Monitoring Reliability and Robustness of Agents for Dynamic Pricing in different (Re-)Commerce Markets

Monitoring der Zuverlässigkeit und Robustheit von Agenten zur dynamischen Bepreisung in unterschiedlichen (Re-)Commerce Märkten

Nikkel Mollenhauer

Universitätsbachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Science
(B. Sc.)

im Studiengang IT-Systems Engineering
eingereicht am 30. Juni 2022 am Fachgebiet
Enterprise Platform and Integration Concepts
der Digital-Engineering-Fakultät der Universität Potsdam

Gutachter	Dr. Rainer Schlosser
Betreuer	Johannes Huegle
	Alexander Kastius

Abstract

Sustainable recommerce markets are growing faster than ever. However, businesses now face the challenge of having to price the same item three times: A price for the new item, one for its refurbished version and the price at which items are bought back from customers. Since these prices are heavily influenced by each other, traditional pricing methods become less effective. To solve this dynamic pricing problem, a simulation framework was built which can be used to train artificial vendors to set optimal prices using Reinforcement-Learning algorithms. Before employing these trained agents in real markets, they must be tested on their reliability and robustness, as even the smallest mistake by the agent can lead to high costs for the business. This thesis introduces a number of ways that agents can be monitored. We come to the conclusion that the most effective tools are... ...todo

Zusammenfassung

Nachhaltige Recommerce-Märkte befinden sich in stetigem Wachstum. Dies stellt Unternehmen jedoch vor die neuartige Herausforderung, dasselbe Produkt mehrfach bepreisen zu müssen: Preise sowohl für die neue und generalüberholte Version sowie ein Ankaufpreis gebrauchter Ware müssen gesetzt werden. Da diese Preise voneinander abhängig sind, greifen traditionelle Methoden der Preisfindung schlechter. Zur Lösung dieses dynamischen Bepreisungsproblems wurde eine Simulationsplattform gebaut, auf der mithilfe von Reinforcement-Learning Algorithmen maschinelle Verkäufer für den Einsatz in realen Märkten trainiert werden können. Bevor dies jedoch geschehen kann müssen die trainierten Modelle auf Zuverlässigkeit und Robustheit überprüft werden, da bereits der kleinste Fehler zu hohen Verlusten aufseiten des Unternehmens führen kann. Diese Arbeit führt Methoden und Tools ein, die für ein solches Monitoring verwendet werden können. Es wird festgestellt, dass die effektivsten Tools dabei...

Übersetze ich die beiden Begriffe, oder lasse ich sie englisch?

...todo

Contents

1 Introduction

This thesis builds upon the bachelors project 'Online Marketplace Simulation: A Testbed for Self-Learning Agents' of the Enterprise Platform and Integration Concepts research group at the Hasso-Plattner-Institute. Therefore, the project will be referenced and all examples and experiments will have been conducted using its framework.

1.1 Objective of the Thesis

This thesis introduces ways to monitor the *Reliability* and *Robustness* of different agents (rule-based as well as trained using various Reinforcement-Learning (RL) approaches) tasked with dynamically pricing products in a Circular Economy marketplace. Since the terms *Reliability* and *Robustness* can be interpreted differently depending on context and personal experience, we will define our usage in the Reliability and Robustness section. Following the term definitions, we will give a short introduction ans explanation of what a Circular Economy market is (The Circular Economy model) as well as what Reinforcement Learning is and how we employ the technique in our framework (Using the simulated marketplace to train agents).

Reliability and Robustness

1. *Reliability*: With *Reliability*, we describe the ability of an agent to be able to transfer knowledge of a certain type of marketplace and/or against a certain opponent over to a different scenario. If Agent A performs well against Agent B on marketplace M, does it perform the same against Agent C on marketplace M, or against Agent B on marketplace N?

2. *Robustness*: *Robustness* is the property that describes how well an agent performs over a longer period of time. In a real-world marketplace, consistency is key to success, so finding profitability outliers and their causes are a central part of evaluating an agent's Robustness.

1.2 The Circular Economy model

The main goal of the aforementioned bachelors project was to develop a realistic online marketplace that simulates a Circular Economy. A market is most commonly referred to as being a *Circular Economy* if it includes the three activities of reduce, reuse and recycle [KRH17]. This means that while in a classical Linear Economy market each product is being sold once at its *new price* and after use being thrown away, in a Circular Economy, recycling and thereby waste reduction is a major focus. In our project, we modelled this by adding two additional price channels, *re-buy price* and *used price*, to the pre-existing *new price* of a product.

The *re-buy price* is defined as the price a vendor is willing to pay a customer to buy back a used product, while the *used price* is defined as the price the vendor sets for products they previously bought back and now want to sell alongside new products (whose price is defined by the *new price*). We will go into more detail of how we modelled different market scenarios in Market scenarios.

In the context of e-commerce, Circular Economy markets are also referred to as *recommerce* markets.

