Fischer, Thomas G.

Working Paper

Reinforcement learning in financial markets - a survey

FAU Discussion Papers in Economics, No. 12/2018

Provided in Cooperation with:
Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics

Suggested Citation: Fischer, Thomas G. (2018) : Reinforcement learning in financial markets - a survey, FAU Discussion Papers in Economics, No. 12/2018, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institute for Economics, Nürnberg

This Version is available at:
https://hdl.handle.net/10419/183139

Discussion Papers in Economics

No. 12/2018

Reinforcement learning in financial markets - a survey

Thomas G. Fischer
University of Erlangen-Nürnberg

ISSN 1867-6707

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institute for Economics
https://www.iwf.rw.fau.de/research/iwf-discussion-paper-series/

Reinforcement learning in financial markets - a survey

Thomas Fischer[a,1]

[a] *University of Erlangen-Nürnberg, Department of Statistics and Econometrics, Lange Gasse 20, 90403 Nürnberg, Germany*

Monday 1st October, 2018

Abstract

The advent of reinforcement learning (RL) in financial markets is driven by several advantages inherent to this field of artificial intelligence. In particular, RL allows to combine the "prediction" and the "portfolio construction" task in one integrated step, thereby closely aligning the machine learning problem with the objectives of the investor. At the same time, important constraints, such as transaction costs, market liquidity, and the investor's degree of risk-aversion, can be conveniently taken into account. Over the past two decades, and albeit most attention still being devoted to supervised learning methods, the RL research community has made considerable advances in the finance domain. The present paper draws insights from almost 50 publications, and categorizes them into three main approaches, i.e., critic-only approach, actor-only approach, and actor-critic approach. Within each of these categories, the respective contributions are summarized and reviewed along the representation of the state, the applied reward function, and the action space of the agent. This cross-sectional perspective allows us to identify recurring design decisions as well as potential levers to improve the agent's performance. Finally, the individual strengths and weaknesses of each approach are discussed, and directions for future research are pointed out.

Keywords: Financial markets, reinforcement learning, survey, trading systems, machine learning.

Email address: thomas.g.fischer@fau.de (Thomas Fischer)

[1]The author has benefited from many helpful discussions with Ingo Klein, Christopher Krauss, and Matthias Schnaubelt.

1. Introduction

As of today, most academic research on applying machine learning to finance, or more specifically, to trading financial assets, is devoted to supervised learning techniques. Already a decade ago, Atsalakis and Valavanis (2009) have surveyed more than 100 articles employing neural networks to predict future returns of financial assets. In addition, a whole set of other supervised methods have been applied to this task - among them, support vector machines (Kim, 2003; Huang et al., 2005), tree-based algorithms (Kumar and Thenmozhi, 2006; Booth et al., 2014; Moritz and Zimmermann, 2014; Krauss et al., 2017) and nearest neighbor classification (Teixeira and De Oliveira, 2010). The general idea of the majority of these works is as follows: First, a predictive model, e.g., a neural network or a random forest, is trained on historical data to forecast the asset's price change using a set of explanatory variables (features). Second, these forecasts are fed into a trading module to derive the actual trading action, e.g., to buy the financial asset in case the forecast surpasses a certain threshold. Despite its unbroken popularity, this two-step approach has several limitations that "may lead to suboptimal performance" (Moody et al., 1998b, p. 5): First, the optimization objective in the predictive model, i.e., the minimization of the forecast error, is not necessarily in line with the ultimate goal of the investor, e.g., the maximization of the return per unit of risk. Second, in most cases, only the forecast itself serves as input to the trading module - additional valuable information that could be obtained from the feature space are discarded (Moody et al., 1998b). Third, exogenous constraints imposed by the environment, e.g., lack of liquidity and transaction costs, are only incorporated into the optimization of the trading component, or - as in the majority of cases - not considered at all.

Reinforcement learning (RL), i.e., "learning what to do, how to map situations to actions, so as to maximize a numerical reward signal" (Sutton and Barto, 1998, p. 3), promises to overcome these limitations. Specifically, the forecasting and the subsequent portfolio construction are integrated in one single step and jointly optimized in line with the objective of the investor. Hereby, the "trading agent" learns by interacting with the environment (or a model thereof), allowing it to incorporate the aforementioned constraints, such as liquidity and transaction costs, into its decision making process. This paper surveys the current state of this research while shedding light on important design decisions. Figure 1 provides an overview of the three identified RL approaches (color coding), as well as their most relevant works:

- *Critic-only approach:* The critic-only approach is the most frequent[2] application of RL in

[2] in terms of publications

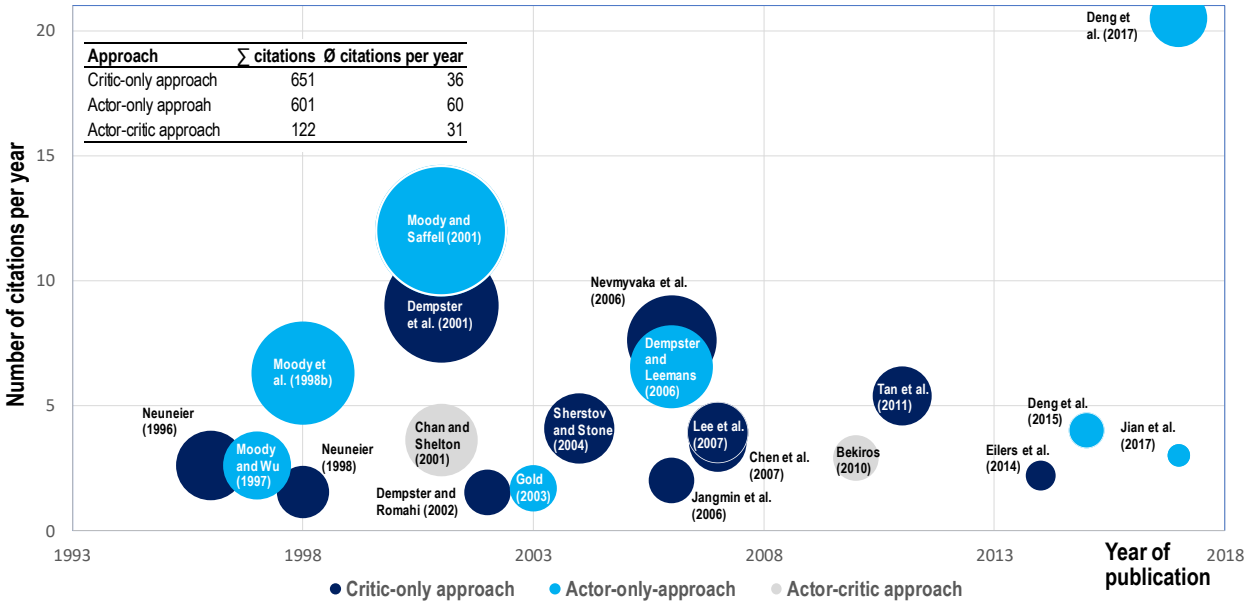

Figure 1: Overview of research papers applying RL in financial markets. The color-coding represents the different RL approaches, i.e., critic-only, actor-only, and actor-critic. The bubble sizes denote the total number of citations obtained from Google Scholar as of 2018-09-16. The position on the x-axis represents the year of publication, the position on the y-axis the average number of citations per year. Works with less than 1.5 citations per year are omitted.

financial markets. The idea of this approach is to learn a value function based on which the agent can compare ("criticize") the expected outcomes of different actions, e.g., "to go long" or "to go short"[3]. During the decision making process, the agent senses the current state of the environment and selects the action with the best outcome according to the value function.

- *Actor-only approach:* The actor-only approach is the second most common approach. Hereby, the agent senses the state of the environment and acts directly, i.e., without computing and comparing the expected outcomes of different actions. The agent hence learns a direct mapping (a policy) from states to actions. The key advantage of this approach is the continuous action space of the agent (e.g., to obtain fine-grained portfolio weights) and the typically faster convergence of the learning process.

- *Actor-critic approach:* The actor-critic approach forms the third category and aims at combining the advantages of the actor-only and the critic-only approach. The key idea is to simultaneously use an actor, which determines the agent's action given the current state of the environment, and a critic, which judges the selected action. Simply speaking, the ac-

[3]Going short means to borrow and sell a security at time t with the objective to buy it back at a lower price (and to return it) at some point in the future. In other words, the investor aims to make a profit from decreasing prices.

tor learns to choose the action which is considered best by the critic and the critic learns to improve its judgment. Despite its potential advantages, actor-critic RL is the least well researched approach in financial markets and has only a limited number of supporting works.

In total, the present paper draws insights from almost 50 publications, hereby contributing to the current state of research in three respects: First, the literature is comprehensively categorized along the three approaches, i.e., critic-only, actor-only, and actor-critic. Second, the most important contributions in each category are discussed in detail, including their most relevant extensions. We hereby aim to use a uniform terminology and notation to allow new researchers to quickly familiarize with this field. Third, for each category, the cross-section of works is further analyzed with regard to the variables used to model the state of the environment, the action space of the agent, as well as the employed reward function. This cross-sectional perspective allows us to identify recurring design decisions as well as promising directions for future research. The remainder of this paper is organized as follows: Section 2 describes the critic-only reinforcement learning approach as well as its various extensions and applications in financial markets. Section 3 is devoted to actor-only agents, i.e., recurrent reinforcement learning, as well as their recent enhancement with deep learning techniques. Section 4 covers actor-critic RL and section 5 focuses on comparison studies. Finally, section 6 contrasts the strengths and weaknesses of the individual approaches and concludes.

2. Critic-only approach

This section discusses the application of critic-only reinforcement learning in financial markets. Table 1 summarizes the main works, their year of publication, as well as the data sample and resolution. Furthermore, the key findings and contents of the respective papers are outlined.

2.1. Baseline approach and relevant extensions

2.1.1. Baseline approach

The application of critic-only reinforcement learning to portfolio management has first been introduced by Neuneier (1996). The key idea, as in all three surveyed approaches, is to treat the management of the portfolio as a Markovian decision problem. The main components are:

- A finite set of *states* S_t summarizing the information the *agent* senses from the *environment* at every time step $t \in \{1, ..., T\}$.
- A set of *actions* A_t which the agent can perform at each time step $t \in \{1, ..., T\}$ to interact with the environment.

Study	Data sample	Resolution	Key findings/contents
Neuneier (1996, 1998)	Artificial FX data, DAX index (1986-1996)	Daily	**Baseline approach:** RL can be successfully applied to portfolio allocation problems; performance exceeds heuristics and supervised learning approach; volatility can be reduced by considering a penalty term.
Bertoluzzo and Corazza (2012)	One single Italian stock (1973-2006)	Daily	The algorithm used to approximate the value function impacts the trading performance; Q-learning works better than kernel-based methods.
Corazza and Bertoluzzo (2014)	Six selected Italian stocks (1985-2014)	Daily	Varying the reward function and hyperparameters (e.g., number of lagged returns used to describe the state) can improve the performance of RL trading systems.
Jin and El-Saawy (2016)	Two selected US stocks (2001-2016)	Daily	Deep Q-learning (DQL) techniques can be applied to approximate the action-value function in RL trading systems.
Dempster et al. (2001), Dempster and Romahi (2002)	GBP-USD currency pair (1994-1998)	Minute-binned	**Evolutionary reinforcement learning:** Genetic algorithms (GA) can improve the performance of RL trading systems, i.e., by finding a suitable state representation.
Chen et al. (2007), Gu et al. (2011)	Selected Japanese stocks (2001-2004)	Daily	Trading systems based on genetic network programming (GNP) can be improved by leveraging RL techniques within judgment and processing nodes.
Lee and Jangmin (2002), Lee et al. (2002, 2003, 2007)	Stocks from the Korean KOSPI 200 index (1999-2005; backtest period differs depending on version of the paper)	Daily	**Divide and conquer - from one to multiple agents:** The trading task (identification of trading opportunities, determination of limit prices) can be divided among multiple collaborating agents that are trained in one integrated learning process.
Tan et al. (2011)	Five selected US stocks (1986-2006)	Daily	**Enhancing existing trading strategies:** RL can be applied to improve trading strategies aiming at the identification of cycles in stock prices.
Eilers et al. (2014)	S&P 500 index, DAX index (2000-2012)	Daily	RL can be applied to improve trading strategies exploiting seasonal effects (e.g., turn-of-the-month and exchange holidays)
Sherstov and Stone (2004)	Penn Exchange Simulator (2003/2004)	High-frequency	**High-frequency applications:** RL can be applied to high-frequency settings. (However, the authors find their RL agent to be outperformed by two benchmark models.)
Cumming et al. (2015)	Eleven currency pairs (2014)	Minute-binned	RL can be applied to build trading systems on intraday FX data.
Nevmyvaka et al. (2006)	1.5 years of limit order data of selected NASDAQ stocks	Milliseconds	**Optimizing execution:** RL can be applied to improve order execution.
Jangmin et al. (2006)	Constituents of the Korean KOSPI and KOSDAQ index (1998-2003)	Daily	**Expanding on the number of assets:** RL can be applied to dynamically allocate investments among several trading opportunities (signals).
Kaur (2017)	Selected US stocks (5 years of data)	Daily	RL can be applied to simultaneously trade a portfolio of two risky assets (buy/sell actions for each of the two stocks are modeled in the action space); sentiment data can be integrated to improve performance of RL trading systems.
Watts (2015)	Artificial data (one risky and one risk-free asset)	-	**Other:** RL can be applied to hedge basis risk.

