Das sonstige Verhalten des Marktes bleibt unverändert. Der Beobachtungsraum ist 18-dimensional (*in-circulation*, eigener Lagerstand und für jeden der vier Konkurrenten drei Preise sowie der Lagerstand). Damit handelt es sich um das komplexeste Szenario, das untersucht wurde.

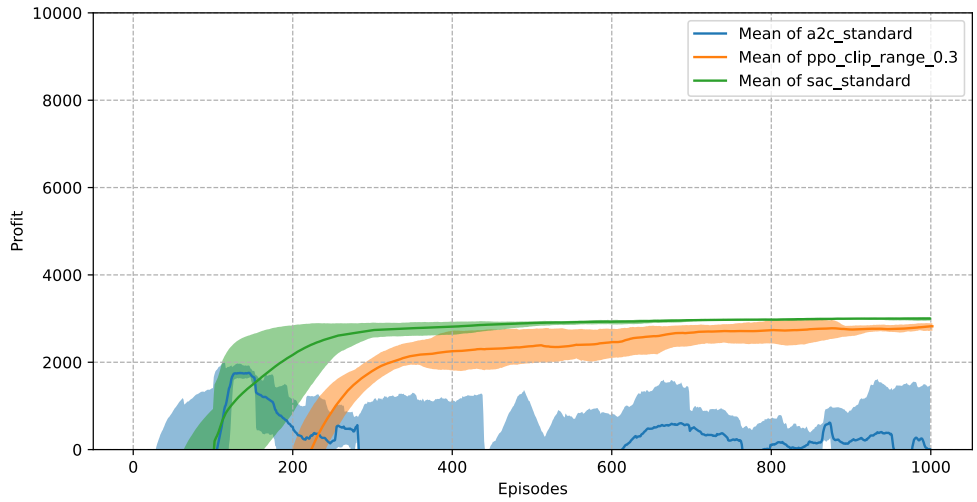

Abbildung 4.13: Lernkurven von vier A2C-, PPO- und SAC-Durchläufen in einem Oligopolszenario über 1000 Episoden mit normaler Rewardfunktion und vollständiger Beobachtung

Grafik 4.13 zeigt die Lernkurven auf diesem Oligopolszenario. Die RL-Agenten erreichen auch hier Lernerfolg, wobei die Gewinne selbstverständlich niedriger ausfallen, da sich die gleiche Anzahl von Kunden auf deutlich mehr Anbieter verteilt. Wie auch in den anderen Experimenten, zeigen PPO und SAC stabile Lernkurven. Bei A2C zeigen sich die bekannten Instabilitäten, aber ihre Leistung in der Spitze bleibt tatsächlich nur leicht hinter der der anderen Algorithmen zurück. Äußerst interessant ist bei diesem Versuch, dass der SAC-Algorithmus besser abschneidet als PPO, der bei den anderen Experimenten in der Spitzenleistung überlegen war. Weil wegen des hohen Trainingsaufwandes nur 1000 Episoden trainiert wurden und die PPO-Kurven am Ende noch einen leichten Anstieg zeigen, kann zwar nicht ausgeschlossen werden, dass PPO zu einem späteren Zeitpunkt aufholt, aber dennoch lässt sich auf Grundlage dieser Lernkurve SAC als überlegen einordnen.

Der mutmaßliche Grund hierfür ist das größere neuronale Netz, mit dem SAC arbeitet. Mit 256 Neuronen in den beiden versteckten Schichten lassen sich kompliziertere Funktionen erlernen als mit nur 64 Neuronen in den versteckten Schichten. Gerade bei der hier erforderlichen deutlich hochdimensionaleren Policyfunktion ist die Notwendigkeit

für eine komplizierte Funktion gegeben. Um diese Vermutung nachzuprüfen, wurde ein zusätzliches Experiment durchgeführt, bei dem PPO Netze mit 256 Neuronen in den versteckten Schichten trainiert. Die Lernkurven sind im Anhang in Abbildung 5.13 abgedruckt. Dabei zeigt sich, dass gute PPO-Durchläufe schließlich die gleiche Performance erreichen wie bei SAC, aber die Trainingsstabilität erheblich niedriger ist.

Der Algorithmenvergleich auf dem Oligopol wurde ebenfalls mit der gemischten Rewardfunktion durchgeführt. Dessen Ergebnisse sind ähnlich, allerdings streuen die SAC-Durchläufe dort deutlich stärker. Die Lernkurven dazu sind im Anhang in Abbildung 5.14 abgedruckt.

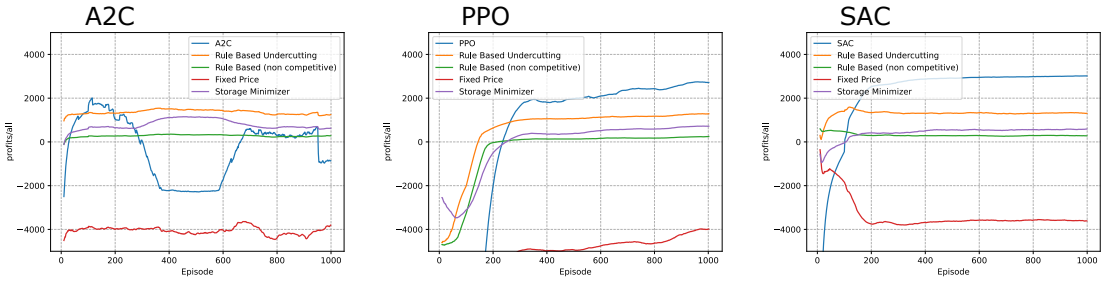

Abbildung 4.14: Profite typischer Trainingsdurchläufe von A2C, PPO und SAC im Vergleich zu ihren regelbasierten Konkurrenten

Abbildung 4.14 zeigt für jeden der drei Algorithmen einen typischen Trainingsdurchlauf. Die Ergebnisse sind dabei äußerst zufriedenstellend. Jeder der RL-Agenten kann während seines Trainingsdurchlaufes alle seine regelbasierten Konkurrenten deutlich übertreffen. Der Agent mit den festen Preisen schneidet erwartungsgemäß am schlechtesten ab, da er weder dazu in der Lage ist, sein Lager zu regulieren, noch auf Preise der Konkurrenten zu reagieren. Der Agent, der nur sein Lager reguliert, landet auf dem vorletzten Platz, während die beiden regelbasierten Agenten, die auch auf Konkurrenzpreise reagieren, passabel abschneiden.

In diesem Oligopolszenario tritt der Effekt, dass ein RL-Agent einen regelbasierten Konkurrenten überholen lässt, nicht auf. Weil die RL-Agenten ihre Konkurrenz hier stets überholen, besteht nicht die Notwendigkeit für die gemischte Rewardfunktion. Deshalb werden diese Versuche auch nicht detailliert betrachtet.

