



School of Business, Economics and Information Systems

Chair of Financial Data Analytics
Prof. Dr. Ralf Kellner

Machine Learning for Algorithmic Trading

Practicability Study on Deep Supervised & Reinforcement Learning

Florian Peschke

Master Thesis

Machine Learning for Algorithmic Trading

Practicability Study on Deep Supervised & Reinforcement Learning

Florian Peschke

Reviewer/Supervisor

Prof. Dr. Ralf Kellner

University of Passau

School of Business, Economics and Information Systems

Chair of Financial Data Analytics

Type: Master Thesis – Business Administration (M.Sc.)

Filed by: Florian Peschke (105259)
florian.peschke@uni-passau.de
Hagenauerstr. 30, 94032 Passau

Attachments: Folder that includes the code, data, and output files.

Docker image: hub.docker.com/r/flo0128/master__thesis

Contents

List of Figures	vii
List of Tables	ix
1. Introduction	1
2. Financial Theory	3
2.1. Efficient Market Hypothesis	3
2.2. Random Walk Hypothesis	3
2.3. Technical Indicators	4
2.4. Financial Return	4
3. Deep Learning	7
3.1. Artificial Neural Networks	7
3.2. Training	9
3.3. Classes of Artificial Neural Networks	12
3.3.1. Feedforward Neural Network	12
3.3.2. Recurrent Neural Network	13
3.4. Model Interpretability	15
4. Reinforcement Learning	19
4.1. Markow Decision Process	19
4.2. Valuation	20
4.3. Algorithms	21
4.4. Deep Q Networks	22
5. Study-Specific Theory	25
5.1. Incremental Metrics	25
5.2. Loss Functions	27
5.3. Activation Functions	27
5.4. Optimizers	29
5.4.1. AdaGrad	30
5.4.2. AdaDelta	30
5.4.3. Adam	31
5.4.4. Rprop	31

5.5. Incremental Higher-Order Derivatives	32
6. Empirical research	35
6.1. Data	35
6.2. Supervised Learning to Predict the CCR	39
6.2.1. Study Design	39
6.2.2. Results	41
6.2.3. Summary & Discussion	44
6.3. Trading via Reinforcement Learning	45
6.3.1. Study Design	46
6.3.2. Results	48
6.3.3. Summary & Discussion	50
7. Summary & Conclusion	53
Bibliography	55
A. Figures	59
B. Tables	65

List of Figures

3.1. Sigmoid activation function and its derivative	14
5.1. Activation Functions	28
5.1. Activation Functions (continued)	29
6.1. Public Storage (PSA) <i>close</i> and <i>diff(log(close))</i>	37
6.2. Box-plots of PSA and all companies associated with the sector <i>real estate</i> (incl. PSA)	38
6.3. Correlation between all companies of the real estate sector – <i>diff(log(close))</i> – [2016, 2021)	39
6.4. Predicted vs. target values for the given window (with the testing dataset containing 15 % of the window’s data)	43
6.5. Performance on testing dataset – total rewards	49
6.6. Performance during training indicating the total rewards at the end of each episode	50
6.7. Realized rewards during training (total reward per iteration) and evalu- ation (cumulative realized rewards)	50
A.1. Evaluation history (incremental metrics) – [2016M01, 2019M01)	59
A.2. Evaluation history (incremental metrics) – [2017M01, 2020M01)	60
A.3. Evaluation history (incremental metrics) – [2018M01, 2021M01)	61
A.4. Detailed evaluation plot of the multi-agent stock environment (stock=PSA)	62
A.5. Detailed evaluation plot of the single-agent stock environment (stock=PSA)	63

List of Tables

6.1. Descriptive statistics of $\text{diff}(\log(\text{close}))$ correlation – [2016, 2021)	38
6.2. Descriptive statistics of correlation between the target ($\text{diff}(\log(\text{close}))$) of PSA) and $\text{diff}(\log(\text{close}))$ of all other companies – [2016, 2021)	38
6.3. Descriptive statistics of correlations between the target ($\text{diff}(\log(\text{close}))$) of PSA) and all other features – [2016, 2021)	40
6.4. Rolling windows, with each containing three years of data	40
6.5. Properties of the best model per rolling window.	42
6.6. Metrics of the best model with regard to the testing dataset	42
6.7. Feature importance for [2018M01, 2021M01) – testing dataset	45
6.8. Evaluation of RL-agents (total rewards)	48
B.1. Variance per sector of the S&P500 – [2016, 2021) – $\text{diff}(\log(\text{close}))$. . .	65
B.2. Features ($n = 72$) – technical indicator details on mrjbq7.github.io/ta-lib/funcs.html	66
B.3. Companies with symbols of sector <i>real estate</i> (S&P500)	67
B.4. Variance per company of sector <i>real estate</i> of the S&P500 – [2016, 2021) – $\text{diff}(\log(\text{close}))$	68
B.5. Hyper-Parameter spaces	69
B.6. Hyper-Parameter spaces (optimizer)	69

Acronyms

AdaGrad	adaptive gradient	30
Adam	adaptive moment estimation	30
AI	artificial intelligence	
ANN	artificial neural network	36
ATR	average true range – volatility	44
CCR	continuous compounding return	37
DL	deep learning	36
DQN	deep Q networks	45
EMA	exponential moving average	44
EMH	efficient market hypothesis	39
FNN	feedforward neural network	41
GaussianNLLLoss	Gaussian negative log likelihood loss	27
GPU	graphics processing unit	36
HL	Huber loss	27
HPT	hyper-parameter tuning	40
HP	hyper-parameter	35
LSTM	long short-term memory	41
MAE	mean absolute error	27
MA	moving average	44
MDP	Markov decision process	19
ML	machine learning	19
MSE	mean squared error	27
PCC	Pearson correlation coefficient	42
PSA	Public Storage	35
R2	coefficient of determination	42
naïve R2	naïve coefficient of determination	36
ReLU	rectified linear unit	48
RL	reinforcement learning	35
RNN	recurrent neural network	13
ROCR	rate of change ratio – price/prevprice	44
Rprop	resilient backpropagation	30
RWH	random walk hypothesis	3
SARSA	state–action–reward–state–action	22
SCR	standard compounding return	4
SGD	stochastic gradient descent	31
SL	supervised learning	7
SRN	simple recurrent network	13

Tanh	tangens hyperbolicus	48
TI	technical indicator	35
TA	technical analysis	4
VAR	variance	44
VM	virtual machine	41

Abstract

In this study, artificial neural networks have been used to predict the next day's log return of one company in the S&P500, utilizing the corresponding historical industry stock and technical indicator data (*real estate*) for training. In order to obtain a more consistent picture of out-of-sample inference capabilities, rolling windows with different test horizons were employed.

Notably, all models were found to be unable to explain the variance of the target variable, indicating that the first form of the efficient market hypothesis is valid and that the next day's return follows the martingale property. Notwithstanding these general results, one model was able to show good intermediary results for the very first batches. In this respect, statements about the importance of single features should be taken with caution, but with reference to the last model, the variance/volatility and momentum/moving average indicators were the most promising.

In a second study on reinforcement learning using deep Q networks, it was shown that over a one-year evaluation period, a single-agent and a multi-agent reinforcement learning environment were not able to outperform a buy-and-hold strategy in terms of the total reward. Nevertheless, the single-agent was superior to the multi-agent during the entire period and to the buy-and-hold strategy during about two-thirds, making reinforcement learning the more promising application for algorithmic trading.

Introduction

Is it possible to predict stock prices and are markets therefore inefficient? Both these questions are controversially discussed in the literature between people in favor and those against the efficient market hypothesis [Fam70; GS80]. In short, the efficient market hypothesis states that prices reflect available information and excess returns are not possible in the long run.

A key question arising from this debate is: Do stock prices follow a random or a deterministic and thus predictable course? The latter would be highly beneficial for algorithmic trading using artificial intelligence, since artificial intelligence is more versatile compared to hard coded trading strategies – under the premise, that artificial intelligence-models behave deterministic, thus explainable, and not random.

Research in this area can be broadly divided into two categories: Use of models based on econometric theory or machine learning theory. As for the first category, a study by Campbell and Thompson [CT07] was able to show that predictive regression models are able to outperform historical average returns (see as well [BCM90; Lew04]). By extending these basic regression models with state-of-the-art model architectures, i.e., machine learning and in particular artificial neural networks in combination with deep learning, Gu et al. [GKX20] could show that returns are predictable to some degree and thus superior to the naïve prediction of zero. A different approach was taken by Nelson et al., who transformed this continuous prediction task into a classification task, i.e., predicting whether the price of a certain stock will increase in the near future or not. By doing so, they succeeded in achieving accuracies greater than 50 % [NPO17].

Study Objective One This leads to the first study question: are deep artificial neural networks able to predict the next day's log return given historical stock and technical indicator data, and if so, on more than one test data horizon? The latter implies a rolling-window study, which strengthens the implications with respect to the validity of the efficient market hypothesis.

In addition to predicting the next day's log return, reinforcement learning can be used to provide a more realistic framework for trading. For example, Deng et al. were

able to set up a deep reinforcement learning framework for financial signal processing with respect to stock index and commodity futures contracts with up-to-the-minute data. They were able to show that reinforcement learning is capable of positive profit and loss scores, depending not on preselected features, but on an automatic feature learning mechanism [Den+17]. Similarly, Zhang et al. demonstrated the benefit of reinforcement learning with respect to trading liquid futures contracts, as they used technical indicators that overall resulted in reinforcement learning algorithms, which could outperform baseline models such as the long-only strategy [ZZR20]. An approach to specifically adapt the deep Q networks-algorithm to finance is proposed by Théate and Ernst [TE21]. Their *Trading Deep Q-Network algorithm (TDQN)* performed on average better than comparable trading strategies such as buy-and-hold and sell-and-hold strategies given individual stocks and stock indices.

Study Objective Two Therefore, the second objective of this work is to gain insights into whether reinforcement learning is beneficial for trading a single stock using deep Q networks compared to a buy-and-hold strategy. In this respect, a single-agent environment competes with a multi-agent environment. The latter has one agent for each position (in the market and out of the market). Moreover, in the optimal case, the first study objective provides clues as to which features are likely to be important for training reinforcement learning-agents.

Structure of this Thesis Methodology is covered in Chapter 2, Chapter 3 and Chapter 4, which introduce financial theory, deep learning as well as reinforcement learning. Following these general theory chapters, Chapter 5 discusses specific theoretical aspects that will be used in the empirical part of this thesis, which is presented in Chapter 6. In addition to describing the data used, it is about the two objectives mentioned above: (1) a section on supervised learning, which attempts to predict the logarithmic return of the closing price for the next day, and (2) simulated trading via reinforcement learning.

Financial Theory

2.1 Efficient Market Hypothesis

According to FAMA [FAM91], the efficient market hypothesis (EMH) is “the simple statement that security prices fully reflect all available information”. Consequently, there should be no loopholes to outperform the market, at least not in the long run.

Regarding these loopholes, the EMH can be divided into three classes according to the information that is considered [MF70].

1. The weak EMH states that no abnormal trading profits are possible using historical information $\mathcal{I}_t^{\text{weak}}$ (historical stock information up to t).
2. The semi-strong form of EMH extends the information set to $\mathcal{I}_t^{\text{semi-strong}}$, which includes all publicly available information up to t .
3. Finally, the strongest form $\mathcal{I}_t^{\text{strong}}$ states that abnormal returns due to trading are not possible when all information, including insider information, is available.

2.2 Random Walk Hypothesis

An implication that follows from EMH is that profitable speculation in the stock market is not possible due to random stock price fluctuations, since markets in general lack memory [Fam65; GP09]. Mathematically, the random walk without drift is defined as $P_t = P_{t-1} + u_t$ with $u_t \sim \mathcal{N}(0, \sigma^2)$ and $\mathbb{E}(P_t) = P_{t-1}$, while the random walk with drift δ is given by $P_t = \delta + P_{t-1} + u_t$ [GP09]. Closely related to random walk hypothesis (RWH) is the martingale property, which states that prices fluctuate randomly such that $\mathbb{E}(\frac{P_{t+1}-P_t}{P_t} | \mathcal{I}_t^{\text{weak}}) = 0$ [Sam73]. Hence, the next day's return is expected to be zero.

2.3 Technical Indicators

Stock exchange data are usually given in the form of opening, closing, high and low prices as well as volume (OHLCV) for a specific date t . Regarding the frequency, one can roughly divide between intraday data (short-term) and non-intraday data (long-term).

Since the information given by the OHLCV data is limited, one can calculate additional information referred to as technical indicators (TIs) derived from technical analysis (TA). In contrast to the EMH and RWH, TA assumes that the future behavior of individual securities can be predicted from the past by identifying patterns that may repeat over time [Fam65].

To find these patterns, various types of pattern recognition methods can be applied, such as momentum and volatility indicators, overlap studies like the exponential moving average, and more.¹

2.4 Financial Return

This section is based on the book *Finanzmarktstatistik* by Schmid and Trede [ST06, chapter 1].

Standard Compounding The standard compounding return (SCR) R_t^s indicates the change in value from P_{t-1} to P_t . Therefore, $P_t = P_{t-1}(1 + R_t^s)$, where the SCR is defined as $R_t^s = \frac{P_t}{P_{t-1}} - 1$. In terms of k periods, the price P_{t+k} can be stated as:

$$P_{t+k} = P_t(1 + R_{t+1}^s)(1 + R_{t+2}^s) \cdots (1 + R_{t+k}^s) = P_t \prod_{i=1}^k (1 + R_{t+i}^s)$$

Using the SCR of the entire period $\frac{P_{t+k}}{P_t} = \prod_{i=1}^k (1 + R_{t+i}^s) \equiv (1 + \bar{R}_t^s(k))^k$, one can calculate the period's average SCR as follows:

$$\bar{R}_t^s(k) = \sqrt[k]{\prod_{i=1}^k (1 + R_{t+i}^s)} - 1$$

¹For a richer list see *Python wrapper for TA-Lib*.

Continuous Compounding Transforming the SCR into continuous time leads to the continuous compounding return (CCR), which is based on the logarithm relationship between P_t and P_{t-1} with $P_t = P_{t-1}\exp(R_t^c)$. Resolving for R_t^c gives the CCR: $R_t^c = \log(\frac{P_t}{P_{t-1}}) = \log(P_t) - \log(P_{t-1})$. For k periods, the price P_{t+k} can be stated by chaining the individual CCRs:

$$P_{t+k} = P_t \exp(R_{t+1}^c) \exp(R_{t+2}^c) \cdots \exp(R_{t+k}^c) = P_t \exp(\sum_{i=1}^k R_{t+i}^c)$$

Because $\frac{P_{t+k}}{P_t} = \exp(\sum_{i=1}^k R_{t+i}^c) \equiv \exp(k\bar{R}_t^c(k))$, the period's average CCR is given by:²

$$\bar{R}_t^c(k) = \frac{1}{k} \sum_{i=1}^k (R_{t+i}^c)$$

²This additivity property of continuous compounding returns is beneficial in measuring the performance of reinforcement learning-agents in terms of cumulative rewards (see Chapter 6).

Deep Learning

Relevant to the study in this paper are deep supervised learning (SL) and deep reinforcement learning (RL), both of which are based on deep learning (DL) using deep artificial neural networks (ANNs).¹

The mathematical formulas in this chapter related to DL are taken from the book *Mathematics for Machine Learning* by Deisenroth et al. [DFO20].

3.1 Artificial Neural Networks

ANNs are by their nature mathematical operations, where a simple ANN, like the feedforward neural network (FNN), is simply a concatenation of K linear functions or modules f_k with $k \in [1, \dots, K]$. Each of these chained functions takes an input \mathbf{x} to output a prediction \hat{y} that approximates the true value y :²

$$\hat{y} = f_K(f_{K-1}(\dots(f_k(\dots(f_1(\mathbf{x}))))))$$

The first layer f_1 takes the raw \mathbf{x} as input, whereas all successive layers f_k with $k > 1$ take the output \mathbf{f}_{k-1} of the previous layer as input.