From now on, when mentioning the general *market* or *marketplace*, we are referencing the Circular Economy marketplace with re-buy prices.

Using the simulated marketplace to train agents

After the initial market was modelled the goal was to train agents using different reinforcement-learning algorithms to dynamically set prices on this marketplace, both in monopolistic scenarios as well as in competition with rule-based vendors which set prices following a strict set of pre-defined rules. These rules can range from simply undercutting the lowest competitor's price to more advanced techniques such as smart inventory management and reliance on previous sales data, which can in some cases and combinations even lead to price-fixing and -gouging between competing vendors (see Vendors for a more in-depth analysis). Furthermore, functionality was added that allows for different Reinforcement learning algorithms to be trained against each other on the same marketplace, as well as functionality for so-called *self-play*, where an agent plays against itself, or more precisely, against its own policy.

Reinforcement-learning agents are trained through a process of trial-and-error. They interact with the market through an observable state and an action which influences the following state. Figure 1.1 illustrates the RL-model in the context of our market. The goal of the agent is to maximize its reinforcement signal, which in our case is the profit the agent made during the last episode, since we want to

re-buy instead of buy-back in the diagram

Create a *nicer* diagram with the three prices

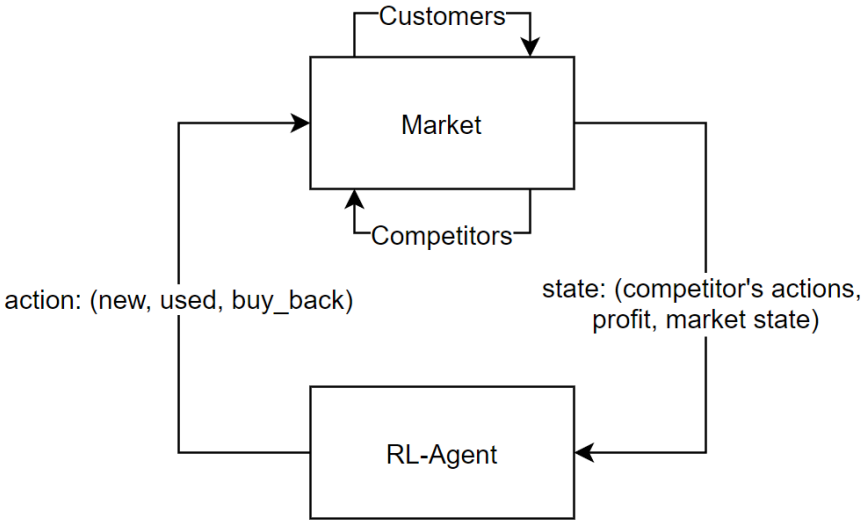

Figure 1.1: The standard reinforcement-learning model in the context of our market.

train agents to maximize profits on real markets. An episode consists of a fixed, configurable number of timesteps, where in each step each vendor (agent) sets their prices and customers make purchasing decisions. By observing which prices lead to which profits (reinforcement signal), the agents get more effective in their pricing decisions over the course of training.

2 Related Work

This section will outline the current industry-standards of how Reinforcement Learning agents are being evaluated, as well as the common problems and misconceptions that can arise from these techniques, together with proposed solutions. We will also give a brief overview over how this knowledge has influenced our simulation framework.

3 Simulating the marketplace

In this section we will outline the requirements and challenges of transferring the vast amount of features and parameters that influence vendor and customer decisions in real recommerce markets to our simulated market environment. We will take a brief look at different market scenarios as well as how our customers work, with a focus on the unique feauture of recommerce markets; the buyback of used items by the vendors. The focus of this chapter will however lie on the vendors, the rule based as well as the ones trained using Reinforcement Learning, comparing their features and how they fare against each other. These comparisons will be done using our various monitoring tools, which will be explained in the following chapter, Approaches to monitoring agents. While results of these experiments will be shortly touched upon, we will go into further detail in the closing chapter, Interpreting the results.

3.1 Market scenarios

In our framework, we implemented a number of 'market blueprints' both for classic Linear Economy markets, as well as for Circular Economy markets both with and without rebuy-prices enabled. There are marketplaces available for each combination of the following two features:

1. Marketplace type: Linear Economy, Circular Economy, Circular Economy with rebuy-prices

2. Market environment: Monopoly, Duopoly, Oligopoly

The marketplace type defines the number of prices the vendor has to set. In a Linear Economy, vendors only set prices for new items, in a Circular Economy prices for refurbished items need to be set as well. In order to simulate a recommerce market, the user can choose a Circular Economy with rebuy-prices. The market environment defines the number of competing vendors in the simulation: Monopolistic and competitive markets are available, where the Duopoly is simply a particular version of an Oligopoly. Depending on the chosen market environment, one, two, or any number of vendors can be chosen to be used in the experiment.