5 Schlussfolgerungen & Ausblick

Obwohl ein Recommerce-Markt zahlreiche Herausforderungen mit sich bringt und die Konkurrenz stark ist, konnten sehr effektive Preisstrategien erfolgreich mit Reinforcement-Learning-Verfahren gefunden werden. Dabei lieferten die Algorithmen A2C, PPO und SAC sehr gute Spitzenergebnisse, während DDPG und TD3 nicht gut abschnitten. Advantage Actor Critic ist den beiden Verfahren PPO und SAC in der Trainingsstabilität deutlich unterlegen, konnte aber sehr gute Policies liefern, wenn die Modellparameter während des Trainings in einem Moment guter Leistung gespeichert wurden. Weiterhin übertraf Advantage Actor Critic die anderen Algorithmen bei der Trainingsdauer und überraschenderweise auch bei der Sample Efficiency. Im Duopol und Monopol erreichte PPO die beste Performance der verglichenen Algorithmen. Über unterschiedliche Szenarien hinweg schnitt er in der Spitzenleistung am besten ab. Seine Trainingskurven verliefen in den Experimenten stabil. Sein Nachteil war in jeder der Analysen die geringere Sample Efficiency. Der Soft Actor Critic Algorithmus lieferte ebenfalls äußerst stabile Trainingsläufe, blieb jedoch in seiner Spitzenleistung auf Duopolszenarien hinter PPO zurück. In der Sample Efficiency hingegen war SAC PPO stets überlegen. Im deutlich höherdimensionalen Oligopol-Markt lieferte SAC die beste Spitzenleistung und war PPO überlegen. Eine genauere Untersuchung dieses Phänomens legt nahe, dass PPO weniger gut komplexe Policyfunktionen trainiert, die durch größere neuronale Netze berechnet werden. Weil die Algorithmen in den unterschiedlichen Szenarien und nach unterschiedlichen Bewertungskriterien unterschiedlich gut abschnitten, kann kein klarer Sieger gekürt werden.

Die Ergebnisse dieser Arbeit sind eine Bestätigung, dass Reinforcement Learning auf realen Märkten zur Optimierung des Pricing angewandt werden kann. Der verwendete Markt ist kompliziert genug und die Ergebnisse stabil genug, sodass erwartet werden kann, dass die Algorithmen auch auf Märkten mit weiteren Parametern wie Qualität oder Saison erfolgreich abschneiden werden. Weiterhin wurde die Lösbarkeit zentraler Probleme für die praktische Anwendung untersucht. Die Ergebnisse fielen positiv aus. So funktionieren die Algorithmen auch dann, wenn einige Aspekte des Marktes nicht beobachtet werden können und die Markov-Eigenschaft damit nicht erfüllt ist. Das Problem, dass die Preisstrategie des Konkurrenten unbekannt ist, kann durch Self-Play gelöst werden. Die dabei trainierten Policies konnten ihre Konkurrenz bei guten Ergebnissen übertreffen, ohne sie jemals im Training gesehen zu haben.

Als Herausforderung verbleibt das nicht genau bekannte Verhalten von Käufern und Eigentümern für das Training auf Simulationsplattformen. Training auf Simulationsplattformen ist aber notwendig, weil wegen des weiterhin hohen Samplebedarfes und schlechter Ergebnisse während des Trainings nicht direkt auf dem echten Markt trainiert werden kann. Um die Simulationsplattform auf den realen Markt anzupassen, muss das Verhalten dieser Marktteilnehmer aus Mangel an akkuraten theoretischen Modellen aus Echtweltdaten geschätzt werden. Das ist jedoch mit verschiedenen Regressionsverfahren

möglich und wurde in vorheriger Forschungsarbeit bereits behandelt, zum Beispiel beim zweistufigen Verfahren in [SB18a].

Eine denkbare Alternative zum Training mit einer Simulationsplattform ist es, auf einem Markt die Daten, die mit einer bisherigen Preisstrategie gesammelt wurden, direkt in einen Experiencebuffer zu schreiben. Mit einem Off-Policy-Verfahren wie SAC könnte damit eine Policy erlernt werden, die auf dem echten Marktplatz bereits akzeptable Ergebnisse liefert. Anschließend könnte auf dem echten Markt noch Fine-Tuning-Training stattfinden. Stabilität und Sample-Efficiency sind die Haupteigenschaften, die ein genutztes RL-Verfahren mitbringen müsste. Ob ein solches Vorgehen praxistauglich ist, sollte Gegenstand künftiger Forschung sein.

Eine weitere Schwierigkeit, die auf realen Marktplätzen auftreten kann und nicht im Umfang dieser Arbeit war, ist ein Gedächtnis der Kunden und Eigentümer. In diesem Modell kommen Kunden und Eigentümer auf den Marktplatz und haben für ihre Entscheidung nur die Informationen eines einzigen Zeitschrittes. Tatsächlich ist aber davon auszugehen, dass zumindest einige Kunden den Preisverlauf beobachten und nicht nur in Abhängigkeit aktueller, sondern auch vergangener oder antizipierter zukünftiger Preise ihre Kaufentscheidung treffen. Stünde dem Agenten dann wie im derzeitigen Modell nur der aktuelle Markt zur Verfügung, so fehlte ihm ein großer Teil der eigentlich benötigten Entscheidungsgrundlage. Um eine längere Liste vergangener Zustände zu verarbeiten, ohne dass von Menschen relevante Features bei der Entwicklung extrahiert wurden, werden andere Arten von neuronalen Netzen benötigt. Wenn diese Informationen eine feste Größe haben, könnte über den Einsatz von CNNs nachgedacht werden, bei variabler Größe fiele die Wahl auf rekurrente Netze oder Transformer-Modelle.

Literatur

[Arc05] New York Times (Archived). *As I.T. Goes, So Goes Forrester?* Accessed: 2022-07-05. 2005. URL: https://web.archive.org/web/20180613040854/https://www.nytimes.com/2005/02/18/business/yourmoney/as-it-goes-so-goes-forrester.html (siehe S. 1).

[ASM21] Julio César Alves, Diego Mello da Silva und Geraldo Robson Mateus. **Applying and Comparing Policy Gradient Methods to Multi-echelon Supply Chains with Uncertain Demands and Lead Times**. In: *Artificial Intelligence and Soft Computing*. Hrsg. von Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz und Jacek M. Zurada. Cham: Springer International Publishing, 2021, 229–239. ISBN: 978-3-030-87897-9 (siehe S. 10).

[Bro+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang und Wojciech Zaremba. **Openai gym**. *arXiv preprint arXiv:1606.01540* (2016) (siehe S. 19).

[Con+22] Peijin Cong, Junlong Zhou, Mingsong Chen und Tongquan Wei. **Personality-Guided Cloud Pricing via Reinforcement Learning**. *IEEE Transactions on Cloud Computing* 10:2 (2022), 925–943. DOI: 10.1109/TCC.2020.2992461 (siehe S. 9).

[FHM18] Scott Fujimoto, Herke van Hoof und David Meger. **Addressing Function Approximation Error in Actor-Critic Methods**. *CoRR* abs/1802.09477 (2018). arXiv: 1802.09477. URL: http://arxiv.org/abs/1802.09477 (siehe S. 15).