Table 1: Overview of surveyed works applying the critic-only approach. Note: FX data refers to foreign exchange data.

- A set of *transition probabilities* between subsequent states which render the environment stochastic. Note: the probabilities are usually not explicitly modeled but the result of the stochastic nature of the financial asset's price process.
- A *reward* (or return) function R_t which provides a numerical feedback value r_t to the agent in response to its action $A_{t-1} = a_{t-1}$ in state $S_{t-1} = s_{t-1}$.
- A *policy* π which maps states to concrete actions to be carried out by the agent. The policy can hence be understood as the agent's rules for how to choose actions.
- A *value function* V which maps states to the total (discounted) reward the agent can expect from a given state until the end of the episode (trading period) under policy π.

Given the above framework, the decision problem is formalized as finding the optimal policy $\pi = \pi^*$, i.e., the mapping from states to actions, corresponding to the optimal value function V^* - see also Dempster et al. (2001); Dempster and Romahi (2002):

$$V^*(s_t) = \max_{a_t} \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1})|S_t = s_t]. \tag{1}$$

Hereby, $\mathbb{E}$ denotes the expectation operator, γ the discount factor, and R_{t+1} the expected immediate reward for carrying out action $A_t = a_t$ in state $S_t = s_t$. Further, S_{t+1} denotes the next state of the agent. The value function can hence be understood as a mapping from states to discounted future rewards which the agent seeks to maximize with its actions. To solve this optimization problem, the Q-Learning algorithm (Watkins, 1989) can be applied, extending the above equation to the level of state-action tuples:

$$Q^*(s_t, a_t) = \mathbb{E}[R_{t+1} + \gamma \max_{a_{t+1}} Q^*(S_{t+1}, a_{t+1})|S_t = s_t, A_t = a_t]. \tag{2}$$

Hereby, the Q-value $Q^*(s_t, a_t)$ equals to the immediate reward for carrying out action $A_t = a_t$ in state $S_t = s_t$ plus the discounted future reward from carrying on in the best way possible. The optimal policy π^* (the mapping from states to actions) then simply becomes:

$$\pi^*(s_t) = \max_{a_t} Q^*(s_t, a_t), \tag{3}$$

i.e., in every state $S_t = s_t$, choose the action $A_t = a_t$ that yields the highest Q-value. To approximate the Q-function during (online) learning, an iterative optimization is carried out with α

denoting the learning rate - see also Sutton and Barto (1998) for further details:

$$Q^*(s_t, a_t) \leftarrow (1-\alpha)\, Q^*(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \right). \tag{4}$$

Neuneier's trading strategies for the DM-USD[4] currency pair and for the DAX index naturally fit into this framework. For the sake of simplicity, we focus on the currency example: At every time step t, the agent senses the environment by observing the state $S_t = s_t$ in form of the current exchange rate, the wealth of its portfolio, and its current position (DM or USD). Based on its knowledge (learned value function and policy), the agent then decides to either invest in DM or in USD for the duration of the next time step (by comparing the respective Q-values), and finally performs the corresponding action $A_t = a_t$. Once this time step is over, the agent receives the reward $R_{t+1} = r_{t+1}$, observes the new state $S_{t+1} = s_{t+1}$, and the process is repeated. One key advantage, especially compared to the static setup in supervised learning, is the high flexibility of the reward function which can be conditioned on states and actions. In Neuneier's case, the reward is the immediate return net of transaction costs and both state- and action-dependent:

$$R_{t+1} = \begin{cases} \frac{z_{t+1}}{z_t}(c_t - \delta_t) - c_t & \text{if current position is DM and } A_t \text{ is USD} \\ \frac{z_{t+1}}{z_t} c_t - c_t & \text{if current position is USD and } A_t \text{ is USD} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Hereby, z_t and z_{t+1} denote the asset's price at time step t and $t+1$ respectively. Further, c_t denotes the agent's capital at time step t and $\delta_t = 0.1 + c_t/100$ are the transaction costs occurring when a switch from DM to USD is performed. We can easily see that the transaction costs lower the reward and penalize the (frequent) change of positions. In fact, Neuneier observes a reduction of position changes by a factor of three compared to a supervised learning benchmark - a strong driver for the 25 percentage points higher profit of the RL agent.

Building upon this initial approach, several refinements have been developed. Neuneier (1998), for example, incorporates the risk aversion of the investor into the reward function. Specifically, he proposes the use of $R_t = \log\left(\frac{PF\ value_t}{PF\ value_{t-1}}\right)$, i.e., the logarithmic change of the value of the portfolio, as it "penalizes losses much stronger than gains" (Neuneier, 1998, p. 942).

Bertoluzzo and Corazza (2012) compare the performance of different algorithms to approximate the value function on an artificial and a real world financial time series. In particular, they

[4]DM refers to Deutsche Mark, the former German currency.

compare temporal difference methods, i.e., Q-learning (Watkins, 1989), with Kernel-based reinforcement learning (Ormoneit and Sen, 2002) using the Sharpe ratio as reward function. In conclusion, the authors find the Q-learning algorithm to generally perform better than Kernel-based reinforcement learning on the artificial data with the caveat that the results vary strongly across the 1000 performed replications. Unfortunately, with regard to the real world data, only one single time series is employed - it hence remains to be seen how these results translate to other assets.

In subsequent work, Corazza and Bertoluzzo (2014) explore the effect of different reward functions and varying hyperparameters. Specifically, the authors compare three different reward functions, i.e., Sharpe ratio, average log return, and $OVER_t$ ratio[5], calculated over $L \in \{5, 21\}$ days, varying numbers of lagged asset returns incorporated in the state ($N \in \{1, 5\}$), and four different ϵ parameter values for the ϵ-greedy exploration. The latter parameter forces the agent to randomly select an action A_t with probability ϵ (instead of selecting the action with the highest Q-value), allowing it to explore additional options in the action space:

$$A_t = \begin{cases} \max_{a_t} Q^*(S_t, a_t), & \text{with probability } 1 - \epsilon \\ \text{random action } a_t, & \text{with probability } \epsilon. \end{cases} \tag{6}$$

Performance-wise, the authors find Sharpe and $OVER_t$ ratio on par, whereas the average log return is lagging behind. With regard to the number of days used to calculate the reward, $L = 5$ performs slightly better than $L = 21$ on the real world data. Finally, the authors report $\epsilon \in \{10\%, 20\%\}$ and one lagged return for the state ($N = 1$) to be the preferable configurations. Overall, we find clear value in this kind of research, in particular, as it provides other researchers with a basis for their own experiments. Going forward, a large set of relevant research questions could be answered in similar studies, for example, is it better to train one single agent with the time series of several stocks (i.e., to learn more general patterns and to leverage a larger number of training samples) or is it better to train one agent per asset (in order to capture its idiosyncratic behavior).

Only recently, Jin and El-Saawy (2016) extend Neuneier's approach by applying deep Q-learning (DQL) techniques. Specifically, the authors make use of ϵ-greedy exploration in combination with experience replay (Lin, 1992). The latter technique is used to update the weights of the artificial neural network (which approximates the Q-value function) using mini batches of randomly selected past experiences, i.e., state-action-reward-state-quadruples $(s_t, a_t, r_{t+1}, s_{t+1})$. Hereby, the agent's

[5] "The sum of the net logarithmic returns and the sum of the absolute value of the same net logarithmic returns at time t both calculated over the last L stock market days" (Corazza and Bertoluzzo, 2014, p. 12).

past experiences are efficiently reused allowing to train the agent on less data, while the randomization can lead to better convergence behavior (Silver, 2015). Result-wise, the authors find their DQL agent to outperform a buy and hold and a rebalance benchmark. Unfortunately, due to the lack of a plain Q-learning agent, the specific value-add of DQL is not reported. Going forward, more research will be required to truly understand how these new techniques can be effectively applied in the context of noisy financial data.

2.1.2. Evolutionary reinforcement learning

The works in this section aim at enhancing and combining the critic-only approach with evolutionary algorithms. Point of departure forms the research of Dempster et al. (2001) who benchmark Q-learning against genetic programming (GP) in a GDB-USD currency trading setup. In their study, both methods receive the trading signals of eight technical indicators as well as the current position as input (state), and decide to keep/switch currencies for the duration of the next time step. Looking at the in- and out-of-sample performance, the authors find strong signs for their Q-learning agent to suffer from overfitting - the impetus for the "evolutionary reinforcement learning approach" in Dempster and Romahi (2002). In this work, the authors begin with the eight technical indicators (from the previous paper) as state variables and use a genetic algorithm to identify the subset ("individual") that yields the best trading performance. Hereby, the authors split the training period into two parts. On the first part of the training data, a separate RL trading agent is trained for each combination of technical indicators generated by the GA. In the second step, the trading performance of each of these agents is evaluated on the remaining part of the training data. The achieved results are then fed back to the GA to judge the fitness of the individuals and to produce a superior generation. After an evolution for 100 generations, the training is stopped and the best performing individual is selected. In their backtests, the authors find the hybrid system to outperform a RL-only system by a clear margin, suggesting that GAs can be successfully employed to optimize the state representation (and potentially other hyperparameters) of RL trading systems. In this respect, only the title of the concisely written paper, i.e., "Intraday FX trading: an evolutionary reinforcement learning approach", might cause misunderstandings, as in fact, the GA mutates sets of state variables and not agents.

Chen et al. (2007) propose another approach at the intersection of evolutionary algorithms and reinforcement learning. Their "GNP-Sarsa" algorithm combines genetic network programming (Hirasawa et al., 2001) with Sarsa learning (Rummery and Niranjan, 1994) and is inspired by earlier works of Potvin et al. (2004). In genetic network programming (GNP), individuals are encoded

as graphs as opposed to binary strings in GA. Each individual consists of a starting node, a set of judgment and processing nodes, as well as transitions between the nodes[6]. In the context of the authors' paper, the judgment nodes receive a candle stick pattern or a technical indicator (e.g., moving average convergence divergence) as input, calculate a buy/sell signal as output, and decide which node to visit next, e.g., a second judgment node to look at a different indicator. The judgment nodes are complemented by so called processing nodes which represent concrete actions, i.e., buy/sell the asset. The authors' key idea is to optimize each of the individuals by equiping it with a dedicated RL agent (one per individual). Specifically, each node of the individual represents a different state for the agent. In case the state is a judgment node, the agent's action determines which candle stick pattern/technical indicator should be used for the judgment. In case the state is a processing node, the actions resemble different threshold values the buy/sell signal needs to exceed to trigger an actual trade. In addition to this RL-based optimization, the evolutionary part of "GNP-Sarsa" produces new individuals by changing their graph structure through crossover and mutation, e.g., by removing or adding transitions between two nodes. Once the last generation is reached, the best performing individual is tested in an out-of-sample trading period. Hereby, the authors find their "GNP-Sarsa" algorithm to outperform a buy and hold strategy on 13 out of 16 selected Japanese stocks and GNP alone in 12 out of 16 cases. Unfortunately, neither Chen et al. (2007), nor its extension to plural subroutines (Gu et al., 2011), include a RL-only benchmark. We therefore cannot judge whether the reported results justify the complexity and presumably high computational costs of the "GPN-Sarsa" architecture.

2.1.3. Divide and conquer - from one to multiple agents

Lee and Jangmin (2002) and the subsequent refinements and enhancements of Lee et al. (2002, 2003, 2007) focus on improving the performance of RL trading systems by splitting the problem into smaller tasks that are solved by collaborating agents. Hereby, four different agents are distinguished:

- *Buy signal agent*: Observes the long-term price development of a stock and identifies opportunities to open a long position.
- *Buy order agent*: Is triggered by the buy signal agent and determines the limit price for the buy order by observing intraday price patterns.
- *Sell signal agent*: Observes the development of the long position (including the accrued profit/loss) and determines when to close the position.

[6]Furthermore, the graph can be enriched by time delays to model additional system dynamics.

- *Sell order agent*: Is triggered by the sell signal agent and determines the limit price for the sell order by observing intraday price patterns.

The above descriptions reveal three key ideas. First, each agent specializes on a certain sub-problem, i.e., identification of the trading opportunity (signaling) and setting the limit price of the order (execution). Second, the agents have different state representations tailored to their specific task, e.g., historical price changes over a long period of time (signaling agents) versus intraday price movements (order agents). Third, the agents need to collaborate to solve the overall trading problem. The latter fact is important and is achieved by the design of the training process, where "each agent has its own goal while interacting with others" (Lee et al., 2007, p. 865). Each training episode begins with the *buy signal agent* analyzing a stock's price development until the current day t. In case no buying opportunity is identified, the episode ends and a new training day begins. In case an opportunity arises, the *buy order agent* is triggered and sets a limit price to purchase the stock on the subsequent day $t+1$. Once a successful purchase is made, the *buy order agent* receives a reward. The closer the limit price is to the low of $t+1$, the higher is this reward. From there on, the *sell signal agent* monitors the stock's price development as well as the accrued profit/loss of the position. During the lifetime of the position, a reward in form of the change of the position's market value is given to the *sell signal agent*. Once the *sell signal agent* decides to close the position, the *sell order agent* is triggered and rewarded similar to the *buy order agent*, i.e., the closer the limit price is to the high of the respective day, the higher is the reward. Finally, based on the realized profit of the round trip trade, the *buy signal agent* is rewarded and a new episode begins. The outlined training approach clearly underlines the high flexibility of RL trading systems and the authors illustrate in a compelling manner how both agent-specific and global goals can be jointly optimized. Also with regard to the results, the authors find their system to achieve better trading performance compared to several benchmarks including a supervised learning model.

