Contracts for Difference: A Reinforcement Learning Approach

Nico Zengeler
Computer Science Institute
Hochschule Ruhr West
Bottrop, Germany
nico.zengeler@hs-ruhrwest.de

Uwe Handmann
Computer Science Institute
Hochschule Ruhr West
Bottrop, Germany
uwe.handmann@hs-ruhrwest.de

***Abstract*—We present a deep reinforcement learning (RL) approach for an automatic trading of contracts for difference (CfD) on indices. Our contribution proves that profitable automatic trading is possible and, if applied in that or a similar way, may lead to resistance lines as observed in derivatives market. As the noisy nature of economic trends complicates predictions, especially in speculative domains like CfD, we do not predict courses but instead train a RL trading agent to learn an overall lucrative policy. Therefore, we simulate a virtual CfD market environment, based on historical trading data. This environment provides a partially observable Markov decision process (POMDP) to reinforcement learners.**

***Index Terms*—Contract for Difference, CfD, Reinforcement Learning, RL, Neural Networks, Long Short-Term Memory, LSTM, Q-learning, Deep Learning**

I. Introduction

Artificial intelligence applications offer new perspectives, possibilities and tools for economic modelling and reasoning [1]. These tools can help employees in the financial industry to assess risks. Especially in speculative business, a statistically appropriate, automated handling of risk can help to avoid large economic losses. Therefore we simulate a derivative market as a partially observable Markov decision process (POMDP) for Reinforcement Learning.
To determine a reward for the agents action, the environment evaluates the trade action on historical market data and returns the financial profit or loss. The agent then tries to find an optimal policy that maximizes the expected rewards. To approximate an optimal policy, we use deep neural networks and evaluate both a feedforward neural network and a recurrent long short-term memory network (LSTM). As a reinforcement learning method, we propose a Q-learning approach with prioritised experience replay. To evaluate the real world applicability of our approach, we also perform a test under real market conditions.
We begin this paper by introducing Contracts for Differences and presenting relevant state-of-the-art research in section II. In section III, we explain our implementation in detail and in section IV we illuminate our evaluation process and the results we obtained. The conclusion in section VI ends this paper with a summary, a short discussion and possible future work.

A. Contracts for Difference

A contract for Difference (CfD), a form of a total return swap contract, allows two parties to exchange the performance and income of an underlying asset for interest payments. In other words, economic players may bet on rising or falling prices and profit, if the real price development matches their bet. Due to the possibility of highly leveraged bets, high wins may occur as well as high losses.
In contrast to other derivatives, such as knockout certificates, warrants or forward transactions, a CfD allows the independent setting of stop-loss and take-profit values. Setting a take-profit and stop-loss value automatically closes the deal, if the underlying course strikes the corresponding threshold. If the asset development does not correspond to the bet, but develops in the opposite direction, a depth arises which can lead to additional financing obligations. A security deposit, also referred to as a margin, fulfils the purpose of hedging the transaction. Since additional funding obligations in the event of default can easily exceed the margin, individual traders can suffer very high losses in a very short time if they have not set a stop-loss value.
Concerning legal aspects, CfD trading currently faces an embargo in the United States of America. According to a general ruling of the Federal Financial Supervisory Authority (Bundesanstalt fr Finanzdienstleistungsaufsicht), a broker in Germany may only offer such speculative options to his customers, if they have no additional liability in case of default, but instead only lose their security deposit.

II. State of the art

Using only historical trade data, [4] investigates a LSTM-based course prediction on the Chinese stock market with accuracies between 64.3% and 77.2%, operating on daily time scales. State-of-the-art stock market prediction usually incorporates external textual information [3, 10, 5]. A deep learning LSTM implementation by [2] learns to predict stock prices based on news text together with pricing information. Concerning stock market prices, [8] proposed a reinforcement learning system to optimize financial objective functions on the basis of certain indicators. Instead of striving for good predictions on stock market prices, we propose a deep reinforcement learning approach. This allows an agent to learn

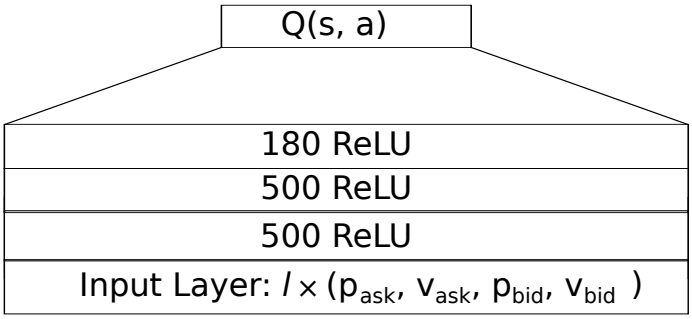

Fig. 1: Our feedforward architecture.