[GB22] Torsten J. Gerpott und Jan Berends. **Competitive pricing on online markets: a literature review**. *Journal of Revenue and Pricing Management* (Juni 2022). ISSN: 1477-657X. DOI: 10.1057/s41272-022-00390-x. URL: https://doi.org/10.1057/s41272-022-00390-x (siehe S. 9).

[Haa+18a] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel und Sergey Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG] (siehe S. 17).

[Haa+18b] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel und Sergey Levine. *Soft Actor-Critic Algorithms and Applications*. 2018. DOI: 10.48550/ARXIV.1812.05905. URL: https://arxiv.org/abs/1812.05905 (siehe S. 17).

[Hes+17] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar und David Silver. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. DOI: 10.48550/ARXIV.1710.02298. URL: https://arxiv.org/abs/1710.02298 (siehe S. 14).

[HGS15] Hado van Hasselt, Arthur Guez und David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. DOI: 10.48550/ARXIV.1509.06461. URL: https://arxiv.org/abs/1509.06461 (siehe S. 14).

[Hil+18] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor und Yuhuai Wu. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018 (siehe S. 19).

[Jos+20] Rajan Jose, Shrikant Krupasindhu Panigrahi, Rashmi Anoop Patil, Yudi Fernando und Seeram Ramakrishna. **Artificial Intelligence-Driven Circular Economy as a Key Enabler for Sustainable Energy Management**. *Materials Circular Economy* 2:1 (Sep. 2020), 8. ISSN: 2524-8154. DOI: 10.1007/s42824-020-00009-9. URL: https://doi.org/10.1007/s42824-020-00009-9 (siehe S. 10).

[KB14] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. URL: https://arxiv.org/abs/1412.6980 (siehe S. 13).

[Kim+16] Byung-Gook Kim, Yu Zhang, Mihaela van der Schaar und Jang-Won Lee. **Dynamic Pricing and Energy Consumption Scheduling With Reinforcement Learning**. *IEEE Transactions on Smart Grid* 7 (2016), 2187–2198 (siehe S. 9, 10).

[Kra+19] Elena Krasheninnikova, Javier García, Roberto Maestre und Fernando Fernández. **Reinforcement learning for pricing strategy optimization in the insurance industry**. *Engineering Applications of Artificial Intelligence* 80 (2019), 8–19. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2019.01.010. URL: https://www.sciencedirect.com/science/article/pii/S0952197619300107 (siehe S. 9).

[KS22] Alexander Kastius und Rainer Schlosser. **Dynamic pricing under competition using reinforcement learning**. *Journal of Revenue and Pricing Management* 21:1 (Feb. 2022), 50–63. ISSN: 1477-657X. DOI: 10.1057/s41272-021-00285-3. URL: https://doi.org/10.1057/s41272-021-00285-3 (siehe S. 9, 19).

[Lap20] M. Lapan. **Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more, 2nd Edition**. Packt Publishing, 2020. ISBN: 9781838820046. URL: https://books.google.de/books?id=O0vODwAAQBAJ (siehe S. 13).

[Lar+21] Thomas Larsen, Halvor Teigen, Torkel Laache, Damiano Varagnolo und Adil Rasheed. **Comparing Deep Reinforcement Learning Algorithms' Ability to Safely Navigate Challenging Waters**. *Frontiers in Robotics and AI* 8 (Sep. 2021). DOI: 10.3389/frobt.2021.738113 (siehe S. 10).

[Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra und Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. DOI: 10.48550/ARXIV.1312.5602. URL: https://arxiv.org/abs/1312.5602 (siehe S. 14).

[Moc+19] Elena Mocanu, Decebal Constantin Mocanu, Phuong H. Nguyen, Antonio Liotta, Michael E. Webber, Madeleine Gibescu und J. G. Slootweg. **On-Line Building Energy Optimization Using Deep Reinforcement Learning**. *IEEE Transactions on Smart Grid* 10:4 (2019), 3698–3708. DOI: 10.1109/TSG.2018.2834219 (siehe S. 10).

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai und Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Hrsg. von H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox und R. Garnett. Curran Associates, Inc., 2019, 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (siehe S. 13, 19).

[RO15] Rupal Rana und Fernando S. Oliveira. **Dynamic pricing policies for interdependent perishable products or services using reinforcement learning**. *Expert Systems with Applications* 42:1 (2015), 426–436. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.07.007. URL: https://www.sciencedirect.com/science/article/pii/S095741741400400X (siehe S. 9).

[SB18a] Rainer Schlosser und Martin Boissier. **Dynamic Pricing under Competition on Online Marketplaces: A Data-Driven Approach**. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, 705–714. ISBN: 9781450355520. URL: https://doi.org/10.1145/3219819.3219833 (siehe S. 9, 38).

[SB18b] Richard S. Sutton und Andrew G. Barto. **Reinforcement Learning: An Introduction**. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html (siehe S. 11–13).

[Sch+15a] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan und Pieter Abbeel. *Trust Region Policy Optimization*. 2015. DOI: 10.48550/ARXIV.1502.05477. URL: https://arxiv.org/abs/1502.05477 (siehe S. 16).

[Sch+15b] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan und Pieter Abbeel. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. DOI: 10.48550/ARXIV.1506.02438. URL: https://arxiv.org/abs/1506.02438 (siehe S. 16).

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford und Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347 (siehe S. 16).

[Sil+14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra und Martin Riedmiller. **Deterministic Policy Gradient Algorithms**. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, I–387–I–395 (siehe S. 14).

[Sil+17a] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan und Demis Hassabis. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. DOI: 10.48550/ARXIV.1712.01815. URL: https://arxiv.org/abs/1712.01815 (siehe S. 28).

[Sil+17b] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel und Demis Hassabis. **Mastering the game of Go without human knowledge**. *Nature* 550:7676 (Okt. 2017), 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270 (siehe S. 28).

[Wil92] Ronald J. Williams. **Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning**. *Mach. Learn.* 8:3–4 (Mai 1992), 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696 (siehe S. 15).