2.1.4. Enhancing existing trading strategies

The literature summarized in this section applies reinforcement learning to enhance and fine-tune existing trading strategies. Inspired by earlier research by Kitchin (1923), Sarlan (2001) and Plummer (2009), Tan et al. (2011) develop a RL-based trading system that automatically identifies and trades stock price cycles. Specifically, the system opens long/short positions as closely as possible to the inflection points of the price time series. The system comprises three components:

- *Cycle identifier and tuner*: The first component aims at extracting cyclic movements in stock prices by identifying a series of peaks and troughs. The identified cycles are then fine-tuned

by shifting them to improve their match with the actual price time series. In both cases, a simple RL agent is employed to determine optimal parameters for the cycle identification (e.g., moving average period) and the required cycle shift.

- *ANFIS*: The second component, an adaptive network fuzzy inference system (ANFIS), aims at predicting inflection points in the cycles, i.e., optimal points to open/close a position. Again, a simple RL agent is employed to fine-tune the hyperparameters of the ANFIS component.

- *Trading agent*: The third component is responsible for the actual trading and aims at opening/closing the position as closely to the inflection points as possible. The component itself is implemented as RL agent and rewarded with the profit resulting from the trade.

In a backtest over a period of 13 years, the authors find their trading system to outperform the market by a clear margin of 50 percentage points and provide valuable ideas on how RL can be leveraged to fine-tune individual trading system components. Given the sample size of five stocks, it would be highly interesting to see a similar approach being applied to the S&P 500 constituents.

Eilers et al. (2014) pursue another attempt at enhancing existing trading strategies. Specifically, the authors focus on improving the trading of three different seasonal effects, i.e., upward biases during exchange holidays (French, 1980), upward biases at the turn-of-the-month (Ariel, 1987), and pre-FOMC (Federal Open Market Committee) announcement drifts (Lucca and Moench, 2015). In their baseline strategy, the authors open a long position on the day before an event and close that position two days later. The goal of the agent is to improve upon this basic strategy. On each day prior to an event, the agent can choose the direction of its position (long, short, do nothing), the holding period (one day or two days), and the leverage (one or two). In total, the action space hence comprises nine different actions resulting from the possible combinations of the aforementioned parameters. To provide the agent with a sense of the environment, the current market condition is described with eleven variables including the type of the event, the open, high, low and close value of the present day, as well as a set of technical indicators. Based on that, the agent is trained with the objective to maximize the return per trade using Q-learning. In a backtesting study on the S&P 500 and the DAX index, the authors find their RL trading system to outperform the baseline strategy by a clear margin. As such, the well structured paper is another valuable inspiration for how RL can be leveraged in finance. One slight improvement vis-a-vis the paper might be to binary encode the "number of the current month" (as categorical state variable) instead of representing it as a single number between 1 and 12 (unless there is predictive value related to the order of months).

2.1.5. High-frequency applications

Sherstov and Stone (2004) develop a RL-based trading system as part of a comparison study within the Penn Exchange Simulator (PXS)[7]. In their paper, the current state of the environment is modeled with just one single variable indicating the stock's recent price trend, i.e., the difference between the stock's current price p_t and its exponential moving average $\overline{p_t} = \beta\overline{p_{t-1}} + (1-\beta)p_t$. At every time step t, the agent decides upon the number of shares to buy/sell (actions) and is rewarded by the change in wealth. Result-wise, the authors find their RL trading agent to be outperformed by two benchmark models, especially on days with high fluctuation in prices. The latter indicates that the state representation is too simplified to capture the complex dynamics of stock prices (Sherstov and Stone, 2004). Nonetheless, the good performance on days with a clear trend is an encouraging sign for further work on the application of RL in high-frequency settings.

2.1.6. Optimizing trade execution

Inspired by the works of Bertsimas and Lo (1998), Almgren and Chriss (2001), El-Yaniv et al. (2001), Tesauro and Bredin (2002), Coggins et al. (2003), and Kakade et al. (2004), Nevmyvaka et al. (2006) apply RL to optimize the execution of trades in a large-scale empirical study. In their paper, the authors aim at minimizing the amount of capital spent for buying a specific number of shares V in a predefined amount of time H (analogously, the authors aim at maximizing the received revenue for selling a specific number of shares). To provide the agent with a comprehensive view of the environment, the state comprises both private variables (number of shares that still need to be executed, elapsed time), as well as several market variables describing the current state of the order book (e.g., the current bid-ask spread). At every time step t, the agent sets a new limit price, observes the executed shares, and senses the new state of the order book. Once all shares are executed or the time H has elapsed, the agent is rewarded with the total proceeds of the order[8]. Based on a large-scale simulation study with three selected stocks on 1.5 years of limit order data on millisecond resolution (which is outstanding for an academic study in terms of the amount of data alone), the authors find their RL execution strategy to clearly outperform a traditional submit and leave strategy. Moreover, the paper is interesting for two reasons. First, it outlines a promising application for RL in finance. Second, this application seems rather "market-ready" - in fact, JPMorgan Chase & Co is actively testing a RL agent for automated trade execution (Noonan, 2017; Terekhova, 2017), hereby underlining its relevance to practice and its disruptive potential.

[7]The PXS is a stock-trading simulator merging agent with real-world orders (Kearns and Nevmyvaka, 2013).
[8]In case H has elapsed, all remaining shares are executed at the current market price.

2.1.7. Expanding on the number of assets

All works described so far either focus on the trading of one single risky asset or on the allocation of the investment among a risky and a risk-free asset. Jangmin et al. (2006) finally propose a RL-based asset allocation strategy which dynamically distributes the investment among multiple trading opportunities. In their framework, two main components are distinguished:

- *Local traders:* The local traders identify and invest in trading opportunities. Specifically, they spot promising stocks, open a long/short position and reverse that position after a certain amount of time. Each local trader is created in two steps. First, a neural network is trained to forecast the stock price change over a fixed amount of time (signal). Second, optimal trading thresholds are derived, i.e., values the signal needs to exceed to trigger an actual trade[9]. A local trader hence consists of the neural network and the fixed threshold values (local policy), and can identify an arbitrary number of trading opportunities each day.

- *Meta policy:* The meta policy completes the system and aims at optimally distributing the investments among the four employed local traders. The meta policy is implemented as critic-only agent with the state consisting of the number of identified trading opportunities (per trader) as well as the available cash. The actions encode the amount of money (discretized to four different values) each local traders may invest per trade - in other words, the meta-policy can choose from $4^4 = 256$ different actions. As reward function, the authors employ the profit ratio over the whole trading period, i.e., there are no intermediate rewards.

In a backtesting study, the authors find their meta policy approach to clearly outperform two handcrafted benchmark policies and report the initial investment to have more than doubled after 17 months. Even though it may be difficult to fully reproduce the paper (e.g., due to the not reported forecast horizon and configuration of the neural networks embedded in the local traders), it is very valuable for another reason. Specifically, it proposes a creative approach to combining supervised learning (local traders) with RL to dynamically allocate investments among multiple signals. Based on these ideas, several extensions could be explored in future research. For example, it would be highly interesting to analyze whether the results can be improved by enriching the state with additional information. Specifically, by adding the magnitude of the individual price change predictions as state variables, the meta policy could judge how certain the local traders are.

Kaur (2017) takes a different and more direct approach to trading several risky assets. In his

[9]The threshold values are derived in a grid-search on part of the training data.

paper, the agent can choose for two stocks independently whether a buy, sell or hold action should be performed for the next time step. The agent can hence choose from $3^2 = 9$ different actions (buy stock 1 and hold stock 2, buy stock 1 and buy stock 2, ...). We can easily see that modeling the actions in this way leads to an exponential growth of the action space, i.e., in case of three stocks, already $3^3 = 27$ actions are possible. Nonsurprisingly, the author himself admits that "this increases the run-time of our system considerably, and limits the performance of the system" (Kaur, 2017, p. 5). In general, we find the simultaneous trading of several assets using the critic-only approach challenging - an area where the actor-only approach (see section 3) and potentially the actor-critic approach (see section 4) seem more suitable. On the other hand, these problems might also be related to not yet having found the right "representation" for the multi-securities trading problem.

2.2. Survey of works along state, actions, and rewards

In the following subsections, we survey the cross-section of works of the critic-only approach along three dimensions, i.e., (i) the information used to model the state of the environment, (ii) the action space of the agent, and (iii) the employed reward function.

2.2.1. State

The state provides the agent with a view of the environment. As with feature engineering in supervised learning, the representation of the state is key to the performance in RL. Table 2 outlines the different types of information incorporated into the state. We make the following observations:

- *Discrete states:* A considerable share of works applies discretization to reduce the size of the state space. When looking at the publication dates however, we find that this mainly applies to the early works where computational resources to train large artificial neural networks as function approximators were limited. Nonsurprisingly, with the advent of GPU hardware in recent years, the majority of works nowadays operate with continuous state variables.

- *Price history and candle sticks:* Almost all works incorporate the asset's recent price history into the state with the number of lagged values varying by study. Corazza and Bertoluzzo (2014) report one lag to be optimal, however this finding appears to be highly problem-dependent. In fact, Sherstov and Stone (2004) report one price trend indicator to be insufficient to capture the complex price dynamics. We further observe that only few studies incorporate information derived from the open, highest, and lowest price. One of these exceptions is Lee et al. (2007), who compute four candle stick indicators representing the body, the upper and lower shadow, as well as the one-day percentage change of the closing price.

Study	Discrete states	Price history	Candle sticks	Current position	PF value/profit	Available cash	Techn. indicators	Macro data	Order book data	Other	Details/remarks
Neuneier (1996, 1998)	x	x		x	x			x			Price history of DAX index and eleven influencing market variables (e.g., interest rates, stock indices; exact variables used are not specified in the paper).
Bertoluzzo and Corazza (2012)		x									One-day percentage returns over the past five days.
Corazza and Bertoluzzo (2014)		x									Past one and past five-day (multi-period) log return.
Jin and El-Saawy (2016)		x		x	x	x					Past stock prices, current position size, current value of the portfolio, available cash.
Dempster et al. (2001), Dempster and Romahi (2002)	x			x			x				Buy/sell signals of eight technical indicators encoded as binary string (e.g., price channel breakout, adaptive moving average, relative strength index); binary indicator for current position (USD vs. GBP).
Chen et al. (2007), Gu et al. (2011)										x	States are the judgment and processing nodes of the genetic network programming graph structure.
Lee and Jangmin (2002), Lee et al. (2002, 2003, 2007)	x	x	x		x		x			x	Agent-specific state representation; *Signaling agents:* turning point matrix (support/resistance levels), current profit of position; *Order agents:* two indicators for short-term price changes, four indicators to represent the most recent candle stick.
Tan et al. (2011)	x									x	Module-specific: *Cycle-finder:* standard deviation of the stock's time series, correlation with Dow Jones index; *Trading agent:* trend indicators over 30, 70, and 90 days.
Eilers et al. (2014)		x	x				x				OHLC (open, high, low, close) of the present day, close values of past three days, two technical indicators (simple moving average, relative strength index), number of the current month, presence of specific event (e.g. turn-of-the-month).
Sherstov and Stone (2004)		x									Difference between current price and an exponential average of previous prices (single variable).
Cumming et al. (2015)		x	x	x							Transformed candle sticks, current position.
Nevmyvaka et al. (2006)	x			x					x	x	Elapsed time and remaining shares to order, four order book-level features, i.e., bid-ask spread, bid-ask volume imbalance, immediate market order costs, signed transaction volume over the past 15 seconds.
Jangmin et al. (2006)							x			x	Number of trading opportunities identified by the local traders, available cash.
Kaur (2017)	x	x		x		x				x	Current position, most recent stock price, available cash, trend in price of asset (extracted from six technical indicators (e.g., moving average)), news sentiment score for each of the two stocks in the portfolio.
Watts (2015)		x			x						Underlying asset price, current wealth, time to payoff.

Table 2: Critic-only approach: Details on information used to describe the current state of the environment. An "x" denotes whether a certain type of data is part of the information set. The "discrete state" column indicates the discretization of the raw features, PF value denotes the value of the to portfolio.

- *Current position, portfolio value, and profit:* Only about half of the studies incorporate the agent's current position (e.g., whether the agent is currently invested in the risky asset) into the state. This observation is surprising as the position information allows the agent to consider the effect of transaction costs. However, when looking at the studies in more detail, we find that some of them do not incorporate transaction costs at all, e.g., Bertoluzzo and Corazza (2012), or apply RL in a setup where switching positions plays no role. Eilers et al. (2014), for example, close all positions after a maximum of two days. With regard to other related variables, i.e., current portfolio value, accumulated profit/loss, and available cash, we find only few studies making use of them.