In the most common usecase of our framework, the training of Reinforcement-Learning agents, only the agent that is to be trained needs to be configured by the user. For this reason, each market environment is equipped with a pre-defined set of competitors that will play against the agent defined by the user. To allow for more control over the simulation, users are however also able to customize this 'competitor-list' to use any vendors they want - as long as they are a valid fit for the marketplace type and the number of competitors chosen matches the market environment. In certain situations, such as the Agent-monitoring tool, multiple Reinforcement Learning agents can be chosen to compete on one market.

Market state random shuffle

3.2 Customers

Customers are at the center of every type of marketplace, which makes them an integral part of our market simulation. However, since each customer in the real market is an individual with different reasonings and makes purchase decisions based upon personal preferences, modelling a realistic depiction of real-world customers proves to be very difficult and time-consuming. For this reason we decided to keep our initial implementation of the customers as simple as possible, taking into account future extension and scalability concerns. As we are focussing on dynamic pricing and therefore allow our vendors to only change the prices of their products, we decided on building customers that value price over any other features a product may have. Additionally, in Circular Economy markets, customers have preferences on whether they would like to buy new or refurbished items.

Once customers have decided to buy a product from any of the available vendors, they turn into an *owner*. In the next step of the simulation, they are offered the option of selling their now used product back to one of the vendors. If they decide to do so, the vendor pays them the advertised *re-buy price* and adds the used product to their inventory, from where it can then be sold as a refurbished procuct in the next step.

3.3 Vendors

Vendors are the main focus of our market simulation.

While our framework will not be able to reproduce all types of pricing models used in the real market, we strive to model as many different models as possible (and feasable in the scope of the project). We will use the following four types of dynamic pricing models, as described in 'Dynamic pricing models for electronic

business' [Nar+05], to group our different approaches and see how each type performs in the context of our simulation framework:

1. *Inventory-based models*: These are pricing models which are based on inventory levels and similar parameters, such as the number of items in circulation. In our framework, almost all rule-based agents consider their inventory levels when deciding which prices to set. The only exception to this rule are the simplest of our agents, the *FixedPriceAgents*, which will always set the same prices, no matter the current market state and competitor actions. We chose to implement this rather incompetent type of agent, as there are still many vendors in the real market which do not make use of dynamic pricing models, but use static pricing methods like these *FixedPriceAgents* instead. Not including these in our framework would therefore remove a fraction of realism.

   Inventory-based models are comparatively easy to implement, as they only depend on data immediately available to the vendor. This has the advantage that rule based agents based on this technique are relatively simple to create and modify. Examples of *Inventory-based agents* in our framework can be found in the *RuleBasedCEAgent*, one of *RuleBasedCEAgent* the first rule based agents we created. This agent does not take pricing decisions of its competitors in the market into account, but simply acts according to its own storage costs. While its performance is not necessarily bad, it is still one of the weakest competitors currently available in the framework.

   Compare this agent with other agents, RL, FixedPrice, better rule-based!

2. *Data-driven models*: *Data-driven* models take dynamic pricing decisionmaking one level further. They utilize their knowledge of the market, such as customer preferences, past sales data or competitor prices, to derive optimal pricing decisions. Aside from the aforementioned *FixedPriceAgents* and the basic *RuleBasedCEAgent*, all of our other rule based agents fall into this category. One of the most prominent examples of a *Data-driven model* is the *RuleBasedCERebuyAgentStorageMinimizer*. This agent bases its prices on two major factors. First, it reacts to its competitors prices by matching its initial prices to the median price of its competitors, and then adjusts them according to current inventory. Notably, all of our *Data-driven models* are also *Inventory-based* to a certain extent, as handling storage plays a big part in a Circular Economy market setting, where used products need to be bought back from customers and subsequently undergo refurbishment while in inventory of the company. *Data-driven models* have proven to be the most competent rule based agents in our recommerce market scenario, in particular the above

described *RuleBasedCERebuyAgentStorageMinimizer*, which only consists of ten lines of code but is still able to outperform seemingly more complex *Inventory-based models* .

Graphs

3. *Game theory models*:Game theory concerns itself with the study of models for conflict and cooperation between rational, intelligent entities [Mye97]. It is therefore often applicable in situations where competing individuals, acting rationally and selfishly interact with each other. In our framework, competitors in the market are influenced in their decisionmaking processes by the actions of other participants of the scenario. While none of our rule based agents have been specifically designed to act according to game theoretic strategies, due to the fact that almost all of them consider their competitors prices in their pricing decision, and due to the nature of Reinforcement Learning trying to maximize their own profits without regard to their competitors performance, behaviour according to Game theory can sometimes be observed. During training, Reinforcement Learning agents observe the market state, which includes prices and sales data not only of themselves but also the other vendors in the market. Using this data, the algorithms try to predict how their prices will influence customer behaviour as well as the competing vendors. In some cases, depending on the concrete behaviour of the competitors, vendors may cooperate in driving prices higher together, in other cases the agents may act more selfishly, sacrificing own profits to stay ahead of their competitors in other regards.

Need graphs to back this claim up! Especially price-graphs, moving up and down etc.