Abbildungsverzeichnis

Appendix

Hyperparameter

Symbol	Erklärung	Standardwert
n_{ep}	Anzahl der Schritte in einer Episode	500
p_{max}	maximal wählbarer Preis für alle drei Kanäle	10
$p_{einkauf}$	Einkaufs- oder Produktionspreis für Neuprodukte	3
p_{lager}	Preis pro eingelagertem Gebrauchtprodukt pro Schritt	0,1
k	Kundenzahl, die pro Schritt den Markt besucht	20
m_{lager}	maximale Anzahl von Gebrauchtprodukten, die im Lager gehalten werden können	100
c	Anteil der Eigentümer, die pro Schritt einen Rückverkauf erwägen	0,05

Tabelle 5.1: Marktparameters mit kurzer Erklärung und für diese Experimente verwendete Standardwerte

Parameter	Wert
Lernrate	10^{-3}
Größe des Experiencebuffers	10^6
Episoden bis zum Beginn des Lernens	100
Größe eines Minibatches	100
Koeffizient für die Polyak-Mittelung (τ)	0,005
Diskontierungsfaktor (γ)	0,99

Tabelle 5.2: Hyperparameter für DDPG und TD3

Parameter	Wert
Lernrate	$7 \cdot 10^{-4}$
Anzahl der Schritte pro Update	5
Diskontierungsfaktor (γ)	0,99

Tabelle 5.3: Hyperparameter für Advantage Actor Critic

Parameter	Wert
Lernrate	$3 \cdot 10^{-4}$
Anzahl der Schritte pro Update	2048
Größe eines Minibatches	64
Anzahl der Epochen pro Update	10
clip_range (ε)	0,2
Diskontierungsfaktor (γ)	0,99

Tabelle 5.4: Hyperparameter für Proximal Policy Optimization

Parameter	Wert
Lernrate	$3 \cdot 10^{-4}$
Größe des Experiencebuffers	10^6
Episoden bis zum Beginn des Lernens	100
Größe eines Minibatches	256
Entropiekoeffizient (α)	automatisch
Koeffizient für die Polyak-Mittelung (τ)	0,005
Diskontierungsfaktor (γ)	0,99

Tabelle 5.5: Hyperparameter für Soft Actor Critic

Definition der unterbietenden, regelbasierten Strategie

Die Notation beschreibt die Sicht von Anbieter 1 aus, wobei Anbieter 2 der Konkurrent ist. Da die beiden Positionen symmetrisch sind, ist das keine Beschränkung der Allgemeinheit. Die regelbasierte Strategie π setzt ihren Neupreis als

$$\pi\left(p_{2,neu}\right)_{neu} = \max\left(p_{2,neu} - 1, p_{einkauf} + 1\right). \tag{5.16}$$

Beim Gebraucht- und Rückkaufpreis wird je nach Lagerstand entschieden. Der Gebrauchtpreis wird in Abhängigkeit des Konkurrenzpreises und des Lagerstandes als

$$\pi\left(p_{2,gebraucht}, n_{lager}\right)_{gebraucht} = \begin{cases} p_{2,gebraucht} + 1 & n_{lager} < m_{lager}/15 \\ p_{2,gebraucht} - 1 & n_{lager} < m_{lager}/8 \\ p_{2,gebraucht} - 2 & \text{sonst} \end{cases} \tag{5.17}$$

gesetzt. Beim Rückkaufpreis werden die gleichen Fallunterscheidungen unternommen und der Preis

$$\pi(p_{2,re}, n_{lager})_{re} = \begin{cases} p_{2,re} + 1 & n_{lager} < m_{lager}/15 \\ p_{2,re} - 1 & n_{lager} < m_{lager}/8 \\ p_{2,re} - 2 & \text{Sonst} \end{cases} \tag{5.18}$$

gewählt. Alle Preise werden auf den Aktionsraum beschränkt (für den Fall, dass die Rechnung Ergebnisse kleiner als null oder größer als p_{max} ermittelt).

Weitere Diagramme zum Lernerfolg

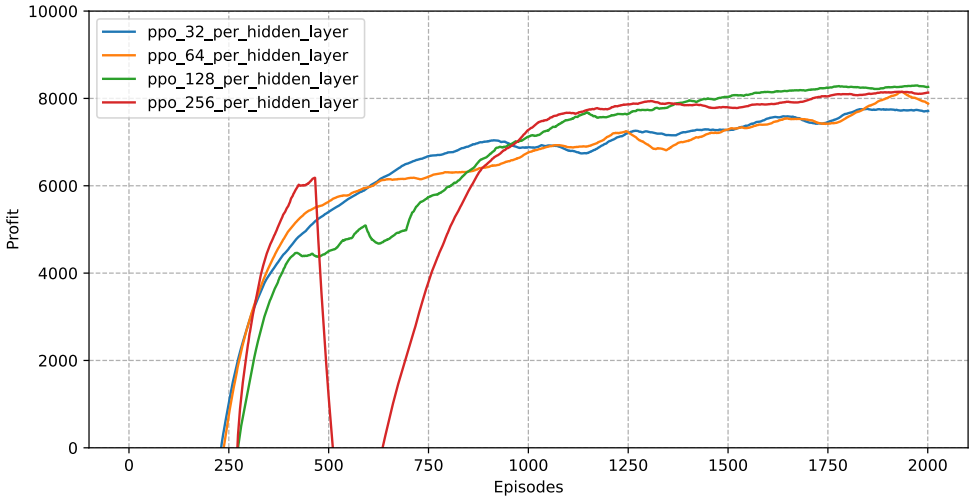

Abbildung 5.1: Lernkurven des Proximal-Policy-Optimization-Algorithmus bei unterschiedlicher Anzahl von Knoten pro versteckter Schicht, aber insgesamt zwei Schichten: Trotz des Unterschiedes in den Netzgrößen und damit sehr großen Unterschieden in der Anzahl der Parameter schneiden die Netze sehr ähnlich ab. Sowohl Lerngeschwindigkeit als auch Spitzenperformance unterscheiden sich kaum. Auffällig ist allerdings, dass der Trainingsdurchlauf mit dem großen Netz einen Absturz aufweist, der ansonsten bei PPO nicht auftritt. Der Durchlauf mit 32 Neuronen pro Schicht erreicht die schlechteste Spitzenperformance.

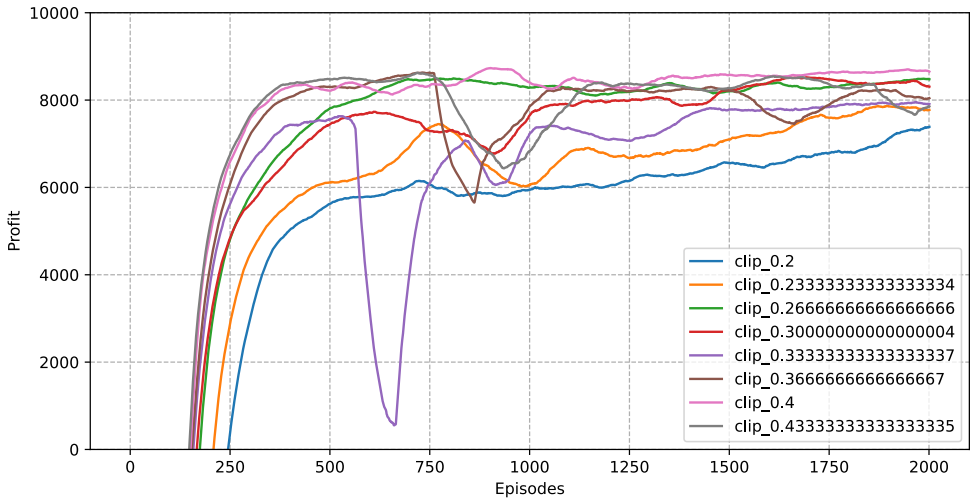

Abbildung 5.2: Lernkurven des Proximal-Policy-Optimization-Algorithmus bei unterschiedlichen Werten für ε: Zwischen 0,2 und 0,3 ist eine Verbesserung in der Leistung zu sehen. Bei größeren Werten setzt sich die Verbesserung nicht fort und die Trainingsstabilität sinkt.