- *Macro data, order book data and other:* Only a small number of studies make use of non price-based data. Neuneier (1996) is one of the few exceptions and includes macroeconomic data such as interest rates. Another example is Kaur (2017) who enriches the state with sentiment scores. In his study, he reports a Sharpe ratio increase from 1.4 to 2.4 - a compelling illustration of the potential of additional data sources.

In general, we find the modeling of the state an highly important aspect of RL trading system design. In fact, Dempster and Romahi (2002) show that selecting a suitable set of variables that generalizes well can strongly benefit the trading performance and suggest the use of genetic algorithms as a way to obtain such a set. When looking at the different types of information, we find price-based features to be clearly a prerequisite for modeling the state. Hereby, the most recent price history (e.g., using the five most recent returns) should be included, but also price changes over longer timeframes, e.g., a twelve months return to capture the stock's momentum, are worth exploring (see Takeuchi and Lee (2013) for valuable inspiration from the supervised learning research). The same applies to the agent's current position, which is required whenever transaction costs need to be considered. Finally, non-price based features deserve more attention. Motivated by the findings of Kaur (2017), future research could explore the potential of other non price-based information including fundamental data, earning estimates, and Google trends. Also, the results for sentiment data of Kaur (2017) are worth being replicated on a larger sample.

2.2.2. Action space and reward function

In this subsection, we finally survey the cross-section of works of the critic-only approach along the action space of the agent and the employed reward function. From panel A ("Actions") of table 3, we make the following observations:

	Panel A: Actions			Panel B: Reward functions			
Study	**Nbr. of actions**	**Discrete actions**	**Details/remarks**	**Profit**	**Sharpe ratio**	**Other**	**Details/remarks**
Neuneier (1996, 1998)	2	x	Invest/do not invest in risky asset.	x		x	Immediate (log) return net of transaction costs, penalty term for variance.
Bertoluzzo and Corazza (2012)	3	x	Buy/sell/neutral.		x		Sharpe ratio over the past 5/20 days.
Corazza and Bertoluzzo (2014)	3	x	Buy/sell/neutral.	x	x	x	Sharpe ratio, ratio of log returns over absolute log returns, avg. log return.
Jin and El-Saawy (2016)	7	x	Share by which the allocation between the two assets is changed (seven discrete percentage values).	x	x		Difference in portfolio value between two time steps, Sharpe ratio.
Dempster et al. (2001), Dempster and Romahi (2002)	2, 3	x	Long/short, long/short/neutral.	x			Immediate return (change in portfolio value) net of transaction costs.
Chen et al. (2007), Gu et al. (2011)	-	x	Actions of the RL agent are parameters for a genetic networking graph algorithm which takes the actual trading decision.	x			(Absolute) capital gain per trade.
Lee and Jangmin (2002), Lee et al. (2002, 2003, 2007)	2, 2, 7, 7	x	*Signaling agents:* Buy/do not buy, hold/sell; *Order agents:* seven actions determining the limit price of the order.	x		x	*Buy signal agent:* profit rate of trade net of transaction costs; *Sell signal agent:* profit rate; *Order agents:* difference between limit and optimum price.
Tan et al. (2011)	-	x	*Cycle finder:* combinations of momentum and moving average period; *Cycle shifter:* days to shift; *ANIFIS module:* tuning parameters; *Trading module:* buy/sell	x		x	*Cycle finder:* quality of cycle; *Cycle shifter:* Pearson's linear correlation coefficient; *ANIFIS module:* Combination of true error and mean correlation; *Trading module:* profit.
Eilers et al. (2014)	9	x	Triple of three decisions, i.e., position type (long/short/neutral), holding period (1 day vs. 2 days), and leverage (1 vs. 2).	x			Profit of the trade.
Sherstov and Stone (2004)	1801	x	Number of shares to buy/sell (value between -900 and 900 incl. "0").	x			Difference in portfolio value (including cash) between two time steps.
Cumming et al. (2015)	3, 2	x	State-dependent, i.e., long/short/idle (in case no asset is in the portfolio) and keep/close position (in case an asset is in the portfolio).	x			Cumulative profit over the trading period.
Nevmyvaka et al. (2006)	11	x	Discrete amount by which the limit of the order should differ from the current bid/ask.			x	Trading/execution costs; immediate reward is linked to executed shares in each step.
Jangmin et al. (2006)	256	x	Amount of cash (four possible values) to allocate to each of the four local traders.	x			Profit ratio over the trading period.
Kaur (2017)	9	x	Buy/sell/hold decision for each of the two stocks ($3^2 = 9$ actions).	x			Cumulative profit over the trading period.
Watts (2015)	3	x	Increase/keep/decrease investment in asset by a fixed amount.			x	Utility of the terminal payoff.

Table 3: Critic-only approach: Details on the modeling of the action space (panel A), as well as the employed reward function(s) (panel B). An "x" denotes whether a certain property (e.g., actions have discrete values) or type of reward function (e.g., profit), is present in the respective study.

- *Number of actions*: In most cases, the number of actions the agent can perform is rather small, i.e., two actions (buy/sell) or three actions (buy/sell/neutral). The main exception is Sherstov and Stone (2004), where each of the 1801 actions (including the "0" action) represents a different number of shares the agent aims to buy/sell at the next time step.

- *Discrete actions*: All actions in the critic-only approach are discrete, i.e., one action refers to one specific amount that should be invested in the risky asset (in most studies, everything is either invested in the risky or in the risk-free asset). In case more fine-grained decisions are required, e.g., one decision to buy one share, one decision to buy two shares, etc., already three different actions are necessary - the reason for the high number of actions in Sherstov and Stone (2004). (Note: using relative changes for the position size as actions may be a better representation - see Jin and El-Saawy 2016.) We further observe an even stronger growth of the action space when several stocks are traded. Kaur (2017), for example, already has $3^2 = 9$ actions resulting from the buy/sell/hold decisions for a portfolio of two stocks.

The above observations clearly illustrate two main limitations of the critic-only approach. First, the modeling of fine-grained decisions leads to significant growth of the action space. Second, an even stronger growth occurs when several stocks are managed simultaneously. In both cases, the large action space leads to an increase of computation time as more and more options need to be explored. Nonetheless, these limitations do not imply that the critic-only approach is not suited for larger-scale applications. In fact, in many cases these limitations can be circumvented by finding another way to formulate the problem - Jangmin et al. (2006), for example, introduce local traders that can identify an arbitrary number of trading opportunities.

Looking at the *reward functions* in panel B of table 3, we find that most studies employ purely profit-based rewards, e.g., absolute profit or average return. The second most common reward function is the Sharpe ratio. Corazza and Bertoluzzo (2014) find this metric, as well as their $OVER_t$ ratio, to yield more consistent results than a purely profit-based reward. In fact, as this observation has also been made in the actor-only approach (see, for example, Moody and Saffell 2001), the Sharpe ratio should be at least included as a benchmark when considering different reward functions. Further, we find "other" reward functions being applied in special cases. Tan et al. (2011), for example, use a variety of custom rewards to optimize specific parts of their trading system. Finally, with regard to immediate vs. terminal rewards (one reward is given at the end of the episode), we cannot draw a definitive conclusion. The majority of works employ immediate rewards (see, for example, Dempster and Romahi 2002), however, we also find works at

the other end of the spectrum. Jangmin et al. (2006), for example, reward their agent only once with the cumulative profit at the end of trading period. For the time being, we hence recommend to start with immediate rewards (to provide the agent with frequent feedback to accelerate learning), however, more research is required to derive a definitive answer.

3. Actor-only approach (recurrent reinforcement learning)

Study	Sample	Resolution	Key findings/contents
Moody and Wu (1997), Moody et al. (1998a,b), Moody and Saffell (2001)	Artificial price time series, S&P 500 index, and three-month treasury bill (1950-1994); GDB-USD currency pair (1996)	S&P 500: monthly; FX: 30-minutes-binned	**Baseline approach:** Recurrent reinforcement learning (RRL) can be successfully applied to manage a portfolio of a risky and a risk-free asset (generalization to several assets in long-only setting is outlined).
Gold (2003)	25 currency pairs (1996)	30-minutes-binned	Complexity of decision function can be increased by considering it as a neural network and adding a hidden layer; result-wise, no improvement of the trading-performance is observed.
Bertoluzzo and Corazza (2007)	Nine financial market indices (1992-2007)	Daily	Investors gain index (as reward function) can improve performance of RL trading systems; proposal of a mechanism to avoid large drawdowns.
Dempster and Leemans (2006)	EUR-USD currency pair (2000-2002)	Minute-binned	**Putting an emphasis on risk:** RRL trading systems can be embedded in a multi-layered risk management system; additional performance tweeks, such as an improved position updating scheme, are outlined.
Deng et al. (2015)	Shanghai index future (2013-2014)	Tick-level	Performance of RRL trading systems can be improved by introducing sparse-coded, task-specific feature representations.
Deng et al. (2017)	Shanghai index future, silver and sugar contract (2014-2015); S&P 500 index (1990-2015)	Futures: minute binned; S&P 500: daily	**Extraction of higher level features - deep recurrent reinforcement learning:** Deep learning techniques can be levered to extract higher level features from the state; Proposal of an extension to back propagation through time to efficiently train the agent.
Jiang and Liang (2017); Jiang et al. (2017)	12 crypto currencies (2014-2017)	30-minutes-binned	Deep RRL can be applied to simultaneously manage a multi-asset portfolio (consisting of 12 crypto currencies).

Table 4: Overview of surveyed works applying the actor-only approach.

This section is devoted to the actor-only or "recurrent reinforcement learning" approach. Table 4 summarizes the main works, their year of publication, as well as the data sample and resolution.

3.1. Summary of baseline approach and relevant extensions

3.1.1. Baseline approach

Instead of approximating a value function to compute the outcomes of different actions in every state S_t, Moody and Wu (1997) take a different approach. In particular, they learn a policy which directly maps states to actions. Point of departure of the "recurrent reinforcement learning algorithm" (Moody and Wu, 1997, p. 300 f.) forms the decision function of a single asset trading

agent:

$$A_t = A(\theta_t; A_{t-1}, I_t) \quad \text{with} \quad I_t = \{z_t, z_{t-1}, z_{t-2}, ...; y_t, y_{t-1}, y_{t-2}, ...\}. \tag{7}$$

Hereby, $A_t \in \{-1, 0, 1\}$ denotes the position (short, neutral, long) of the trading agent for the next time step, A_{t-1} its current position, and θ_t refers to the learned parameters of the agent. (Note: the subscript "$_t$" indicates that the parameters change at every time step in case the agent is trained using online learning.) Further, I_t denotes the information set used to represent the state, i.e., lagged values of the asset's price time series z_t as well as other external variables y_t. An example for a concrete decision function is given in Moody and Saffell (2001),

$$A_t = \text{sign}\left(uA_{t-1} + v_0 r_t + v_1 r_{t-1} + ... + v_m r_{t-m} + w\right), \tag{8}$$

where $r_t = z_t - z_{t-1}$ and $\theta_t = \theta = \{u, v_i, w\}$ with $i \in \{0, 1, ..., m\}$. The goal of the agent is to optimize some utility function U_t which is linked to its decision function. For the sake of simplicity, we follow the authors and make use of the additive profit utility function:

$$U_t(\theta) = P_T = \sum_{t=1}^{T} R_t = \mu \sum_{t=1}^{T} \{r_t^f + A_{t-1}(r_t - r_t^f) - \delta_t \left|A_t - A_{t-1}\right|\}. \tag{9}$$

In the above equation, P_T denotes the cumulative profit at the end of the trading period and R_t the absolute profit/loss at each of its T time steps. Further, μ denotes the position size (a fixed number of shares of stock z), r_t and r_t^f the absolute price change between period t and $t-1$ of the risky and risk-free asset respectively. Finally, δ_t denotes the transaction costs occurring when the agent switches positions. (Note: P_0, A_T and A_0 are usually set to zero.) From the above equations, we can easily see that the agent's decision/action A_t at time step t is impacted by its previous decision A_{t-1}, i.e., the agent's current position. This recurrent relationship has led to the term "recurrent reinforcement learning" and renders the optimization problem path-dependent. We therefore seek a set of parameters θ that maximize the utility function U_t over all T time steps. To solve this problem, the authors apply an online approach based on gradient ascent which optimizes the parameters θ by repeatedly computing the value of U_t (forward pass) and adjusting θ using gradient ascent. The gradient is given in the following equation - for further details see Moody et al. (1998b):

$$\frac{dU_t(\theta)}{d\theta} = \frac{dU_t}{dR_t}\left\{\frac{dR_t}{dA_t}\frac{dA_t}{d\theta} + \frac{dR_t}{dA_{t-1}}\frac{dA_{t-1}}{d\theta}\right\}. \tag{10}$$

The second novelty of the RRL approach is a "differential Sharpe ratio" utility function which is based on Sharpe's measure of risk-adjusted returns (Sharpe, 1966) but can be optimized in an online manner. In their simulation study, the authors find this utility function to yield more consistent results than a purely profit-maximizing objective - a similarity to the findings of Corazza and Bertoluzzo (2014) for the critic-only approach.