Each function f_k has an individual set of parameters $\boldsymbol{\theta}_{k-1} \in \boldsymbol{\theta}$ that always includes a weight tensor \mathbf{A}_{k-1} and optionally a bias tensor \mathbf{b}_{k-1} . The output of such standard linear layer is given by: $\mathbf{f}_k = f_k(\mathbf{f}_{k-1}, \boldsymbol{\theta}_{k-1}) = \mathbf{f}_{k-1} \mathbf{A}_{k-1}^T + \mathbf{b}_{k-1}$ with $k \in [1, \dots, K]$, where $\mathbf{f}_0 \doteq \mathbf{x}$.^{3 4}

Since the linear module as such is obviously limited to linear behavior, nonlinearity can be introduced by activation functions ϕ_k , which are wrapped around each linear layer f_k : $\phi_k(f_k(\mathbf{f}_{k-1}, \boldsymbol{\theta}_{k-1}))$.

¹SL uses data labeled by experts, while RL is a trial-and-error process to find a strategy that maximizes the so-called (total) reward.

²The target can as well be a vector \mathbf{y} that is approximated by $\hat{\mathbf{y}}$.

³A deep ANN is given if $K \geq 3$ such that f_1 is the input module or layer, f_k with $k \in [2, \dots, K-1]$ are the hidden modules and f_K is the output module.

⁴If \mathbf{f}_{k-1} is a row vector with $\mathbf{f}_{k-1} \in \mathbb{R}^{1 \times d_{\text{in}}}$, then $\mathbf{A}_{k-1} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and $\mathbf{b}_{k-1} \in \mathbb{R}^{1 \times d_{\text{out}}}$.

Forwardpropagation The forward pass takes the input \mathbf{x} and passes it through the first layer f_1 , then takes the output \mathbf{f}_1 and passes it through the second layer f_2 and again uses the output, this time \mathbf{f}_2 , as the input to the third layer f_3 , until it reaches the last output layer f_K that outputs the approximated value of \mathbf{y} , which is given by $\hat{\mathbf{y}}$ with $\hat{\mathbf{y}} \approx \mathbf{y}$.

Loss With the approximation $\hat{\mathbf{y}} \approx \mathbf{y}$ it is possible to compare the target \mathbf{y} with the forecast $\hat{\mathbf{y}}$ to quantify the forecast error by a loss function.

For example, if \mathbf{y} is a real number \mathbb{R} , one can calculate the mean squared error (MSE), which is given by $\mathcal{L}^{\text{MSE}}(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2$, or the Gaussian negative log likelihood loss (GaussianNLLLoss) $\mathcal{L}^{\text{GNLL}}(\hat{\mathbf{y}}, \mathbf{y}, \sigma^2) = \frac{1}{2}(\log(\max(\sigma^2, \epsilon)) + \frac{(\hat{\mathbf{y}} - \mathbf{y})^2}{\max(\sigma^2, \epsilon)}) + c$.

Backpropagation Each loss function used in DL must be differentiable to update the individual parameters per layer that are included in the ANN's parameter set $\boldsymbol{\theta}$. The intuition underlying this approach is that the loss indicates how far the predicted value(s) $\hat{\mathbf{y}}$ are off. Taking this into account and the fact that the loss function is differentiable, it is now possible to partially differentiate the loss \mathcal{L} with respect $\boldsymbol{\theta}$. Given this gradient information, one can then update $\boldsymbol{\theta}$ such that the error would be reduced.

To calculate this gradient information, the chain rule is applied, which partially differentiates the loss \mathcal{L} with respect to all parameters $\boldsymbol{\theta} = \{\mathbf{A}_0, \mathbf{b}_0, \dots, \mathbf{A}_{K-1}, \mathbf{b}_{K-1}\}$:⁵

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{K-1}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \boldsymbol{\theta}_{K-1}} \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{K-2}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \boldsymbol{\theta}_{K-2}}\end{aligned}$$

Generalizing the chain rule to each individual parameter set $\boldsymbol{\theta}_j$, where $\boldsymbol{\theta}_j = \{\mathbf{A}_j, \mathbf{b}_j\}$ with $j \in [0, \dots, K-1]$, yields:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_j} = \frac{\partial \mathcal{L}}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}} \dots \frac{\partial \mathbf{f}_{j+2}}{\partial \mathbf{f}_{j+1}} \frac{\partial \mathbf{f}_{j+1}}{\partial \boldsymbol{\theta}_j}$$

⁵If the target is a vector \mathbf{y} , then the prediction \mathbf{f}_K is a vector, too.

Optimization Step The previous section explained how to perform backward propagation through the computation graph of the ANN with respect to the loss value. Based on this backward propagation, it is possible to access the gradient signals (slope) of the loss function with respect to the parameters $\nabla_{\theta}\mathcal{L}(\theta)$. To reduce the loss, one increases parameter elements with high negative values and decreases parameter elements with high positive values.

For the update mechanism described here, optimization algorithms are required to use the gradients $\nabla_{\theta_t}\mathcal{L}(\theta_t)$ at the time step t and update the existing parameters. Such an algorithm can for example apply a simple update like $\theta_{t+1} = \theta_t - \gamma_t(\nabla_{\theta_t}\mathcal{L}(\theta_t))^T$, with a step size γ .

Finding either the global minima or the best local minima from a set of local minima is a challenge that all optimization algorithms face. Such searching can be guided by the step size or learning rate γ . For example, if γ is too low, the optimizer may get stuck in a non-optimal minima, while in the opposite case, if γ is too high, a minima may not be reached at all.

3.2 Training

Let the data be $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{y} \in \mathbb{R}^{m \times 1}$, such that n indicates the number of observations and m indicates the number of variables.⁶

Batching With thousands, millions, or even more observations, computing the loss, gradients, and optimization steps would be infeasible due to computer memory and/or computational limitations. For this reason, the observations are split into N batches. One approach is to take each batch individually, calculating the loss \mathcal{L}_n and gradients $\nabla_{\theta_t}\mathcal{L}_n(\theta_t)$ to do one optimization step $\theta_{t+1} = \theta_t - \gamma_t(\nabla_{\theta_t}\mathcal{L}_n(\theta_t))^T$. Alternatively, one could accumulate the gradients of all N batches to perform the optimization step: $\theta_{t+1} = \theta_t - \gamma_t(\nabla_{\theta_t}\mathcal{L}(\theta_t))^T = \theta_t - \gamma_t \sum_{n=1}^N (\nabla_{\theta_t}\mathcal{L}_n(\theta_t))^T$.

Epoch One iteration over all N batches is called an epoch. In general, after a very few epochs, the parameters θ have not yet reached their optimal values, such that the loss is not minimal. For this reason, it is a common approach to repeat the described epoch step multiple times, always doing the forward and backward pass and updating the parameters. This can theoretically be repeated indefinitely.

⁶In terms of input \mathbf{X} , these variables are also referred to as input features.

Validation Generally, additional epochs help to minimize the loss. However, too many runs can lead to overfitting of the model to the training dataset. As the loss with respect to the training dataset continues to decrease, the model appears more and more promising for the task at hand. However, it is losing its generalization ability. Inference on out-of-sample data will show much weaker performance because the parameters are calibrated with respect to minimizing the loss on the training data as much as possible.

To avoid this, it is common to take the initial dataset with $\mathbf{X} \in \mathbb{R}^{n \times m}$ and $\mathbf{y} \in \mathbb{R}^{n \times 1}$ and split it into a training, validation and test set

$$\begin{aligned} \mathbf{X}^{\text{train}} \in \mathbb{R}^{n_{\text{train}} \times m} & \quad \wedge \quad \mathbf{y}^{\text{train}} \in \mathbb{R}^{n_{\text{train}} \times 1} \\ \mathbf{X}^{\text{val}} \in \mathbb{R}^{n_{\text{val}} \times m} & \quad \wedge \quad \mathbf{y}^{\text{val}} \in \mathbb{R}^{n_{\text{val}} \times 1} \\ \mathbf{X}^{\text{test}} \in \mathbb{R}^{n_{\text{test}} \times m} & \quad \wedge \quad \mathbf{y}^{\text{test}} \in \mathbb{R}^{n_{\text{test}} \times 1} \end{aligned}$$

with $n_{\text{train}} + n_{\text{val}} + n_{\text{test}} = n$. The training dataset is the only dataset of all three on which the model is directly trained.

To identify overfitting, one checks the inference performance on the validation dataset after each training epoch, for example. If the loss on the training dataset steadily decreases, but the loss on the validation dataset increases after, say, a few epochs, the ANN begins to overfit to the training data and suffers from its inference quality on unseen data.

Contrary to overfitting is underfitting, which is not as problematic as overfitting, since the ANN underperforms both on the training and on the unseen data. Of course, this is also problematic, but not as much as overfitting, since underfitting is directly noticeable on the training dataset, while overfitting is not. One strategy to overcome underfitting is to extend the ANN-architecture.

Regularization Overfitting can be further avoided or at least reduced by various regularization techniques. Three of them are: (1) implementing a dropout, (2) an early stopping mechanism, and (3) batch normalization. The latter is appropriate for normalizing individual batches, as the name implies, but can also act as a regularization mechanism [IS15].

Dropout A dropout can reduce overfitting by randomly setting elements of the layer’s inputs or outputs to zero during training, given the probability p [Sri+14]. In addition to zeroing elements, *PyTorch* scales the remaining elements by a factor of $\frac{1}{1-p}$ [Pas+19]. While the dropout mechanism is active during training, it is inactive during evaluation since the problem of overfitting arises only during the former.

Early Stopping Since the dropout is directly implemented into the artificial neural network architecture, a simpler but still powerful method can be used to prevent overfitting of the model from outside. Following each training epoch, the performance of the model is evaluated on the validation set to compare the metrics to the results of the training set. If they start to diverge, i.e., the validation metrics start to deteriorate, it is possible to stop the training early, before it has completed the specified number of (maximum) epochs. This not only leads to better evaluation results, but is also memory and time efficient, since not all defined epochs have to be run.

Batch Normalization A third method to avoid overfitting is batch normalization. Similar to the dropout mechanism, it sits directly inside the ANN-architecture and normalizes each mini-batch.⁷ One advantage is that higher learning rates can be used while being less cautious about initializing the parameters. It might further regularize the ANN, making the dropout mechanism obsolete [IS15].

$$\mathbf{y}^i = \text{scale}(\hat{\mathbf{x}}^i) + \text{shift} = \frac{\mathbf{x}^i - \mathbb{E}(\mathbf{x}^i)}{\sqrt{\text{Var}(\mathbf{x}^i) + \epsilon}} \boldsymbol{\gamma} + \boldsymbol{\beta}$$

whereby $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameter vectors, while ϵ provides numerical stability to the denominator.

Evaluation Since the validation dataset is known to the model to some extent due to data leakage,⁸ one needs an additional evaluation or testing dataset that must be ensured to be unknown to the model. This dataset can then be used to determine the actual inference quality of the fitted model.

⁷If a batch is given by $\mathbf{X}_n \in \mathbb{R}^{d_{bn} \times d_{seq} \times d_{in}}$, then a mini-batch is given by $\mathbf{X}_n^i \in \mathbb{R}^{d_{seq} \times d_{in}}$ with $i \in [1, \dots, d_{bn}]$.

⁸For example, by adjusting the learning rate or early stopping the training according to a measure of the validation results.

Metrics Besides the loss, other metrics can be employed to measure the quality of the ANN's output. With regression tasks where y and thus \hat{y} are real numbers, measures such as the coefficient of determination (R2), naïve coefficient of determination (naïve R2),⁹ or the Pearson correlation coefficient (PCC) between y and thus \hat{y} may be good candidates.

3.3 Classes of Artificial Neural Networks

The simplest artificial neural network is the feedforward neural network, which just concatenates linear layers and their respective activation functions in a pure forward direction, while the more complex recurrent neural networks allow each layer to circulate and thus reuse values.

3.3.1 Feedforward Neural Network

The plain FNN has K linear layers, each of which performs a linear transformation with respect to the input \mathbf{f}_{k-1} and a subsequent transformation using an activation function ϕ_k .¹⁰ All linear layers feature a weight \mathbf{W}_{k-1} and an optional bias \mathbf{b}_{k-1} , which form the standard linear equation $f_k(\mathbf{f}_{k-1}, \boldsymbol{\theta}_{k-1}) = \mathbf{f}_{k-1} \mathbf{A}_{k-1}^T + \mathbf{b}_{k-1}$ with $\mathbf{f}_0 \doteq \mathbf{x}$ and $k \in [1, \dots, K]$.

The functioning within the layers can be explained in more detail by looking at the linear operations of the first level, focusing on the shapes of the \mathbf{f}_0 , \mathbf{A}_0 and \mathbf{b}_0 :

Input From $\mathbf{f}_0 \doteq \mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^{1 \times m}$ follows that $\mathbf{f}_0 \in \mathbb{R}^{1 \times m}$, where m is the input size of \mathbf{x} – respective \mathbf{f}_0 .¹¹ Since the input size is relevant for all layers, it is renamed to d_{in} (input dimension).

Times Weight The weight matrix $\mathbf{A}_0 \in \mathbb{R}^{d_{\text{out},0} \times d_{\text{in},0}}$ has an output dimension $d_{\text{out},0}$, which acts as a hyper-parameter that sets the so-called unit size of this layer. Multiplying \mathbf{f}_0 by \mathbf{W}_0^T yields a new vector $\mathbf{a}_0 \in \mathbb{R}^{1 \times d_{\text{out},0}}$.

⁹The naïve R2 simply replaces the mean $\mathbb{E}(y)$ with 0, implying a naïve prediction of zero for the next day's return.

¹⁰A linear output is achieved by using the unity as a pseudo activation function

¹¹Here, \mathbf{x} has the shape $1 \times m$ and is thus a row vector. However, it can be a matrix or 3D-tensor \mathbf{X} as well.

Plus Bias Finally, \mathbf{a}_0 is being added to the bias $\mathbf{b}_0 \in \mathbb{R}^{1 \times d_{\text{out},0}}$.

Results in Output Addition of \mathbf{a} and \mathbf{b} gives the layer's output $\mathbf{f}_1 = \mathbf{f}_0 \mathbf{A}_0^T + \mathbf{b}_0 = \mathbf{a}_0 + \mathbf{b}_0$ with $\mathbf{f}_1 \in \mathbb{R}^{1 \times d_{\text{out},0}}$. The output \mathbf{f}_1 of the first layer f_1 is transformed via the activation function ϕ_1 and then transferred to the next layer f_2 that has an input size $d_{\text{in}_1} \doteq d_{\text{out}_0}$ and a predefined output size d_{out_1} . Continuing this forward pass until the last layer is reached, which has a particular and important predefined output size $d_{\text{out}_{K-1}}$. If the target \mathbf{y} is scalar, $\hat{\mathbf{y}}$ must be scalar and thus $d_{\text{out}_{K-1}} = 1$.

3.3.2 Recurrent Neural Network

Compared to a feedforward neural network, a recurrent neural network (RNN) allows for circular movements within the computation graph of the artificial neural network.

