Monitoring of Agents for Dynamic Pricing in different Recommerce Markets

Monitoring von Agenten zur dynamischen Bepreisung in unterschiedlichen Recommerce-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. Michael Perscheid
Betreuer	Dr. Rainer Schlosser 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: One 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, their fitness must be monitored and evaluated, 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 für gebrauchte Ware müssen gesetzt werden. Da diese Preise voneinander abhängig sind, greifen traditionelle Methoden der Preissetzung 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 bezüglich ihrer Eignung überprüft und bewertet 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...

...todo

Contents

1 Introduction

This thesis builds upon the bachelor's 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.

Alex: 'Klassendiagramm/ FMC-Diagramm für die Übersicht? Du nennst recht Häufig Komponenten, eine Gesamtübersicht wie sie interagieren wäre gar nicht schlecht.'

1.1 Objective of the Thesis

This thesis introduces ways to monitor and evaluate different agents (rule-based as well as trained using various Reinforcement-Learning approaches) tasked with dynamically pricing products in a *Circular Economy* marketplace. We will first introduce the general concept of a *recommerce* market (1.2 The Circular Economy model) and the structure of the framework we built (). In 2 Related Work we will explore other aproaches to market simulations for dynamic pricing, the general concept of Reinforcement-Learning as well as novel approaches to evaluating those agents. This will be followed by an overview of the specific features of a *recommerce* market that we implemented in 3 Simulating the marketplace. In 4 Approaches to monitoring agents we will give a detailed explanation of the different tools we built and used to monitor and evaluate our different agents. These tools will be put into context in 5 The *recommerce* workflow, where the different parts of the framework will be joined together and its modularity is highlighted. Finally, we will conduct a number of experiments using our tools in 6 Analyzing Graphs.

nameref in intro, where the diagram will be

Highlight modularity in that chapter

1.2 The Circular Economy model

The main goal of the aforementioned bachelor's project was to develop an online marketplace that simulates a realistic Circular Economy market environment. 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, a focus is put on recycling and thereby waste reduction. In our project, we first started by modelling the simpler Linear Economy, upon which we then built the more complex Circular Economy markets. This was done by adding two additional price channels, *re-buy price* and *used price*, to the pre-existing *new price* of a product. Refer to Figure 1.1 for an overview of the product lifecycle in a Circular economy.

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.

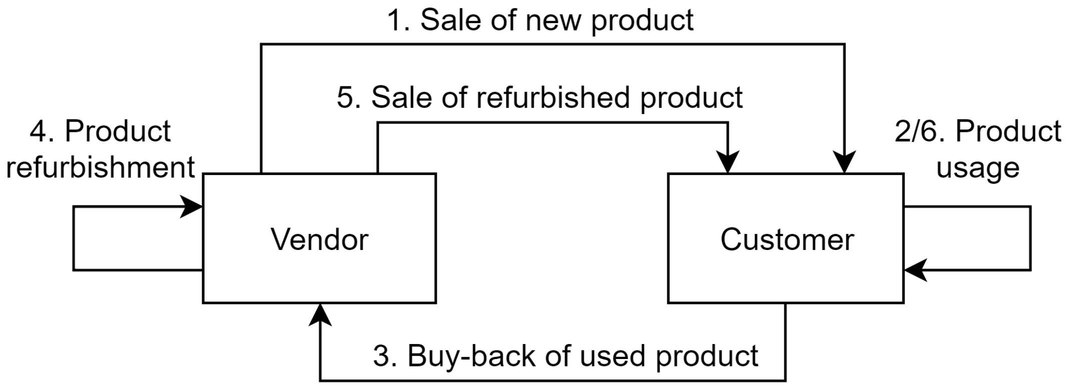


Figure 1.1: The product lifecycle in a Circular Economy. In a Linear Economy, the product lifecycle ends with step No. 2

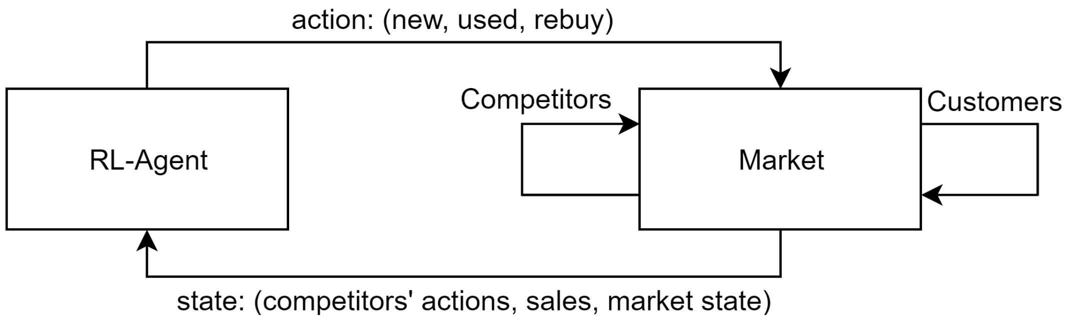

action: (new, used, rebuy)
RL-Agent
Competitors
Market
Customers
state: (competitors' actions, sales, market state)

Figure 1.2: The standard reinforcement-learning model in the context of our market.

In the context of e-commerce, Circular Economy markets are also referred to as *recommerce* markets. From now on, when mentioning a general *market* or *marketplace*, we are referencing a Circular Economy marketplace with re-buy prices.

1.3 Reinforcement-Learning

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 Game theory models 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 1.2 illustrates the RL-model in the context of our market. The goal of the agent is to maximize its *reinforcement signal*, which in the case of our simulation framework is the profit the agent made during the last episode, since we want to train agents to maximize profits on real markets. An episode consists of a fixed, but pre-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 approaches to modelling and simulating recommerce marketplaces, as well as give an introduction into the concepts of Reinforcement-Learning, the technology the framework is built for. Additionally, novel and unconventional approaches to monitoring and evaluation of these Reinforcement-Learning agents will be presented.

Dynamic Pricing simulations

The topic of dynamic pricing techniques is well explored, with the earliest mathematical models having been developed over a century ago [den15]. With the emergence of e-commerce and the ability to cheaply and quickly change prices, as well as the freedom of information, including being able to know competitor's prices in real time, the topic has gained importance even further. The role that autonomous agents will play in this new market environment, both in the form of vendors and customers, has also long been a topic of discussion and speculation alike [KHG00]. Dynamic pricing methods come in many different shapes and forms, from simple greedy algorithms over customer-choice methodology to machine-learning models [Gee+19]. Due to this high number of options to choose from, and to enable researchers to evaluate their algorithms' performance, a unified platform for comparison is needed. The real market is not really an option for this, as it is near impossible to create equal opportunities for each pricing method to be able to reliably compare them to each other. This is one of the major reasons for simulating the real market - development, testing and comparison of new pricing methods.

Nowadays, another market dimension is creating new challenges for businesses: Circular Economy models (see 1.2 The Circular Economy model) which include two or more prices per product make it hard to price items manually using pricing experts or even regular dynamic pricing models, which may not be able to correctly adapt to interdependencies between prices. This is creating a demand for dynamic pricing methods using self-learning technologies, such as Reinforcement-Learning. And again, to train, test and evaluate such pricing agents, simulated environments are needed. In the case of Reinforcement-Learning, this is not only optional but required, as Reinforcement-Learning agents need to learn how a market environment works and how to interact with it before they become viable for use on the real market, as the pricing decision they take early on in the training process lack any experience and are therefore taken randomly, until enough knowledge about the market was collected.