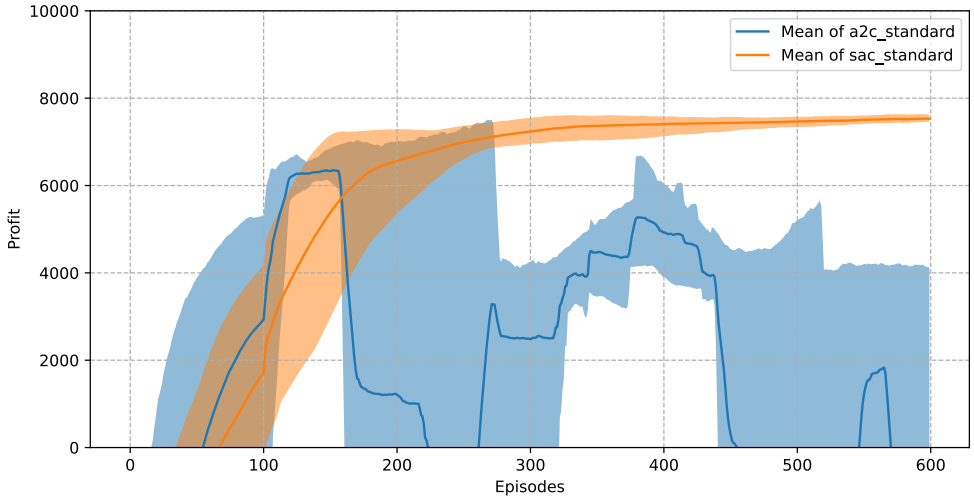

Abbildung 5.3: Lernkurven von vier SAC-Durchläufen über 600 Episoden mit A2C als Vergleich

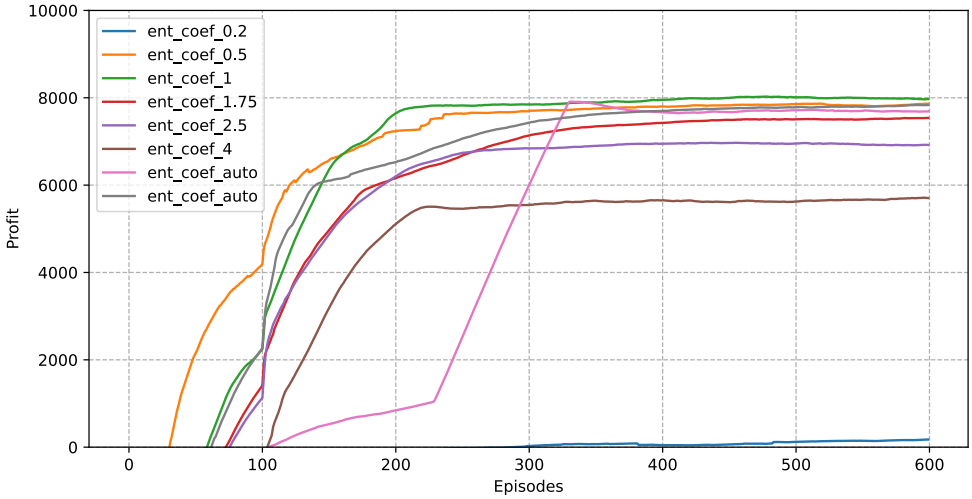

Abbildung 5.4: Lernkurven des Soft-Actor-Critic-Algorithmus bei unterschiedlichem Entropiekoeffizienten (sechs fest gewählt und zwei automatisch angepasst): Hohe Werte sind eine Regularisierung und führen dazu, dass die Policy »zufälliger« ist. Die Exploration und Stabilität nehmen zu, allerdings leidet die Spitzenperformance. Das liegt daran, dass die Policy weniger genau gelernt wird, und die Aktionsauswahl stärker streut. Bei dem Durchlauf mit $\varepsilon = 0{,}2$ ist die Performance deutlich niedriger, was vermutlich an zu geringer Exploration liegt. Der Durchlauf mit $\varepsilon = 1$ hat sogar geringfügig bessere Ergebnisse geliefert als die mit automatischer Ermittlung des Entropiekoeffizienten.

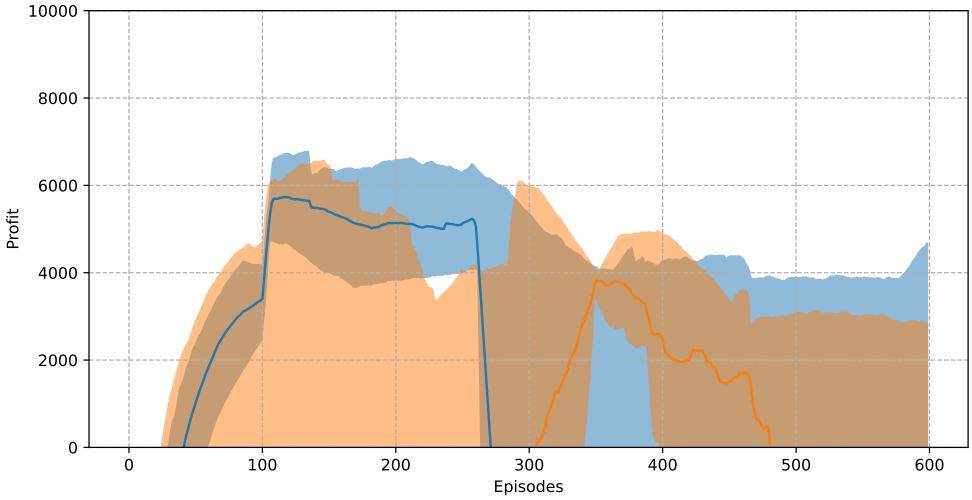

Abbildung 5.5: Lernkurven des Advantage-Actor-Critic-Algorithmus bei gemischter Rewardfunktion (orange) im Vergleich zur Standardfunktion (blau): Man sieht, dass einige Durchläufe mit gemischter Rewardfunktion einen ähnlichen Verlauf nehmen wie die mit Standardrewardfunktion. Andere Läufe machen kaum Gewinn und leiden unter starker Instabilität.