Simple RNN The simple recurrent network (SRN) uses a hidden state that permits such circular movements within the calculation graph. This hidden state \mathbf{h}^t is indexed to time. Given an input sequence $(\mathbf{x}^{t-k}, \dots, \mathbf{x}^t)$ from the time step $t - k$ to t , the hidden state is the output of a linear layer transformed by an activation function

$$\mathbf{h}^t = \phi^h(\mathbf{a}^t) = \phi^h(f(\mathbf{h}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta}^h)) = \phi^h(\mathbf{W}\mathbf{h}^{t-1} + \mathbf{U}\mathbf{x}^t + \mathbf{b})$$

\mathbf{h}^t could be recursively unfolded to revalidate the time dependence it contains, by substituting \mathbf{h}^{t-1} for its formula and replacing \mathbf{h}^{t-2} again with its formula until reaching the first state at $t - k$, which is randomly generated. Since the goal remains to predict a target value, the hidden state is used to forecast the corresponding target sequence $(\mathbf{y}^{t-k}, \dots, \mathbf{y}^t)$ given by the linear layer and a subsequent activation function

$$\hat{\mathbf{y}}^t = \phi^o(\mathbf{o}^t) = \phi^o(f(\mathbf{h}^t, \boldsymbol{\theta}^o)) = \phi^o(\mathbf{V}\mathbf{h}^t + \mathbf{c})$$

The parameter set $\boldsymbol{\theta}^h = \{\mathbf{W}, \mathbf{U}, \mathbf{b}\}$ contains two weight matrices \mathbf{W} and \mathbf{U} , and a bias \mathbf{b} , whereby the parameter set $\boldsymbol{\theta}^o = \{\mathbf{V}, \mathbf{c}\}$ has one weight matrix \mathbf{V} and one bias \mathbf{c} . The first activation function ϕ^h is usually the tangens hyperbolicus (Tanh) $\text{Tanh}(x) = \tanh(x) \doteq \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$, while the second activation function ϕ^o is target specific.

Vanishing/Exploding Gradients The likelihood of vanishing or exploding gradients increases with deep ANNs, since the chain rule multiplicatively connects the individual derivatives. For vanishing gradients, the problem is that the updating signal the optimizer receives to update the parameters θ is very small, which can cause the parameters to remain unchanged, making it difficult to train the ANN and thence reach a minima. For exploding gradients, the problem is reversed, as the update values are so high that the parameters can change drastically, even leading to values so large that computers can no longer process them.

Besides the depth of the ANN, a possible cause can be the activation functions and especially their derivative. When the value of the derivative with large values (positive and negative) is very low, gradients can disappear (are near or even zero). One such problematic activation function is the Sigmoid function shown in Figure 3.1, whose derivative $\frac{d}{dx}\sigma(x)$ gets really small for large positive and negative values.

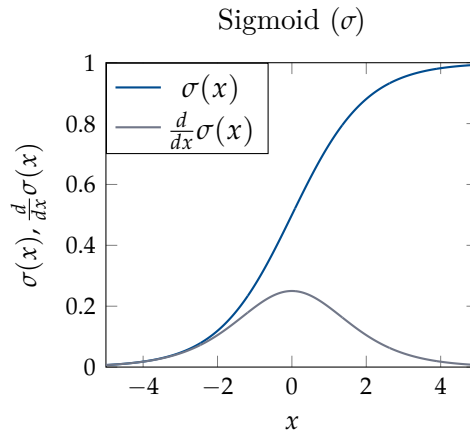


Fig. 3.1.: Sigmoid activation function and its derivative

LSTM In order to overcome the vanishing and exploding gradient problem that deeply chained SRN can face, gates are introduced into the recurrent architecture. With these gates, additional memory is established that is shared over time and updated with new information at each time step t .

Generally, the long short-term memory (LSTM)-architecture has three of these gates that, like a gatekeeper, decide what (information) stays in, what is thrown out and what is allowed to come in: the (1) input \mathbf{i}^t , (2) forget \mathbf{f}^t and (3) output \mathbf{o}^t gate. In addition to these three gates, there is again a hidden state \mathbf{h}^t as well as a hidden cell state \mathbf{c}^t that are indexed to time, and an input activation $\tilde{\mathbf{c}}^t$ [Pas+19].

$$\begin{aligned}
\mathbf{i}^t &= \phi^g(\mathbf{W}_i \mathbf{x}^t + \mathbf{U}_i \mathbf{h}^{t-1} + \mathbf{b}_i) \\
\mathbf{f}^t &= \phi^g(\mathbf{W}_f \mathbf{x}^t + \mathbf{U}_f \mathbf{h}^{t-1} + \mathbf{b}_f) \\
\mathbf{o}^t &= \phi^g(\mathbf{W}_o \mathbf{x}^t + \mathbf{U}_o \mathbf{h}^{t-1} + \mathbf{b}_o) \\
\tilde{\mathbf{c}}^t &= \phi^a(\mathbf{W}_{\tilde{c}} \mathbf{x}^t + \mathbf{U}_{\tilde{c}} \mathbf{h}^{t-1} + \mathbf{b}_{\tilde{c}})
\end{aligned}$$

Given these four equations, one can calculate the hidden cell state \mathbf{c}^t and the hidden state \mathbf{h}^t .

$$\begin{aligned}
\mathbf{c}^t &= \mathbf{f}^t \odot \mathbf{c}^{t-1} + \mathbf{i}^t \odot \tilde{\mathbf{c}}^t \\
\mathbf{h}^t &= \mathbf{o}^t \odot \phi^h(\mathbf{c}^t)
\end{aligned}$$

Each gate and the input activation contain a parameter set $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{U}, \mathbf{b}\}$, which holds two weights and one bias. The forget gate is the key element to prevent gradients from vanishing or exploding, as it regulates what information from the past to keep or forget, keeping gradients stable over time and layer stacks [HS97].

Usually, the activation function ϕ^g is the Sigmoid function $\text{Sigmoid}(x) = \sigma(x) \doteq \frac{1}{1+\exp(-x)}$, while the other two activation functions ϕ^a and ϕ^h are Tanh.

3.4 Model Interpretability

Because deep artificial neural networks can be difficult to understand due to their extensive function chaining and nonlinear transformations, examination of an ANN is a topic in itself. It usually involves three components: (1) the inputs, (2) the layers and their neurons, and (3) the output of the ANN.

To figure out how one component affects another, various attribution algorithms may be employed. Attribution can be categorized into primary, layer, and neuron attribution, where the first examines the degree to which inputs contribute to the output of the ANN, the middle quantifies the contribution of neurons per layer on the output, and the last describes how much each input contributes to the activation of neurons per layer.¹²

Focusing on primary attribution, i.e., how much each input feature contributes to the ANN's output, one can exploit the property that ANNs are differentiable in order to calculate higher-order derivatives such as the Jacobian and Hessian matrix.

¹²This categorization is used by *Captum* [Kok+20] – a Python library that offers various attribution algorithms to explain *PyTorch* models.

Jacobian Matrix Given the input $\mathbf{x} \in \mathbb{R}^{1 \times d_{\text{in}}}$ and the ANN as f_K , the first order partial derivative of f_K with respect to \mathbf{x} is called the Jacobian matrix \mathbf{J}_f and is given by:

$$\frac{\partial f_K}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_K}{\partial x_1} & \cdots & \frac{\partial f_K}{\partial x_i} & \cdots & \frac{\partial f_K}{\partial x_{d_{\text{in}}}} \end{bmatrix}$$

with $\frac{\partial f_K}{\partial \mathbf{x}} \in \mathbb{R}^{1 \times d_{\text{in}}}$ and $i \in \{1, \dots, d_{\text{in}}\}$. Each element $\frac{\partial f_K}{\partial x_i}$ of $\frac{\partial f_K}{\partial \mathbf{x}}$ indicates the influence of x_i on the output of f_K . It can thus be viewed as an importance indicator, with large $\frac{\partial f_K}{\partial x_i}$ showing that the feature x_i is significant with respect to the output of f_K .

Hessian Matrix Partial differentiation of the Jacobian matrix \mathbf{J}_f with respect to the same input \mathbf{x} yields the Hessian matrix \mathbf{H}_f :

$$\frac{\partial^2 f_K}{\partial^2 \mathbf{x}} = \begin{bmatrix} \frac{\partial^2 f_K}{\partial^2 x_1} & \cdots & \frac{\partial^2 f_K}{\partial x_1 \partial x_j} & \cdots & \frac{\partial^2 f_K}{\partial x_1 \partial x_{d_{\text{in}}}} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f_K}{\partial x_i \partial x_1} & \cdots & \frac{\partial^2 f_K}{\partial x_i \partial x_j} & \cdots & \frac{\partial^2 f_K}{\partial x_i \partial x_{d_{\text{in}}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f_K}{\partial x_{d_{\text{in}}} \partial x_1} & \cdots & \frac{\partial^2 f_K}{\partial x_{d_{\text{in}}} \partial x_j} & \cdots & \frac{\partial^2 f_K}{\partial^2 x_{d_{\text{in}}}} \end{bmatrix}$$

with $\frac{\partial^2 f_K}{\partial^2 \mathbf{x}} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$ and $i, j \in [1, \dots, d_{\text{in}}]$. \mathbf{H}_f shows the curvature of f given the specific inputs.

Interpreting Gradients The special feature of Jacobian and Hessian is that the input is not a scalar but a vector. However, compared to a function that is differentiated a first and a second time with respect to one scalar input, the same implications still hold.

For example, let f be $f(x) = x^2$, then $\frac{\partial f}{\partial x} \equiv \frac{df}{dx} = 2x$ and $\frac{\partial^2 f}{\partial^2 x} \equiv \frac{d^2 f}{dx^2} = 2$. The global minima of f is at $x = 0$ ($\text{argmin}_x f(x) = 0$). Looking at the first order derivative of f , i.e. $2x$, one can observe in which direction x has to be changed to reach this minima.¹³ If x is negative, increase it and vice versa. Larger gradients indicate a greater influence on the output of f .

¹³The first order derivative gives the slope of the tangent line at a given point $f(x)$. For example, the slope at $x = -2$ is $\frac{d}{dx} f(-2) = 2 \cdot (-2) = -4$. Thus, increasing x would decrease the output value of f in absolute values, which in turn indicates that we have moved closer to the global minima at $x = 0$.

The second order derivative of f is 2, which is positive and means that f is upward sloping (convex) $\forall x$. However, if $\frac{d^2f}{dx^2}$ were negative, the curvature of f would be downward sloping (concave) $\forall x$ and if it were zero, there would be a saddle point. Generally, if $\frac{d^2f}{dx^2} \neq 0$, f cannot be linear. This logic also applies to the Hessian matrix \mathbf{H}_f .

Reinforcement Learning

Another machine learning (ML) approach is reinforcement learning (RL), which is sequential in nature. Here, an agent performs one action per time step and state, leading to a reward that indicates how good that action was. It is then the agent's goal to maximize this sequence of rewards (total reward) by finding an optimal strategy to select actions.

Mathematics in this chapter is based on the book *Reinforcement Learning: An Introduction* by Sutton and Barto [SB20].

4.1 Markov Decision Process

RL is based on the Markov decision process (MDP), which is time-dependent and has various components to describe an environment in which an agent acts. At a given time step t , the agent is in the state S_t , which is a subset of the state set \mathcal{S} ,¹ taking an action A_t from the available action set $\mathcal{A}(s)$ to transition to the next state S_{t+1} and receiving a reward R_{t+1} , which is in the reward set \mathcal{R} .² This trajectory $(S_t, A_t, S_{t+1}, R_{t+1})_{t=0}^{T-1}$ is repeated many times, where 0 is the starting point, and T is the ending point, such that S_T is the final state, which is a subset of the state set \mathcal{S}^+ .^{3 4}

It is then possible to calculate different probabilities with the trajectory at hand. For example, the probability of transition to the next state s' and receiving a reward r is given by $p(s', r | s, a) \doteq \mathcal{P}(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$, $\forall s', s \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$.^{5 6}

¹ \mathcal{S} does not contain the final state.

²Lowercase letters stand for an ordinary, uppercase letters for a random variable.

³Because the sequence of all trajectories is finite, i.e., has a final state, it is called episodic. However, there may be no final state, indicating a continuing task.

⁴ \mathcal{S}^+ is the set of all states and includes the final state.

⁵ $\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1$, $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$.

⁶From this, the state-transition probabilities can be computed:

$$p(s' | s, a) \doteq \mathcal{P}(S_t = s' | S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

4.2 Valuation

The challenge for the agent is to decide at each time step which action $A_t \in \mathcal{A}(s)$ to take. Since each action leads to a reward R_{t+1} , adding all subsequent rewards up to the episode's end gives the total reward $G_t \doteq \sum_{\tau=t}^{T-1} R_{\tau+1}$ that the agent seeks to maximize. Typically, more recent values are considered more informative or valuable than those more distant in the future. Adjusting the reward sequence by a discount factor thus yields $G_t = \sum_{\tau=t}^{T-1} \gamma^{\tau-t} R_{\tau+1}$.

Value Functions In this sense, each state and each action available per state can be valued, as indicated by $v(s)$ and $q(s, a)$, respectively. $v(s)$ indicates the expected total reward in state S_t given all available actions A_t from the action set $\mathcal{A}(s)$, while the latter $q(s, a)$ gives the expected total reward when taking the action $A_t = a \in \mathcal{A}(s)$.

Now the question arises on what basis to select the actions. Such basis is called a strategy or, more precisely, a policy π . It maps states to probabilities for selecting each possible action, denoted by $\pi(a | s)$. Under this policy, the state and action values can be represented as follows:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[\sum_{\tau=t}^{T-1} \gamma^{\tau-t} R_{\tau+1} | S_t = s] \\ q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[\sum_{\tau=t}^{T-1} \gamma^{\tau-t} R_{\tau+1} | S_t = s, A_t = a] \end{aligned}$$

Since the state value is related to the action value, the latter can be rewritten like this:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s') | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \end{aligned}$$

Optimal Policy With respect to $v_\pi(s)$ and $q_\pi(s, a)$, one is interested in finding the optimal policy π_* that maximizes both value functions.⁷ A strategy π' is defined as optimal iff the total expected reward is greater than or equal to that of another

⁷If π_* is the optimal policy that maximizes $v(s)$, then π_* also maximizes $q_\pi(s, a)$.

strategy π' for all states $\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s)$. From this follows the optimal state value function $v_*(s)$ and action value function $q_*(s, a)$:

$$\begin{aligned} v_*(s) &\doteq \max_{\pi} v_\pi(s) \\ &= \max_{a \in \mathcal{A}(s)} q_*(s, a) \\ \text{with } q_*(s, a) &\doteq \max_{\pi} q_\pi(s, a) \\ &= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \end{aligned}$$

$\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}(s)$.

Bellman Optimality Equations If $v_*(s)$ is optimal under π_* , then $v_*(s)$ must be equal to the expected return for the best action from that state, as indicated by the Bellman optimality equation:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_*(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_*(s')] \end{aligned}$$

Similarly, the optimal action-value function can be stated by the Bellman optimality equation:

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{4.1}$$

4.3 Algorithms

In the course of RL, $v_*(s)$ and $q_*(s, a)$ are unknown. However, both can be estimated based on past iterations, where one iteration is the trajectory $(S_t, A_t, S_{t+1}, R_{t+1})_{t=0}^{T-1}$. With further iterations i , the estimates converge to the optimal values $v_i(s) \rightarrow v_*(s) \wedge q_i(s, a) \rightarrow q_*(s, a)$ as $i \rightarrow \infty$.

Q Learning To improve the estimate of the action value function, one must update the old estimate $Q^{\text{old}}(S_t, A_t)$ with some other value to obtain the new estimate $Q^{\text{new}}(S_t, A_t)$. This update step has the generalized form: new estimate \leftarrow old estimate + learning rate \cdot (target – old estimate). Where (target – old estimate) is the temporal difference.

In terms of Q-learning, the old estimate is given by $Q^{\text{old}}(s, a)$ and the target is derived from the Bellman equation leading to the Q-learning formula

$$Q^{\text{new}}(S_t, A_t) \leftarrow Q^{\text{old}}(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q^{\text{old}}(S_t, A_t)) \quad (4.2)$$

Since the target maximizes the action-value of the next state with respect to the actions available, Q-learning is *off-policy*.

SARSA Rather than maximizing the action value of the next state with respect to the available actions, the state–action–reward–state–action (SARSA) algorithm takes the action that is considered best, i.e., it is *on-policy*.

$$Q^{\text{new}}(S_t, A_t) \leftarrow Q^{\text{old}}(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q^{\text{old}}(S_t, A_t))$$

4.4 Deep Q Networks

One drawback of reinforcement learning is that a lookup table, which maps the states and their respective actions to the value functions, is memory inefficient and even infeasible if the number of states and available actions is very large. For this reason, rather than trying to store each value individually, they are approximated.