Judith: 'Diesen letzten Absatz verstehe ich nicht so ganz. Hast Du das mit der CE nicht in dem 1.2 Genauso erklärt? Dann könntest Du hier den ersten Satz des Absatzes deutlich kürzen, bzw. Den kannst Du sowiese nochmal Teilen. Warum motivierst Du hier RL? Würde das nicht besser in das RL Kapitel passen? '

Visualization - State-of-the-art

The process of training Reinforcement-Learning agents for any kind of task always comes with the requirement for visualizing the data collected during training. This allows for an analysis of an algorithm's performance and gives insights into its strengths and weaknesses.

For the past years, going back as far as 2018, one of the most used frameworks overall and the most used for machine learning was TensorFlow [Ove21]. Aside from its API for model building, TensorFlow also provides a front-end visualization toolkit called TensorBoard, which can be used independent of other TensorFlow tools. TensorBoard provides an API for tracking and visualizing important metrics such as loss and accuracy, and allows developers to easily integrate their own metrics as well. During an experiment, data can be visualized live during the training process, allowing developers to quickly gain insights into the performance of the algorithms. In our recommerce market simulation framework, we use the TensorBoard in conjunction with our own tools for data visualization, see Approaches to monitoring agents.

Do I refer to our own project in the related work chapter?

What-If-Tool (part of Tensorboard)

Visualization - Novel approaches

Model visualization

Aside from visualizing results of a training run, developers may also want to visualize the model itself. Tools such as the *Graph Visualizer* [Won+18] are a step in the right direction, but the study found that while developers are satisfied with the visualization tool itself, they would prefer to be able to edit and influence the graph model directly. The *SimTF* tool developed by Leung et al. [CLH18] attempts to solve this problem. The same as the *Graph Visualizer*, the *SimTF* tool is based on TensorFlow. The authors describe it as a *library for neural network model building*. *SimTF* acts as a middleman between the visually constructed neural network graph model and the TensorFlow API, allowing developers to modify the visualized model to influence the training network.

In our simulation framework, the neural network model itself is quite static, users can only choose from a pre-set selection of network sizes, and all changes to hyperparameters must be done through configuration files ahead of a training run.

Evaluation

On the other side of the visualization tools we have those which provide insights into currently running or completed training runs, such as the previously mentioned *TensorBoard*. Besides simply visualizing different collected data, the goal of most of these tools is to allow the developers to get a sense of the trained algorithm's performance, and to evaluate and compare it against previously trained agents using the same algorithm or other algorithm's completely.

When developing new algorithms, the most common barrier to proper comparison with current state-of-the-art algorithms is a lack in reproducibility of results. Benchmark environments such as those provided by OpenAI Gym [Bro+16] lower this barrier by providing unified interfaces against which many algorithms have already been tested. However, the effects of extrinsic factors (such as hyperparameters) and intrinsic factors (such as random seeds for environment initialization) make proper, reliable reproducability a challenge, something for which current evaluation practices do not account for [Hen+17]. Jordan et al. [Jor+20] propose a new, *comprehensive evaluation methodology for reinforcement learning algorithms* *that produces reliable measurements of performance.* The authors describe the goal of their new method as not trying to find methods (algorithms) that maximize performance with optimal hyperparameters, but rather to find those that do not require extensive hyperparameter tuning and can therefore easily be applied to new problems. This leads to a preference for algorithms that are not aimed at being the best at problems which are already solved (which applies to the aforementioned benchmark environments), but instead those which are most likely to succeed in new, unexplored environments.

Kurve zum monitoring, genauer erklären, wie das mit den hyperparametern ist

Judith: 'Generell hat mir in dem Kapitel so ein bisschen der Vergleich zu unserem Framework gefehlt, Ist vllt auch schwierig, wenn Du keine Paper hast (ich kenne das Dilemma) Für Tensorboard ist das gut, vllt findest Du noch ein Paper, dann kannst Du schreiben, wie Mustermensch et. Al. Aber Du kannst glaube ich auch so Dinge schreiben, wie die machen das, das machen wir eher nicht so, sondern so und so anders '

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 and put to use in Analyzing Graphs.

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.

Overview diagram of the whole framework

In the most common usecase of our framework, the training of Reinforcement-Learning agents, only the agent (or multiple agents in the case of training multiple Reinforcement-Learning agents against each other) that is to be trained needs to be configured by the user, as each market environment is equipped with a predefined 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 change these competitors 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.

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.

Most customers' behaviour can be classified into one of a (non-exhaustive) number of categories, such as *Perfectionistic* or *Impsulsive* consumers, proposed in [EIT13] (see Table 6.1). As we are focussing on dynamic pricing and our vendors can only change/influence the prices of their products, we decided on primarily building customers that value price over any other features a product may have, thereby incentivizing our vendors to make the most of their pricing power. This behaviour coincides with the shopping style of the *Price Conscious* or *'Value for money'* consumer.

I could remove this paragraph, since we focus on Circular Economies

To make Linear markets a little more complex and add another dimension than just the *new price* of a product for vendors to consider, random quality values are assigned to each vendor's products. Customers in this economy model were modelled take this quality value into account when making their purchasing decisions, further reinforcing the shopping style mentioned above. As Circular Economy markets are inherently more complex than their linear counterparts (through the addition of two new price channels, which influence each other through their effect on customers), it was decided to remove the additional layer of quality values from these markets.

Within each simulated step, after the various vendors have set their prices, purchase decisions are made by the customers. To save on computational time, probability distributions are used to determine what part of the total number of customers will decide to take which action. See below the way these distributions are calculated for an exemplary Circular Economy marketplace:

Cite Nick?

▶ **Definition 3.1.** Let P_{in} be the price of the new and P_{ir} the price of the refurbished product of vendor i. Using these prices, for each vendor, we define R_{in} as the *preference ratio* regarding P_{in}:

$$R_{in} := \frac{10}{P_{in}} - e^{P_{in}-8}$$

and similarly, R_{ir} as the *preference ratio* regarding P_{ir}:

$$R_{ir} := \frac{5.5}{P_{ir}} - e^{P_{ir}-5}$$

◀

Graph for the preference ratios in Appendix?

In order to be usable as a probability distribution, the different ratios need to add up to 1. We use *softmax* to normalize our values, after which each *preference ratio* will be within the interval $(0, 1)$, with all ratios adding up to 1 as needed.

Following this, the market uses a *multinomial* distribution to draw n samples, with n being the number of customers making a pruchasing decision in this step of the simulation.

As mentioned above, the current decisionmaking process of customers in our simulation is still quite basic. Diverging from the real market introduces a number of parameters and circumstances that can be used to make customer behaviour more realistic. Through the modular approach when building the framework, any customer behaviour implemented in the future can easily be added to the pool of available options, as long as it manifests in the form of a probability distribution, at which point the number of customers can be split between different distributions when drawing from the *multinomial* distribution (or any other distribution if so desired).

Owners

Once a customer has 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 product in the next step.