Graphs

Make sure to get some comparisons between these algorithms going!

4. *Machine learning models*: All of our Reinforcement learning agents fall into the category *Machine learning models*. As it is not the focus of this thesis, we will not go into detailed explanations of the various models used, but instead focus on how they perform in relation to each other. The algorithms used by us can be grouped together into the following three categories:

   - *Q-Learning*: *Q-Learning* agents were the first Reinforcement Learning agents we introduced in our framework, as it is one of the easier Reinforcement-Learning algorithms to implement [KLM96]. However, the drawback of using *Q-Learning* in our framework is that it can only be applied to discrete action and state spaces. In the initial exploration-phase of our simulation framework this was not a problem, as relatively small action spaces were used, adapted to this limitation. This means that when using *Q-Learning*, only 'whole' prices can be set, and any decimal places must be omitted. This of course limits the framework in its realism, as the fine-tuning of prices using decimal places can be

This must not be 'cited', right?

critical in influencing customer decisions. It should be mentioned that while this limitation is known to us, our simulation framework still operates on discrete action and state spaces, both to allow algorithms like *Q-Learning* to still function and to not overly complicate the simulation, as many of the other components (such as customer behaviour, see Customers) are still very basic in their nature as well. The *Q-Learning* used in our framework have been custom implemented by us, using the *pytorch*[1] framework. While approaches for *Q-Learning* algorithms that can work with continuous action and state spaces have been presented in the past ([GWZ99], [MPD02]), we have chosen not to implement such an algorithm in our framework, but rather explore more approaches other than *Q-Learning*, introduced below.

- *Actor-Critic*: *Actor-Critic* algorithms are more complex than Q-Learning algorithms, and have therefore been implemented later in the process. They are structured different then Q-Learning algorithms in the way that they are 'split' into two parts: The *actor* is responsible for selecting actions, while the *critic* is responsible for critizing the actions taken by the *actor*, thereby improving its performance [SB18]. Similar to the *Q-Learning* agents, the *Actor-Critic* algorithms have also been implemented by us. In total, one discrete and two continuous *Actor-Critic* agents can be used, in addition to those provided through *Stable-Baselines*, see the next section.
- *Stable-Baselines*: Stable-Baselines provides a number of open-source implementations for various Reinforcement-Learning algorithms. In our framework, we use the latest version of these algorithms, through *Stable-Baselines3* [Raf+21]. The advantage when using algorithms provided through Stable-Baselines lies in the fact that they need close to no custom implementation from our end, we can instead interact with them through very simple interfaces. This cuts down on the amount of time and effort that needs to be spent developing, implementing and maintaining these powerful algorithms, and allows us to introduce a higher number of algorithms than would otherwise be possible. Currently, five different *Stable-Baselines3* algorithms are used in our simulation framework, see the *Stable-Baselines3* documentation for more information [Bas22]:
  - *A2C*: Advantage-Actor-Critic

1 https://pytorch.org/docs/stable/index.html

- *DDPG*: Deep deterministic policy gradient
- *PPO*: Proximal Policy Optimization
- *SAC*: Soft Actor-Critic
- *TD3*: Twin-delayed deep deterministic policy gradient

We will be using some of these algorithms in our experiments and briefly compare them with our Rule-Based approaches using our various monitoring tools, which are introduced in Approaches to monitoring agents.

3.4 Diverging from the real market

We do not claim in any way that our simulation framework is exhaustive or complete. There are a number of parameters influencing market dynamics, small and great, which have not been modelled due to time constraints or their unpredictability in their affect on the market state or customer decisions. Seasonality of demand, customer retention and loyalty through branding and unforeseen effects on market dynamics such as global events (for example, the Covid-19 pandemic) are just some of the places where our market simulation diverges from the real market. However, all of these factors are either very rare in nature (global, unforeseen events), only applicable to a subset of markets (brand loyalty) or can initially be discarded as their effect on market dynamics is predictable, making them a lower priority than other parameters (seasonality).

Of course, this results in the fact that our framework can not be used out-of-the-box for any kind of market. However, great care was taken during the development process to build each part of the simulation in a modular way, so that new functionality can easily be integrated into it. Marketplaces, customers and vendors (rule-based as well as those using Reinforcement learning) have all been implemented as classes disconnected from each other, so that new variants of each can easily be added to the pool of available options to be used in experiments. The same goes for our monitoring tools, all of which have been built to work with any (valid) combination of input parameters.

The lack of realism on certain areas of the framework does not directly impact the way that our monitoring tools work or can be used. It must however be noted that the interpretation of results must always be based on the knowledge of these limitations.

4 Approaches to monitoring agents

In this section we will take a look at the different approaches we took to monitoring agents in our framework, explaining the reasons why we chose to implement specific features and how they help us in determining an agents strengths and weaknesses.