Mnih et al. proposed deep Q networks (DQN). An algorithm that employs artificial neural networks (ANNs) to approximate action values [Mni+13]. One challenge, however, is how to train ANNs, since training requires a differentiable loss function to update the parameters of the network. But because the target is not directly available, it must be predicted by an ANN as well. Thus, y_t in

$$\mathcal{L}_t(\theta_t) = \mathbb{E}[(y_t - Q(S_t, A_t, \theta_t))^2]$$

is replaced by a feasible value, which is derived from the Bellman equation. This so-called target network is parameterized by θ_{t-1} :

$$y_t = \mathbb{E}[R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_{t-1}) | S_{t+1}]$$

As the loss function must be differentiated with respect to the current parameters $\nabla \boldsymbol{\theta}_t \mathcal{L}_t(\boldsymbol{\theta}_t)$, $\boldsymbol{\theta}_{t-1}$ of the target network is fixed. Ultimately, the current action value can be updated using Equation (4.2) – only adapted to be used with ANNs. DQN itself is an *off-policy* algorithm, since it learns the greedy strategy $a = \max_a Q(S_t, A_t; \boldsymbol{\theta})$, but still allows for exploratory behavior.

Study-Specific Theory

5.1 Incremental Metrics

Since the batch-format input to the first layer of the artificial neural network (ANN) is $\mathbf{X}_n \in \mathbb{R}^{d_{bn} \times d_{seq} \times d_{in}}$, the batch-format output is $\hat{\mathbf{y}}_n \in \mathbb{R}^{d_{bn} \times 1}$ with the corresponding targets $\mathbf{y}_n \in \mathbb{R}^{d_{bn} \times 1}$, where N gives the total number of batches with $n \in [1, \dots, N]$.¹

In this study, three metrics that are appropriate for a regression task are used: (1) coefficient of determination (R²), (2) naïve coefficient of determination (naïve R²), and (3) Pearson correlation coefficient (PCC).² In sample average notation with a bar line indicating the sample mean, these metrics are calculated as follows:

$$\begin{aligned}
 (1) : R^2 & \doteq \frac{SSE}{SST} = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2} \\
 & \doteq 1 - \frac{SSR}{SST} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \quad R^2 \in (-\infty, 1] \\
 (2) : \text{naïve } R^2 & \doteq 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i y_i^2}, \quad \text{naïve } R^2 \in (-\infty, 1] \\
 (3) : \rho(y, \hat{y}) & \doteq \frac{\sum_i ((y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}}))}{\sqrt{\sum_i (y_i - \bar{y})^2} \sqrt{\sum_i (\hat{y}_i - \bar{\hat{y}})^2}}, \quad \rho(y, \hat{y}) \in [-1, 1]
 \end{aligned}$$

The Problem with the Mean Value However, it is not possible to calculate the individual values for each batch and average them to obtain metric values for the entire dataset. This results from the fact that the mean value per batch differs between all batches and especially from the mean value of the whole dataset.

Hence, the mean must be retransformed into its summation notation: $\mu_x = \frac{1}{n_x} \sum_i x_i$. To allow for incremental updates with batch-format values, one has to adjust the formula slightly:

¹ $d_b \leftarrow d_{\text{batch size}} \wedge d_{seq} \leftarrow d_{\text{sequence length}} \wedge d_{in} \leftarrow d_{\text{input size}}$.

²The naïve R² is based only on the assumption of the naïve prediction that the next day's return is zero [GKX20]. However, when the targets are normalized to mean zero, there is little or no difference between the R² and the naïve R².

$$\mu(x)_{1:n} = \frac{1}{\sum_n d_{b_n}} \sum_{n=1}^N \sum_{i=1}^{d_{b_n}} x_{n,i}$$

From this results: if $n = N$, then $\mu(x)_{1:N} \equiv \mu_x$ – where μ_x is the mean of the whole dataset.³

In a similar fashion one can write the incremental variance⁴ in sample notation as:

$$\begin{aligned} \sigma(x)_{1:n}^2 &= \frac{1}{\sum_n d_{b_n}} (\sum_n \sum_{i=1}^{d_{b_n}} x_{n,i}^2 - \frac{1}{\sum_n d_{b_n}} \sum_n (\sum_{i=1}^{d_{b_n}} x_{n,i})^2) \\ &\doteq \frac{1}{\sum_n d_{b_n}} \text{error}(x)_{1:n} \end{aligned}$$

and the incremental covariance⁵ as:

$$\sigma(x, y)_{1:n} = \sum_n \sum_{i=1}^{d_{b_n}} x_{n,i} y_{n,i} - \frac{1}{\sum_n d_{b_n}} \sum_n (\sum_{i=1}^{d_{b_n}} x_{n,i} \sum_{i=1}^{d_{b_n}} y_{n,i})$$

Dropping the Mean Value from the Formulas Applying this logic to all three metrics above yields the three incremental versions, again in sample notation:^{6 7}

$$\begin{aligned} R_{1:n}^2 &\doteq 1 - \frac{\sum_n \sum_{i=1}^{d_{b_n}} (y_{n,i} - \hat{y}_{n,i})^2}{\text{error}(y)_{1:n}}, \quad R_{1:n}^2 \in (-\infty, 1] \\ \text{naïve } R_{1:n}^2 &\doteq 1 - \frac{\sum_n \sum_{i=1}^{d_{b_n}} (y_{n,i} - \hat{y}_{n,i})^2}{\sum_n \sum_{i=1}^{d_{b_n}} y_{n,i}^2}, \quad \text{naïve } R_{1:n}^2 \in (-\infty, 1] \\ \rho(y, \hat{y})_{1:n} &\doteq \frac{\sigma(y, \hat{y})_{1:n}}{\sqrt{\text{error}(y)_{1:n}} \sqrt{\text{error}(\hat{y})_{1:n}}}, \quad \rho(y, \hat{y})_{1:n} \in [-1, 1] \end{aligned}$$

³Incremental calculation has the advantage that it is much more memory efficient than storing all values and calculating the metric from scratch after each batch.

⁴

$$\sigma_x^2 \doteq \frac{1}{n} \sum_{i=1}^n (x_i - \mu_x)^2 = \frac{1}{n} \sum_i x_i^2 - \frac{2}{n} \mu_x \sum_i x_i + \mu_x^2 = \frac{1}{n} \sum_i x_i^2 - \mu_x^2 = \frac{1}{n} (\sum_i x_i^2 - \frac{1}{n} (\sum_i x_i)^2)$$

⁵

$$\begin{aligned} \sigma(x, y) &\doteq \sum_{i=1}^n ((x_i - \mu_x)(y_i - \mu_y)) = \sum_i (x_i y_i) - 2n \mu_x \mu_y + n \mu_x \mu_y \\ &= \sum_i (x_i y_i) - n \frac{1}{n} \sum_i x_i \frac{1}{n} \sum_i y_i = \sum_i (x_i y_i) - \frac{1}{n} \sum_i x_i \sum_i y_i \end{aligned}$$

⁶All incremental metrics in this section originated from the author of this thesis. The basic idea, however, is taken from the incremental Q-value algorithm described in [SB20, chapter 2].

⁷These three metrics are implemented for use with *PyTorch* by subclassing the base `Metric` from the Python package `torchmetrics`. See the attached code file `metrics.py` in the `fin_dl/torch` folder.

5.2 Loss Functions

Three loss functions are employed in the study of this thesis: (1) the mean squared error (MSE), (2) Huber loss (HL) and (3) Gaussian negative log likelihood loss (GaussianNLLLoss).

MSE The MSE is a typical loss used for regression problems to quantify the squared difference between the true and predicted values: $\mathcal{L} = \mathcal{L}^{\text{mse}}(\hat{y}, y) = \|\hat{y} - y\|^2$.

HL However, since the MSE is not robust, being sensitive to outliers, a second approach is to use a threshold for switching between the MSE and the mean absolute error (MAE), given by HL [Hub64]. Using the threshold δ leads to

$$\mathcal{L}_i = \mathcal{L}_i^{\text{Huber}}(\hat{y}_i, y_i) = \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, & \text{if } |\hat{y}_i - y_i| < \delta \\ \delta(|\hat{y}_i - y_i| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

GaussianNLLLoss The GaussianNLLLoss assumes that the ground-truth values are generated from a Gaussian distribution with both mean and variance being outputs of the ANN, where the predicted values function as the mean and the variance is given by a one vector or matrix that is updated during the optimization step.⁸

$$\mathcal{L} = \mathcal{L}^{\text{GaussianNLLLoss}}(\hat{y}, y, \sigma^2) = \frac{1}{2}(\log(\max(\sigma^2, \epsilon)) + \frac{(\hat{y} - y)^2}{\max(\sigma^2, \epsilon)}) + c$$

The use of the GaussianNLLLoss [NW94] is motivated by the fact that the targets (continuous compounding returns (CCRs)) are normally scaled using the quantile information.⁹

5.3 Activation Functions

Since the long short-term memory (LSTM) cell already has defined activation functions (Sigmoid and Tanh), the activation functions used in this work will be

⁸ ϵ is a stabilization factor and c a constant.

⁹The targets do not necessarily need to be scaled, but in this study, scaling the inputs as well as the targets lead to a more stable training.

related to the feedforward neural network (FNN)-architecture only. Besides the following formulas, all functions are illustrated in Figure 5.1.

SELU $\text{SELU}(x) = \text{selu}(x) \doteq \text{gain} \cdot (\max(x, 0) + \min(0, \alpha(\exp(x) - 1)))$ with gain and α being hyper-parameters defaulting to 1.67 and 1.05 in *PyTorch*.

Tanh $\text{Tanh}(x) = \tanh(x) \doteq \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$

CELU $\text{CELU}(x) = \text{celu}(x) \doteq \max(0, x) + \min(0, \alpha(\exp(\frac{x}{\alpha}) - 1))$ with the hyper-parameter α that defaults to 1.0 in *PyTorch*.

SiLU $\text{SiLU}(x) = \text{silu}(x) \doteq x \cdot \text{sigmoid}(x)$

Softplus $\text{Softplus}(x) = \text{softplus}(x) \doteq \frac{1}{\beta} \log(1 + \exp(\beta x))$ with the hyper-parameter β set to 1.0 in *PyTorch*.

Mish $\text{Mish}(x) = \text{mish}(x) \doteq x \cdot \tanh(\text{softplus}(x))$

ELU where α in *PyTorch* has the standard value 1.0.

$$\text{ELU}(x) = \text{elu}(x) \doteq \begin{cases} x, & \text{if } x > 0 \\ \alpha(\exp(x) - 1), & \text{otherwise} \end{cases}$$

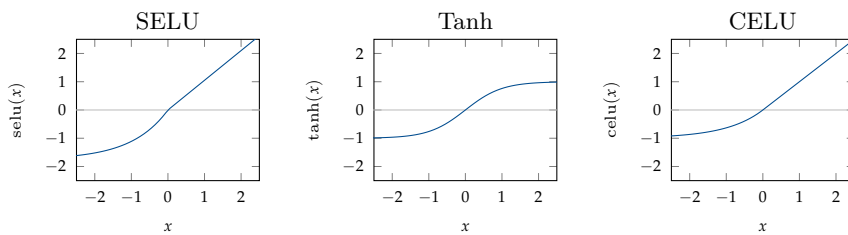


Fig. 5.1.: Activation Functions

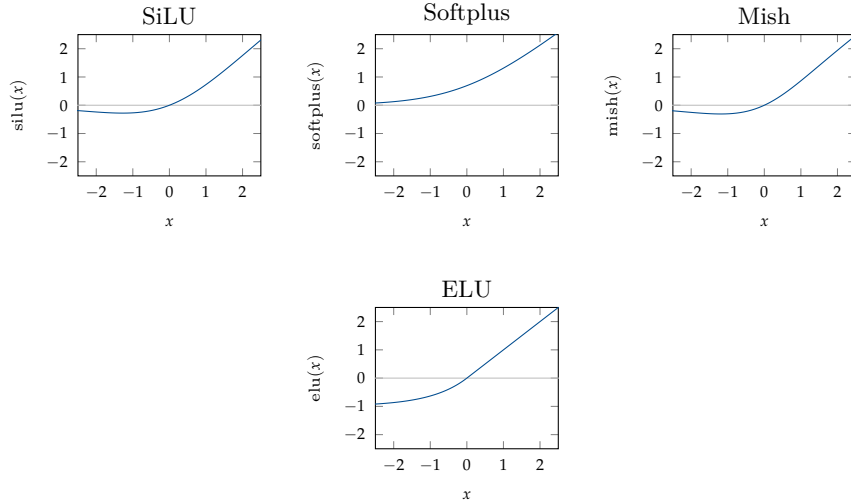


Fig. 5.1.: Activation Functions (continued)

Decision not to use the ReLU The widely used activation function rectified linear unit (ReLU) is not used in this study because the Hessian matrix was only a null tensor.¹⁰ Consider a simple linear function $f(x) = x$, which has the first derivative $\frac{d}{dx}f(x) = 1$ and the second derivative $\frac{d^2}{dx^2}f(x) = 0$, the latter indicates linearity.

If the Hessian matrix contains only zeros, this might indicate that the function from which it was calculated is linear. Nevertheless, this issue or anomaly and its implications cannot be explored further here. Consequently, ReLU and linear variants of it (like PReLU or LeakyReLU) are not used in this study.

5.4 Optimizers

Optimizers are a key component in training ANNs, as they can strongly influence the performance. Several optimization algorithms are available to adjust the updating behavior towards finding an optimal minimum. Since this is not a simple task and depends on many factors, different algorithms exist.

Crucially, for an optimization algorithm it is important not to get stuck in a local semi-optimal minima, but at the same time not to explore too much without finding one at all. An important parameter is the learning rate, which allows to skip semi-optimal local minima and to explore further. Its value can be high, e.g. at

¹⁰See the attached file `relu_hessian.py`. The same is true for other versions such as PReLU or LeakyReLU.

the beginning of the training, and is then reduced either within the optimization algorithm or by some kind of learning rate scheduler.

In this study, four optimizers are used: (1) adaptive gradient (AdaGrad), (2) AdaDelta, (3) adaptive moment estimation (Adam), and (4) resilient backpropagation (Rprop).^{11 12}

5.4.1 AdaGrad

AdaGrad is a stochastic optimization algorithm that automatically adjusts the learning rate γ to the parameters in $\boldsymbol{\theta}$. It adapts to the size of the parameters, such that parameters associated with more frequent features receive a smaller update compared to less frequent features [DHS11].

It takes as input the learning rate γ , the parameters $\boldsymbol{\theta}_t$ indexed to time, the weight decay λ , the learning rate decay η , the stabilization scalar ϵ , the gradients \mathbf{g}_t and the accumulated quadratic gradients \mathbf{G}_t ($\mathbf{G}_0 \leftarrow \mathbf{0}$) (see Algorithm 1).

Algorithm 1: AdaGrad

```

1 for  $t \leftarrow 1, \dots$  do
2    $\mathbf{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_K(\boldsymbol{\theta}_{t-1})$ 
3    $\tilde{\gamma} \leftarrow \frac{\gamma}{1+\eta(t-1)}$ 
4   if  $\lambda \neq 0$  then
5      $\mathbf{g}_t \leftarrow \mathbf{g}_t + \lambda \boldsymbol{\theta}_{t-1}$ 
6   end
7    $\mathbf{G}_t \leftarrow \mathbf{G}_{t-1} + \mathbf{g}_t^2$ 
8    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \tilde{\gamma} \frac{\mathbf{g}_t}{\sqrt{\mathbf{G}_t + \epsilon}}$ 
9 end
```

5.4.2 AdaDelta

The AdaDelta optimization algorithm is an enhancement of the AdaGrad optimization algorithm in that it allows learning rates to decay continuously during training, making it much smoother than AdaGrad [Zei12].