Not exactly, but an owner is created

In our simulation, all owners are memoryless, meaning that they do not remember the original price they paid for the product. Additionally, each vendor in the market is obligated to buy back any product, independent of the original vendor. In each step, every owner has the option to either keep the product for another step, discard it (meaning it is removed from circulation and not sold back to a vendor), or sell it to any one of the vendors in the market. Similarly to the way we compute customer purchasing decisions, the decisions owners take are also represented through probability distributions. When deciding if and to which vendor a product should be sold, the following formula is used:

▶ **Definition 3.2.** We keep the definitions from Definition 3.1. Additionally, let P_{ib} be the price at which vendor i is willing to buy back an owner's product. For each vendor, we define P_{im} as the purchase option with the lowest price:

$$P_{im} := \min(P_{in}, P_{ir})$$

We define the likelihood R_{ib} that the owner will return their product to vendor i as follows:

$$R_{ib} := 2 \cdot e^{(P_{ib} - P_{im})/P_{im}}$$

Additionally, the owner's *discard preference* R_d is updated for each vendor as follows:

$$R_d := \min(R_d, \frac{2}{P_{ib} + 1})$$

Meaning the owner is more likely to discard their product if re-buy prices are low across the board. ◀

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). Due to the modular nature of the framework, it is possible for users to easily create and add their own pricing strategies in the form of a vendor that can play on a market. This allows users to not only add other Reinforcement-Learning algorithms, but also to define new Rule-Based strategies. For this reason, we also do not restrict the usage of our monitoring tools to just Reinforcement-Learning agents, but allow users to monitor any combination of vendors.

Do I need to motivate *why* we allow this more?

We will use 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.

Inventory-based models

These are pricing models which are based on inventory levels and similar parameters, such as the number of items in circulation (items which are currently in use by customers). 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. The prices these agents will set are pre-determined through the user's configuration of the experiment, and will not change over the course of the market simulation.

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 which fall into this category are relatively simple to create and modify. Examples of *Inventory-based agents* in our framework can be found in the *RuleBasedCEAgent*, one of 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, always trying to keep a balance between having enough (refurbished) products to sell back to customers and keeping storage costs low. While its performance is not necessarily bad, it is still one of the weakest competitors currently available in the framework. For the full implementation of the agent's policy, see Listing 6.1.

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

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 with the median price of the competitors, and then adjusts them according to current inventory. Again, the full implementation of this vendor's policy is available in Listing 6.2 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 to prove it

Game theory models

If any section needs to be removed here (e.g. for page length), this should be the one

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, most competitors in the market are influenced in their decisionmaking processes by the actions of other participants of the scenario. This is especially true for the Reinforcement-Learning agents, which base their policy on the received market states, which include their competitor's actions. 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.

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

Machine learning models

All of our Reinforcement-Learning agents fall into this category. As they are not the focus of this thesis, we will not go into detailed explanations of the various models used. The following will give a short overview over the different algorithms used in our framework.

There are two 'origins' of the algorithms in our framework. In the earlier phases, Q-Learning and Actor-Critic algorithms were custom implemented by us. Later on, we used the Stable-Baselines library to incorporate a greater number of pre-implemented algorithms into our framework. While these are not as easily configurable, they can quickly be used without much additional work.

- *Q-Learning*: *Q-Learning* agents were the first Reinforcement Learning agents we introduced in our framework, as its algorithm is one of the easier ones to implement [KLM96]. The *Q-Learning* algorithm used in our framework is implemented using the *pytorch*[1] framework. However, the drawback of using *Q-Learning* in our framework is that it can only be applied to discrete action and state spaces. This means that when using *Q-Learning*, only 'whole' prices can be set by our vendors, 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 critical in influencing customer decisions. 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. Even now, 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. While approaches for *Q-Learning* algorithms that can work with continuous action and state spaces have been presented in the past ([GWZ99], [MPD02]),

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

we have chosen not to implement such an algorithm in our framework, but rather explore approaches other than *Q-Learning*, such as *Actor-Critic* algorithms, introduced below.

Section is quite superficial, should I make it more specific?

- *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 [Bas22a]:
  - *A2C*: Advantage-Actor-Critic
  - *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

Add something about market circumstances, customers, and close with modularity argument. Chapter needs more structure

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 effect on the market state or customer decisions. Seasonality of demand, customer retention, 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, following its inability to model these circumstances. 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).

The factors mentioned above all have an impact on both Linear and Circular markets, but there are also a number of parameters that are specific to Circular or even recommerce markets in particular. In the modern recommerce market, more and more consumers make their initial purchasing decisions with the product's eventual resale value already in mind [TP19]. Additionally, a great number of different motivations for choosing second-hand or refurbished products over traditional new ones can be identified. An exemplary study conducted in 2008 ([RG08]) classified such motivations into 15 different categories (see Table 6.2), all of which could be used to add dimensions to customer behaviour in our simulation framework, thereby making the whole simulation more realistic.

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.

Diagram for modularity here/reference here

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 an 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 completed a trining run, 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.)

   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

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

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 these sections 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.

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

4.2 Monitoring during a training session

When talking about monitoring agents during a training session, we are always talking about Reinforcement-Learning agents, since Rule-Based agents always perform the same and cannot be trained. But even though they cannot be trained, our monitoring tools can still be applied to Rule-Based agents as well, as users may want to compare different Rule-Based strategies against each other, or measure the strategy's performance on a market before training a Reinforcement-Learning agent.

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 Analyzing Graphs.

TensorBoard

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.

Citation or footnote?

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

List some/all of the data provided through tensorboard. Maybe some here and the rest in appendix table

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 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, which can change over the course of a training run. 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?

Small graphic which also shows the agent getting worse after more episodes

[2] https://www.tensorflow.org/

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

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. 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 markets: 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 chosen marketplace, meaning that the agents will be monitored independently from each other. In this case, each agent will play against a 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 reached by simply starting multiple monitoring sessions with a different agent each, this is not the case. Using this flag instead, 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, 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 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.

   Judith: 'Idea for future: Allow custom market state initialization'

   Judith: 'For whole section: Highlight some keywords to

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 allowed 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 preset 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.

Judith: 'Absatz ist etwas unstrukturiert'

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* (for a more detailed overview see Vendors). Reinforcement Learning agents can only be monitored on the 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 rulebased 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.

Refer to an explanation in the workflow chapter

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. For an even more detailed view, scatterplots containing the exact values for each episode are also created for the same metrics. 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.

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 agent's capacity to adapt to different circumstances, 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.

Judith: 'Oh gott, jetzt auch noch der exampleprinter, wozu so viele verschiedene Methoden? Vllt ist es nicht schlecht zu Beginn des Kapitels einmal grob die drei verschiedenen Aufgaben der Tools zu umreißen?'

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

Judith: 'Warum hat der Exampleprinter gegenüber agent monitoring seine Daseinsberechtigung? Same for tensorboard'

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

Judith: 'Und wie funktioniert das jetzt? Welche Daten nutzt der?'

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 market simulation framework is to provide a simple-to-use but powerful tool 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 introduce the most important parts of the workflow.

Judith: 'Maybe change chapter order of this with Chapter 4?'

When working with our simulation framework, users can choose from two options: First, it is possible to use our tool via a custom Command line interface (CLI). Alternatively, users can interact with the framework through a web-interface, which utilizes Docker for remote-deployment of tasks issued by the user. For detailed insights into our web-interface and remote deployment processes, see [Her22].