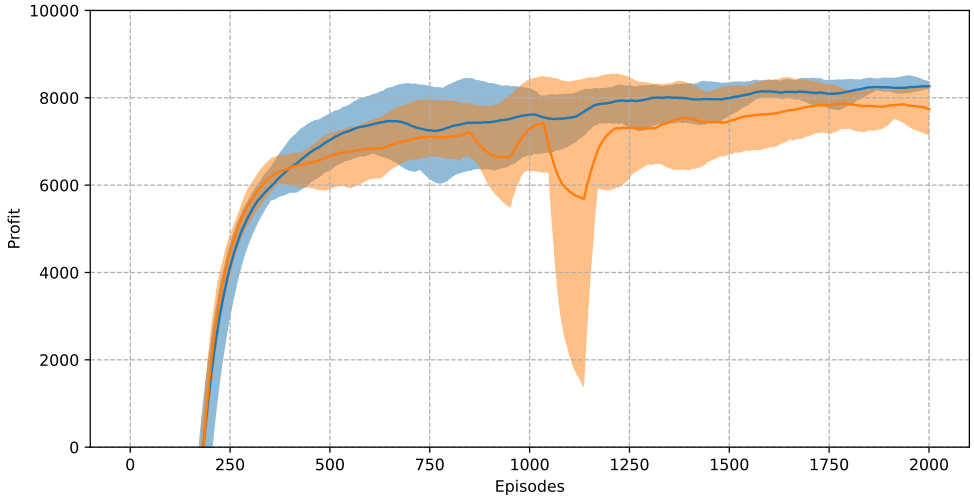

Abbildung 5.6: Lernkurve des Proximal-Policy-Optimization-Algorithmus bei gemischter Rewardfunktion (orange) im Vergleich zur Standardfunktion (blau): Bei gemischter Rewardfunktion sind die Returns etwas schlechter, wie es angesichts des veränderten Optimierungskriteriums zu erwarten ist. Weiterhin verringert sich die sehr hohe Stabilität von PPO beim Training mit gemischter Rewardfunktion etwas. Diese entstehende Instabilität ist aber gering und beeinträchtigt nicht die Ergebnisse.

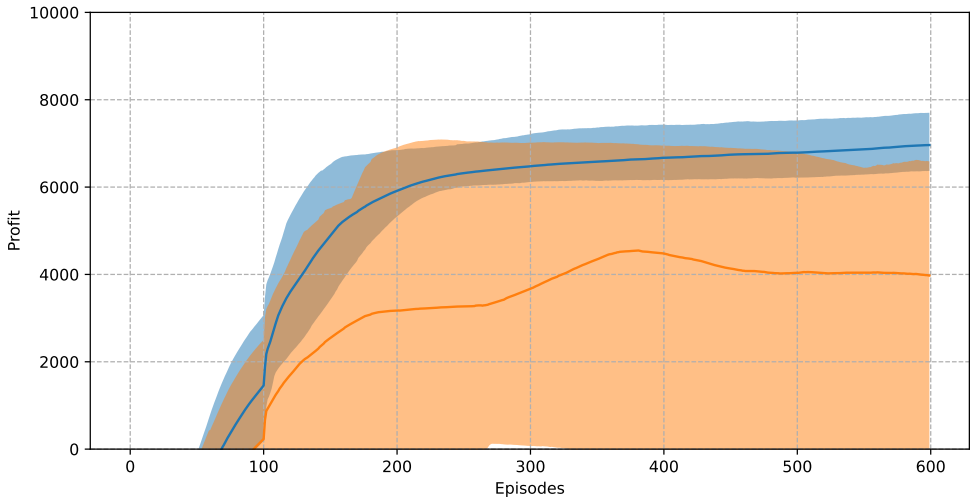

Abbildung 5.7: Lernkurve des Soft-Actor-Critic-Algorithmus bei gemischter Rewardfunktion (orange) im Vergleich zur Standardfunktion (blau): Ähnlich wie bei A2C in 5.5 gibt es Durchläufe mit gemischter Rewardfunktion, die ähnlich zu den Durchläufen mit Standardrewardfunktion sind. Es gibt aber auch Durchläufe, die nie die Gewinnzone erreichen. Diese Durchläufe erreichen sogar bezüglich der gemischten Rewardfunktion die besten Ergebnisse, weil sie erfolgreich eine Schwäche beim regelbasierten Konkurrenten finden. (siehe 5.8)

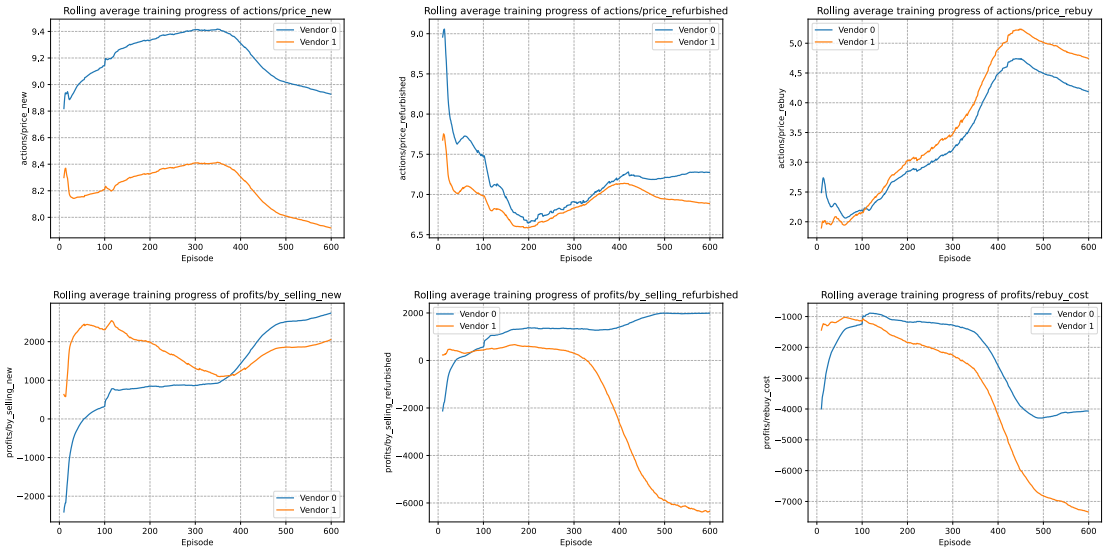

Abbildung 5.8: Detaillierter Blick in einen SAC-Trainingsdurchlauf gegen den regelbasierten Konkurrenten bei gemischter Rewardfunktion: Der SAC-Agent (Vendor 0) nutzt eine Schwäche des Konkurrenten (Vendor 1) aus und leert dessen Lager so, dass dieser Strafe für nicht bereitgestellte Produkte zahlen muss. Gleichzeitig versucht der regelbasierte Konkurrent, Produkte zurückzukaufen und gibt bei hohen Rückkaufpreisen dafür zu viel Geld aus.