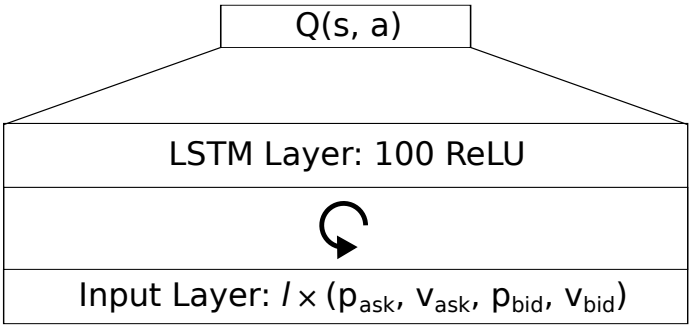

Fig. 2: Our LSTM architecture.

to act on arbitrary time scales on a speculative CfD market. To the best of our knowledge, no deep reinforcement learning appliances exist for CfD trading.

III. Method

We aim to find optimal trading policies in adjustable environment setups, using Q-learning, as proposed by [11]. To boost the training efficiency, we employ a prioritized experience replay memory for all our models [9]. We use AdaGrad updates as a weight update rule [6]. As for an observed state s, the underlying POMDP presents a tick chart of length l. The tick chart consists of a sequence of ask and bid prices, with the corresponding ask and bid trade volumes. We denote the price values per tick as p_{ask}, p_{bid} and the trade volumes as v_{ask}, v_{bid}.

A. Models

We investigate both a feedforward and a LSTM architecture. Both architectures feature the same input and output layers setup, but have different hidden layers. The input layer contains the state s, in form of a tick data sequence with length l. As for the output layer, the neural network approximates the Q-values $Q(s, a)$ for each action in the action space $a \in A$. To approximate these values, we use an output layer of $|A|$ neurons, each neuron with linear activation. Each action $a \in A$ may invoke a different trade order.

1) Feedforward: Our feedforward neural network features a hidden part of three dense layers, as sketched in figure 1. The first two dense layers consist of 500 rectifying linear units with a small bias of 0.1. We use the He weight initialisation with a uniform distribution to initialise the weights. To obtain a roughly the same number of weights as we have in our LSTM architecture, we append a third layer with 180 rectifying linear units, also with a bias of 0.1. For an input length of $l = 500$ and an action space size of $|A| = 3$, the feedforward network has a total of $840,540$ parameters.

2) LSTM: We use a LSTM network with forget gates as proposed by [7]. The architecture consists of a single recurrent layer with 100 rectifying linear units, as shown in figure 2. We initialise the gates weights using the default normal distribution. For a fixed input length of $l = 500$ and an action space size of $|A| = 3$, the LSTM network has a total of $840,300$ parameters.

B. Environment

We implement a simple market logic that operates on historical trading data on a tick time scale as a basis for a POMDP. The environment processes the agents trading actions without a delay, which simplifies analytical investigations but discards the important factor of latency. To increases the information content presented to the agent in each observation, we remove equal successive ticks. This results in a reduced input sequence length but discards the information about how long a particular state lasts.
As a state s, the environment presents a sequence of l unique ticks, starting from a random point t in trade history x. We adjust each state for the mean value:

$$s = x[t : t + l] - \bar{x}[t : t + l]$$

For each state s, the agent chooses an action a. If the agent chooses the action $a = 0$ to open no trade, the agent receives a reward of 0 and observes the next state. An episode in this environment terminates if the agent chooses to open a deal with an action $a \neq 0$. When the agent performs an action, the simulation runs forward until the market price reaches either the take-profit or the stop-loss value. The environment then returns the achieved financial profit or loss as a reward, scaled by a constant factor. Algorithm 1 in the appendix shows the simulated CfD market logic and the corresponding Q-learning procedure in pseudo code.