For each step t , the gradients are denoted by \mathbf{g}_t , the learning rate by γ , the weight decay by λ , the decay by ρ , the quadratic average gradients by \mathbf{v}_t ($\mathbf{v}_0 \leftarrow \mathbf{0}$) and

¹¹In the following, the strict matrix notation is abandoned in favor of computational matrix operations, which are common in programming languages such as Python.

¹²The following pseudocodes are adapted from [Pas+19].

the accumulated past gradients \mathbf{u}_t ($\mathbf{u}_0 \leftarrow \mathbf{0}$). At every time step t the following calculations given in Algorithm 2 are performed.

Algorithm 2: AdaDelta

```

1 for  $t \leftarrow 1, \dots$  do
2    $\mathbf{g}_t \leftarrow \nabla_{\theta} f_K(\theta_{t-1})$ 
3   if  $\lambda \neq 0$  then
4      $\mathbf{g}_t \leftarrow \mathbf{g}_t + \lambda \theta_{t-1}$ 
5   end
6    $\mathbf{v}_t \leftarrow \rho \mathbf{v}_{t-1} + (1 - \rho) \mathbf{g}_t^2$ 
7    $\Delta \mathbf{x}_t \leftarrow \frac{\sqrt{u_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} \odot \mathbf{g}_t$ 
8    $\mathbf{u}_t \leftarrow \rho \mathbf{u}_{t-1} + (1 - \rho) \Delta \mathbf{x}_t^2$ 
9    $\theta_t \leftarrow \theta_{t-1} - \gamma \Delta \mathbf{x}_t$ 
10 end

```

5.4.3 Adam

Adam is an extension of the stochastic gradient descent (SGD) optimization algorithms, AdaGrad and RMSProp and is well suited for non-stationary targets and problems with very noisy and/or sparse gradients [KB15].

In addition to the parameters θ_t indexed to time, it takes the learning rate γ , two exponential decay rates for the moment estimates β_1 and β_2 , and a weight decay λ as inputs.

Since it focuses on estimating moments, two moments \mathbf{m}_t and \mathbf{v}_t ($\mathbf{m}_0, \mathbf{v}_0 \leftarrow \mathbf{0}$) and their estimates $\hat{\mathbf{m}}_t$ and $\hat{\mathbf{v}}_t$ are calculated. If the loss is to be minimized, the optimization step is illustrated in Algorithm 3.

5.4.4 Rprop

Riedmiller and Braun proposed the Rprop algorithm, which focuses only on the sign of the gradients [RB93]. Here, at each time step t , for each element i in the gradients \mathbf{g}_t , it is determined whether the sign has changed with respect to the same element, but in the previous gradients \mathbf{g}_{t-1} . Since the algorithm depends only on the sign, the parameter updating step is not blurred by large gradient sizes, as can be the case with other adaptive learning methods.

Algorithm 3: Adam

```
1 for  $t \leftarrow 1, \dots$  do
2    $\mathbf{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_K(\boldsymbol{\theta}_{t-1})$ 
3   if  $\lambda > 0$  then
4      $\mathbf{g}_t \leftarrow \mathbf{g}_t + \lambda \boldsymbol{\theta}_{t-1}$ 
5   end
6    $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$ 
7    $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$ 
8    $\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1 - \beta_1^t}$ 
9    $\hat{\mathbf{v}}_t \leftarrow \frac{\mathbf{v}_t}{1 - \beta_2^t}$ 
10   $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \gamma \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}$ 
11 end
```

The optimization step further takes a pair of increase and decrease factors $\eta_{+/-}$ and a pair of minimum and maximum allowed step sizes $\Gamma_{\min/\max}$ as inputs with the pseudocode given by Algorithm 4.

Algorithm 4: Rprop

```
1 for  $t \leftarrow 1, \dots$  do
2    $\mathbf{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_K(\boldsymbol{\theta}_{t-1})$ 
3   forall Parameter gradients  $\mathbf{g}_t^i \in \mathbf{g}_t \wedge$  single gradient  $\mathbf{g}_t^{ijk}$  do
4     if  $\mathbf{g}_{\text{previous}}^{ijk} \mathbf{g}_t^{ijk} > 0$  then
5        $\eta_t^{ijk} \leftarrow \min(\eta_{t-1}^{ijk} \eta_+, \Gamma_{\max})$ 
6     else if  $\mathbf{g}_{\text{previous}}^{ijk} \mathbf{g}_t^{ijk} < 0$  then
7        $\eta_t^{ijk} \leftarrow \max(\eta_{t-1}^{ijk} \eta_-, \Gamma_{\min})$ 
8        $\mathbf{g}_t^{ijk} \leftarrow 0$ 
9     else
10       $\eta_t^{ijk} \leftarrow \eta_{t-1}^{ijk}$ 
11    end
12  end
13   $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boldsymbol{\eta}_t \text{sign}(\mathbf{g}_t)$ 
14   $\mathbf{g}_{\text{previous}} \leftarrow \mathbf{g}_t$ 
15 end
```

5.5 Incremental Higher-Order Derivatives

When calculating higher order derivatives such as the Jacobian and Hessian matrices, the question that arises is how to accomplish this for the entire dataset, since both

provide the gradients with respect to one input (observation) vector. However, the dataset contains many observations grouped into batches, and each batch may even contain mini-batches of sequenced observations.

Consequently, the Jacobian and Hessian matrices have to be some kind of reduced version of all the individual matrices, which can be achieved, for example, through summation. Similar to [KNR22], it is possible to accumulate the squared Jacobian and Hessian to draw the square root later. This has the advantage of positive and negative values not cancelling each other out. However, the disadvantage is a loss of information, since all values are positive. Adding the sign per element of the normally accumulated Jacobian and Hessian, where positive and negative values can cancel each other out, might help.

To keep memory requirements low, an incremental version can be used that iterates over all (mini-)batches per dataset and accumulates the squared and normal Jacobian and Hessian matrices as well as the number of individual Jacobian and Hessian matrices. From this, the first and second moments can be calculated, as seen in Algorithm 5 and Algorithm 6.

Algorithm 5: Incremental mean and variance of Jacobians

Input : f_K, \mathbf{X} with $\mathbf{X}_n \in \mathbb{R}^{d_b \times d_{\text{seq}} \times d_{\text{in}}}$
Initialize: $\mathbf{\Sigma} \leftarrow \mathbf{0} \in \mathbb{R}^{d_{\text{in}}}, \mathbf{\Sigma}^2 \leftarrow \mathbf{0} \in \mathbb{R}^{d_{\text{in}}}, \text{len} \leftarrow 0$

- 1 **forall** $n \in [1, \dots, N]$ **do**
- 2 $\mathbf{J}_n \leftarrow \text{jacobian}(f_K, \mathbf{X}_n)$
- 3 $\mathbf{J}_n^* \leftarrow \text{einsum}(\dots \text{bij} \rightarrow \text{bij}, \mathbf{J}_n)$
- 4 $\mathbf{\Sigma} \leftarrow \mathbf{\Sigma} + \text{einsum}(\text{bij} \rightarrow \text{j}, \mathbf{J}_n^*)$
- 5 $\mathbf{\Sigma}^2 \leftarrow \mathbf{\Sigma}^2 + \text{einsum}(\text{bij} \rightarrow \text{j}, (\mathbf{J}_n^*)^2)$
- 6 $\text{len} \leftarrow \text{len} + d_b \cdot d_{\text{seq}}$
- 7 **end**
- 8 $\sigma^2 \leftarrow \frac{1}{\text{len}}(\mathbf{\Sigma}^2 - \frac{1}{\text{len}}(\mathbf{\Sigma})^2)$
- 9 $\boldsymbol{\mu} \leftarrow \frac{1}{\text{len}}\mathbf{\Sigma}$
- 10 $\boldsymbol{\mu}_{\text{signed}} \leftarrow \frac{1}{\text{len}}\text{sign}(\mathbf{\Sigma})\sqrt{\mathbf{\Sigma}^2}$
- 11 $\boldsymbol{\mu}_{\text{unsigned}} \leftarrow \frac{1}{\text{len}}\sqrt{\mathbf{\Sigma}^2}$

Algorithm 6: Incremental mean and variance of Hessians

Input : f_K, X with $X_n \in \mathbb{R}^{d_b \times d_{\text{seq}} \times d_{\text{in}}}$ and $X^{\text{mini}} \in \mathbb{R}^{d_{\text{seq}} \times d_{\text{in}}}$

Initialize: $\Sigma \leftarrow \mathbf{0} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$, $\Sigma^2 \leftarrow \mathbf{0} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$, $\text{len} \leftarrow 0$

```
1 forall  $n \in [1, \dots, N]$  do
2   forall  $X^{\text{mini}} \in X_n$  do
3      $H_n \leftarrow \text{hessian}(f_K, X^{\text{mini}})$ 
4      $H_n^* \leftarrow \text{einsum}(\text{abcdef} \rightarrow \text{bcf}, H_n)$ 
5      $\Sigma \leftarrow \Sigma + \text{einsum}(\text{bij} \rightarrow \text{ij}, H_n^*)$ 
6      $\Sigma^2 \leftarrow \Sigma^2 + \text{einsum}(\text{bij} \rightarrow \text{ij}, (H_n^*)^2)$ 
7      $\text{len} \leftarrow \text{len} + d_b \cdot d_{\text{seq}}$ 
8   end
9 end
10  $\sigma^2 \leftarrow \frac{1}{\text{len}}(\Sigma^2 - \frac{1}{\text{len}}(\Sigma)^2)$ 
11  $\mu \leftarrow \frac{1}{\text{len}}\Sigma$ 
12  $\mu_{\text{signed}} \leftarrow \frac{1}{\text{len}}\text{sign}(\Sigma)\sqrt{\Sigma^2}$ 
13  $\mu_{\text{unsigned}} \leftarrow \frac{1}{\text{len}}\sqrt{\Sigma^2}$ 
```

Empirical research

This empirical chapter is two-folded: The first part (Section 6.2) deals with predicting the logarithmic returns of one target company with regard to three individual rolling windows, whereas the second part (Section 6.3) applies reinforcement learning (RL) given a single stock trading environment, to compare the total reward of a single and a multi-agent, and both with a buy-and-hold strategy.

6.1 Data

Data Basis The database for the following studies contains the typical daily OCHLV (Open, Close, High, Low, Volume) values for all companies in the S&P500 from the beginning of 2010 to the end of 2021. In addition, technical indicators (TIs) are calculated per company using *Technical Analysis Library* (TA-Lib).^{1 2}

Processing In a second step, each time series per company is tested for non-stationarity using the augmented Dickey-Fuller test. If the test indicates non-stationarity,³ the difference is taken and then retested for non-stationarity. This process is repeated a maximum of seven times, and if the time series is still non-stationary, it is dropped. The same applies to time series that contain more than 5 % NaNs (not a number) or zeros.⁴

Trimming Since the entire dataset is very large, tuning, fitting, and computing the higher-order derivatives for each rolling window is computationally very demanding. Thus, the dataset had to be reduced and trimmed to cover the time horizon from the beginning of 2016 to the end of 2020. Additionally, the focus was placed on the *real estate* sector, with Public Storage (PSA) being the target company.⁵

¹ta-lib.org

²Please take a look at `technical_indicators.py` for all calculated TIs as well as the hyper-parameters (HPs) used.

³P-value threshold is set to 5 %.

⁴The final set of 72 features used is given in Table B.2.

⁵The *real estate* sector was selected because it has a low variance (see Table B.1) with respect to the target variable $\text{diff}(\log(\text{close}))$ and at the same time contains 30 companies, which seems

Trimming and reducing the dataset is motivated by the following reasons:

1. The main reason is that using the entire dataset to tune and train the artificial neural network (ANN) even on a graphics processing unit (GPU) is computationally expensive, especially in terms of computing the Hessian for all data points can take hours or even days.
2. Focusing on a few companies and only one company as a target allows the number of tuning trials to be increased, which in turn opens up the possibility of testing the performance of more and different architectures, loss and activation functions, and optimizers to see if one can achieve a positive naïve coefficient of determination (naïve R²) and perhaps find a general model architecture that works best for the given task and data. Moreover, testing more model architectures gives a sharper picture regarding the overall adequacy of using ANNs for the task at hand. Especially if the results are not very good, this could be an indicator to focus on the data rather than the technical aspects of deep learning (DL).⁶
3. Using all the data from the S&P 500 may result in a model that predicts the mean, or generally a very uniform sequence of values, as the characteristics of each stock's data disappear, i.e., cancel each other out. However, since the data of other companies provide further and hopefully more valuable information for forecasting than just the data of the target companies, the model is trained on companies of the same sector as the target company (*real estate*).⁷

Target Variable Because the logarithmic return of *close* with respect to PSA is the target feature to be predicted, the raw *close* as well as the logarithmic return of *close* over the entire time horizon is presented in Figure 6.1.

The closing price shows interesting upward and downward characteristics that are well suited for the RL-part in Section 6.3, as the price does not follow a continuous upward trend, which can allow for transactions (buy, sell) that may lead to a higher

appropriate for training – not too meager, but computationally practical. Similarly, the company PSA was selected because it has the lowest variance among all companies in the *real estate* sector (see Table B.4). Focusing on the variance is motivated by the hypothesis that more stable values lead to more stable and eventually better predictive results.

⁶Selecting only one company as a target is mainly motivated by the reduced complexity, making it easier to interpret and show/plot the results. If the results appear promising, extending the data to more or even all companies and observations can be easily done by changing only a few lines of code. It should therefore be seen as a test balloon.

⁷Companies are listed in Table B.3.

total return over the entire period compared to a simple buy-and-hold strategy.⁸ In addition, the data includes a short market shock (steep downturn of *close* and high positive and negative continuous compounding returns (CCRs)) in March 2020. So it will be interesting to observe whether this feature affects the RL-agent, since it is an outlier period with respect to the logarithmic return and falls exactly in the evaluation period.

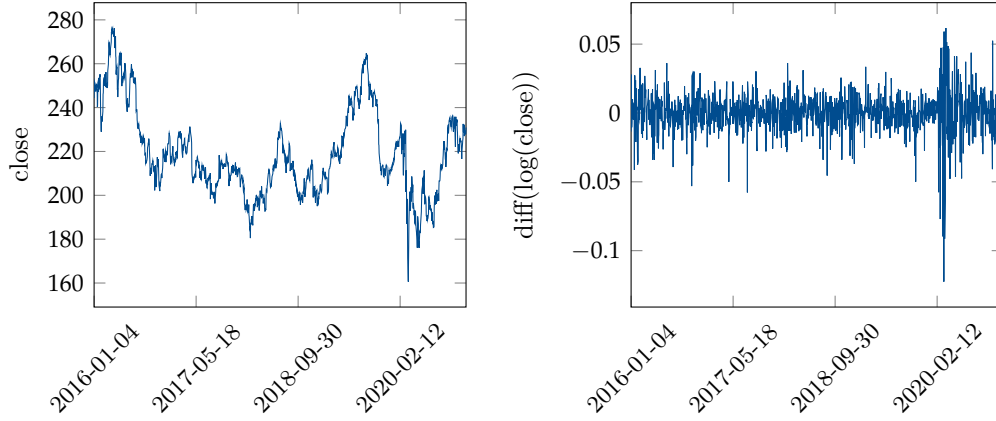


Fig. 6.1.: PSA *close* and $\text{diff}(\log(\text{close}))$

Box plots of $\text{diff}(\log(\text{close}))$ Since all firms associated with the *real estate* sector are used to train the ANN in Section 6.2, box plots are given in Figure 6.2 to show the quantiles and whiskers for PSA and the *real estate* sector with respect to the target variable $\text{diff}(\log(\text{close}))$. It can be seen that both box-plots are very similar, suggesting that PSA is not an outlier, but can be strongly represented by the descriptive statistics of the $\text{diff}(\log(\text{close}))$ data for the sector as a whole.⁹

Correlation Additionally, one can conduct a correlation analysis to observe whether the input data is suitable for predicting the target variable. Figure 6.3 shows the correlation between the variable $\text{diff}(\log(\text{close}))$ of all companies over the entire trimmed horizon. As can be seen more compactly in Table 6.1, the correlation is 59 % on average, which is the same as the median. Thus, the correlation between all $\text{diff}(\log(\text{close}))$ time series can be considered medium to high.