Figure 5.1 depicts the common workflow activities in our framework. For all possible tasks, the user needs to provide configuration files, which define both the task to be worked on as well as parameters which influence market and agent behaviour (Configuring the run). After the configuration files have been validated, the simulation framework initialized the requested marketplace and agents, and then starts the requested task, for which there are currently three options provided through the CLI or web-interface:

- Training
- (Agent-)Monitoring
- Exampleprinter

At the end of the respective task, the user is always provided with the various diagrams and data collected during the respective task, which can then be used to start subsequent tasks.

5.1 Configuring the run

Configuration is one of the most important aspects of the workflow. 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 behaviour and thereby 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.

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

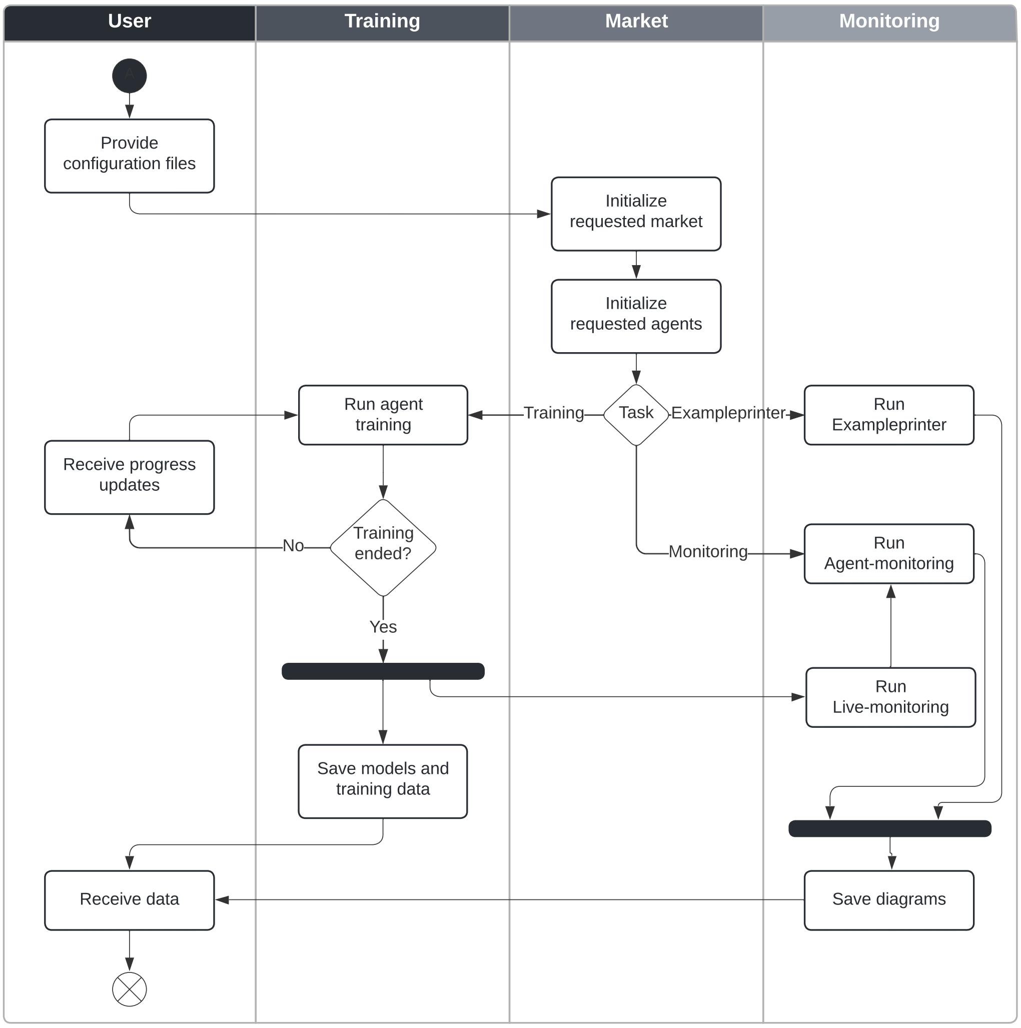


Figure 5.1: Swimlane diagram depicting possible workflows, possible webserver interaction omitted

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

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

Judith: 'Das ganze wie nutze ich die Konfiguration auf dem Webserver könntest Du meiner Meinung nach kürzer fassen'

Example screenshot of the webserver. Configuration and running page?

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

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

5.2 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 automatically used to record metrics and give insights into the current training run, by creating visualizations 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, or even worse, starts to deteriorate in performance, 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.

Citation for the 'possibility' of this

Judith: 'Auch wenn ich das training abbreche?' -> Mention this in chapter 6, run a long experiment and stop it prematurely. Live-Monitoring is not run, but models are still saved

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.

This sentence is a bit harsh. Make it less 'Crucial'

6 Analyzing Graphs

In this chapter, we will put the tools and workflows described in the previous sections to the use. We will train a Reinforcement-Learning Agent, and then monitor it using all of the tools at our disposal, highlighting where each tool is most useful, and what could be improved.

I do have an additional, fully trained experiment on an Oligopoly, but I feel like using both would be very confusing, as there are a lot of diagrams already

Setting up the experiment

Before starting our monitoring, we will need to conduct an experiment, where a Reinforcement-Learning algorithm is being trained on a market environment. For our experiment, we will train a Reinforcement-Learning Agent using the SAC-algorithm [Bas22b] on a Duopoly marketplace with rebuy prices. The agent will be trained playing against a Rule-Based agent, more specifically the *RuleBasedCERebuyAgentStorageMinimizer*, as presented in Data-driven models. The configuration files for this experiment can be found in Figure 6.1 and Figure 6.2. We will refer to this experiment as the *SAC-Duopoly*. To ensure that we are evaluating an agent with a performance that is to be expected with the provided parameters, we will conduct the experiment multiple times to be able to identify outliers in the data. All diagrams except those shown in Figure 6.1 (which contains diagrams from each of the four experiment runs) have been taken from the same experiment run, denoted as SAC-Duopoly_1 in Figure 6.1 (a).