IV. Evaluation

To evaluate our approach, we use a DE30 CfD index with a nominal value of €25 per lot at a 5% leverage. We reflect the boundary conditions of the chosen asset in our simulation by setting an adequate reward scaling factor. A small trade volume of 0.01 lot leads to a reward scaling factor of $c = 0.25$ for the training and testing procedure.
As a data basis for our market simulation, we recorded about a million unique trade ticks as a data basis for the training environment and about half a million unique ticks for our testing procedure. In this evaluation, we use the models as described in III with an action space of size $|A| = 3$. The action $a = 0$ does not cause a trade order but makes the agent wait and observe the next tick. To open a long position, the agent would choose $a = 1$, while the action $a = 2$ would

cause the opening of a short position.

To find good training parameters for our models, we have conducted a grid search in a space of batch size, learning rate and input sequence length. We evaluate batch sizes $b \in \{10, 50, 100\}$, learning rates $\eta \in \{10^{-4}, 10^{-5}, 10^{-6}\}$ and input sequence lengths $l \in \{50, 100, 250\}$. By comparing the final equities after $1,000$ test trades we find an optimal parameter configuration. For the feedforward architecture, we find the optimal training parameter configuration in $(b = 100, l = 50, \eta = 10^{-5})$. Concerning the single layer LSTM network, we find the best test result for in $(b = 10, l = 50, \eta = 10^{-4})$.

A. Training

For each memory record, we have a starting state s_1, the chosen action a, the follow-up state s_2 alongside with the achieved reward r and a variable e. The variable e tells us if the replayed experience features a closed trade, thereby ending in a terminal state. In a training run, the agent performs a total of $250,000$ learning steps. For each learning step, we sample a batch of b independent experiences (s_1, a, s_2, r, e) from the prioritised replay memory. Then, we apply an AdaGrad weight update to the neural network, based on the difference between the predicted and the actual Q-values.

B. Test

To evaluate our models, we perform tests on unseen market data. If for an optimal action a the expected reward $Q(s, a) < 0$, the agent does not execute the order, as we want our agent to achieve a profit and not a minimal loss. This increases the time between trades at the benefit of more likely success. We test each feedforward and LSTM network by performing a total of $1,000$ test trades on unseen data. Each test run starts with an equity of €1000.

From the action distribution in figure 3, we can see that both the feedforward and the LSTM agent tend to open short positions more frequently. To perform a trade, the feedforward network observes for 2429 ticks on average, while the LSTM network waits for $4,654$ tick observations before commiting to any trade action. While the LSTM network tends to wait and observe, by choosing the action $a = 0$ more frequently, the feedforward network makes decisions faster. We can see an increase in equity for both our models in figure 3. Furthermore, the LSTM network seems to have a conceptual advantage due to its immanent handling of sequences. Looking at the differences in the profit distribution, as shown in figure 3, we find that the LSTM network achieves more low profits.

V. Real world application

For a real world proof of concept, we use the API and login credentials as provided by X Open Hub. We let our real world test run for ten trading days, in which the agent opened and closed 16 trades without manual interference. All of the test trades resulted in a positive return value, so we can't make a statement about the inherent risk of a defaulting position. Figure 6 shows the individual profits achieved by the agent