An in-depth look at the correlation of PSA's $\text{diff}(\log(\text{close}))$ with the time series of all other companies is provided in Table 6.2 as a descriptive format. It indicates a similarly high to medium correlation (with mean 53 % and median 54 %), suggesting

⁸It is assumed that it is more difficult to beat a buy-and-hold strategy in a bull market than in a bear market.

⁹The data of PSA is included in the training and validation dataset.

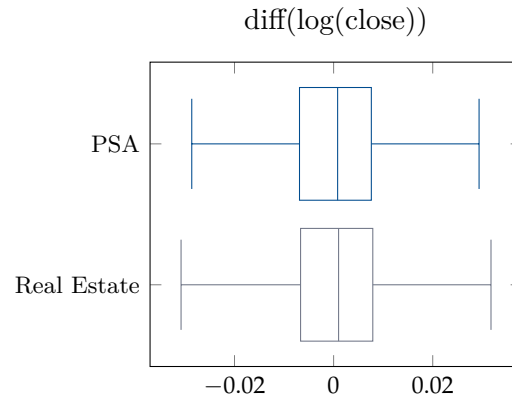


Fig. 6.2.: Box-plots of PSA and all companies associated with the sector *real estate* (incl. PSA)

that these time series may, at least at first glance, be suitable to predict $\text{diff}(\log(\text{close}))$ of PSA.

Tab. 6.1.: Descriptive statistics of $\text{diff}(\log(\text{close}))$ correlation – [2016, 2021)

mean	std	min	25%	50%	75%	max
0.59	0.14	0.22	0.50	0.59	0.68	0.92

Tab. 6.2.: Descriptive statistics of correlation between the target ($\text{diff}(\log(\text{close}))$ of PSA) and $\text{diff}(\log(\text{close}))$ of all other companies – [2016, 2021)

mean	std	min	25%	50%	75%	max
0.53	0.11	0.30	0.46	0.54	0.59	0.84

Looking at the correlation between PSA's $\text{diff}(\log(\text{close}))$ and all features of all companies, the correlation with respect to the descriptive statistics has decreased, as can be seen in Table 6.3. The mean is now only about 21 % and the median is about 25 %, suggesting that some variables may be more important than others with respect to the prediction task. But because correlation does not necessarily imply causality, approaches to attribute importance, such as higher-order derivatives, can be more informative in this regard as they quantify the influence of individual features with respect to the output.

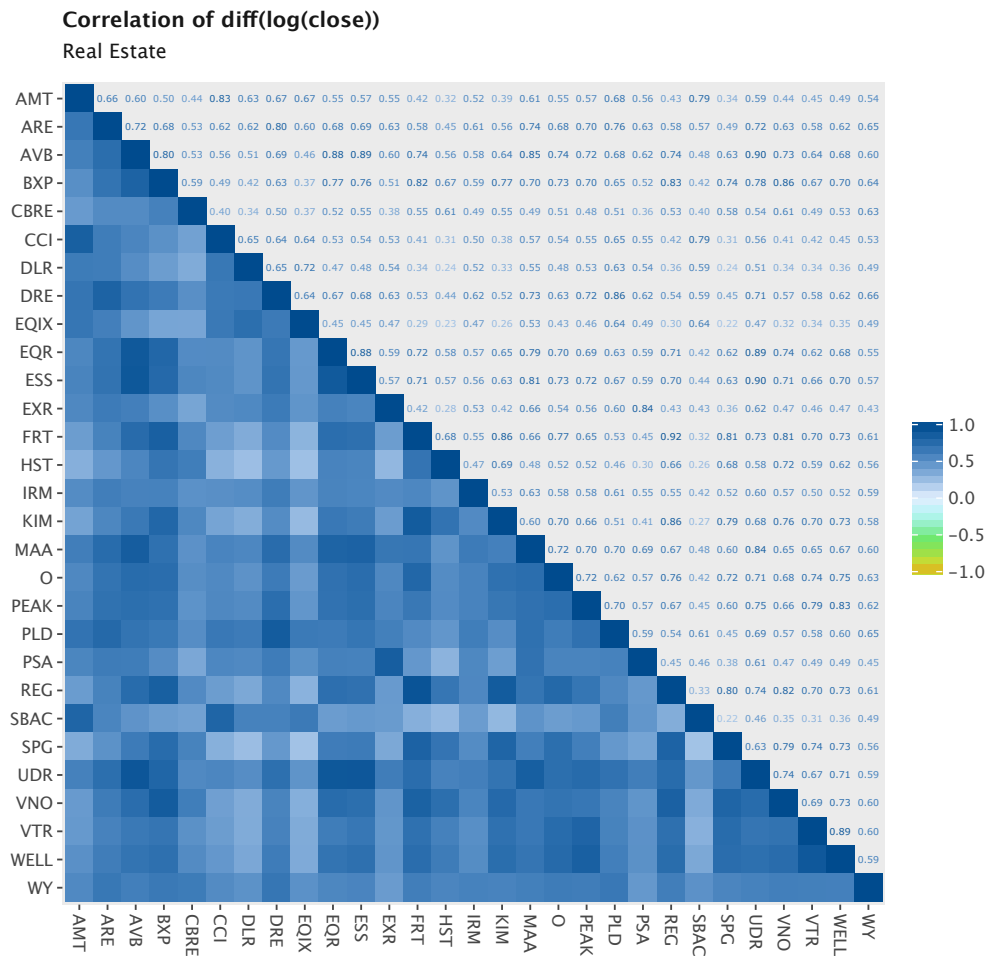


Fig. 6.3.: Correlation between all companies of the real estate sector – diff(log(close)) – [2016, 2021)

6.2 Supervised Learning to Predict the CCR

It is the aim of this section to show whether the logarithmic return of the next day can be predicted using historical stock data and technical indicators. Based on the results, it can then be concluded if the efficient market hypothesis (EMH) holds or not. To obtain a representative overview, three windows are used. Each window has its own tuned model that predicts the $diff(log(close))$ of PSA.

6.2.1 Study Design

Rolling Windows Each window contains three years of data, each one year apart, as shown in Table 6.4.

Tab. 6.3.: Descriptive statistics of correlations between the target ($\text{diff}(\log(\text{close}))$) of PSA) and all other features – [2016, 2021)

mean	std	min	25%	50%	75%	max
0.21	0.25	-0.64	-0.00	0.25	0.40	0.84

Tab. 6.4.: Rolling windows, with each containing three years of data

No.	Label	Start	End
1	[2016M01, 2019M01)	2016-01-01	2018-12-31
2	[2017M01, 2020M01)	2017-01-01	2019-12-31
3	[2018M01, 2021M01)	2018-01-01	2020-12-31

Per window, the data is split so that the training set contains the first 65 %, the validation set contains the next 20 %, and the testing set contains the remaining last 15 % of the entire window data.

Batching is applied in sizes of ten (trading days) with a sequence length of five (trading days) per mini-batch. The target variable is $\text{diff}(\log(\text{close}))$, which is shifted back by one time step.¹⁰

Each window conducts 75 tuning trials with the objective to maximize the naïve R2 of the validation dataset. Per trial, one artificial neural network architecture and several hyper-parameters are drawn. After these 75 trials, the model with the highest objective value is selected and fitted again to then predict the $\text{diff}(\log(\text{close}))$ of PSA given the testing dataset.

Tuning Training an ANN requires at least three main parts: (1) an ANN architecture with layers that have parameters to be trained, (2) a loss function that is differentiable to compute the gradients with respect to the parameters, and (3) an optimizer that takes the gradient signals and updates the parameters.

Each of the three parts has different HPs that can be optimized to profile the effects on the loss. Tuning, then, is a method of optimizing the HPs of a model with respect to an objective, which can be the loss or any other metric.^{11 12}

¹⁰Each feature per mini-batch contains $(x_{t-4}, x_{t-3}, x_{t-2}, x_{t-1}, x_t)$ and the respective target mini-batch contains only y_{t+1} .

¹¹One could also tune the HPs of the dataset such as the observations per batch, the batch size, and the sequence length. However, this is not done here since the focus of hyper-parameter tuning (HPT) is limited to the three parts of ANN training.

¹²The sampling space of the hyper-parameters is given in Table B.5 and Table B.6.

Architecture First of all, the model can be a feedforward neural network (FNN) or long short-term memory (LSTM) network.¹³ Each of these types has a predefined layer stack that includes the main linear layer and the activation function, in case of the FNN, or the LSTM cell, and a regularization layer. The latter is either a dropout layer, a batch regularization layer, or no regularization layer at all.

The HPs are drawn in the following order:

1. Determine the ANN type: either FNN or LSTM.
2. Draw the number of stacks N_{stack} .
3. Determine whether to use the dropout layer (with the HP p representing the dropout probability of setting an element to zero), the batch normalization layer (has two HP: ϵ and momentum) or build the ANN without regularization modules.
4. If ANN-type=FNN: draw the output sizes of \mathbf{A} and an activation function. If ANN-type=LSTM: draw the hidden size of each LSTM cell.

Optimizer Each trial selects an optimizer from the four available (see Section 5.4). It is also set up with its default parameters or values drawn by the tuning sampler.

6.2.2 Results

With each rolling window, 75 tuning trials are conducted in the manner described above using *Optuna's TPE (Tree-structured Parzen Estimator) algorithm* [Aki+19] with a maximum of five training and validation epochs per trial and 100 epochs for fitting the best model after tuning.

An early trial stopping mechanism is attached to the tuning to stop unpromising trials, while the training process of the best model uses an early stopping mechanism to prevent overfitting, combined with a learning rate scheduler.¹⁴ ¹⁵

¹³To be precise, because each mini-batch contains sequences, it is a time window-based FNN.

¹⁴All three monitor the naïve R2 of the validation set. Early trial pruning stops the trial immediately if the metric is below -2.5 or after three epochs if it has not improved. The learning rate scheduler reduces the learning rate by the default factor of 0.1 if the metric has not improved after three epochs, while training stops completely after five epochs if the metric has not improved.

¹⁵The study was conducted on a Google Cloud virtual machine (VM) with an NVIDIA® T4 GPU.

Window Models Table 6.5 gives the key properties of the best model, i.e., the model that showed the highest naïve R2 for the validation dataset during tuning and was taken for the inference task.^{16 17}

Tab. 6.5.: Properties of the best model per rolling window.

Window	Model	N_{stacks}	Regularization	Optimizer	Loss	Activation
1	FNN	4	Dropout	Adadelta	MSE	SiLU
2	LSTM	6	None	Adadelta	MSE	
3	LSTM	4	None	Adagrad(default_args)	Huber	

Window Details The most important part is the performance of each model with respect to the testing dataset. A look at Table 6.6 indicates that the performance of all models is not very high. All coefficient of determination (R2) values are negative, which means that the models predict even worse values compared to just predicting the mean of the target, which is close to zero. Same goes for the naïve R2 values, which are all close to zero except for one positive value of 0.37 % (second window). It is possible to see the reason for this inadequate performance by comparing the target values with the predicted values (see Figure 6.4).

Tab. 6.6.: Metrics of the best model with regard to the testing dataset

Window	Interval	R2	naïve R2	PCC
1	[2016M01, 2019M01)	-0.0009	-0.0009	0.0133
2	[2017M01, 2020M01)	-0.0054	0.0037	0.1022
3	[2018M01, 2021M01)	-0.0314	-0.0054	0.1326

For the first two windows, the models tend to predict values that are close to zero and thus close the mean value of the target. It follows that the models are not able to partially explain the variance of the target values. Looking at the predictions of the last model, the picture has changed – variance is visible. It even has the highest Pearson correlation coefficient (PCC) of about 13 %, but is not able to explain the variance of the target values, too.

¹⁶Highest validation score of naïve R2 during tuning: first window (0.0013), second window (-0.0019) and third window (0.0028).

¹⁷These models have different architectures, losses, and optimizers. Therefore, it is not possible to generalize with respect to the individual components and HPs that are best suited for the prediction task. The complete model architecture with dimensions per layer is given in the appended folder under `best_model/architecture`.

The obvious conclusion from these results is that the first EMH-form holds. One could even interpret that models one and two intuitively understood the first EMH-form and martingale property that the next day's return is zero.¹⁸

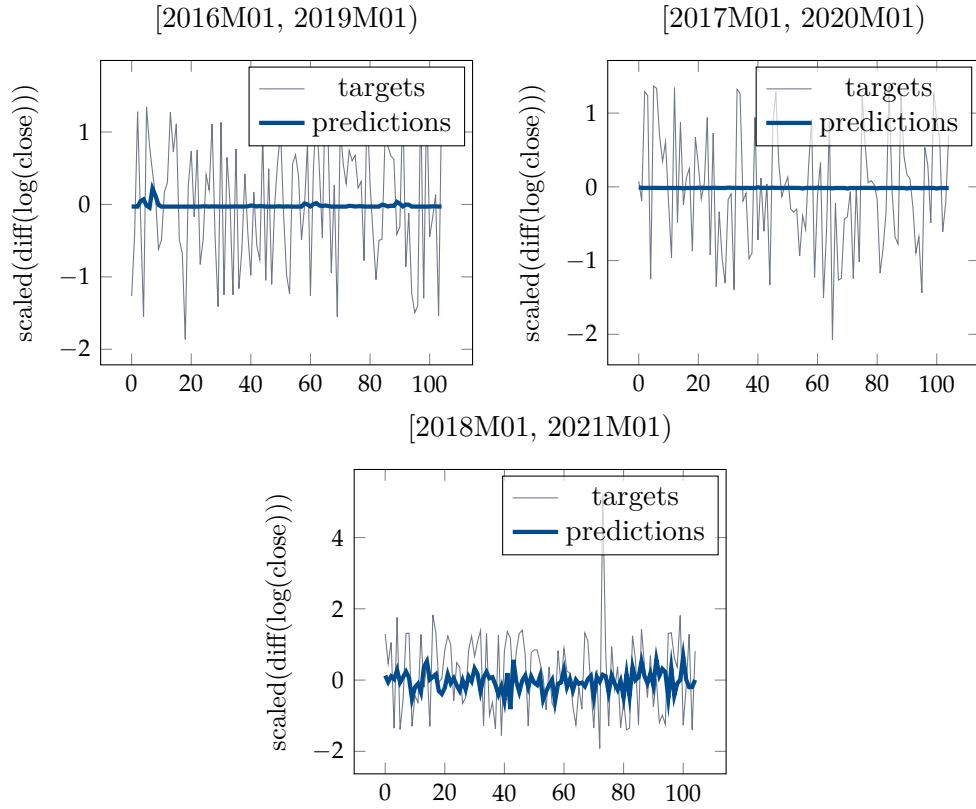


Fig. 6.4.: Predicted vs. target values for the given window (with the testing dataset containing 15 % of the window's data)

Neglecting the first two models, the last one is still the most promising, as it has the highest variability in terms of its predictions and thus has a greater chance of being able to outperform the mean and explain some of the variance of the target. This can be seen by looking at the evaluation (batch) history illustrated in Figure A.3.¹⁹ The naïve R2 is positive until the seventh batch, with the highest value of 0.21 (21 %). The curve of the R2 is a bit lower, which is (only) positive for the first three batches, reaching a maximum value of 0.17 (17 %). Finally, the PCC shows a similar curvature, starting high and ending low, with a maximum value of 0.68 (68 %).

While the last model's batch histories of all metrics (Figure A.3) start high and end low, the batch histories of all metrics with respect to the first and second

¹⁸However, some caution is warranted here since the target values are scaled and do not represent raw logarithmic returns.

¹⁹The plot of the first and second models is shown in Figure A.1 and Figure A.2, respectively.

show exactly the opposite curvature (see Figure A.1 and Figure A.2). As such, the implications regarding evaluation length are ambiguous, as the first two models benefit from a longer horizon, unlike the last model. Nevertheless, long evaluation horizons can be problematic because the parameters of the model do not learn the true values of the already predicted ones. For this reason, a shortening of the evaluation horizon could be beneficial.