Building upon this research, Gold (2003) performs a comprehensive empirical study analyzing the effect of more complex decision functions. Point of departure is the interpretation of the decision function from equation 8 as a single layer neural network (perceptron),

$$A_t = \text{sign}\left(\sum_{i=0}^{L} v_i r_{t-i} + v_{L+1} A_{t-1} + w\right), \tag{11}$$

whereby v is the weight vector, w the activation threshold and r_i with $i \in \{0, 1, ..., L\}$ the input of the neural network. Based on this notion, Gold introduces a hidden layer, allowing the agent to learn "a more complex and in theory more powerful trading rule" (Gold, 2003, p. 363):

$$x_j = \tanh\left(\sum_{i=0}^{L} V_{i,j} r_{t-i} + V_{L+1,j} A_{t-1} + w_j\right), \tag{12}$$

$$A_t = \text{sign}\left(\sum_{j=0}^{N} v'_j x_j + w'\right). \tag{13}$$

Hereby, the assets' lagged returns r_i are first transformed in the hidden layer of the neural network (equation 12) and then further transformed at the output layer (equation 13) resulting in the final trading decision of the agent. Note: V is a weight matrix connecting each of the $L + 1$ inputs with each of the $N + 1$ hidden neurons and w is a vector containing all activation thresholds of the hidden layer. In his study, Gold varies the number of neurons of the network, the learning rate, the weight decay parameter of the learning algorithm, the number of epochs used for training, as well as the length of the training and test windows. The backtest is performed on half-hourly binned bid-ask prices of 25 different currency pairs - 10 pairs were used for tuning, 15 pairs for out-of-sample testing. (Note: it may be better to split the sets by time rather than currency pairs to reduce the risk of a look-ahead bias.) Gold makes the following observations:

- The performance of the single layer neural network is better than the performance of the neural network with two layers. Gold assumes that the latter mermorizes noise (overfitting).

- The number of lagged returns used as input has a stronger impact on the performance than the number of hidden neurons. Gold finds four lagged returns and 16 hidden neurons to achieve the best results, however, the results vary strongly across the analyzed currency pairs.
- The optimal length of the training and trading window differs strongly by currency pair.
- Learning rate and optimal number of training epochs are strongly dependent on each other with lower learning rates generally requiring more training epochs. Gold recommends the use of an intermediate learning rate and intermediate number of training epochs.
- Slight weight decay can improve the results of the two layer neural network, however, does not improve the performance of the single layer neural network.

In general, we observe that the optimal parameters vary heavily by currency pair. In fact, Gold notes that "we have found the optimal fixed parameters for currency trading in general as the possible combinations of parameters is very large and many of the parameters have a complex interdependence" (Gold, 2003, p. 369). From a practical point of view, it may hence be beneficial to determine currency pair-specific parameters. Beyond these results, the concisely written paper is important for another reason. By considering the decision function as a neural network and introducing a hidden layer, Gold (2003) can be understood as an early precursor of the deep recurrent reinforcement learning approach presented in section 3.1.3.

3.1.2. Putting an emphasis on risk

Inspired by Venturi (2003), Dempster and Leemans (2006) expand upon the baseline approach by embedding it into a three-layered FX trading system. The system aims at optimizing its trading behavior based on the investor's risk preference and to dynamically adapt to changing market conditions. In total, the proposed trading system consists of the following three layers:

- *Basic trading system:* Core of the trading system is a RRL agent resembling the one of the baseline approach. (Note: the authors propose some performance tweaks, for example, an improved position updating scheme, which are worth considering.) At every time step t, the agent proposes the position (EUR or USD) for the next time step.
- *Risk and performance management layer:* The second layer takes the actual trading decision and performs four tasks. First, it introduces a threshold value y, which the trading signal needs to exceed in order to trigger an actual trade. Second, it maintains a trailing stop-loss (x basis points above and below the maximum value of the position since its inception) to

secure profits and to limit losses. Third, it enforces a "cool-down period" to temporarily suspend trading activities whenever the position was closed out. Fourth, it fully suspends trading when the maximum drawdown exceeds z to prevent potentially even larger losses.

- *Parameter optimization layer:* The third layer aims at optimizing the overall system with respect to the investor's utility function U, i.e.:

$$\max_{\delta,\eta,\rho,x,y} U(\overline{R}, \Sigma). \tag{14}$$

Hereby, $\overline{R}$ denotes to the average returns and Σ a custom risk measure employed by the authors. In total, five different parameters are tuned, i.e., the stop-loss level x, the trading threshold y, the transaction cost parameter δ (which steers the trading agent's position switching behavior), the adaption parameter η (used to compute the differential Sharpe ratio), as well as the learning rate ρ (used during the update of the trading agent's weights). To find a solution for this optimization problem, the authors perform a "one-at-a-time random search optimization" (Dempster and Leemans, 2006, p. 9) which tunes one parameter at a time while keeping the other parameters constant.

Performance-wise, the authors find their trading system to achieve an annual profit of 26% in a two-year backtest on the USD-EUR currency pair. Beyond these results, the well-structured and comprehensively written paper is particularly interesting for two reasons. First, it lays out levers to tweak the performance (e.g., the improved position updating scheme). Second and more importantly, it illustrates how the RL trader can be embedded into a larger trading system - thereby even closer aligning it to the objective of the investor (i.e., by limiting drawdowns).

3.1.3. Extraction of higher level features - deep recurrent reinforcement learning

For most of the past decade, new research on applying actor-only reinforcement learning to the financial markets domain has been very limited. However, RRL seems to regain momentum with several new publications in 2017 alone. These new works aim at combining RRL with deep learning techniques to improve the agent's sense of the environment, i.e., by extracting higher-level features from the state variables. We hence In the following, we discuss two of these recent "deep recurrent reinforcement learning" (DRRL) approaches.

Deng et al. (2017) make one of the first attempts to combine deep learning with recurrent reinforcement learning in financial markets. In their paper, the authors consider the trading problem as an "online decision making problem involving two critical steps" (Deng et al., 2017, p. 653).

In the first step, the agent senses the environment and summarizes the current market condition in form of 20 higher level features. In the second step, the agent executes an action given this summary of the market environment. Following this idea, the authors propose a trading system consisting of two interlinked components:

- *Deep neural network (DNN) for feature learning:* The DNN component extracts higher level features from the 50 raw variables used to model the state of the environment[10]. The DNN itself comprises three hidden layers with 128 neurons each, and one output layer with 20 neurons - one for each higher level feature.

- *Recurrent neural network (RNN) for decision making:* The RNN forms the second part of the system and takes the actual trading decision. The network resembles the decision function from the baseline approach, whereby the input (state) consists of the 20 features extracted by the DNN as well as the current position of the agent, i.e., the previous output of the RNN[11].

To train both components in an integrated manner, the authors proceed in two steps. First, they perform a pre-initialization of the parameters (weights) of the two components. For the DNN, for example, they make use of auto encoders to obtain initial weights. Second, they fine-tune the overall system with "task-aware backpropagation through time" (task-aware BPTT) - a custom training algorithm aimed at avoiding vanishing gradients in the DNN structure[12]. To evaluate the performance of their trading system, the authors perform a backtest on minute by minute data of three futures markets (Chinese stock index future, silver and sugar future) as well as the S&P 500 index (daily resolution). The reported Sharpe ratios amount up to 21.1 - a value that seems very high compared to typical Sharpe ratios reported in practice. By contrast, the paper illustrates in a compelling manner, how feature extraction capabilities of deep learning can be leveraged in the context of RRL trading systems. Further research could create a better understanding of how the network configuration (e.g., number of hidden layers and number of neurons per layer) affects the results. For example, it might be worth experimenting with fewer neurons to mitigate the risk of overfittingIn this respect, future research could also analyze whether the agent can be effectively trained using observations of several stocks, i.e., to increase the total number of samples and to learn more general patterns holding true for several stocks.

Jiang et al. (2017) make another attempt at combining deep learning techniques with recurrent

[10] 45 past price changes, momentum changes over the 3 hours, 5 hours, 1 day, 3 days and 10 days.
[11] Note: this is a difference to Gold (2003) where the previous output is also transformed in the hidden layer.
[12] A detailed description of the training algorithm can be found in Deng et al. (2017).

reinforcement learning to build a trading agent for crypto currencies. In contrast to the previous section where a single asset is traded, the authors trade a portfolio of eleven crypto currencies and a cash position[13]. The objective of the agent is hence to create a sequence of portfolio weights $w_1, w_2, ..., w_T$ that maximizes the average cumulative return R of the portfolio. As in the earlier paper, deep learning is applied to extract higher level features from the state. Hereby, the authors experiment with three different neural networks, i.e., a convolutional neural network (CNN), a recurrent neural network (RNN), and a long short-term memory network (LSTM). In order to train the overall system efficiently, the authors propose an online stochastic batch learning scheme and store the portfolio compositions in a central memory. Performance-wise, the authors find their deep learning trading agents to clearly outperform their benchmarks and report 4-fold returns in a 50 days backtest. Irrespective of the financial performance, the paper is interesting for two reasons. First, it outlines another approach to combine deep learning techniques with RL trading agents. Second and more importantly, the paper is the first empirical study applying actor-only RL to manage a portfolio of several (crypto) assets. Going forward, it would be nice to see the outlined approach being applied to more liquid asset classes (e.g., the S&P 500 constituents) and longer holding periods (e.g., several days), where microstructural effects play a minor role. Furthermore, different reward functions could be compared, for example, to check whether using the Sharpe ratio also leads to better results in case several assets are traded simultaneously.

3.2. Survey of works along state, action space, and rewards

In the following subsections, we survey the cross-section of works of the actor-only approach along three dimensions, i.e., (i) the information used to model the state of the environment, (ii) the action space of the agent, and (iii) the employed reward function.

3.2.1. State

Table 5 outlines the types of information used for the state. We make the following observations:

- *Discrete states:* All surveyed RRL trading agents are operating on continuous state variables, i.e., the different variables are not discretized. This is an advantage over earlier works of the critic-only approach, where the discretization was necessary to limit computational costs.

- *Price history and candle sticks:* The majority of works incorporates the asset's recent price history into the state with the number of lagged values varying by study. As in the critic-only approach, we find only few studies making use of additional information derived from

[13] More precisely, bitcoin is referred to as "cash" in the authors' study.

Study	Discrete states	Price history	Candle sticks	Currentposition	PF value/ profit	Available cash	Technical indicators	Macro data	Order bookdata	Other	Remarks
Moody and Wu (1997), Moody et al. (1998a,b), Moody and Saffell (2001)		x		x							Up to 84 variables, i.e., lagged first differences of the asset price, current position, macroeconomic data, e.g., yield curve slope, 6 months difference in AAA bond yield, 6 months difference in TBill yield (not all variables used are outlined).
Gold (2003)		x		x							Lagged first differences of the price time series, current position. (Note: lagged first differences are z-scored prior to being fed as inputs.)
Bertoluzzo and Corazza (2007)		x		x							Lagged log returns of the portfolio, current position.
Dempster and Leemans (2006)		x		x							Lagged first differences of the price time series, current position, risk preference of the investor (as additional, external variable); experiments of the authors to feed 14 additional technical indicators did not improve the performance except when the number of lagged returns was reduced.
Deng et al. (2015)		x		x			x		x	x	More than 80 raw features comprising trend indicator, oscillator indicator (relative strength index), price changes, volume patterns (adopted on balance volume and volume weighted moving average), order book features (bid-ask spread, bid-ask volume imbalance, signed transaction volume), moving averages and standard deviation to measure momentum of the price movement. (Note: dimensionality reduction and sparse encoding are performed prior to feeding the features into the RL system.)
Deng et al. (2017)		x		x						x	Current position and 45 raw features comprising the price changes of the past 45 minutes, as well as the momentum change over the previous 3 hours, 5 hours, 1, 3, and 10 days. (Note: raw-features are pre-processed to obtain a fuzzy representation.)
Jiang and Liang (2017), Jiang et al. (2017)		x	x	x							Current positions (portfolio weights), normalized high, low and closing prices over a period of 50 time steps (each state is hence a sequence of the feature values at 50 consecutive time steps).

Table 5: Actor-only approach: Details on information used to describe the internal state of the agents by study. An "x" denotes whether a certain type of data is part of the information set. The "discrete state" column indicates the discretization of the raw features, PF value denotes the value of the portfolio.

the open, highest and lowest price. An exception is the paper of Jiang et al. (2017) which includes the high, low and close of the asset price at 50 consecutive time steps.

- *Current position, portfolio value, and profit:* All studies of the actor-only approach incorporate the agent's current position into the state. In case a portfolio is managed, see Jiang et al. (2017), the current portfolio weight of each position is included in the state.

- *Macro data, order book data and other:* As opposed to the critic-only approach, other features, such as order book and macroeconomic data, are barely used. Deng et al. (2015) have the richest state representation with more than 80 variables including technical indicators (e.g., relative strength index), volume patterns (adopted on balance volume), as well as order-book level features (e.g., bid-ask spread, bid-ask volume imbalance, and signed transaction volume).

From the surveyed works, we find the agent's state being primarily modeled with price-based features, whereby more recent research focuses on the extraction of higher level features using deep learning techniques (see, for example, Deng et al. 2017 and Jiang et al. 2017). As in the critic-only approach, these price-based features, as well as the agent's current position, can be considered as the minimum information set for modeling the state. Future research could explore the value of additional data sources such as earning estimates, fundamental data, and investor sentiment.