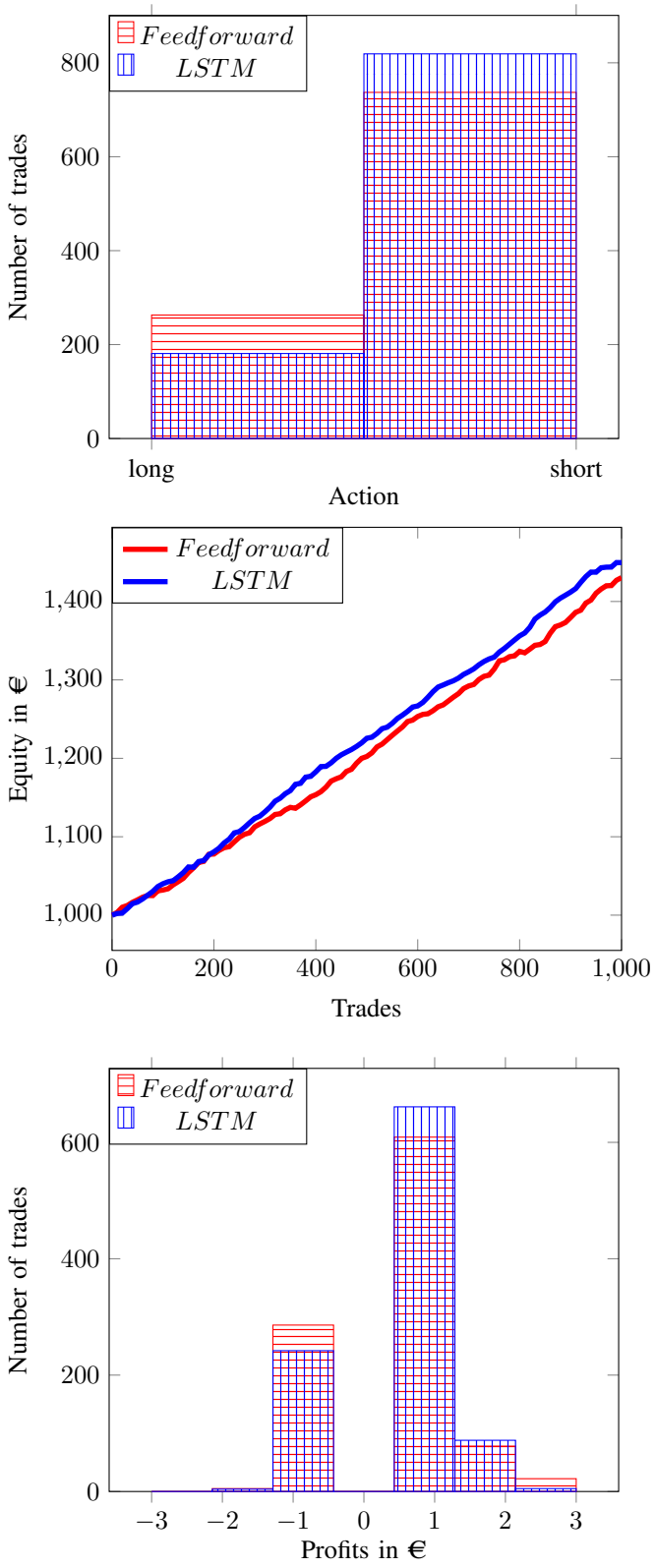

Fig. 3: Test results. Top: action distribution. Mid: equity development. Bottom: profit distribution.

and the corresponding increase in equity.
When operating with a single open trade per time, as originally intended in our learning process, we observed the effects of latency in the real world, such that the observed state actually contains past information. This causes the agent to decide an action upon old values, lowering the temporal precision. Also, the agents orders come to pass late, such that the state has already changed and the agent misses its intended position to set the take-profit and stop-loss values. To accommodate for the latency problems, we increase the action space size to $A = |10|$ and introduce a function $d_{profit}(a)$ to map a to a certain delta. This delta creates a distance for the stop-loss and take-profit values from the opening price, for example:

$$d_{profit}(a) := \begin{cases} 0, & : a = 0 \\ 2, & : a = 1, a = 6 \\ 5, & : a = 2, a = 7 \\ 10, & : a = 3, a = 8 \\ 25, & : a = 4, a = 9 \\ 50, & : a = 5, a = 10 \end{cases}$$

This workaround introduced some slack into the strategy to accomodate for the various problems introduced by latency. Like $d_{profit}(a)$, a function $d_{loss}(a)$ returns a stop-loss delta. Setting $d_{loss}(a) = d_{profit}(a)$ means to set a stop-loss value as far away from the opening price as the take-profit value. This may, for example, lead to an optimal high frequency trading strategy at chance-risk-equality. Conceptually, these adjustable delta values allow the agent to anticipate different magnitudes of price changes. Also, a user may influence the automatic trading system during runtime by changing the parameters of $d_{profit}(a)$ and $d_{loss}(a)$.
In our real world example, set the $d_{loss}(a) = 20.0 \cdot d_{profit}(a)$. This decreases the risk of immediate default, but potentially leads to a high loss. Also we superimpose a rule-based hedging system, which allows the agent to open one long position, one short position and a third arbitrary position contemporaneous. To make full use of the margin provided by the demo account, we have increased the trading volume from 0.01 to 0.33.
We have designed an LSTM architecture with an additional layer of 250 LSTM units, as shown in figure 4. Using these settings, we have trained the LSTM network with a training parameter configuration of $(b = 50, l = 250, \eta = 10^{-5})$. In the corresponding learning dynamics, as shown figure 5, we can see that the agents maintain potential high wins while trying to reduce losses, which result in an overall surplus during training. To keep the agent up to date with the real world market data, we have trained the network out of trading time. Each weekend, a training run with $250,000$ learning steps optimized the network from scratch, using the last weeks data plus the past data from previous weeks.