Feature Importance Considering the feature importance of models that do not perform well with respect to the Jacobian and Hessian matrices is questionable, since both compute the derivative of the model's output with respect to the inputs. But if the predictions are not good, one might also question the results of the Jacobian and Hessian matrices.

However, since the third model shows variance in its predictions and good intermediate results at least for the first batches, the Jacobian and the diagonal Hessian of the whole dataset are briefly discussed. Since the Hessian is only calculated for the testing dataset due to computational limitations, only the Jacobian of the testing set is discussed here to make both comparable.

Table 6.7 shows the five main features in terms of variance and signed mean per Jacobian and Hessian matrix, all of which have the first two features in common, i.e., the *variance (VAR)* of *close* and the first difference of *exponential moving average (EMA)*. The following three ranks share the same four features (all as first difference values): *rate of change ratio – price/prevprice (ROCR)*, *average true range – volatility (ATR)*, *moving average (MA)*, and *volume*.²⁰

In summary, the variance/volatility and momentum/moving average indicators play an important role in predicting the log return of the closing price.²¹

6.2.3 Summary & Discussion

The key finding must be that with the data and models used, it is not possible to predict the next day's logarithmic return, which supports the first form of the efficient market hypothesis. This indicates that the historical stock data and technical indicators are irrelevant for the inference task, since the next day's price is

²⁰The range of values with respect to the signed mean of the Hessian matrix is between $2 \cdot 10^{-5}$ and $1 \cdot 10^{-6}$. The values are close to zero, but not exactly zero. Whether such small values still indicate nonlinearity of f_K is up for debate, but cannot be answered in this study.

²¹All results regarding the importance of the features for all windows can be found in the attached folder `feature_importance`.

Tab. 6.7.: Feature importance for [2018M01, 2021M01) – testing dataset

Rank	$\mu_{\text{signed}}(J)$	$\sigma^2(J)$	$\mu_{\text{signed}}(\mathbf{H}_{\text{diag}})$	$\sigma^2(\mathbf{H}_{\text{diag}})$
1	VAR	VAR	VAR	VAR
2	diff(EMA)	diff(EMA)	diff(EMA)	diff(EMA)
3	diff(ROCR)	diff(ATR)	diff(ATR)	diff(ATR)
4	diff(ATR)	diff(ROCR)	diff(MA)	diff(MA)
5	diff(MA)	diff(MA)	diff(volume)	diff(volume)

given by today’s price, and the next day’s return is expected to be zero (martingale property).

Another important result is the low variance of the model’s output. In terms of risk behavior, the models – at least the first two – can be described as risk averse, while the third model is more risk affine and was even able to show strong results for the first few predicted values. However, the overall performance is not convincing. So it seems to be more a question about whether the input is causally related to the target or whether the model configuration is not optimal. Since three windows and 225 tuning experiments with different architectures and hyper-parameters could not show solid results, it is likely the causal relationship that needs to be focused on rather than the technical side of deep learning.

From the feature importance of the last model with respect to the test dataset, one can cautiously infer that variance/volatility and momentum/moving average based indicators are valuable.

6.3 Trading via Reinforcement Learning

RL is focused exclusively on *PSA*’s own data and the RL algorithm used is deep Q networks (DQN), which is provided by *rllib* [Lia+18].

In this section, the aim is to show if an RL-agent can generally outperform a simple buy-and-hold strategy as well as to demonstrate whether a multi-agent environment is superior to a single-agent environment.

6.3.1 Study Design

At the beginning, the environment \mathcal{E} must be defined. \mathcal{E} is a single stock trading environment that has access to the stock data (OCHLV as well as the technical indicator data) \mathbf{X}^{PSA} at time step t with $t \in [1, \dots, T]$.

Since \mathcal{E} is a single stock trading environment, the agent takes a position $P_t \in \mathcal{P} = \{\text{out}, \text{long}\}$ and can perform an action $A_t \in \mathcal{A}(p) \subseteq \mathcal{A} = \{\text{buy}, \text{sell}, \text{none}\}$. Depending on the action, the position may change, which is given by the two options: $P_t = \text{out}, A_t = \text{buy} \rightarrow P_{t+1} = \text{long} \wedge P_t = \text{long}, A_t = \text{sell} \rightarrow P_{t+1} = \text{out}$. The action $A_t = \text{none}$ is a passive action, while the other two are active actions.

In addition, \mathcal{E} does not allow for a cash account and the option to invest a portion in shares and keep the other portion as cash. Beyond that, short selling of shares is not possible, since the agent has no alternative investment options and thus no possible opportunity costs given a single share. It is also assumed that the agent can buy or sell a share at the given closing price of t .

Whether an action was good or bad is indicated by the reward R_{t+1} , which can be positive or negative and is derived from the continuous compounding return of *close* with $r_{t+1} = \log \frac{x_{t+1}^{\text{close}}}{x_t^{\text{close}}}$.²² The last part missing is the state information S_t that is given by the observation and thus stock data $\mathbf{x}_t^{\text{PSA}} = (\mathbf{x}_t^{\text{PSA}}; \text{OCHLV}, \mathbf{x}_t^{\text{PSA}}; \text{TI})$.²³

Single-Agent The single-agent has a positional information $P_t \in \mathcal{P} = \{\text{out}, \text{long}\}$ and takes actions $A_t \in \mathcal{A}(p) \subseteq \mathcal{A} = \{\text{buy}, \text{sell}, \text{none}\}$. Thus, the environment \mathcal{E} stated above has not changed.

Multi-Agent The single-agent as defined here has one weakness, and that is noise. This noise is caused by having three actions available at each state and time step t , while only one action results in a position change. For example, with $P_t = \text{out}$, the agent can only transition to $P_{t+1} = \text{long}$ taking $A_t = \text{buy}$, such that $A_t = \text{sell}$ and $A_t = \text{none}$ result in the same position $P_{t+1} = \text{out}$ and the same reward R_{t+1} (see Equation (6.1)).

²²Additionally, transaction fees are charged when changing the position, such that when $P_t = \text{out} \wedge P_{t+1} = \text{long}$, x_t^{close} is adjusted to $x_t^{\text{close}*} = x_t^{\text{close}}(1 + \text{fee}^{\text{buy}})$ and if $P_t = \text{long} \wedge P_{t+1} = \text{out}$, $x_t^{\text{close}*} = x_t^{\text{close}}(1 - \text{fee}^{\text{sell}})$ with $\text{fee}^{\text{buy}} = 0.02 \wedge \text{fee}^{\text{sell}} = 0.02$.

²³Since the implications regarding the feature importance of the previous section are not strong, all TIs are used.

$$\begin{aligned}
P_t = \text{out} : A_t = \text{buy} \rightarrow P_{t+1} = \text{long} \wedge A_t = \text{none}, A_t = \text{sell} \rightarrow P_{t+1} = \text{out} \\
P_t = \text{long} : A_t = \text{sell} \rightarrow P_{t+1} = \text{out} \wedge A_t = \text{none}, A_t = \text{buy} \rightarrow P_{t+1} = \text{long}
\end{aligned} \tag{6.1}$$

To reduce this noise and amplify the signal per action A_t , all positions are represented by an individual agent. This leads to a new action definition $\mathcal{A}(p) \subset \mathcal{A} = \{\text{buy}, \text{sell}, \text{none}\}$, with $\mathcal{A}(P_t = \text{out}) \in \mathcal{A}^{\text{out}} = \{\text{buy}, \text{none}\}$ and $\mathcal{A}(P_t = \text{long}) \in \mathcal{A}^{\text{long}} = \{\text{sell}, \text{none}\}$ such that $\mathcal{A}^{\text{out}}, \mathcal{A}^{\text{long}} \subset \mathcal{A}$ with $\mathcal{A}^{\text{out}} \cap \mathcal{A}^{\text{long}} = \emptyset$.

In contrast to a single-agent, each action per set \mathcal{A}^{out} and $\mathcal{A}^{\text{long}}$ is associated with a unique position P_{t+1} and a reward R_{t+1} , resulting from taking an available action A_t in S_t given P_t :

$$\begin{aligned}
P_t = \text{out} : A_t = \text{buy} \rightarrow P_{t+1} = \text{long} \wedge A_t = \text{none} \rightarrow P_{t+1} = \text{out} \\
P_t = \text{long} : A_t = \text{sell} \rightarrow P_{t+1} = \text{out} \wedge A_t = \text{none} \rightarrow P_{t+1} = \text{long}
\end{aligned}$$

Performance Measure For the purpose of comparing an agent's performance, the baseline is given by the cumulative logarithmic return, which is just the cumulative return if one buys the stock at time $t = 0$ and holds it for the entire time period. An agent is evaluated according to its total return over the entire time horizon, which is given by the cumulative realized reward. An additional performance measure is the cumulative optimal reward, which is simply the reward when taking the optimal action in S_t and P_t . It is always positive and indicates whether an agent's action was non-optimal or optimal.

$$\begin{aligned}
R_{t+1}^{\text{realized}} &= \begin{cases} \text{sign}(P_{t+1}) \log\left(\frac{x_{t+1}^{\text{close}}}{x_t^{\text{close}}}\right), & \text{if } P_t \neq P_{t+1} \\ \text{sign}(P_{t+1}) \log\left(\frac{x_{t+1}^{\text{close}}}{x_t^{\text{close}}}\right), & \text{otherwise} \end{cases} \\
R_{t+1}^{\text{true}} &= \log\left(\frac{x_{t+1}^{\text{close}}}{x_t^{\text{close}}}\right) \\
R_{t+1}^{\text{optimal}} &= \max_{a \in \mathcal{A}(P_t)} R_{t+1}
\end{aligned}$$

with $\text{sign}(P_{t+1} = \text{out}) \doteq -1$ and $\text{sign}(P_{t+1} = \text{long}) \doteq 1$.

Settings The data horizon remains the same, starting in January 2016 and ending in late December 2020. Since no validation dataset is used, the training data spans four years [2016, 2019] and the analysis includes data from one year (2019, 2020]. In addition, PSA is again the research subject. However, this time the training

Tab. 6.8.: Evaluation of RL-agents (total rewards)

Agent	Optimal Reward	True Reward	Realized Reward
Single	3.6118	0.0775	-0.0987
Multi	3.8672	0.0775	-0.3837

uses only PSA's own data, which is given by the standardized OCHLV and TI data (normally distributed using quantile information).²⁴

During 250 training episodes, all agents (single and multiple) perform actions by trial and error, hopefully learning which actions are best in a given state S_t . DQN uses feedforward neural networks with three hidden layers (output sizes: [256, 128, 64]),²⁵ and each training episode starts with a random date to prevent the agent from remembering only the bare sequence and its optimal actions.

6.3.2 Results

The main findings are shown in Table 6.8, where both the single-agent and the multi-agent perform worse than a buy-and-hold strategy with respect to the cumulative log return at the period's end.

With respect to the testing horizon, the true cumulative logarithmic return *close* is 0.0775 (buy-and-hold strategy), while the optimal return is 3.6118 for the single-agent and 3.8672 for the multi-agent.²⁶

Figure 6.5 shows the aggregated rewards and thus cumulative logarithmic returns of both agents, realizing a negative total reward, which is less than the true aggregated reward. However, the total reward curve of the single-agent is above the true reward curve most of the time, even during the stock market shock in March 2020. Only after about eight months do they cross, such that the total realized reward curve becomes negative and falls below the true reward curve. For the multi-agent, the cumulative realized reward curve lies underneath the true reward curve the majority

²⁴This decision is again due to computational constraints, but also to make the results more interpretable when focusing on only one company.

²⁵Which activation functions *rllib* uses with respect to DQN is not clear at first glance, but likely given by the default configuration: hidden layers have the tangens hyperbolicus (Tanh), while the final output layer has a rectified linear unit (ReLU).

²⁶They differ because the optimal cumulative reward depends on the positions and thus on the actions.

of time, with only a few exceptions.²⁷ A look at Figure A.4 and Figure A.5 reveals that the single-agent has more variability in its positions, while the multi-agent stays out of the market most of the time.

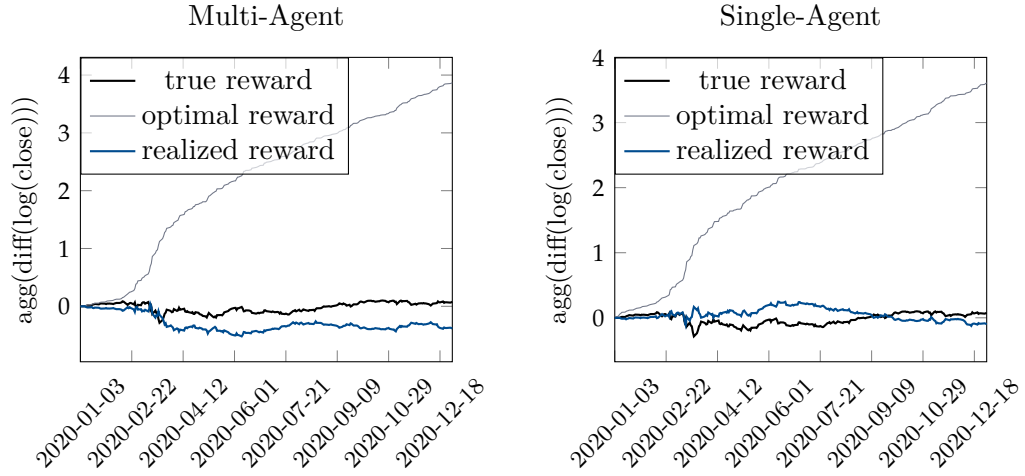


Fig. 6.5.: Performance on testing dataset – total rewards

In terms of performance, the single-agent dominates the multi-agent (see right diagram in Figure 6.7), however, the multi-agent achieves higher total rewards during training, as can be seen in the left diagram in the same figure. The detailed training plots can be found in Figure 6.6 and clearly show that both agents improve with each episode and significantly outperform the buy-and-hold strategy.

A final striking feature is that the multi-agent drops significantly after the brief stock crisis in March 2020, while the single-agent shows a strong downward and upward movement, followed by an increase in performance (see right graph in Figure 6.7). Since both agents were trained with the same data, such large differences are likely due to randomness, especially given that the multi-agent shows better training results. However, this cannot be validated without repeating the training and evaluation process multiple times to then take averages.

²⁷Detailed evaluation plots with mapped positions and actions can be found in Figure A.4 and Figure A.5, showing that both agents change positions and thus take actions. Thus indicating that both do learn, even if the performance is generally worse than with a buy-and-hold strategy.