3.2.2. Action space and reward function

In this subsection, we survey the cross-section of works along the agent's action space and the employed reward function. From panel A ("Actions") of table 6, we make the following observations:

- *Output layer*: Following Gold (2003), the agent's decision function can be considered as a neural network receiving the state as input and returning an action as output, i.e., the position to take in the next time step. From this perspective, the action space is determined by the number of output neurons as well as their activation function. When looking at the surveyed literature, we find two different designs. The majority of works use one single output neuron with *tanh* activation function. This design is used when one single asset is traded - with the continuous output of *tanh* (a value between -1 and 1) denoting the share of capital for the long, respectively the short position[14]. The second design is applied to manage a portfolio of several assets. Hereby, the number of neurons is equal to the number of assets in the portfolio and the *softmax* activation function ensures that all portfolio weights add up to 1.

[14]For the long-only case, *tanh* can be replaced by *sigmoid*.

	Panel A: Actions			Panel B: Reward functions			
Study	**Output layer**	**Discrete actions**	**Details/remarks**	**Profit**	**Sharpe ratio**	**Other**	**Details/remarks**
Moody and Wu (1997), Moody et al. (1998a,b), Moody and Saffell (2001)	1 neuron (tanh), multiple neurons (softmax)	(x)	Tanh: share of capital to invest in a long/short position of the risky asset; Softmax: continuous portfolio weight for each asset in the portfolio. Note: "(x)" denotes, that both discretized and non-discretized actions are employed.	x	x	x	Profit, differential Sharpe ratio, differential downside deviation ratio.
Gold (2003)	1 neuron (tanh)	x	Tanh: long/short(/neutral) position in single asset (outputs are discretized with sign).		x		Differential Sharpe ratio.
Bertoluzzo and Corazza (2007)	1 neuron (tanh)	x	Tanh: long/short(/neutral) position in single asset (outputs are discretized with sign).			x	Investor's gain index (ratio between cumulative positive and negative rewards).
Dempster and Leemans (2006)	1 neuron (tanh)	x	Tanh: Long/short position of fixed amount in single asset; positions are only opened in case a certain threshold is exceeded (the threshold is determined during optimization of the trading system).			x	Custom risk measure incorporating the cumulative profit, the average profit per time interval, and the trader's personal risk aversion (external parameter).
Deng et al. (2015)	1 neuron (sigmoid)	x	Sigmoid: Long/short/neutral position of fixed amount; Sigmoid output is discretized with sign. (Note: as sigmoid only takes values between 0 and 1, it might be the case that the authors used tanh.)	x			Profit net of transaction costs. (Note: to reduce the number of position changes, transaction costs are incorporated with a quadratic penalty term.)
Deng et al. (2017)	1 neuron (tanh)	x	Tanh: long/short(/neutral) position in single asset (outputs are discretized with sign).	x	x		Total profit, moving Sharpe ratio.
Jiang and Liang (2017); Jiang et al. (2017)	12 neurons (softmax)		Softmax: continuous portfolio weights for each of the 12 assets (long only).	x			Average log return.

Table 6: Actor-only approach: Details on the modeling of the action space (panel A), as well as the employed reward function(s) (panel B). An "x" denotes whether a certain property (e.g., actions are discretized) or type of reward function (e.g., profit), is present in the respective study.

- *Discrete actions*: The majority of studies discretize the action values, i.e., the agent can either take a long position, a short position, or fully stay out of the market (neutral). In most cases, this discretization is realized by simply taking the sign of the decision function output (see, for example, Bertoluzzo and Corazza 2007 and Deng et al. 2017). Hereby, it is important to note that only the final output is discretized - in the decision function, *tanh* is still used to compute the derivatives required for gradient ascent (Gold, 2003).

With regard to the *reward function* (panel B of table 6), we find half of the studies employing purely profit-based rewards, e.g., absolute profit or average return. The second most common reward function is the differential Sharpe ratio (Moody and Wu, 1997). As in the critic-only case, some authors find this metric to yield preferable results vis-a-vis purely profit-related rewards - see, for example, Moody and Saffell (2001). Overall, the actor-only approach is particularly appealing for its continuous actions. Consequently, the approach is well suited to build trading agents that gradually adjust their positions, or that dynamically re-allocate investments among different assets. With respect to the latter setting, future research could explore how additional constraints, such as maximum allocation per individual asset or economic sector, can be efficiently incorporated.

4. Actor-critic approaches

The actor-critic approach combines the actor-only with the critic-only approach. Table 7 summarizes the main works, their year of publication, as well as the data sample and resolution.

Study	Sample	Resolution	Objective
Li et al. (2007)	NASDAQ index, S&P 500 index, IBM (1989-2004)	Daily	**Baseline approach:** Actor-critic reinforcement learning can be applied to the trading domain; reported forecasting performance is better than actor-only model and supervised learning benchmark.
Bekiros (2010)	NASDAQ index, FTSE100 index, NIKKEI255 index (1984-2009)	Daily	**Fuzzy actor-critic reinforcement learning:** Actor-critic RL techniques can be leveraged to optimize trading systems based on fuzzy inference rules.
Chan and Shelton (2001)	Simulated data	-	**Market making:** Actor-critic RL can be applied to market making.

Table 7: Overview of surveyed works applying the actor-critic approach.

4.1. Baseline approach and relevant extensions

4.1.1. Baseline approach

As shown in table 7, and despite its promise to combine the advantages of actor-only and critic-only RL (Konda and Tsitsiklis, 2000; Grondman et al., 2012), there are very few studies applying

actor-critic-based RL in financial markets. To outline the general approach, we follow Si and Wang (2001), who propose an actor-critic RL implementation using action-dependent heuristic dynamic programming (ADHDP), as well as Li et al. (2007), who employ it in the trading domain[15]. As suggested by its name, actor-critic RL systems consist of two components, the actor and the critic:

- *Actor:* The actor determines the agent's actions A_t and forms the policy of the system. At every time step t, the actor receives the current state S_t of the environment as input, and computes the agent's action A_t as output. From this perspective, the actor is similar to the agents in the actor-only approach presented in section 3.

- *Critic:* The critic has the task of evaluating the actor's actions. Hereby, the critic receives the current state S_t as well as the actor's action A_t as input. Based on this information, the critic calculates the discounted future reward for performing action A_t in state S_t, i.e., it calculates a "quality score" for the action using a learned value function. In this respect, the critic resembles the actor-only system outlined in section 2.

The key idea of the actor-critic approach is to gradually adjust the policy parameters (weights) of the actor in a way that its actions maximize the total discounted future reward predicted by the critic. Looking more closely, the optimization problem consists of two interdependent parts. First, given the actions of the actor, the prediction error made by the critic, i.e., the difference between predicted and actual discounted future reward, needs to be minimized by adjusting the critic's parameters W_c. Second, given the approximation of the value function of the critic (which is used to estimate the future rewards), the actor's policy parameters W_a need to be adjusted to maximize these total future rewards. During training, the parameters of both components are incrementally updated using gradient ascent. Hereby, the updates are performed in an alternating manner, i.e., W_c is kept constant while W_a is updated and vice versa. For a more detailed description of the training algorithm please refer to the original paper of Si and Wang (2001).

The actor-critic RL agent of Li et al. (2007) closely follows the above design. In the author's study, both actor and critic are implemented as neural networks, whereby the output of the actor-network forms part of the input vector of the critic-network. The objective of the agent is to improve the predicted price change $\hat{Z}(t+l)$ of a financial asset over the next l time steps. Hereby, the final prediction $\hat{Z}(t+l)$ is the sum of the prediction made with an Elman recurrent network

[15]Note: Similar to Q-learning, ADHDP can be applied to a wide set of problems. Also, there exist several other techniques to build actor-critic RL agents, see, for example, Grondman et al. (2012).

(Elman, 1990) $\hat{Z}_{SL}(t+l)$ and the prediction of the RL agent $\hat{Z}_{RL}(t+l)$:

$$\hat{Z}(t+l) = \hat{Z}_{SL}(t+l) + \hat{Z}_{RL}(t+l). \tag{15}$$

By splitting the overall forecast into the supervised learning (SL) and the reinforcement learning (RL) part, the authors aim at capturing the "general market inertia" using the Elman network and capturing the "synthesized investor's abnormal decision" using the RL agent (Li et al., 2007, p. 234). Result-wise, the reported directional accuracy values show a clear improvement vis-a-vis a forecast solely based on the supervised Elman network, as well as the joint forecast of the Elman network and an actor-only RL agent. The latter is an indication for the potential performance benefits of actor-critic RL agents and should be a motivation for future research.

4.1.2. Fuzzy actor-critic reinforcement learning

Bekiros (2010) pursues another attempt at building an actor-critic RL trading system using fuzzy programming techniques. Specifically, the trading decision (long/short) is the output of a first-order Sugeno fuzzy system (Sugeno, 1985), which is optimized using actor-critic RL. The overall system comprises two parts, i.e., the fuzzy system and its RL optimization:

- *Fuzzy system:* The fuzzy system receives the current state of the environment as input S_t, and returns the trading decision as output. The input is a triple consisting of the past two logarithmic one-day returns of the asset (r_t, r_{t-1}), and the asset's change in 20 days volatility compared to the previous day $(\sigma_t - \sigma_{t-1})$. From input to output, three steps are performed. First, each element of the input is fuzzyfied, i.e., the degree to which it can be classified as "high" or as "low" is calculated using a membership function of the following form:

$$\mu_{M_j^i}(S_{t,j}) = \begin{cases} 1 - \left(2|S_{t,j} - a_j^i|/b_j^i\right), & \text{if } 2|S_{t,j} - a_j^i| \leq b_j^i \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

  Hereby, i denotes the rule the membership function belongs to (see below), $S_{t,j}$ refers to the j-th element of the input vector at time step t, and a_j^i and b_j^i are the (tuning) parameters of the membership function. By defining two potential classes ("high" vs. "low") for each of the three elements of the input vector, 2^3 possible combinations emerge, i.e., ("low", "low", "low"), ("low", "low", "high"), and so forth. In the next step, a corresponding rule R^i is

defined for each of these eight combinations:

$$R^i : \text{IF } S_{t,1} \text{ is } M_1^i \text{ AND } S_{t,2} \text{ is } M_2^i \text{ AND } S_{t,3} \text{ is } M_3^i \text{ THEN } = z_t^i. \tag{17}$$

In the above equation, the left hand side is referred to as the premise, whereby $M_j^i \in \{\text{“high”}, \text{“low”}\}$ with $j \in \{1, 2, 3\}$ and $i \in \{1, 2, ..., 8\}$. The right hand side of the equation is the rule's output and calculated as follows,

$$z_t^i = h^i + c^i s_{t,1} + d^i s_{t,2} + k^i s_{t,3}, \tag{18}$$

whereby h^i, c^i, d^i, and k^i are the coefficients of the output of rule i. Finally, in the last step, the weighted output (i.e., the predicted rate of return for tomorrow) across all eight rules is computed:

$$\hat{V}_t(S_t) = \frac{\sum_{R^i \in A(S_t)} w_{R^i}(S_t) z_t^i}{\sum_{R^i \in A(S_t)} w_{R^i}(S_t)}. \tag{19}$$

Hereby, the weight factors w_{R^i} for the individual rules are the multiplicative conjunction of the membership values of the input vector elements, i.e., $w_{R^i} = \prod_{j=1}^{3} \mu_{M_j^i}(S_{t,j})$. In case the final output $\hat{V}_t(S_t)$ is positive, i.e., the asset's price is expected to rise, the resulting action is “long” - otherwise the action is “short”.

- *Actor-critic RL optimization:* The actor-critic RL optimization aims at finding optimal values for the parameters a_j^i and b_j^i of the membership functions (equation 16), as well as optimal coefficients h^i, c^i, d^i and k^i for the polynomial output rules (equation 18). Hereby, optimal refers to parameter values that minimize the reinforcement signal, i.e., the mean squared error E_t between the forecasted $\hat{V}_t(S_t)$ and the actual one-day ahead return y_t. The optimization itself is carried out in an alternating manner. First, the membership function parameters (actor part) remain fixed and the coefficients for the first-order polynomial outputs (critic part) are computed using the singular value decomposition method (Golub and Reinsch, 1971). In the second step, the reinforcement signal is computed and the membership parameters (actor part) are updated using gradient ascent. After each iteration, the reinforcement signal is computed on a validation set and the training is stopped once it rises compared to the previous iteration (a potential indication for overfitting).

In a backtest study, the author finds his “adaptive fuzzy actor-critic reinforcement learning” approach to clearly outperform a recurrent neural network, a Markov-switching model, and a buy

and hold strategy. The reported cumulative returns and Sharpe ratios are positive and reasonably high - with the caveat that the results vary strongly depending on the underlying and time frame for testing.

4.2. Survey of works along state, action space, and rewards

In the following subsections, we survey the cross-section of works along three dimensions, i.e., (i) the information used to model the state of the environment, (ii) the action space of the agent, and (iii) the employed reward function.

4.2.1. State

Study	Discrete states	Price history	Candle sticks	Currentposition	PF value/ profit	Available cash	Technical indicators	Macro data	Order bookdata	Other	Remarks
Li et al. (2007)		x								x	Normalized first order differences of the recent price history, most recent reinforcement signal.
Bekiros (2010)		x								x	Last two logarithmic one-day returns, change in 20 days volatility compared to the previous day.
Chan and Shelton (2001)	x			x					x	x	Inventory level, order imbalance, market quality measures (i.e., bid-ask spread and price continuity).