VI. Conclusion

We have contributed a parametrisable training environment that lets reinforcement learning agents learn different strategies for CfD trading. Our neural network implementations serve as a proof of concept for artificially intelligent trading automata

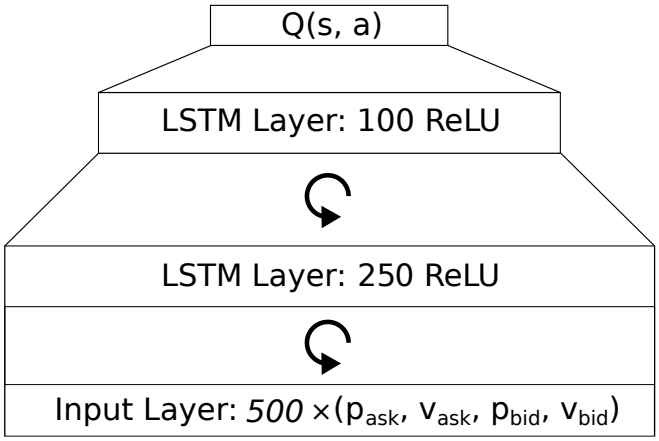

Fig. 4: A more sophisticated LSTM architecture for a real world application.

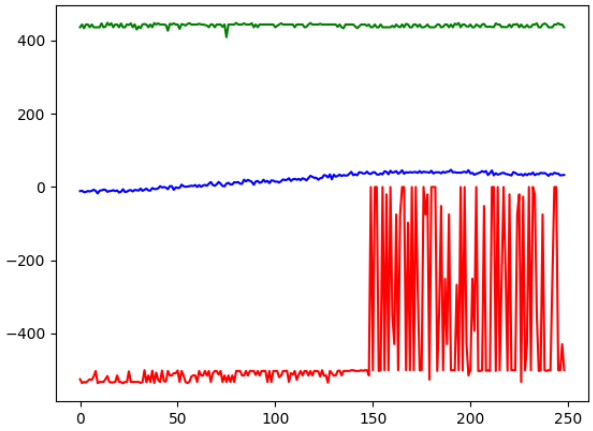

Fig. 5: The learning dynamics of our real world LSTM example.

that operate on tick time scales. As our approach conceptually allows to learn trading strategies of arbitrary time scales, a user may provide minutely, hourly or even daily closing prices as a data basis for training as well. Considering high frequency trading strategies, we state that feedforward networks suit best due to their low inference and training time at very promising test results. During our investigations of real world applicability, we found the order latency and the available equity as the main factors of a profitable and secure automated high frequency trading approach. Comparing the results of the recurrent LSTM network to the simple feedforward network, we state that assuming sequences in this kind of trading data may slightly improve results. Another key observation of our investigation reveals the importance of using a training setup that matches the real trading conditions. Furthermore we observed that if we perform trades without a stop-loss value, the CfD remains open and matches the observation of resistance lines as found in derivatives market.