4.1 When to monitor what

Our *workflow* (which will be explained in more detail in The *recommerce* workflow) can generally be split into two parts when it comes to monitoring and evaluating agents: *during* and *after* training. When talking about the *workflow* we refer to the process of configuring and starting a training session, where a Reinforcement-Learning agent is being trained on a specific marketplace against competitors. The *workflow* also includes the subsequent collection of data used to evaluate the agent's performance. We are also introducing the term of the *complete agent* in this section, which will be used to refer to both Reinforcement-Learning agents that have been fully trained as well as rule-based agents which do not need training.

As mentioned above, we split our monitoring tools into the following two categories:

1. During training: Having data available as soon as possible without having to wait for a long training session to end is crucial to an efficient workflow. Our framework enables us to collect, visualize and analyze data while a training session is still running. This enables us to filter out agents with sub-par performance prematurely. This can be done using user-defined threshholds or on the basis of previoulsly collected data (e.g. only keeping agents that are performing better than at least 50% of other agents after the same amount of training in comparable scenarios.)

2. On complete agents: After a training session has finished we have a complete and final set of data available for an agent, which enables us to perform more thorough and reliable tests. These can include simulating runs of a marketplace to gather data on the agent's performance in different scenarios and against different competitors, or running a static analysis of the agent's

Implement this feature. It should at least be able to temporarily halt the training to start an intermediate monitoring session

Also implement this feature. Allow users to set rules for when a training session should be terminated if the agent is not performing well at a certain point. Also, it should be able to give the program a set of datapoints and have it set rules if possible

policy in different market states. The tools available for trained agents are in the same way also usable on rule-based agents.

If we also conduct experiments in this chapter, mention it here

In the following sections, we will take a look at all of the tools our framework provides for monitoring agents, distinguishing between the two general types of monitoring mentioned above. The goal of this section is to give a short overview of each tool, how and why they were implemented and what value they offer to the framework as a whole and to the analysis of agent reliability and robustness in particular. We will also discuss features that are currently not available, explaining how they could benefit the entire workflow or enrich the overall experience.

4.2 Monitoring during a training session

Is this a footnote? This info should be obvious, but maybe should be mentioned for completeness?

When talking about monitoring agents during a training session, we are of course talking about Reinforcement-Learning agents, since Rule-Based agents always perform the same and cannot be trained. Monitoring agents while they are still being trained enables us to be more closely connected to the training process. Ultimately, the goal of such monitoring tools is to be able to predict the estimated 'quality' of the final trained agent as reliably as possible while the training is still going. Users must however be careful when interpreting the results of monitoring tools that work on 'incomplete' agents, we will go more into this in Interpreting the results.

Tensorboard

Citation or footnote?

Question for the tutors: Do I give an example of such code?

The *Tensorboard* is an external tool from the from the Reinforcement-Learning library *Tensorflow*[2]. With just a few lines of code a training session can be connected to a Tensorboard. We are then able to pass any number of parameters and metrics we deem interesting or useful to the Tensorboard, which then offers visualizations for each of them, updating live as the training progresses. Though the Tensorboard does not interpret data in any way, it is an immensely useful tool for quickly and easily recording data and offering a first rough comparison of competitors in the market.

Create some sample diagrams. Perhaps these can be re-used in the workflow section, so they could perhaps be moved to the Appendix for reference?

Live-monitoring

Unlike the Tensorboard, the monitoring tools summarised under the term *live-monitoring* were completely and from the ground up built by our team. For most

2 https://www.tensorflow.org/

of the visualizations, the *matplotlib*[3] library was used. Live-monitoring aims to be more configurable and in-depth with the metrics it offers than the aforementioned Tensorboard. During a training session, 'intermediate' models, as we will call them, are being saved after a set number of episodes. These models contain the current policy of the agent and can be used the same as models of complete agents, the only difference being the quality of the agent, as those with a lower amount of trained episodes generally perform worse. These intermediate models can then be used by a range of monitoring tools available to us. Since the models only contain the current policy of an agent but not the history of states and actions preceding the model, we need to run separate simulations on these models to be able to analyze and evaluate them. For this, we utilize our *agent-monitoring* toolset, which will be explained in more detail in the Agent-monitoring section.

make the current live-monitoring so that graphs are actually being created while the training is running, not like it is now with graphs only being created afterwards

Have we introduced the concept of episodes before? Should a Glossary be introduced, or do we do this stuff in the introduction?

Does this need a citation?

4.3 Monitoring complete agents

The following tools offer a wide range of functionalities for monitoring complete agents. We can use these tools to both determine the reliability and robustness of one certain agent, but also to compare different agents, either during a joined monitoring session or by comparing results of monitoring runs on identically parametrized market situations.

Agent-monitoring

Using the *Agent-monitoring* toolset, users can configure a custom market simulation, using the following parameters:

1. Episodes: This parameter decides how many independent simulations are run in sequence. At the start of each episode, the market state will be reset. Within an episode, each vendor runs through 50 timesteps, during each of which a price is set (depending on the chosen economy type, this can range from only one price for new items to three prices, including a re-buy price for used items) and a set number of customers interact with the vendors.