Experiment results

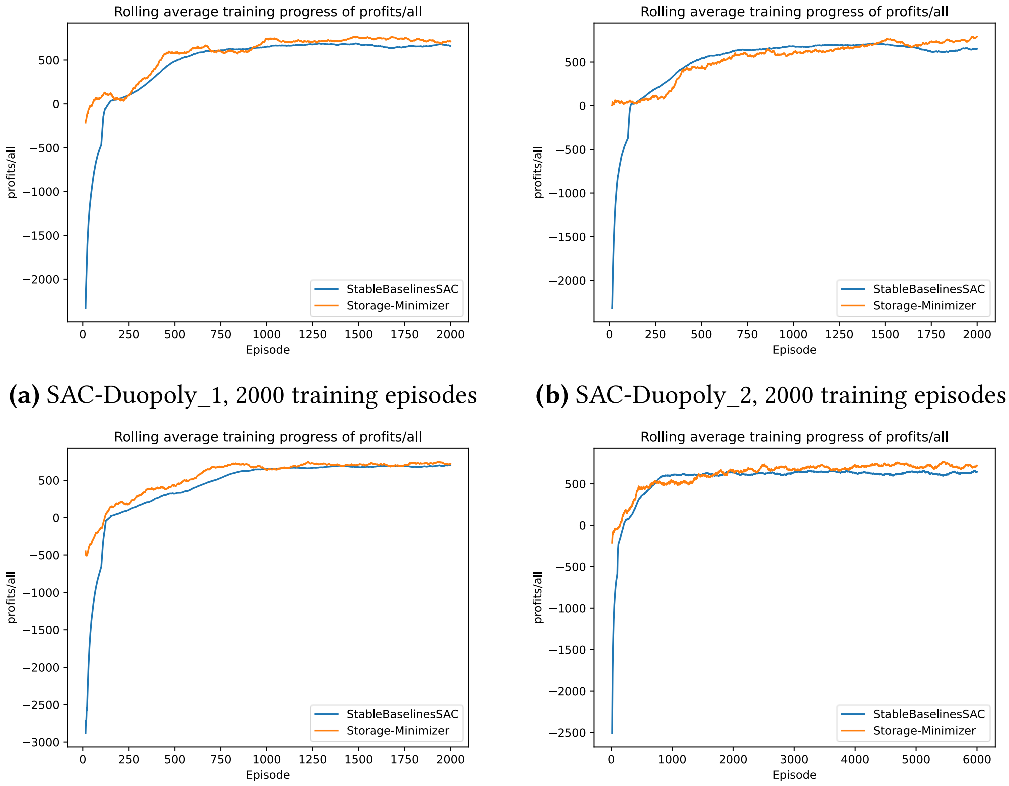
In the following sections we will use our different monitoring tools on the results of the SAC-Duopoly experiment. We will start with the Live monitoring tool, which runs automatically after the training run has compelted and creates over 90 graphs and diagrams already. Due to this high number of available diagrams, we will always only look at a curated selection, highlighting those which give the best insights into the trained agent. We have also dedicated a section to diagrams which are being created, but not as useful to the user, where we will try to identify the reason for the lack of information that can be gained from these diagrams (Useless diagrams)

Live- and Agent-monitoring

This section will focus on the diagrams created by the Live-monitoring tool after training, which always runs an Agent-monitoring session as well, to immediately provide the user with many additional useful diagrams without the need to run the tool manually.

A commonly asked question when deciding on the quality of a Reinforcement-Learning Agent is their *stability*. If an algorithm is stable, the trained agent will

cite?



(a) SAC-Duopoly_1, 2000 training episodes **(b)** SAC-Duopoly_2, 2000 training episodes

(c) SAC-Duopoly_3, 2000 training episodes **(d)** SAC-Duopoly_4, 6000 training episodes

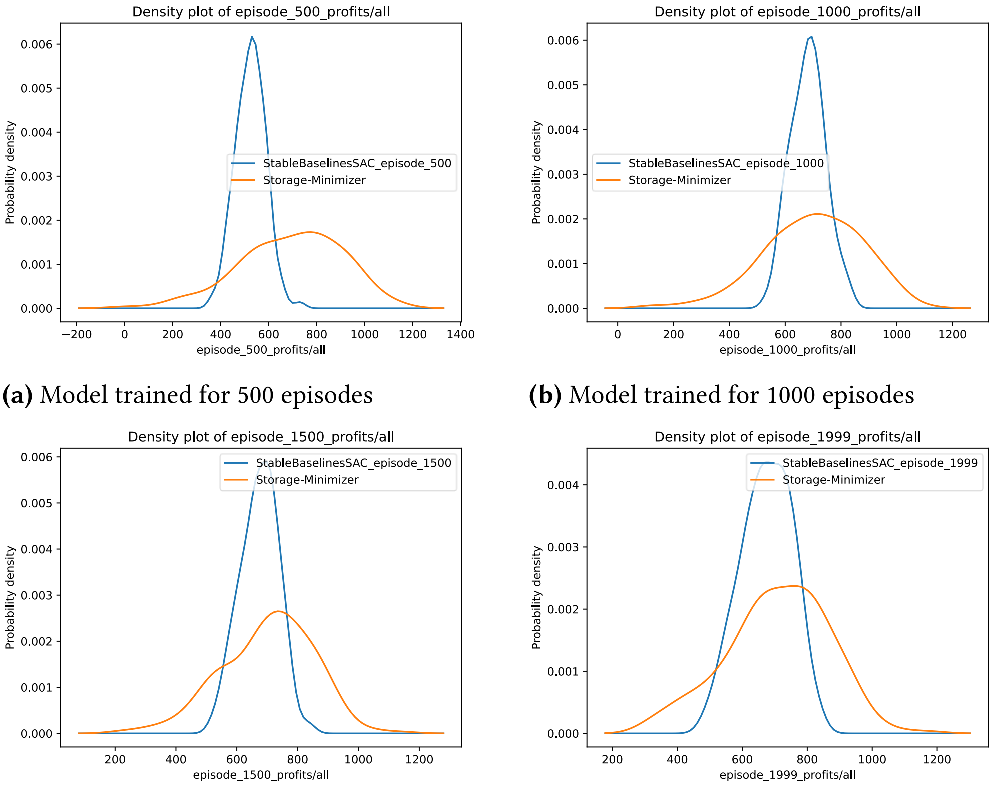
Figure 6.1: Profit per episode of four different training runs of an SAC-Agent on a Duopoly market.

produce similar results over multiple training sessions, on the condition that the parameters do not differ. Not only the rewards achieved at the end of the training will be very similar, but also the amount of episodes needed to reach certain thresholds. In the case of the SAC-Duopoly experiment, we ran the same configuration four times: The first three experiments were run using the exact same parameters, for the fourth experiment the amount of training episodes were tripled, meaning that the SAC-Agent had more time to alter its policy. Figure 6.1 shows the results of these four training sessions, created using the Live-monitoring tool and visualizes the data collected during the training process. Although many other graphs are created, the visualization of the total profit of the agent is the most convenient to use when evaluating an agent's performance, as the total profit is the parameter which the agent is trying to optimize.

Table/List of all graphs needed

Figure 6.1 shows the stability of the SAC-Agent very well. The agent not only reached the break-even threshold of a reward of 0 after around 150 episodes in each of the four training runs, but no matter the total length of training (see the model in Figure 6.2 (d), which was trained three times as long as the others), the profit would always stabilize and stay at around 670. Had the monitoring tools shown that the agent performs worse than expected in some of the experiments, we could conclude that this particular algorithm is not fit for the type of work required by our market simulation.

We can also observe that the profits of the SAC-Agent and the Rule-Based Agent (in the case of this experiment, a *RuleBasedCERebuyAgentStorageMinimizer*) seem to be closely linked to each other in this particular scenario. In the beginning of



(a) Model trained for 500 episodes

(b) Model trained for 1000 episodes

(c) Model trained for 1500 episodes

(d) Model trained for 1999 episodes

Figure 6.2: Probability densities for achieving a certain profit for four different training stages of the SAC-Duopoly experiment.

each experiment, when the Reinforcement-Learning agent still knows very little about the market and makes great losses, the Rule-Based agent also has a hard time to perform well. However, as soon as the SAC-Agent starts to perform better, the Rule-Based agent is also able to increase its mean profits at around the same rate as the SAC-Agent. Even more interestingly, the agents not only increase their profits at the same rate, but also very quickly arrive at a point where they earn the same mean amount as the other.