Table 8: Actor-critic approach: Details on information used to describe the internal state of the agents by study. An "x" denotes whether a certain type of data, e.g., historical prices, is part of the information set or not. The "discrete state" column indicates the discretization of the raw features to reduce the size of the state space. Further, "PF value" denotes the value of the portfolio.

From table 8, which summarizes the different types of information used to model the state, we make the following observations:

- *Discrete states:* Two out of three studies do not discretize the different variables describing the state. The same applies to the actor's action - a clear advantage over the discrete actions in the critic-only approach.
- *Price history and candle sticks:* As in the other two RL approaches, the majority of works incorporates the asset's recent price history as state variables. Worth mentioning is Bekiros (2010), who adds the change of the stock's conditional volatility as additional variable.
- *Current position, portfolio value, and profit:* None of the surveyed studies incorporates the current position, portfolio value, or profit. This omission is surprising at first sight, however

closely related to the problem definition in the respective papers (no transaction costs/static holding periods). However, in a setup with dynamic holding periods, including the current position and incorporating transaction costs are likely to improve the trading performance.

- *Macro data, order book data and other:* Two out of three studies solely rely on state variables derived from price data. Chan and Shelton (2001) is an exception as the authors use order-book-level information, such as order book imbalance, for their market making agent.

4.2.2. Action space and reward function

	Panel A: Actions	**Panel B: Reward functions**			
Study	**Remarks**	**(Cum.) profit**	**Sharpe ratio**	**Other**	**Remarks**
Li et al. (2007)	The agent's action is used to revise the price change forecast of a supervised learning model.			x	The reward at each time step compares the sign of the actual price change with the predicted sign of the price change.
Bekiros (2010)	The actions of the RL agent are the (optimized) parameters for the fuzzy system.			x	Mean squared error between predicted and actual one-day ahead return (minimization objective).
Chan and Shelton (2001)	Change of bid and change of ask price for the next time step; the values by which the current bid and ask are changed are discretized	x		x	Custom reward function combining profit and market quality, as well as profit-only reward function.

Table 9: Actor-critic approach: Details on the modeling of the action space (panel A), as well as the employed reward function(s) (panel B). An "x" denotes whether a certain type of reward function (e.g., profit), is employed in the respective study.

In this subsection, we survey the cross-section of works along the action space and the employed reward function. From panel A ("Actions") of table 9, we observe that the agent's action space varies strongly among the works and is often not directly linked to the final trading decision. In Li et al. (2007), for example, the agent's action is used to revise the price change forecast of a supervised learning model. In Bekiros (2010), the trading decision itself is made by a Sugeno fuzzy system which is optimized by the actor-critic RL agent.

In terms of *reward function*, the above picture is repeated. The surveyed papers mostly focus on improving the price change forecast rather than the resulting profit or Sharpe ratio. However, this does not imply that actor-critic RL trading systems can't be optimized using these metrics - in fact, it would be highly interesting to see future research implementing such an approach.

5. Comparison studies and further works

This section is devoted to studies comparing the different RL approaches as well as selected other works. From table 10 we make the following observations:

- *No clear winner:* As denoted by the underlined "x", which highlights the best performing agent in the respective study, the results vary across the surveyed works. Duerson et al. (2005) find the Q-learning agent (critic-only) to yield the best results, however, Du et al. (2009) report the opposite with RRL achieving superior performance. Casqueiro and Rodrigues (2006), the most cited comparison study, draws a similar conclusion: "Experimental results were not very conclusive" (Casqueiro and Rodrigues, 2006, p. 1409). In their paper, Q-learning and RRL are on par in two out of three cases, and both outperformed by a naive buy and hold strategy in an experiment on a third time series.

- *Focus on actor-only and critic-only agents:* As of today, there is no comprehensive study benchmarking all three approaches, i.e., actor-only, critic-only, and actor-critic.

Overall, these contradictory findings are likely to be driven by three factors, i.e., the high degree of noise in financial data, the immense solution space with regard to designing the RL trading system (e.g., selected variables for the state, employed reward function), and the sample sizes. Future research could hence provide valuable answers by comparing the different RL approaches in a large empirical setting, for example, by applying the agents to all S&P 500 constituents. In the meantime, a pragmatic way to not having to choose a specific approach is proposed by Gao and Chan (2000). The authors ensemble the actions of an actor-only and a critic-only agent to derive the final trading decision. Similar to supervised learning, see, for example, Dietterich (2000), the "ensemble agent" is reported to yield better results than each of the agents individually.

Finally, the works of Kearns and Nevmyvaka (2013) are worth mentioning. In their article, the authors provide three case studies on the application of RL in high frequency trading. The first case study focuses on optimizing trade execution with RL and is a summary of earlier works of the authors (see Nevmyvaka et al. (2006) and section 2.1.6). The second case study addresses the prediction of short-term price movements from the state of the order book. Finally, the third case study highlights how RL can be applied to optimize the allocation of trades to different dark pools. Overall, all three studies provide valuable insights with a focus on relevant variables for the state as well as ways to interpret the policies learned by the agents.

Study	Sample	Resolution	Approach					Study setting and result
			Critic-only	Actor-only	Actor-critic	Supervised	Other	
Duerson et al. (2005)	Five selected US technology stocks (exact time frame of data sample is not reported)	Daily	<u>x</u>	x			x	*Setting:* Comparison of four trading systems, i.e., Q-learning agent (critic-only), short-term discounted history reinforcement learner (critic-only), recurrent reinforcement learning agent (actor-only), turning point predictor (other), and one naive trading strategy (buy and hold) in single-asset setting; objective of the agent is to maximize the total return by re-allocating the investment among a risky and a risk-free asset at every time step. *Result:* Critic-only approaches yield the best results. Note: the paper indicates "a possible problem" related to the formula for the return R_t and its differential for the actor-only model (Duerson et al., 2005, p. 5) - a potential explanation for the poor performance of the actor-only model.
Casqueiro and Rodrigues (2006)	One simulated time series, Lisbon Stock Exchange PSI 20 index, EDP stock (exact time frame of data sample is not reported)	Weekly	x	x		x	x	*Setting:* Comparison of Q-Learning, recurrent reinforcement learning, Sharpe ratio maximization (Choey and Weigend, 1997), supervised learning, and two naive trading strategies (buy and hold strategy, "only invest in risk-free asset" strategy) in single-asset setting; objective of the agent is to maximize the total return by re-allocating the investment among a risky and a risk-free asset at every time step. *Result:* Q-learning (critic-only) and recurrent reinforcement learning (actor-only) show similar performance with better results (in terms of profit and Sharpe ratio) than the other methods in two out of three cases; on one of the real-world data sets, the naive buy and hold strategy yields the best results; Sharpe ratio maximization performed worst across all three time series.
Du et al. (2009)	Simulated time series; S&P 500 index (exact time frame of data sample is not reported)	1200 discrete time intervals	x	<u>x</u>				*Setting:* Comparison of Q-Learning (critic-only) with recurrent reinforcement learning (actor-only) in single-asset setting; objective of the agent is to maximize the total return by re-allocating the investment among a risky asset (market portfolio) and a risk-free asset (cash) at every time step. *Result:* Recurrent reinforcement learning yields better results than Q-learning.

Table 10: Overview of comparison studies. An "x" in the five columns underneath "Approach" denotes that the respective approach was included in the study. In case the "<u>x</u>" is underlined, the authors find the respective approach to yield the best performance. In case none of the "x" is underlined in a row, the reported results are inconclusive.

6. Conclusion

The present paper surveys the literature at the intersection of RL and financial markets. Drawing insights from almost 50 publications, three different RL approaches are distinguished - each with its individual strengths and weaknesses:

Critic-only approach: The critic-only approach is the most popular RL approach in financial markets. The core of the method is a learned value function which allows the agent to compare different actions. During the decision making process, the agent observes the current state of the environment and chooses the action with the best outcome according to the value function. The main advantage of the critic-only approach is the high flexibility of the reward function and its applicability to a wide set of problems. Specifically, the reward function does not need to be differentiable. This property allows to model even complex reward schemes with several *if-else* branches, for example, to send a large negative reinforcement signal, in case a certain drawdown is exceeded. Moreover, due to the explicit use of a discount factor, the preference between immediate and future reward can be carefully controlled. Probably the most noticeable limitation is the agent's discrete action space closely related to Bellman's "curse of dimensionality" (Bellman, 1957). This becomes particularly obvious when N assets are traded or when very fine-grained trading decisions are required. In fact, "the number of discrete actions that must be considered is exponential in N" (Moody and Saffell, 2001, p. 888), rendering the approach unfeasible for large numbers of stocks N. (Note: by formulating the decision problem differently, the exponential increase in actions can be mitigated to some extend - see section 2.1.7). Finally, some authors find the approach to be "brittle" in presence of noise (Brown, 2000) or not to converge under certain conditions (Baird and Moore, 1999). Going forward, more research could be devoted to enriching the state with other data sources (see, for example, Kaur 2017 who made an attempt using sentiment data), to finding a better problem representation when trading multiple securities, and to the application of DQL techniques. Moreover, the question whether immediate or terminal rewards (one reward at the end) perform better and under what circumstances, should be further explored.

Actor-only approach (RRL): The actor-only approach is the second most popular approach. Instead of approximating a value function, the policy, i.e., the mapping from states to actions, is learned directly. In financial markets, the approach has been introduced by Moody and Wu (1997). The main advantages are the continuous action space, as well as the usually faster convergence and higher transparency (Moody and Saffell, 2001; Bekiros, 2010). Having continuous actions, the agent can carefully interact with the environment, for example, to gradually increase an investment. Moreover, by using multiple output neurons in combination with a softmax activation function, a

portfolio consisting of several assets can be managed simultaneously. The most noticeable disadvantage of the actor-only approach is the need for a differentiable reward function, limiting the reward schemes that can be modeled. Going forward, more research could be devoted to better understanding the impact of different network architectures (e.g., type and number of hidden layers) for deep RRL agents and the effect of different reward functions for multi-security portfolios. Furthermore, the value of non price-based information, e.g., sentiment data, could be explored.

Actor-critic approach: Actor-critic RL constitutes the third approach and aims at combining the advantages of actor-only and critic-only RL (Konda and Tsitsiklis, 2000; Grondman et al., 2012). As suggested by its name, actor-critic RL comprises two agents, the actor and the critic. The actor determines the actions and forms the policy of the system. At every time step, the actor receives the current state as input, and computes the agent's action as output. The critic evaluates these actions. Hereby, it receives the current state as well as the actor's action as input, and computes the discounted future reward as output. The key idea is to gradually adjust the policy parameters of the actor in a way that it maximizes the reward predicted by the critic. Despite the ambition to combine the advantages of both agents, there are only few studies employing actor-critic RL in financial markets. These works include Li et al. (2007), who develop an RL agent to improve the forecast of stock returns obtained by an Elman network, as well as Bekiros (2010), who combine actor-critic RL with fuzzy logic. Unfortunately, neither of these agents is comparable to the critic-only and actor-only agents discussed above. Future research could hence develop an actor-critic agent whose action resemble the trading decisions. Based on that, it should be analyzed whether the ambition of combining the advantages of actor-only and critic-only RL can be realized.

Looking at the three approaches and given the absence of a large-scale comparison study, the actor-only approach currently appears to be the best suited approach for financial markets. The main reasons (see also Moody and Saffell 2001) are its continuous actions (to carefully control the investment), its usually small number of parameters (which make it less prone to overfitting), its good convergence behavior (which results in faster training), and its recent improvements with deep learning techniques. Going forward and following the call of Britz (2018), it would be nice to see more attention being devoted to RL in finance - in particular as the findings can be valuable for RL research in general: Trading agents have to make decision based on imperfect information while interacting with a large number of market participants - both similarities to complex multiagent environments. Market conditions face constant change with history sometimes repeating itself - agents hence should adapt without forgetting what they have already learned. Finally, due to the good availability of data, agents can be easily tested or even deployed in a live environment.

Bibliography

Almgren, R., Chriss, N., 2001. Optimal execution of portfolio transactions. Journal of Risk 3, 5–40.

Ariel, R. A., 1987. A monthly effect in stock returns. Journal of Financial Economics 18 (1), 161–174.

Atsalakis, G. S., Valavanis, K. P., 2009. Surveying stock market forecasting techniques – Part II: Soft computing methods. Expert Systems with Applications 36 (3), 5932–5941.

Baird, L. C., Moore, A. W., 1999. Gradient descent for general reinforcement learning. In: Proceedings of the international conference on Advances in Neural Information Processing Systems. pp. 968–974.

Bekiros, S. D., 2010. Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach. Journal of Economic Dynamics and Control 34 (6), 1153–1170.

Bellman, R., 1957. A Markovian decision process. Journal of Mathematics and Mechanics, 679–684.

Bertoluzzo, F., Corazza, M., 2007. Making financial trading by recurrent reinforcement learning. In: Proceedings of the international conference on Knowledge-Based and Intelligent Information and Engineering Systems. Springer, Berlin, Heidelberg, pp. 619–626.