2. Plot interval: A number of diagram types enable the user to view averaged or aggregated data over a period of time. The plot interval parameter decides the size of these intervals. Smaller intervals mean more accurate but also more convoluted data points. Computational time also increases with a smaller interval size.

These diagrams can be seen/are explained in...

3 https://matplotlib.org/stable/index.html

3. Marketplace: Using this parameter, the user can set the marketplace on which the monitoring session will be run. See Market scenarios for an explanation of the different available marketplaces.

4. Separate marketplaces: This parameter is a boolean flag that determines the way in which the monitoring session will handle the agents given by the user. If the flag is enabled, each agent will be initialized on a separate instance of the set marketplace, meaning that the agents will be monitored independently from each other. In this case, each agent will play against the pre-defined set of competitors, depending on the chosen marketplace (see also Market scenarios). This functionality is most useful when directly comparing two or more agents against each other, as all of them will be playing under the same circumstances, without influencing each other. While it may seem like the same results could be reahced by simply starting multiple monitoring sessions with a different agent each, this is not the case. Using this flag, it is ensured that all agents get the exact same starting conditions for each episode. Normally, the market state is randomly shuffled at the beginning of each episode (see Market scenarios), which would lead to inconsistent results. By running multiple marketplaces in parallel using the *Agent-monitoring* tool, we can ensure that all agents are given the same states at the start of an episode, to to match the simulations as closely as possible. However, we cannot influence the competitors' behaviour in the different marketplaces, as these are always dependent on the agents' actions and can therefore not be replicated on different markets. If the flag is disabled, the monitoring tool will initialize only one marketplace and set the passed agents to directly compete against each other on this marketplace. In this case, the number of agents that must be passed is determined by the number of competing vendors defined by the market environment. This functionality is most useful when evaluating a single agent, trying to determine its specific strengths and weaknesses against certain opponents, something that would not be possible with the static pre-set competitors. In this sense, the configuration can be used in two ways: To either have multiple trained agents play against each other to find out which one beats the other, or to monitor a single agent against a custom selection of rule-based competitors not available otherwise. In order to confirm trends seen during the training process, the same configuration can be re-used for this tool to have the trained agent play against the same competitors on the same market again. Similarly, micro-adjustments can be made to analyze how the trained agent reacts to changes to his learnt environment.

5. Agents: Depending on the chosen marketplace, only a select number of valid agents can be chosen to be monitored, as each agent is built to interact with a specific type of marketplace. First off, all agents belong to one of the two major categories: *Reinforcement Learning agent* or *Rulebased agent*. Reinforcement Learning agents can only be monitored on the same marketplace type they were trained on, while the market environment may be changed. Rulebased agents can similarly only be used on the marketplace type they were built for, as each of them makes assumptions about the number of prices they will need to set. This leads to not all marketplace types having the same amount of rule based vendors available. Following their importance for our simulation framework, the Linear Economy has the least and most often weakest vendors available, while the more refined competitors are most of the times only available as a version compatible with the Circular Economy with rebuy prices.

During each episode and for each vendor, all actions and market events are being recorded using *watchers*. At the end of a monitoring session, the collected data is evaluated in different visual formats. First of all, all data that would be available to see in the *tensorboard* during a training session is visualized using density plots. These plots can be used to compare the vendors in a more granular way, if for example the effect of a parameter on the customer's sell-back behaviour of used items should be tested. Another view that is created is a bar diagram containing the cumulative profits per episode for each agent plotted against each other. This allows for a quick overview to see which agent had an overall better performance.

Do I keep this new word? If yes, it should be explained at least a little bit!

The most common usecase of the *Agent-monitoring* tool is to test trained agents against competitors other than the ones it was trained against. This is done to test the *Reliability* of the agent, an important factor in deciding an agent's quality, as its competitors in the real market will differ from any it has encountered in training, due to the sheer vastness of options when it comes to dynamic pricing models available nowadays, see Vendors.

Exampleprinter

The *Exampleprinter* is a tool meant for quickly evaluating an agent in-depth. When run, each action the monitored agent takes is being recorded, in addition to market states and events, such as the number of customers arriving and the amount of products thrown away. For certain market scenarios such as the *CircularEconomyRebuyPriceDuopoly*, which simulates a circular economy model with rebuy prices in which two vendors compete against each other, these actions and events are also

being summarised as an animated graphic, where each time-step is being illustrated.

Add in a graphic here. Perhaps have two of the time-steps to 'simulate' the animation

Policyanalyzer

The *Policyanalyzer* is our only tool which does not simulate a run of the market scenario. Instead, the tool can be used to monitor an agent's reaction to different market events. The user can decide on up to two different features to give as an input, such as the competitor's new and used prices, and the Policyanalyzer will feed all possible input combinations to the agent and record its reactions.

Diagram

4.4 Features for the future

This section is meant as a collection of ideas and approaches for tools that could further enhance the workflow, but which have not been implemented and therefore not been tested for their feasability and usefulness.