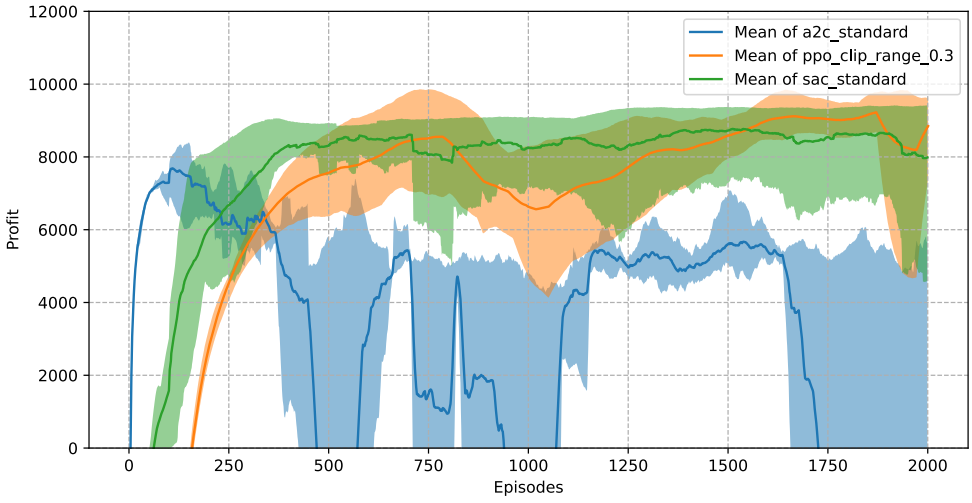

Abbildung 5.9: Lernkurven von vier A2C-, PPO- und SAC-Durchläufen beim Self-Play bei gemischter Rewardfunktion; die Algorithmen wurden über 2000 Episoden trainiert

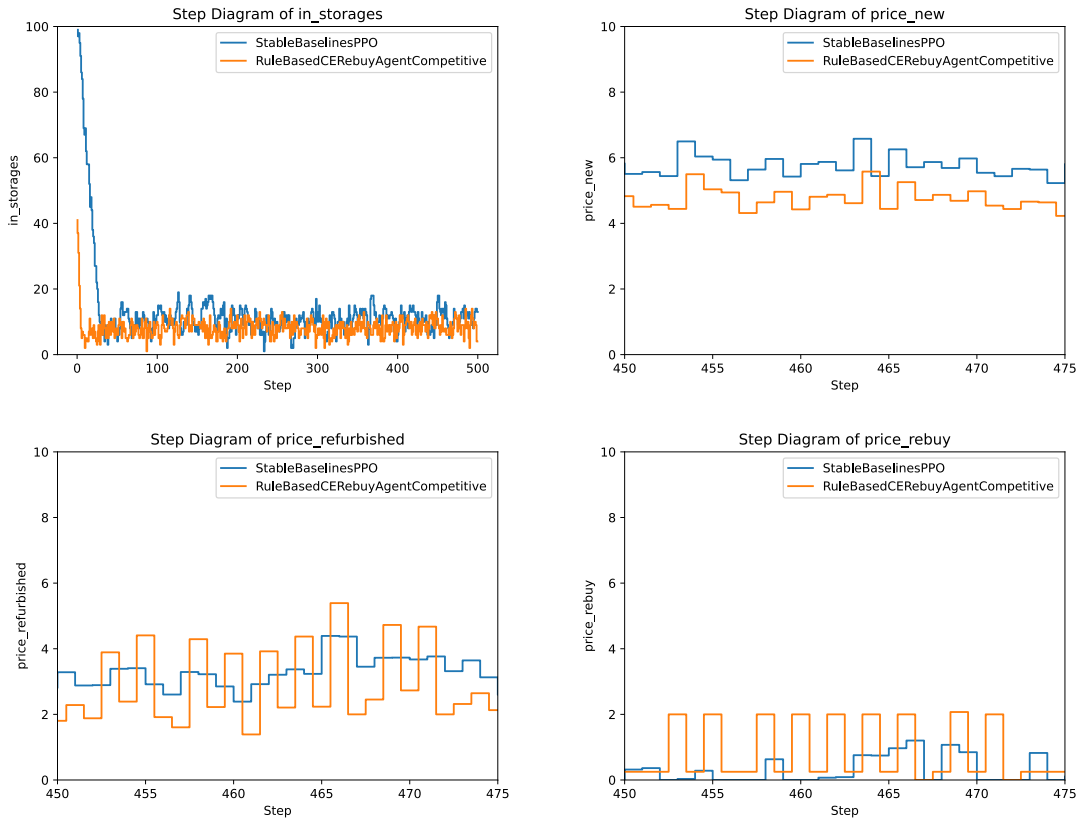

Abbildung 5.10: Details einer Episode, in der ein PPO-Agent (trainiert mit Self-Play und gemischter Belohnungsfunktion) gegen den unterbietenden regelbasierten Wettbewerber antritt

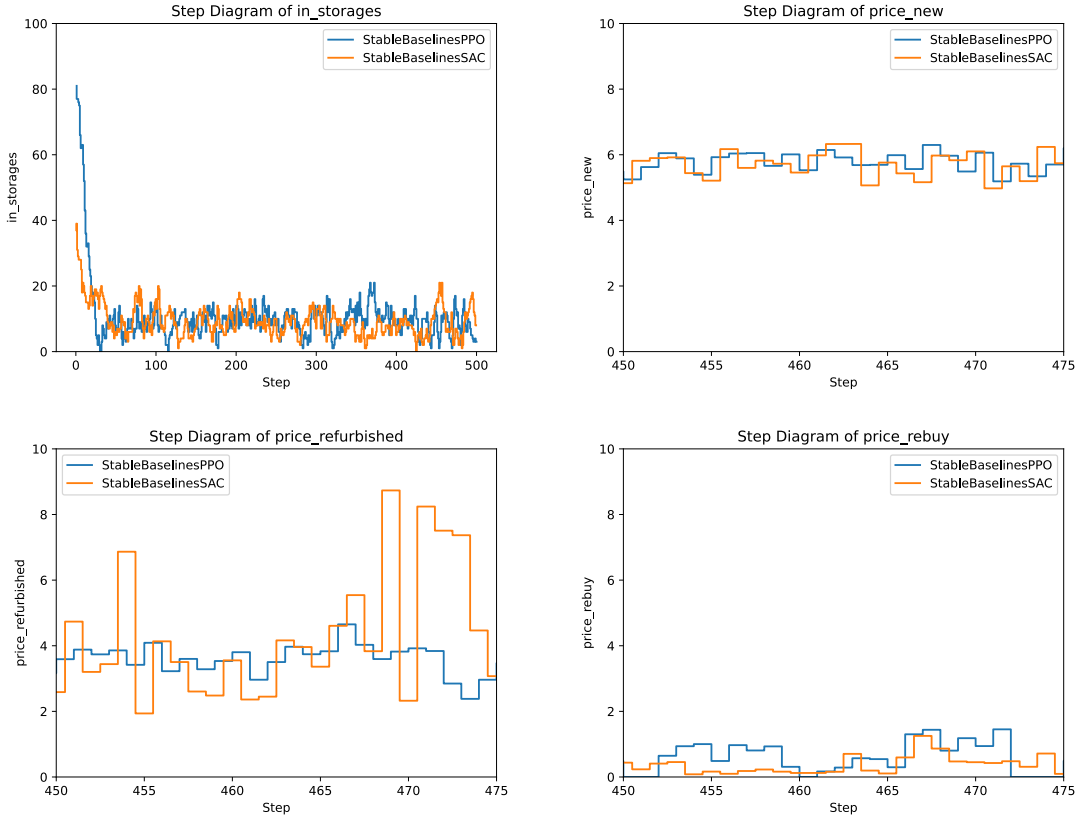

Abbildung 5.11: Details einer Episode, in der ein PPO-Agent (trainiert mit Self-Play und gemischter Belohnungsfunktion) gegen einen ebenso trainierten SAC-Agenten antritt