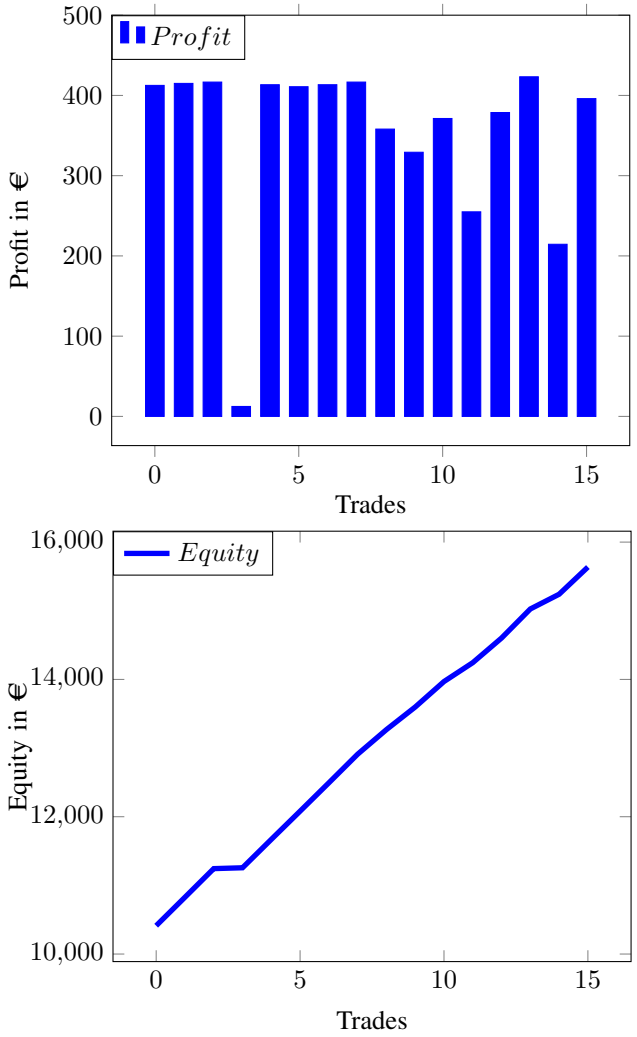

Fig. 6: Top: The profits achieved in our real world example. Bottom: The equity development of our demo account.

A. Discussion

Although our contribution proves that artificially intelligent trading automata may learn different strategies if given an appropriate training environment, we did not fully test the effects of changing the training environment parameters. We did not investigate the effect of adjusting the chance-risk-ratio at training time, neither did we study the effect of larger replay memory sizes, other learning rules that AdaGrad or other learning schemes than Q-learning. Instead of using batch normalisation with a division by the standard deviation, we only subtracted the mean value. Also, our training environment currently comes with the drawback of not running in parallel; the underlying market simulation does not proceed until the agent has committed to an action. As we had sudden jumps in our recorded tick data due to missing records that may lead to impossible profits or losses, we manually set profit thresholds in order to avoid flawed values. As the demo account that we have used for our real world example had a limited validity, which left us only 20 trading days on the DE30 index, we did not observe enough trades to make a reliable statement about the long time reliability of that concrete strategy.

B. Future work

So far we have only used course data of a single trading symbol. The application of our method on other assets, especially in the field of currency exchange, may give us a deeper understanding of machine learning in economic domains. In future work, we may study the benefit of aggregated input sequences of different assets. For example, a convolutional neural network may correlate course data from different sources, in order to improve an optimal policy. More sophisticated methods of transfer learning may enable us to reuse already acquired knowledge to improve the performance on unknown assets. For instance, we may use progressive neural networks to transfer knowledge into multiple action spaces. This allows to learn trading on multiple assets simultaneously, making use of correlations in a large input space.
Also, the trade proposals of our agents may serve as an input for more sophisticated trading algorithms that employ prior market knowledge. As an example, a rule-based system that incorporates long term market knowledge may make use of the agents proposals to provide a fully automatic, reliable trading program. Such a rule-based system might prevent the agent to open positions if above or below a certain threshold, that a human operator may set according to his or her prior knowledge of the market. Furthermore, future work may consider the integration of economic news text as input word vectors.

References

[1] P. Aghion, B. Jones, and C. Jones. *Artificial intelligence and economic growth*. Tech. rep. National Bureau of Economic Research, 2017.

[2] R. Akita et al. “Deep learning for stock prediction using numerical and textual information”. In: *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. IEEE. 2016, pp. 1–6.

[3] J. Bollen, H. Mao, and X. Zeng. “Twitter mood predicts the stock market”. In: *Journal of computational science* 2.1 (2011), pp. 1–8.

[4] K. Chen, Y. Zhou, and F. Dai. “A LSTM-based method for stock returns prediction: A case study of China stock market”. In: *2015 IEEE International Conference on Big Data (Big Data)*. IEEE. 2015, pp. 2823–2824.

[5] X. Ding et al. “Deep learning for event-driven stock prediction”. In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[6] J. Duchi, E. Hazan, and Y. Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[7] F. Gers, J. Schmidhuber, and F. Cummins. “Learning to forget: Continual prediction with LSTM”. In: (1999).