5 The *recommerce* workflow

The main goal of the bachelor's project is to provide a simple-to-use but powerful interface for training Reinforcement-Learning algorithms on highly configurable markets for users in both a research and a business context. To achieve this, multiple components had to be developed and connected to create the workflow we now provide. This section will go over the most important parts of the workflow, focusing on the way each of them supports the monitoring capabilities of the framework.

Better word for components

Diagram for the whole workflow?

5.1 Configuring the run

Unarguably, the most important part of the whole workflow is its configuration. Without it, each simulation and training session would produce similar, if not the same results. By tweaking different parameters of a run, market dynamics can be changed and agent performance be influenced. The goal of our monitoring tools is to enable users to assess the extent to which each parameter influences certain characteristics of the training and/or monitoring session, and to enable them to make informed decisions for subsequent experiments.

Is this 'zu wertend'?

Ultimately, all configuration is done using various .json files which contain key-value pairs of the different configurable items. We further differentiate between different groups of configurations, which means that hyperparameters influencing the market, such as maximum possible prices or storage costs, are being handled separate from parameters needed for Reinforcement-Learning Agents, such as their learning rates, allowing users to easily change and tweak parameters involving different parts of the framework.

Our main goals when building our configuration tools were to make sure that users are able to quickly and safely configure their experiments in a way that is also easily reproducible. To be as user-friendly as possible, a lot of validation logic has been implemented to be able to catch invalid configurations as early as possible, making sure that whenever a training session is started, it is confirmed that the configuration is valid and the training will not fail due to wrongly set parameters at any point. Since our project has also been deployed to a remotely accessible, high-performant machine, we decided on creating an approachable web-interface

for our configuration tools, which can now be used for both configuring, starting and evaluating experiments.

The webserver/Docker-API

Example screenshot of the webserver. Configuration and running page?

On our webserver, users can upload .json files containing any number of (valid) key-value pairs, or create configurations using a self-validating form, always making sure that it contains only valid values. For example, if the user chooses a Circular Economy market scenario with re-buy prices enabled, the user will not be able to configure agents that cannot operate on such a market.

Aside from uploading or configuring complete, ready-to-go configurations, users can also choose to upload or create partial configurations, which can then be combined with other configuration objects to create a complete, valid configuration. This allows users to create multiple incomplete configurations containing only select parameters and then test all permutations of these configurations to observe the effect the different parameter combinations have on the experiment.

Mention that this currently needs to be done manually, future work section?

An example of one such approach can be found below:

Example of a mix and match of parameters

After having configured the experiment, the webserver also offers the option of starting multiple runs of the same configuration simultaneously. Most of the times this is recommended, as singular runs are prone to misleading results due to the trained agents training with specific market states, which in some cases may lead to behaviour unique to this starting configuration. By running the same configuration multiple times using different starting circumstances for both the agent that is to be trained as well as its competitors, a configuration's effect on the agent's performance can be disconnected from the pseudo-random market states which influence the vendor's decision making processes.

Example: Two completely different runs with the same configuration

Is this perhaps too much interpretation for this section?

A configuration can be interpreted as producing 'reliable' agents if any number of simulations all converge on similar agent performances (meaning similar mean rewards). The more singular runs diverge from the mean, the more the agent's performance is dependent on the initial market state, meaning that its performance is less stable, as it will produce unpredictable results on new, unknown market scenarios. A higher number of runs that produce agents which perform on similar levels of performance means that the specific configuration, to be exact the parameters responsible for the Reinforcement-Learning algorithm, produces 'reliable' agents: No matter the state of the market, the agent can always be expected to perform on the same level that was observed during the training process, setting prices that lead to similar profits. Reliability is therefore one of the most important factors in determing an agent's overall quality, as the ultimate goal is always to be able to deploy the agent onto the real market, which will always differ from

mean or median? What do I use?

the simulated environments the agent experienced during training, simply due to the fact that real markets are being influenced by so many more parameters than one could model in our, or for that matter any simulation framework. It should be noted that the chosen Reinforcement Learning algorithm drives the reliability of a configuration, as many of the other hyperparameters most often have no impact on this aspect of an experiments results.

5.2 Choosing what to show the user during training

During the training process, we of course record all actions the different vendors take, both for the training agent and its competitors, be they Reinforcement Learning agents as well, or Rulebased agents. Market states and events are also being recorded, to be able to match them to the agent's actions later on. Users of course want to have an indicator of how the training run is going so far, both to inform themselves about the total progress of the training, but also about the quality of the agent they are training. We now have to decide what information is shown to the user, making sure that the user is well informed, whilst also holding back on displaying too much information at once to prevent confusion, as the underlying data and states are ever-evolving, leading to many datapoints being invalidated shortly after their creation. This potential overflow of information is the reason why we decided on displaying minimal information to the user while the training session is still running: Training progress and current performance, in the form of the current overall mean profit.