This might lead one to believe that the two competitors' policies align closely with each other. To validate this claim, another type of diagram created by the Live monitoring tool can be used: The density plots. These diagrams visualize probability densities for the various datasets recorded. While the diagrams shown in Figure 6.1 simply visualize data that was collected during the training run, density plots are created by running the Agent-monitoring tool, where the marketplace is simulated an additional time. This allows us to use the 'intermediate' models we saved during the training run (see Live-monitoring) to compare the Reinforcement-Learning Agent's policies at different points in time during the training.

This is explanatory, perhaps move it to the approaches chapter

Figure 6.2 shows the density plots of profit-per-episode for four different training stages of the SAC-Duopoly experiment. From this it can be concluded that the claim that the two competitors' policies align closely is incorrect: even though the mean reward within an episode is always very similar (Figure 6.1 (a)), the SAC-Agent achieves rewards which lie closer together, while the Rule-Based agent's rewards have more of a spread.

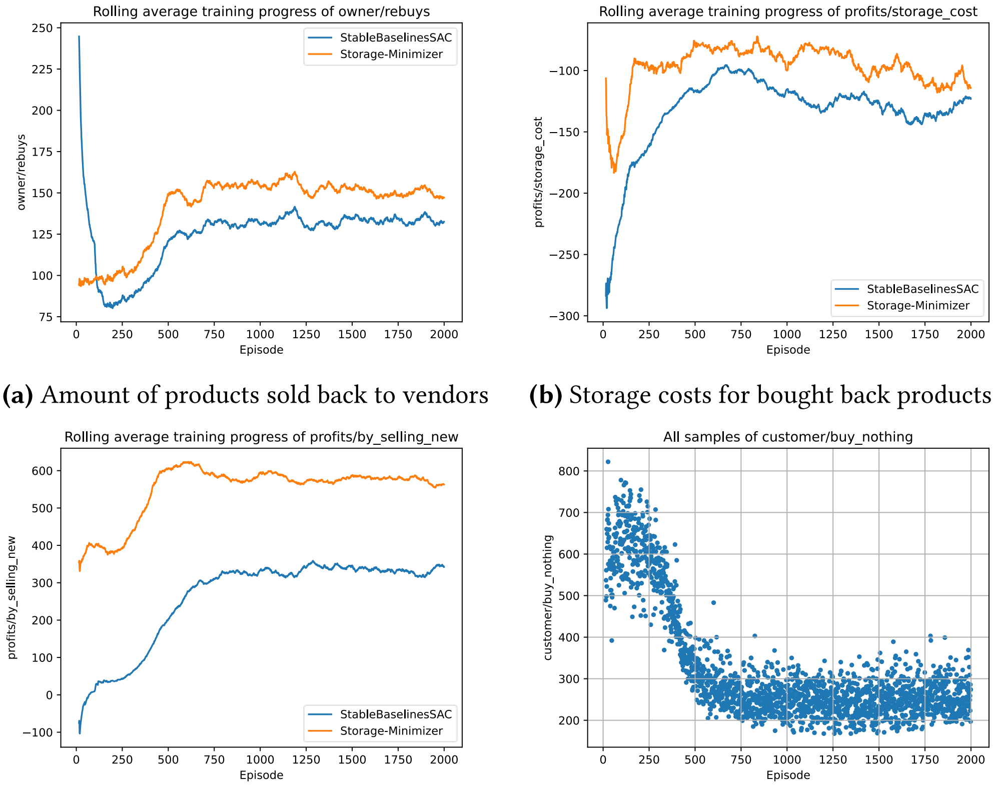
We can also observe that the models which were trained for longer are not necessarily better or even the same quality of the models with less experience. There is a significant improvement going from the model shown in Figure 6.2 (a) to the one in Figure 6.2 (b), this shift in the probability density curve can be correlated with the maximum mean rewards the two models could achieve during the training: For the model trained on 500 episodes, this was around 480, for the other it was about 650 (Figure 6.1 (a)). Both of these values have the respective highest probability of being reached during the second simulation after the training has concluded, visualized as maxima in the density plots. Going from the model trained on 1000 episodes to the next one, which was trained on 1500 episodes (Figure 6.2 (c)), both the probability densities as well as the mean rewards stay very close to each other. From this we can conclude that training the SAC-Agent for more than 1000 episodes is very likely to not have an effect on the maximum reward achievable by the algorithm. Going from the model trained on 1500 episodes to the last one, saved at the end of the training Figure 6.2 (d), we can however see a deterioration in performance: While the mean rewards hardly changed (Figure 6.1 (a)), the probability density curve got wider at its base, extending out further towards a reward of 400, and lowering the maximum at a reward of about 700 from the previous 0.6% to only 0.4%. This means that the model which was trained for the longest time produces less predictable results than those trained less, which may at first seem counterintutitive, but is a common phenomenom when training Reinforcement-Learning agents, known as *Catastrophic Forgetting* (see also [Cah11]). The insights gained by our monitoring tools combined with the fact that we save 'intermediate' models of the Reinforcement-Learning agent during training allows us to find the optimal trained model to use for further investigation and eventual deployment on the real market. In the case of the SAC-Duopoly experiment, the optimal model would be the one trained for 1000 episodes.

Further investigation

Evaluating a model based on just the mean profits achieved by the monitored agents may not be enough for many users, which is why our Live-monitoring offers many other useful diagrams as well. In this section, we will take a look at a small selection of them.

Users may ask themselves how the profits achieved by the different vendors are split between the two available retail channels of new and refurbished products, how many products were bought back from customers or how much the vendors had to pay for storage of these used products. For all of these questions, the Live-monitoring (together with the Agent-monitoring) tool offers two types of diagrams: First, the simple lineplots shown in Figure 6.1 can be used, as well as scatterplots which visualize the exact data recorded during the episode, instead of the trends shown by the lineplots. Figure 6.3 shows a number of different metrics such as the ones mentioned above, all of which are available as both linepots and scatterplots.

Many connections can be made when evaluating different diagrams side-by-side, such as the rather obvious observation of the initially high storage costs of the SAC-Agent in Figure 6.3 (b) being caused by a very high number of products that were bought back from customers (Figure 6.3 (a)), which is in turn a cause of high rebuy prices set by the Reinforcement-Learning Agent, making it more likely that customers sell back their products. High storage and rebuy costs will have then caused a policy change to set lower rebuy prices, thereby de-incentivizing



(a) Amount of products sold back to vendors **(b)** Storage costs for bought back products

(c) Profit from selling new products **(d)** Number of customers that bought nothing

Figure 6.3: Diagrams visualizing various datapoints collected during training of the SAC-Duopoly experiment.

customers to sell back as many products as they used to and lowering the agent's storage costs.

The next pair of diagrams tells us two things: First, the vendors' initially low profits from selling new products as shown in Figure 6.3 (c), or in the bigger picture, low profits overall (Figure 6.1 (a)) were not caused by the vendors setting prices that are too low, but rather too high. This is confirmed by an initially very high number of customers which chose to not buy any of the products offered by the vendors (Figure 6.3 (d)). Additionally, when comparing the total profits of the two vendors with the profits gained from selling only new products (Figure 6.1 (a) and Figure 6.3 (c)), we can come to the conclusion that the SAC-Agent must have learned to prioritize selling refurbished products over new ones and keeping storage costs low, since we know that overall profits for the two vendors are about the same in the later parts of the training, but the SAC-Agent sells a lot less new products than its Rule-Based counterpart.