[8] John E Moody and Matthew Saffell. “Reinforcement learning for trading”. In: *Advances in Neural Information Processing Systems*. 1999, pp. 917–923.

[9] T. Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015).

[10] M. Vargas, B. De Lima, and A. Evsukoff. “Deep learning for stock market prediction from financial news articles”. In: *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. IEEE. 2017, pp. 60–65.

[11] C. Watkins and P. Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

M. Sc. Nico Zengeler studied IT Security and Applied Computer Science at the Ruhr-University Bochum from 2010 to 2017. From 2012 to 2016 he worked as a research assistant at the Department of Energy Plant Technology and from 2016 to 2017 at the Institute of Neuroinformatics, both at the Ruhr-University Bochum. Currently, he is a PhD student at the Ruhr-University Bochum and employed at the Institute of Computer Science at the Ruhr West University of Applied Sciences.

Prof. Dr. Uwe Handmann worked at the Chair of Theoretical Biology at the Institute of Neuroinformatics at the Ruhr-University Bochum as a research associate focusing on neuroinformatics, image processing, driver assistance systems and sensor data fusion from 1995 to 2001. Since 1999 he was head of the working group ”Image Processing”. Dr. Handmann received his Dr.-Ing. degree in 2000. From 2000 to 2008, Dr. Handmann was group leader and lead architect in the field of biometric systems at L1 Identity Solutions AG (formerly ZN Vision Technologies AG). In 2008, he accepted a professorship at the University of Applied Sciences Sdwestfalen and headed the laboratory for computer science, image processing and media technology. In 2010, Dr. Handmann was offered a professorship at the Ruhr West University of Applied Sciences. At the Bottrop campus he heads the Institute of Computer Science.

APPENDIX

CfD environment for Q-learning

Algorithm 1 CfD Q-learning

procedure TRAINING
  $x \leftarrow$ *cleaned historic trade data*
  $l \leftarrow$ *input sequence length*
  $c \leftarrow$ *constant factor to scale rewards*
  $M \leftarrow$ *experience replay memory*
  $\pi \leftarrow$ *Network policy, including exploration*
  $d_{profit}, d_{loss} \leftarrow$ *delta values for take-profit and stop-loss*
  while learning **do**
    $t \leftarrow$ *random point of time in trading history*
    while $t \leq len(x) - l$ **do**
      $reward \leftarrow 0$
      $terminal \leftarrow 0$
      $state_1 \leftarrow x[t : t + l] - mean(x[t : t + l])$
      $action \leftarrow \pi(state_1)$
      if $action \mathrel{\widehat{=}}$ *no trade* **then**
        $t \leftarrow t+1$
      else
        if $action \mathrel{\widehat{=}}$ *long* **then**
          take-profit $\leftarrow x[t + l] + d_{profit}(action)$
          stop-loss $\leftarrow x[t + l] - d_{loss}(action)$
          while $t \leq len(x) - l$ **do**
            $t \leftarrow t+1$
            $price \leftarrow x[t + l]$[*bid price*]
            if $price \geq$ *take-profit* **then**
              $reward \leftarrow c \cdot (price - take\text{-}profit + d_{profit}(action))$
              $terminal \leftarrow 1$
            if $price \leq$ *stop-loss* **then**
              $reward \leftarrow c \cdot (price - stop\text{-}loss - d_{loss}(action))$
              $terminal \leftarrow 1$
        if $action \mathrel{\widehat{=}}$ *short* **then**
          take-profit $\leftarrow x[t + l] - d_{profit}(action)$
          stop-loss $\leftarrow x[t + l] + d_{loss}(action)$
          while $t \leq len(x) - l$ **do**
            $t \leftarrow t+1$
            $price \leftarrow x[t + l]$[*ask price*]
            if $price \leq$ *take-profit* **then**
              $reward \leftarrow c \cdot (take\text{-}profit - price + d_{profit}(action))$
              $terminal \leftarrow 1$
            if $price \geq$ *stop-loss* **then**
              $reward \leftarrow c \cdot (stop\text{-}loss - price - d_{loss}(action))$
              $terminal \leftarrow 1$
      $state_2 \leftarrow x[t : t + l] - mean(x[t : t + l])$
      append to $M(state_1, action, state_2, reward, terminal)$
      if $len(M) \geq batchsize$ **then**
        experience $\leftarrow$ *sample*(M)
        learn(experience)
        update priority(experience)