The first choice and a save bet when looking at datapoints that will be shown to users is the current training progress. Each training run is defined in its length by a parameter that sets the number of episodes that should be simulated. Each episode consists of an independently initialized market state on which a configurable amount of steps is run. Within each step, the vendors take turns in observing the current market state and then setting their prices for the new and used products and depending on the chosen marketplace, a rebuy price. We use the training progress to drive the frequency with which data is being shown to the user using the console: Every 10 episodes, we output the current progress, containing both the current number of steps and episodes completed, as well as the mean reward over all episodes the agent has completed. Additionally, to give a more visual component to the training progress a progress bar is displayed, which also tries to estimate the training-time remaining, based on the previous length the simulation of an episode took.

Should episodes and steps be defined/explained earlier than this? In the introduction? Specific section/Chapter for important terms and phrases?

Using only these two datapoints, the user knows the two most important things:

How far along in the training process he is, and how the agent is currently performing.

Small paragraph for rl-vs-rl and self-training having different output

The monitoring workflow

Within the recommerce workflow, there are two points in time where our monitoring tools are or can be used. While a training session is running, the Tensorboard tool is used to give insights into the current training run and detailed explanations of current states and actions. By saving intermediate models, the Live-monitoring enables the user to compare agent models saved at different times within the training process. This can be especially useful when trying to determine the most effective amount of training steps after which a model does not improve further, to optimize future runs. After the training has finished, or at any point disconnected from a training session, our tools described in Monitoring complete agents can be used to further analyze the trained models.

After training

After training has completed, the most crucial phase of the workflow starts. As has been mentioned before, the end-goal of users using the simulation framework is to be able to deploy trained agents into real markets to set prices independently of human inputs or guidance. Starting from this premise, the goal of the training process is therefore to model the marketplace as realistically as possible, so that the trained agents can transfer the knowledge from their training to the real market 'without noticing a difference'. Before deploying agents to the real market, they need to be tested on their reliability and robustness, to make sure that their policies are not just effective under certain circumstances, but that they are also able to adapt to different market scenarios and behave accordingly.

Which datapoints prove to be/are most effective?

6 Interpreting the results

Graphs and diagrams are available...

...comparing with other agents/models

...which hyperparameters influence the results in what ways?

...can we augment e.g. Grid-Search with our analysis?

-> Would need to make results ‘machine-readable’ again

7 Conclusions & Outlook

Bibliography

[Bas22] Stable Baselines3. *RL Algorithms*. Accessed: 2022-06-06. 2022. URL: https://stable-baselines3.readthedocs.io/en/master/guide/algos.html (see page 11).

[GWZ99] Chris Gaskett, David Wettergreen and Alexander Zelinsky. **Q-Learning in Continuous State and Action Spaces**. In: *Advanced Topics in Artificial Intelligence*. Ed. by Norman Foo. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, 417–428. ISBN: 978-3-540-46695-6 (see page 11).

[KLM96] Leslie Pack Kaelbling, Michael L. Littman and Andrew W. Moore. **Reinforcement Learning: A Survey**. *Journal of Artificial Intelligence Research* 4 (1996), 237–285. DOI: https://doi.org/10.1613/jair.301. URL: https://www.jair.org/index.php/jair/article/view/10166 (see page 10).

[KRH17] Julian Kirchherr, Denise Reike and Marko Hekkert. **Conceptualizing the circular economy: An analysis of 114 definitions**. *Resources, Conservation and Recycling* 127 (2017), 221–232. ISSN: 0921-3449. DOI: https://doi.org/10.1016/j.resconrec.2017.09.005. URL: https://www.sciencedirect.com/science/article/pii/S0921344917302835 (see page 2).

[MPD02] José del R. Millán, Daniele Posenato and Eric Dedieu. **Continuous-Action Q-Learning**. *Machine Learning* 49:2 (Nov. 2002), 247–265. ISSN: 1573-0565. DOI: 10.1023/A:1017988514716. URL: https://doi.org/10.1023/A:1017988514716 (see page 11).

[Mye97] Roger B Myerson. **Game theory: analysis of conflict**. Harvard university press, 1997, 1 (see page 10).

[Nar+05] Y. Narahari, C. V. L. Raju, K. Ravikumar and Sourabh Shah. **Dynamic pricing models for electronic business**. *Sadhana* 30:2 (Apr. 2005), 231–256. ISSN: 0973-7677. DOI: 10.1007/BF02706246. URL: https://doi.org/10.1007/BF02706246 (see page 9).

[Raf+21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus and Noah Dormann. **Stable-Baselines3: Reliable Reinforcement Learning Implementations**. *Journal of Machine Learning Research* (2021) (see page 11).

[SB18] Richard S Sutton and Andrew G Barto. **Reinforcement learning: An introduction**. MIT press, 2018 (see page 11).

Declaration of Authorship

I hereby declare that this thesis is my own unaided work. All direct or indirect sources used are acknowledged as references.

Potsdam, 7th June 2022

Nikkel Mollenhauer