Useless diagrams

Which graphs are how useful?

What limitations are there?

What can we do in the future to improve the graphs or the workflow?

Can we see which hyperparameters influence the results in what ways using the diagrams?

How can we improve the workflow (e.g. Grid-Search) with our analysis?

Bibliography

[Bas22a] Stable Baselines3. *RL Algorithms.* Accessed: 2022-06-06. 2022. URL: https://stable-baselines3.readthedocs.io/en/master/guide/algos.html (see page 14).

[Bas22b] Stable Baselines3. *SAC.* Accessed: 2022-06-17. 2022. URL: https://stable-baselines3.readthedocs.io/en/master/modules/sac.html (see page 27).

[Bro+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang and Wojciech Zaremba. **OpenAI Gym** (2016). DOI: 10.48550/ARXIV.1606.01540. URL: https://arxiv.org/abs/1606.01540 (see page 6).

[Cah11] Andy Cahill. **Catastrophic forgetting in Reinforcement-Learning Environments**. MA thesis. University of Otago, 2011. URL: http://hdl.handle.net/10523/1765 (see page 30).

[CLH18] Jui-Hung Chang, Yin Chung Leung and Ren-Hung Hwang. **A Survey and Implementation on Neural Network Visualization**. In: *2018 15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN).* 2018, 107–112. DOI: 10.1109/I-SPAN.2018.00026 (see page 6).

[den15] Arnoud V. den Boer. **Dynamic pricing and learning: Historical origins, current research, and new directions**. *Surveys in Operations Research and Management Science* 20:1 (2015), 1–18. ISSN: 1876-7354. DOI: https://doi.org/10.1016/j.sorms.2015.03.001. URL: https://www.sciencedirect.com/science/article/pii/S1876735415000021 (see page 5).

[EIT13] Jacqueline K Eastman, Rajesh Iyer and Stephanie P Thomas. **The impact of status consumption on shopping styles: An exploratory look at the millennial generation**. *Marketing Management Journal* 23:1 (2013), 57–73 (see pages 10, 37).

[Gee+19] Ruben van de Geer, Arnoud V. den Boer, Christopher Bayliss, Christine S. M. Currie, Andria Ellina, Malte Esders, Alwin Haensel, Xiao Lei, Kyle D. S. Maclean, Antonio Martinez-Sykora, Asbjørn Nilsen Riseth, Fredrik Ødegaard and Simos Zachariades. **Dynamic pricing and learning with competition: insights from the dynamic pricing challenge at the 2017 INFORMS RM & pricing conference**. *Journal of Revenue and Pricing Management* 18:3 (June 2019), 185–203. ISSN: 1477-657X. DOI: 10.1057/s41272-018-00164-4. URL: https://doi.org/10.1057/s41272-018-00164-4 (see page 5).

[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 13).

[Hen+17] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup and David Meger. *Deep Reinforcement Learning that Matters.* 2017. DOI: 10.48550/ARXIV.1709.06560. URL: https://arxiv.org/abs/1709.06560 (see page 6).

[Her22] Judith Herrmann. *Skalierbares Lernen in der Cloud.* 2022 (see page 23).

[Jor+20] Scott Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang and Philip Thomas. **Evaluating the Performance of Reinforcement Learning Algorithms**. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, 4962–4973. URL: https://proceedings.mlr.press/v119/jordan20a.html (see page 6).

[KHG00] Jeffrey O. Kephart, James E. Hanson and Amy R. Greenwald. **Dynamic pricing by software agents**. *Computer Networks* 32:6 (2000), 731–752. ISSN: 1389-1286. DOI: https://doi.org/10.1016/S1389-1286(00)00026-8. URL: https://www.sciencedirect.com/science/article/pii/S1389128600000268 (see page 5).

[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 13).

[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 1).

[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 13).

[Mye97] Roger B Myerson. **Game theory: analysis of conflict**. Harvard university press, 1997, 1 (see page 13).

[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 12).

[Ove21] Stack Overflow. *Stack Overflow Annual Developer Survey*. Accessed: 2022-06-12. 2021. URL: https://insights.stackoverflow.com/survey (see page 6).

[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 14).

[RG08] Dominique Roux and Denis Guiot. **Measuring Second-Hand Shopping Motives, Antecedents and Consequences**. *Recherche et Applications en Marketing (English Edition)* 23:4 (2008), 63–91. DOI: 10.1177/205157070802300404. eprint: https://doi.org/10.1177/205157070802300404. URL: https://doi.org/10.1177/205157070802300404 (see pages 15, 38).

[SB18] Richard S Sutton and Andrew G Barto. **Reinforcement learning: An introduction**. MIT press, 2018 (see page 14).

[SK86] George B. Sprotles and Elizabeth L. Kendall. **A Methodology for Profiling Consumers' Decision-Making Styles**. *Journal of Consumer Affairs* 20:2 (1986), 267–279. DOI: https://doi.org/10.1111/j.1745-6606.1986.tb00382.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1745-6606.1986.tb00382.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1745-6606.1986.tb00382.x (see page 37).

[TP19] Linda Lisa Maria Turunen and Essi Pöyry. **Shopping with the resale value in mind: A study on second-hand luxury consumers**. *International Journal of Consumer Studies* 43:6 (2019), 549–556. DOI: https://doi.org/10.1111/ijcs.12539. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ijcs.12539. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ijcs.12539 (see page 15).

[Won+18] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mané, Doug Fritz, Dilip Krishnan, Fernanda B. Viégas and Martin Wattenberg. **Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow**. *IEEE Transactions on Visualization and Computer Graphics* 24:1 (2018), 1–12. DOI: 10.1109/TVCG.2017.2744878 (see page 6).

Appendix

Consumer Characteristic	Description
Perfectionistic, High-Quality Conscious	Consumer searches carefully and systematically for the very best quality in products
Brand Conscious, ‘Price = Quality’	Consumer is oriented towards buying the more expensive, well-known brands
Novelty and Fashion Conscious	Consumers who like new and innovative products and gain excitement from seeking out new things
Recreational and Shopping Conscious	Consumer finds shopping a pleasant activity and enjoys shopping just for the fun of it
Price Conscious/ Value for the Money	Consumer with a particularly high consciousness of sale prices and lower prices in general
Impulsive/ Careless	Consumer who buys on the spur of the moment and appears unconcerned about how much he/she spends
Confused by Overchoice	Consumer perceiving too many brands and stores from which to choose and experiences information overload in the market
Habitual/ Brand Loyal	Consumer who repetitively chooses the same favorite brands and stores

Table 6.1: Consumer Shopping Styles, taken from [EIT13], including information from [SK86]