Bertoluzzo, F., Corazza, M., 2012. Testing different reinforcement learning configurations for financial trading: Introduction and applications. Procedia Economics and Finance 3, 68–77.

Bertsimas, D., Lo, A. W., 1998. Optimal control of execution costs. Journal of Financial Markets 1 (1), 1–50.

Booth, A., Gerding, E., Mcgroarty, F., 2014. Automated trading with performance weighted random forests and seasonality. Expert Systems with Applications 41 (8), 3651–3661.

Britz, D., 2018. Introduction to learning to trade with reinforcement learning.
URL http://http://www.wildml.com/2018/02/introduction-to-learning-to-trade-with-reinforcement-learning/

Brown, T., 2000. Policy vs. value function learning with variable discount factors. In: Proceedings of the NIPS workshop on reinforcement learning: Learn the policy or learn the value function.

Casqueiro, P. X., Rodrigues, A. J., 2006. Neuro-dynamic trading methods. European Journal of Operational Research 175 (3), 1400–1412.

Chan, N. T., Shelton, C., 2001. An electronic market-maker. Technical report, MIT, Cambridge, MA.

Chen, Y., Mabu, S., Hirasawa, K., Hu, J., 2007. Trading rules on stock markets using Genetic Network Programming with Sarsa learning. In: Proceedings of the 9th annual conference on Genetic and Evolutionary Computation. ACM, pp. 1503–1503.

Choey, M., Weigend, A. S., 1997. Nonlinear trading models through Sharpe ratio maximization. International Journal of Neural Systems 8 (4), 417–431.

Coggins, R., Blazejewski, A., Aitken, M., 2003. Optimal trade execution of equities in a limit order market. In: Proceedings of the conference on Computational Intelligence for Financial Engineering. IEEE, pp. 371–378.

Corazza, M., Bertoluzzo, F., 2014. Q-learning-based financial trading systems with applications. Social Science Research Network Working Paper Series, University Ca' Foscari of Venice.

Cumming, J., Alrajeh, D., Dickens, L., 2015. An investigation into the use of reinforcement learning techniques within the algorithmic trading domain. Master's thesis, Imperial College London.

Dempster, M. A., Leemans, V., 2006. An automated FX trading system using adaptive reinforcement learning. Expert Systems with Applications 30 (3), 543–552.

Dempster, M. A., Payne, T. W., Romahi, Y., Thompson, G. W., 2001. Computational learning techniques for intraday FX trading using popular technical indicators. IEEE Transactions on Neural Networks 12 (4), 744–754.

Dempster, M. A. H., Romahi, Y. S., 2002. Intraday FX trading: An evolutionary reinforcement learning approach. In: Proceedings of the international conference on Intelligent Data Engineering and Automated Learning. Springer, Berlin, Heidelberg, pp. 347–358.

Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q., 2017. Deep direct reinforcement learning for financial signal representation and trading. IEEE Transactions on Neural Networks and Learning Systems 28 (3), 653–664.

Deng, Y., Kong, Y., Bao, F., Dai, Q., 2015. Sparse coding-inspired optimal trading system for HFT industry. IEEE Transactions on Industrial Informatics 11 (2), 467–475.

Dietterich, T. G., 2000. Ensemble methods in machine learning. In: Proceedings of the international workshop on Multiple Classifier Systems. Springer, Berlin, Heidelberg, pp. 1–15.

Du, X., Zhai, J., Lv, K., 2009. Algorithm trading using Q-learning and recurrent reinforcement learning. Working paper, Stanford University.

Duerson, S., Khan, F., Kovalev, V., Malik, A. H., 2005. Reinforcement learning in online stock trading systems. Working paper, Georgia Tech University.

Eilers, D., Dunis, C. L., von Mettenheim, H.-J., Breitner, M. H., 2014. Intelligent trading of seasonal effects: A decision support algorithm based on reinforcement learning. Decision Support Systems 64, 100–108.

El-Yaniv, R., Fiat, A., Karp, R. M., Turpin, G., 2001. Optimal search and one-way trading online algorithms. Algorithmica 30 (1), 101–139.

Elman, J. L., 1990. Finding structure in time. Cognitive Science 14 (2), 179–211.

French, K. R., 1980. Stock returns and the weekend effect. Journal of Financial Economics 8 (1), 55–69.

Gao, X., Chan, L., 2000. An algorithm for trading and portfolio management using Q-learning and Sharpe ratio maximization. In: Proceedings of the international conference on Neural Information Processing. pp. 832–837.

Gold, C., 2003. FX trading via recurrent reinforcement learning. In: Proceedings of the international conference on Computational Intelligence for Financial Engineering. IEEE, pp. 363–370.

Golub, G. H., Reinsch, C., 1971. Singular value decomposition and least squares solutions. In: Linear Algebra. Springer, Berlin, Heidelberg, pp. 134–151.

Grondman, I., Busoniu, L., Lopes, G. A., Babuska, R., 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews) 42 (6), 1291–1307.

Gu, Y., Mabu, S., Yang, Y., Li, J., Hirasawa, K., 2011. Trading rules on stock markets using Genetic Network Programming-Sarsa learning with plural subroutines. In: Proceedings of the SICE annual conference. IEEE, pp. 143–148.

Hirasawa, K., Okubo, M., Katagiri, H., Hu, J., Murata, J., 2001. Comparison between Genetic Network Programming (GNP) and Genetic Programming (GP). In: Proceedings of the congress on Evolutionary Computation. Vol. 2. IEEE, pp. 1276–1282.

Huang, W., Nakamori, Y., Wang, S.-Y., 2005. Forecasting stock market movement direction with support vector machine. Computers & Operations Research 32 (10), 2513–2522.

Jangmin, O., Lee, J., Lee, J. W., Zhang, B.-T., 2006. Adaptive stock trading with dynamic asset allocation using reinforcement learning. Information Sciences 176 (15), 2121–2147.

Jiang, Z., Liang, J., 2017. Cryptocurrency portfolio management with deep reinforcement learning. arXiv preprint arXiv:1612.01277v5.

Jiang, Z., Xu, D., Liang, J., 2017. A deep reinforcement learning framework for the financial portfolio management problem. arXiv preprint arXiv:1706.10059.

Jin, O., El-Saawy, H., 2016. Portfolio management using reinforcement learning. Working paper, Stanford University.

Kakade, S. M., Kearns, M., Mansour, Y., Ortiz, L. E., 2004. Competitive algorithms for VWAP and limit order trading. In: Proceedings of the 5th ACM conference on Electronic Commerce. ACM, pp. 189–198.

Kaur, S., 2017. Algorithmic trading using reinforcement learning augmented with hidden Markov model. Working paper, Stanford University.

Kearns, M., Nevmyvaka, Y., 2013. Machine learning for market microstructure and high frequency trading. High Frequency Trading: New Realities for Traders, Markets, and Regulators.

Kim, K.-J., 2003. Financial time series forecasting using support vector machines. Neurocomputing 55 (1-2), 307–319.

Kitchin, J., 1923. Cycles and trends in economic factors. The Review of Economic Statistics, 10–16.

Konda, V. R., Tsitsiklis, J. N., 2000. Actor-critic algorithms. In: Proceedings of the conference on Advances in Neural Information Processing Systems. pp. 1008–1014.

Krauss, C., Do, X. A., Huck, N., 2017. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. European Journal of Operational Research 259 (2), 689–702.

Kumar, M., Thenmozhi, M., 2006. Forecasting stock index movement: A comparison of support vector machines and random forest. Indian Institute of Capital Markets 9th Capital Markets Conference Paper. Available at SSRN: https://ssrn.com/abstract=876544.

Lee, J. W., Jangmin, O., 2002. A multi-agent Q-learning framework for optimizing stock trading systems. In: Proceedings of the international conference on Database and Expert Systems Applications. Springer, Berlin, Heidelberg, pp. 153–162.

Lee, J. W., Kim, S.-D., Lee, J., Chae, J., 2003. An intelligent stock trading system based on reinforcement learning. IEICE Transactions on Information and Systems 86 (2), 296–305.

Lee, J. W., Park, J., Jangmin, O., Lee, J., Hong, E., 2007. A Multiagent Approach to Q-Learning for Daily Stock Trading. IEEE Transactions on Systems, Man, and Cybernetics, part A (systems and humans) 37 (6), 864–877.

Lee, J. W., Zhang, B.-T., et al., 2002. Stock trading system using reinforcement learning with cooperative agents. In: Proceedings of the 19th international conference on Machine Learning. Morgan Kaufmann Publishers Inc., pp. 451–458.

Li, H., Dagli, C. H., Enke, D., 2007. Short-term stock market timing prediction under reinforcement learning schemes. In: Proceedings of the IEEE international symposium on Approximate Dynamic Programming and Reinforcement Learning. IEEE, pp. 233–240.

Lin, L.-J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning 8 (3-4), 293–321.

Lucca, D. O., Moench, E., 2015. The pre-FOMC announcement drift. The Journal of Finance 70 (1), 329–371.

Moody, J., Saffell, M., 2001. Learning to trade via direct reinforcement. IEEE Transactions on Neural Networks 12 (4), 875–889.

Moody, J., Saffell, M., Liao, Y., Wu, L., 1998a. Reinforcement learning for trading systems and portfolios: Immediate vs future rewards. In: Decision Technologies for Computational Finance. Springer, Berlin, Heidelberg, pp. 129–140.

Moody, J., Wu, L., 1997. Optimization of trading systems and portfolios. In: Proceedings of the conference on Computational Intelligence for Financial Engineering (CIFEr). IEEE, pp. 300–307.

Moody, J., Wu, L., Liao, Y., Saffell, M., 1998b. Performance functions and reinforcement learning for trading systems and portfolios. Journal of Forecasting 17 (56), 441–470.

Moritz, B., Zimmermann, T., 2014. Deep conditional portfolio sorts: The relation between past and future stock returns. Working paper, LMU Munich and Harvard University.

Neuneier, R., 1996. Optimal asset allocation using adaptive dynamic programming. In: Advances in Neural Information Processing Systems. pp. 952–958.

Neuneier, R., 1998. Enhancing Q-learning for optimal asset allocation. In: Advances in Neural Information Processing Systems. pp. 936–942.

Nevmyvaka, Y., Feng, Y., Kearns, M., 2006. Reinforcement learning for optimized trade execution. In: Proceedings of the 23rd international conference on Machine Learning. ACM, pp. 673–680.

Noonan, L., 2017. JPMorgan develops robot to execute trades.
URL https://www.ft.com/content/16b8ffb6-7161-11e7-aca6-c6bd07df1a3c

Ormoneit, D., Sen, Ś., 2002. Kernel-based reinforcement learning. Machine Learning 49 (2-3), 161–178.

Plummer, T., 2009. Forecasting financial markets: the psychology of successful investing. Kogan Page Publishers, London, Philadelphia.

Potvin, J.-Y., Soriano, P., Vallée, M., 2004. Generating trading rules on the stock markets with Genetic Programming. Computers & Operations Research 31 (7), 1033–1047.

Rummery, G. A., Niranjan, M., 1994. Online Q-learning using connectionist systems. University of Cambridge, Department of Engineering, Cambridge, England.

Sarlan, H., 2001. Cyclical aspects of business cycle turning points. International Journal of Forecasting 17 (3), 369–382.

Sharpe, W. F., 1966. Mutual fund performance. The Journal of Business 39 (1), 119–138.

Sherstov, A. A., Stone, P., 2004. Three automated stock-trading agents: A comparative study. In: Proceeedings of the international workshop on Agent-Mediated Electronic Commerce. Springer, Berlin, Heidelberg, pp. 173–187.

Si, J., Wang, Y.-T., 2001. Online learning control by association and reinforcement. IEEE Transactions on Neural Networks 12 (2), 264–276.

Silver, D., 2015. Lecture 6: Value Function Approximation. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf.

Sugeno, M., 1985. Industrial applications of fuzzy control. Elsevier Science, New York.

Sutton, R. S., Barto, A. G., 1998. Reinforcement learning: An introduction. Vol. 1. MIT press, Cambridge.

Takeuchi, L., Lee, Y.-Y., 2013. Applying deep learning to enhance momentum trading strategies in stocks. Working paper, Stanford University.

Tan, Z., Quek, C., Cheng, P. Y., 2011. Stock trading with cycles: A financial application of ANFIS and reinforcement learning. Expert Systems with Applications 38 (5), 4741–4755.

Teixeira, L. A., De Oliveira, A. L. I., 2010. A method for automatic stock trading combining technical analysis and nearest neighbor classification. Expert Systems with Applications 37 (10), 6885–6890.

Terekhova, M., 2017. JPMorgan takes AI use to the next level.
URL https://www.businessinsider.de/jpmorgan-takes-ai-use-to-the-next-level-2017-8?r=US&IR=T

Tesauro, G., Bredin, J. L., 2002. Strategic sequential bidding in auctions using dynamic programming. In: Proceedings of the first international joint conference on Autonomous Agents and Multiagent Systems: Part 2. ACM, pp. 591–598.

Venturi, S., 2003. Evolutionary algorithms for currency trading. Ph.D. thesis, Centre for Financial Research, Judge Institute of Management, University of Cambridge.

Watkins, C. J. C. H., 1989. Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge.

Watts, S., 2015. Hedging basis risk using reinforcement learning. Working Paper, University of Oxford.