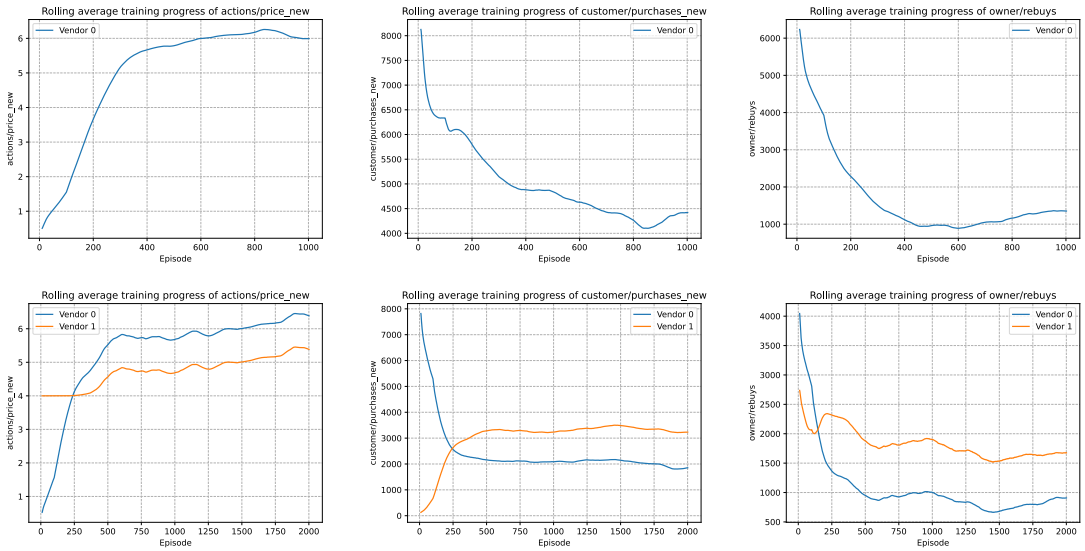

Abbildung 5.12: Messungen in PPO-Trainingsdurchläufen auf einem Monopol (oben) und einem Duopol (unten): Dass die gesetzten Neupreise trotz Monopol nicht höher sind, liegt daran, dass die Kaufbereitschaft der Kunden mit steigenden Preisen sinkt. Auf dem gleichen Preisniveau verkauft der PPO-Agent im Monopol doppelt so viele Neuprodukte, die Aktivität auf dem Rebuy-Markt ist ebenfalls etwa doppelt so hoch.

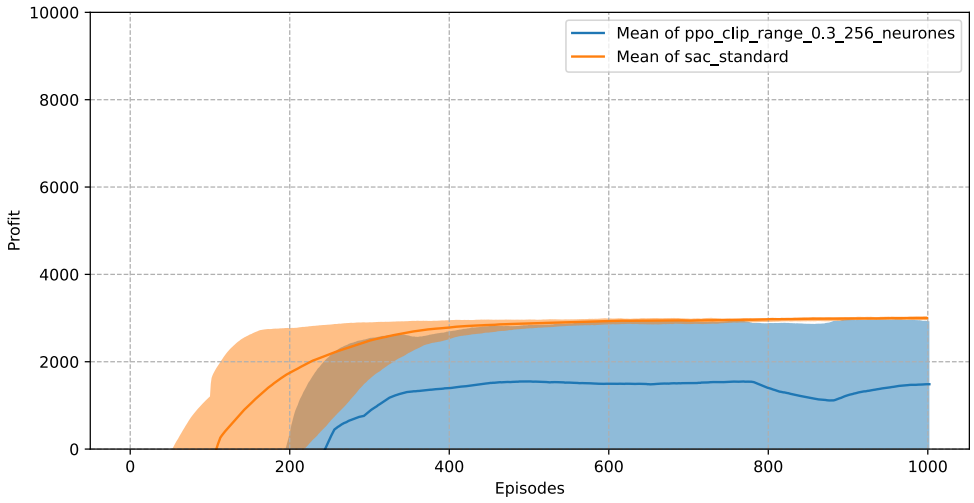

Abbildung 5.13: Lernkurven von vier PPO- und SAC-Durchläufen in einem Oligopolszenario über 1000 Episoden, wobei PPO mit der gleichen Netzwerkarchitektur wie SAC trainiert: Dabei erreichen die PPO-Agenten in ihren Leistungsspitzen Werte von 2800, 2900, 3010 und -1750. Die SAC-Agenten erreichen Werte zwischen 2950 und 3070. Damit ist der beste PPO-Durchlauf nicht mehr schlechter als die SAC-Durchläufe, allerdings sind die PPO-Durchläufe mit dem größeren Netz erheblich instabiler.

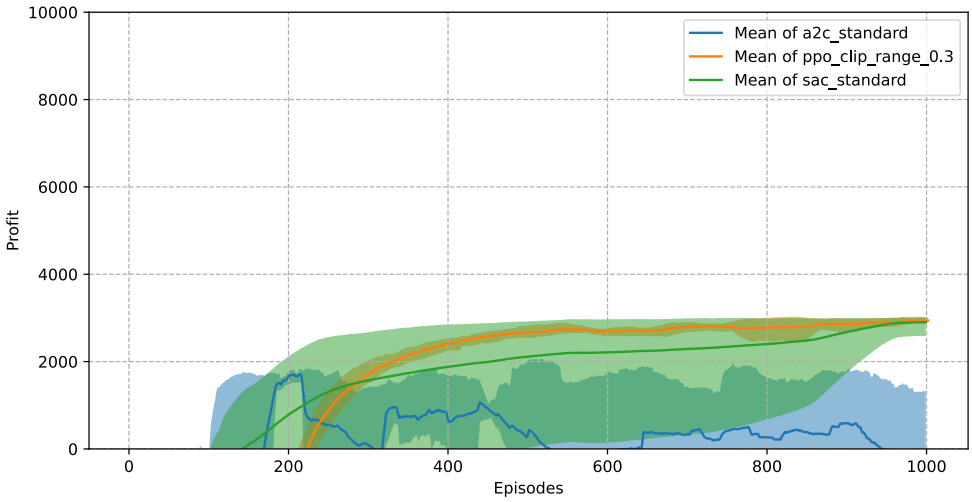

Abbildung 5.14: Lernkurven von vier A2C-, PPO- und SAC-Agenten in einem Oligopolszenario über 1000 Episoden mit gemischter Rewardfunktion und vollständiger Beobachtung

Eigenständigkeitserklärung

Ich erkläre hiermit, dass diese Arbeit eigenständig und ohne fremde Hilfe entstanden ist. Alle direkt oder indirekt verwendeten Quellen werden als Referenzen angegeben.

Potsdam, 18. Juli 2022 

Jan Niklas Groeneveld