I - Economic dimensions
1. ECO1 - Buying cheaper, spending less (anxiety expressed in regard to expenditure) 2. ECO2 - Paying fair prices 3. ECO3 - Allocative role of price (what is obtained for a particular budget) 4. ECO4 - Bargain hunting
II - Dimensions relating to the nature of the offering
5. OFF1 - Originality 6. OFF2 - Nostalgia 7. OFF3 - Congruence 8. OFF4 - Self-expression
III - Dimensions relating to the recreational aspects of second-hand channels
9. CIR1 - Social contact 10. CIR2 - Stimulation 11. CIR3 - Treasure hunting
IV - Power dimensions
12. PUIS1 - Smart shopping 13. PUIS2 - Power over the seller
V - 14. ETH - Ethical and ecological dimension
VI - 15 ANT-OST - Anti-ostentation dimension

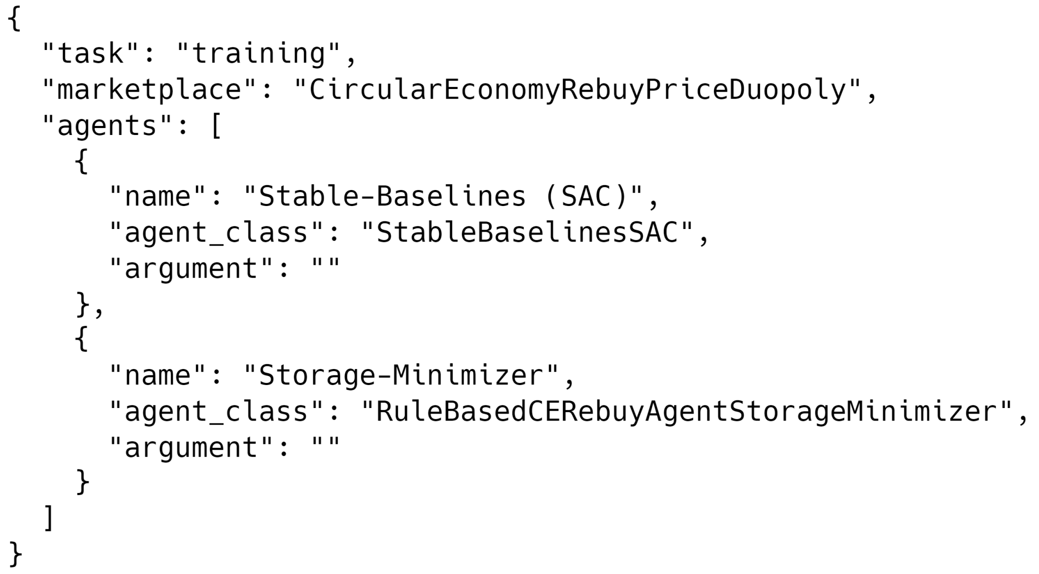
Table 6.2: 15 areas of motivation toward second-hand shopping, taken from [RG08] (descriptions omitted)

Listing 6.1: Policy implementation of the *RuleBasedCEAgent*, simplified for readability

```
def policy(market_state) -> tuple:
  products_in_storage = market_state[1]
  price_refurbished = 0
  price_new = config_market.production_price
  rebuy_price = 0
  if products_in_storage < config_market.max_storage/15:
    # fill up the storage immediately
    price_refurbished = config_market.max_price * 6/10
    price_new += config_market.max_price*6/10
    rebuy_price = price_refurbished-1

  elif products_in_storage < config_market.max_storage/10:
    # fill up the storage
    price_refurbished = config_market.max_price * 5/10
    price_new += config_market.max_price * 5/10
    rebuy_price = price_refurbished - 2

  elif products_in_storage < config_market.max_storage/8:
    # storage content is ok
    price_refurbished = config_market.max_price * 4/10
    price_new += config_market.max_price * 4/10
    rebuy_price = price_refurbished // 2

  else:
    # storage too full, get rid of some refurbished products
    price_refurbished = config_market.max_price * 2/10
    price_new += config_market.max_price * 7/10
    rebuy_price = 0

  price_new = min(9, price_new)
  return (price_refurbished, price_new, rebuy_price)
```

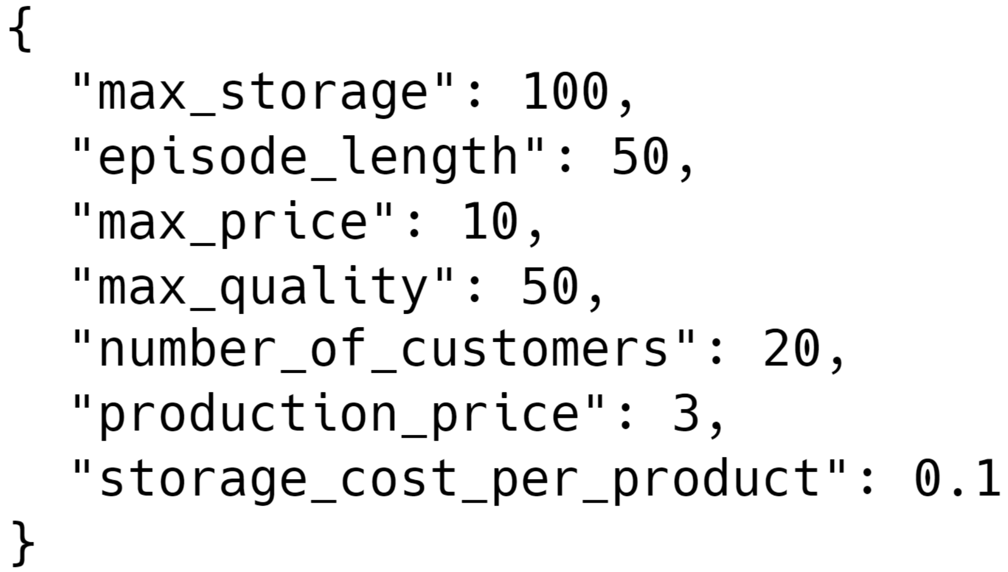
Listing 6.2: Policy implementation of the *RuleBasedCERebuyAgentStorageMinimizer*, simplified for readability

```
def policy(observation) -> tuple:
  own_storage = observation[1].item()
  refurbished_prices, new_prices, rebuy_prices =
      get_competitor_prices()

  price_new = max(median(new_prices)-1,
      config_market.production_price+1)

  if own_storage < config_market.max_storage/15:
    # fill up the storage immediately
    price_refurbished = max(new_prices + refurbished_prices)
    rebuy_price = price_new - 1

  else:
    # storage too full, get rid of some refurbished products
    rebuy_price = min(rebuy_prices)-config_market.max_price*10
    price_refurbished = int(np.quantile(refurbished_prices, 0.25))

  return clamped_prices(price_refurbished, price_new, rebuy_price)
```



```
{
  "task": "training",
  "marketplace": "CircularEconomyRebuyPriceDuopoly",
  "agents": [
    {
      "name": "Stable-Baselines (SAC)",
      "agent_class": "StableBaselinesSAC",
      "argument": ""
    },
    {
      "name": "Storage-Minimizer",
      "agent_class": "RuleBasedCERebuyAgentStorageMinimizer",
      "argument": ""
    }
  ]
}
```

Figure 6.1: The environment_config.json of the SAC-Duopoly Experiment



```
{
  "max_storage": 100,
  "episode_length": 50,
  "max_price": 10,
  "max_quality": 50,
  "number_of_customers": 20,
  "production_price": 3,
  "storage_cost_per_product": 0.1
}
```

Figure 6.2: The market_config.json of the SAC-Duopoly Experiment

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, 18th June 2022

Nikkel Mollenhauer