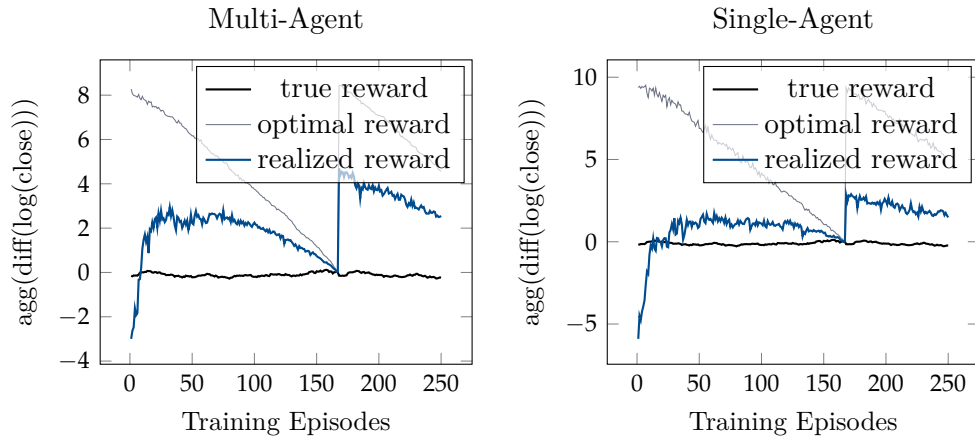


Fig. 6.6.: Performance during training indicating the total rewards at the end of each episode

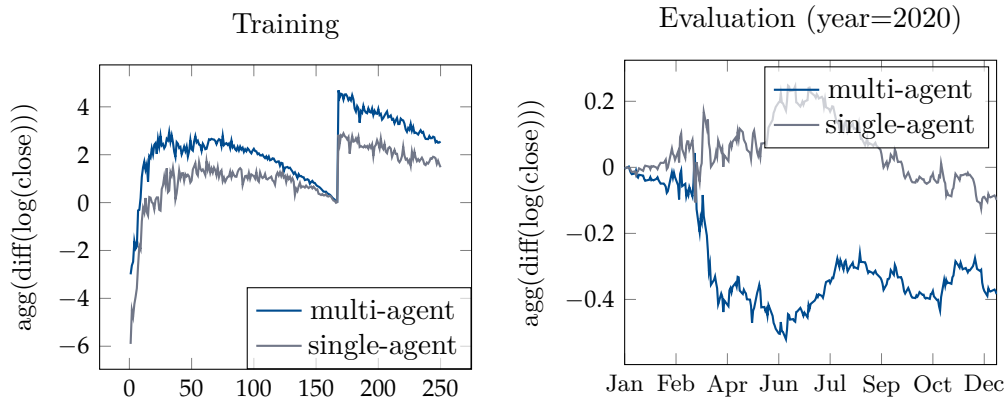


Fig. 6.7.: Realized rewards during training (total reward per iteration) and evaluation (cumulative realized rewards)

6.3.3 Summary & Discussion

In summary, under the conditions of this study, a single-agent environment is superior to a multiple-agent environment.²⁸ Compared to a buy-and-hold strategy, both RL-agents are inferior. But as the performance of the single-agent shows, there exists a possibility to outperform the market. So the question is how to increase this performance.

²⁸However, this is not a general statement, but only applies to this particular study design and should only be taken with caution, as chance could play a role. Repeating this study several times would only allow for a more general statement to be made.

Only speculation and conjecture can be made here about this: First and foremost, one should focus on the data and features on which the agent is trained. In general, the input data must be causally related (directly or indirectly) to the target variable. Otherwise, it should be difficult for an agent to make connections between the states, positions, and actions on the one hand and the feedback (reward) received on the other hand. It is thus likely not a weakness or inability of the RL-algorithm, but the lack of relevant information in the input data, which makes it difficult for the agent to learn how the stock market behaves.

Assuming that the data are suitable, another question is whether to include more data from other companies and variables, or reduce the number to a few hand-picked features. Finally, it might also make sense here to shorten the test horizon, apply continual learning, and perhaps use intraday data for high frequency trading simulations.

Summary & Conclusion

Objective One The question posed here was whether deep artificial neural networks can predict the next day's logarithmic return, and if so, over more than one evaluation period.

With daily stock data for all companies in the S&P500 *real estate* sector from the beginning of 2016 to the end of December 2020, tuned artificial neural network models (feedforward neural network and long short-term memory) were not able to explain the variance of the target's logarithmic return across three rolling windows. Therefore, the first efficient market hypothesis form cannot be discarded.

A peculiarity regarding the prediction of the model is that two of the three models show little to no output variance. They have either learned nothing or, to put it carefully, they have learned the martingale property that the next day's return is zero. However, the model with the highest variance was able to perform very well on the first few batches, suggesting that shorter evaluation periods might be more appropriate. In addition, looking at the feature importance of this model regarding the test dataset demonstrated that variance/volatility and momentum/moving average indicators were most promising.

Objective Two Concerning reinforcement learning and deep Q networks, both agents (single and multi-agent) were unable to outperform the buy-and-hold strategy in terms of the total realized reward. However, within this study design, the single-agent outperformed the multi-agent and even the buy-and-hold strategy during the first eight months of the testing period. Therefore, it may be reasonable to shorten the evaluation dataset here.

Relevance for Algorithmic Trading From this practicability study follows that predicting the next day's logarithmic return to exploit market inefficiencies is not possible using only historical stock data and technical indicators, and therefore not practical for automatic trading systems. In contrast, reinforcement learning seems more promising, but lacks reliability, since an agent must consistently outperform a baseline over multiple time periods with different stock market characteristics to be of practical relevance. In this sense, it must be explainable why the agent

chooses a particular action among the options available to it, since random behavior in trading decisions can be costly.

Final Thought Based on both studies, the most important question from the author's point of view is what drives the stock market. Because only then it is possible to select the right data, which must be causally related to the target variable, since artificial neural networks are not magic boxes that can give great predictions out of thin air. Without causal relationship between the input and target variables, artificial intelligence and all other predictive modeling approaches are likely to be in the dark remaining unstable and thus unreliable. Consequently, one should focus on causality rather than correlation. In this context, knowledge of institutional economics and game theory might be useful.

Bibliography

- [Aki+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019 (cit. on p. 41).
- [BCM90] RONALD J. BALVERS, THOMAS F. COSIMANO, and BILL MCDONALD. “Predicting Stock Returns in an Efficient Market”. In: *The Journal of Finance* 45.4 (1990), pp. 1109–1128. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1990.tb02429.x> (cit. on p. 1).
- [CT07] John Y. Campbell and Samuel B. Thompson. “Predicting Excess Stock Returns Out of Sample: Can Anything Beat the Historical Average?”. In: *The Review of Financial Studies* 21.4 (Nov. 2007), pp. 1509–1531. eprint: <https://academic.oup.com/rfs/article-pdf/21/4/1509/24453399/hhm055.pdf> (cit. on p. 1).
- [DFO20] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020 (cit. on p. 7).
- [Den+17] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. “Deep Direct Reinforcement Learning for Financial Signal Representation and Trading”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.3 (2017), pp. 653–664 (cit. on pp. 1, 2).
- [DHS11] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159 (cit. on p. 30).
- [FAM91] EUGENE F. FAMA. “Efficient Capital Markets: II”. In: *The Journal of Finance* 46.5 (1991), pp. 1575–1617. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1991.tb04636.x> (cit. on p. 3).
- [Fam70] Eugene F. Fama. “Efficient Capital Markets: A Review of Theory and Empirical Work”. In: *The Journal of Finance* 25.2 (1970), pp. 383–417 (cit. on p. 1).
- [Fam65] Eugene F. Fama. “Random Walks in Stock Market Prices”. In: *Financial Analysts Journal* 21 (1965), pp. 55–59 (cit. on pp. 3, 4).
- [GS80] Sanford J. Grossman and Joseph E. Stiglitz. “On the Impossibility of Informationally Efficient Markets”. In: *The American Economic Review* 70.3 (1980), pp. 393–408 (cit. on p. 1).
- [GKX20] Shihao Gu, Bryan Kelly, and Dacheng Xiu. “Empirical Asset Pricing via Machine Learning”. In: *The Review of Financial Studies* 33.5 (Feb. 2020), pp. 2223–2273. eprint: <https://academic.oup.com/rfs/article-pdf/33/5/2223/33209812/hhaa009.pdf> (cit. on pp. 1, 25).

- [GP09] D.N. Gujarati and D.C. Porter. *Basic Econometrics*. Economics series. McGraw-Hill Irwin, 2009 (cit. on p. 3).
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80 (cit. on p. 15).
- [Hub64] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101 (cit. on p. 27).
- [IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167 (cit. on pp. 10, 11).
- [KNR22] Ralf Kellner, Maximilian Nagl, and Daniel Rösch. “Opening the black box: Quantile neural networks for loss given default prediction”. In: *Journal of Banking and Finance* 134 (2022) (cit. on p. 33).
- [KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015) (cit. on p. 31).
- [Kok+20] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, et al. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG] (cit. on p. 15).
- [Lew04] Jonathan Lewellen. “Predicting returns with financial ratios”. In: *Journal of Financial Economics* 74.2 (2004), pp. 209–235 (cit. on p. 1).
- [Lia+18] Eric Liang, Richard Liaw, Robert Nishihara, et al. “RLlib: Abstractions for Distributed Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 3053–3062 (cit. on p. 45).
- [MF70] Burton G. Malkiel and Eugene F. Fama. “EFFICIENT CAPITAL MARKETS: A REVIEW OF THEORY AND EMPIRICAL WORK*”. In: *The Journal of Finance* 25.2 (1970), pp. 383–417. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1970.tb00518.x> (cit. on p. 3).
- [Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602 (cit. on p. 22).
- [NPO17] David M. Q. Nelson, Adriano C. M. Pereira, and Renato A. de Oliveira. “Stock market’s price movement prediction with LSTM neural networks”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 1419–1426 (cit. on p. 1).
- [NW94] D.A. Nix and A.S. Weigend. “Estimating the mean and variance of the target probability distribution”. In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN’94)*. IEEE, 1994 (cit. on p. 27).

- [Pas+19] Adam Paszke, Sam Gross, Francisco Massa, et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035 (cit. on pp. 11, 14, 30).
- [RB93] Martin Riedmiller and Heinrich Braun. “A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm”. In: *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*. 1993, pp. 586–591 (cit. on p. 31).
- [Sam73] Paul A. Samuelson. “Proof That Properly Discounted Present Values of Assets Vibrate Randomly”. In: *The Bell Journal of Economics and Management Science* 4.2 (1973), pp. 369–374 (cit. on p. 3).
- [ST06] F. Schmid and M.M. Trede. *Finanzmarktstatistik*. Springer Berlin Heidelberg, 2006 (cit. on p. 4).
- [Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958 (cit. on p. 11).
- [SB20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2020 (cit. on pp. 19, 26).
- [TE21] Thibaut Théate and Damien Ernst. “An application of deep reinforcement learning to algorithmic trading”. In: *Expert Systems with Applications* 173 (2021), p. 114632 (cit. on p. 2).
- [Zei12] Matthew D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method”. In: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701 (cit. on p. 30).
- [ZZR20] Zihao Zhang, Stefan Zohren, and Stephen Roberts. “Deep Reinforcement Learning for Trading”. In: *The Journal of Financial Data Science* 2.2 (2020), pp. 25–40. eprint: <https://jfds.pm-research.com/content/2/2/25.full.pdf> (cit. on p. 2).

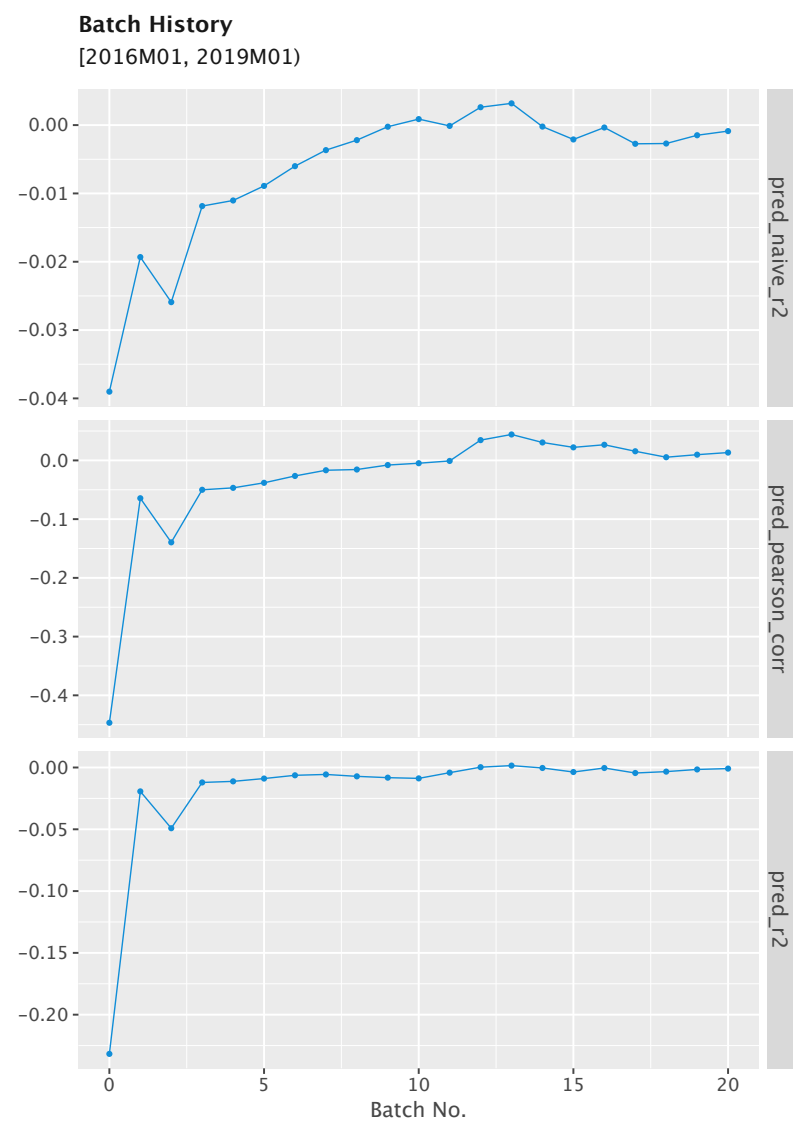


Fig. A.1.: Evaluation history (incremental metrics) – [2016M01, 2019M01]

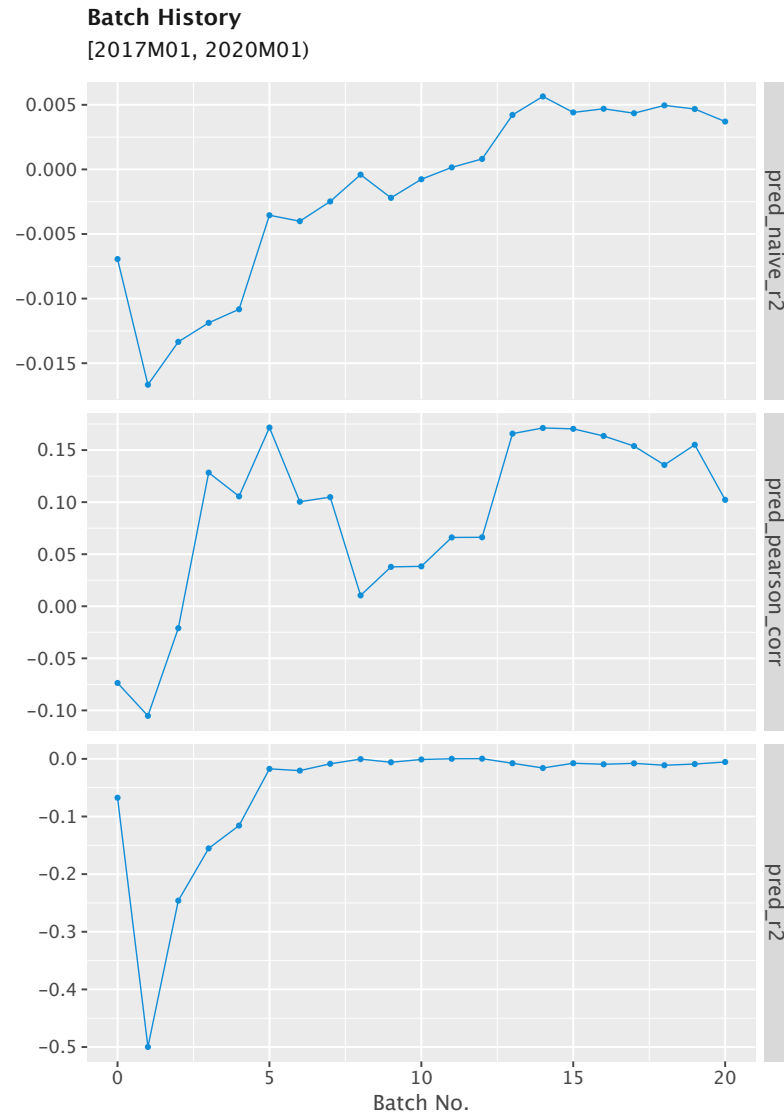


Fig. A.2.: Evaluation history (incremental metrics) – [2017M01, 2020M01)

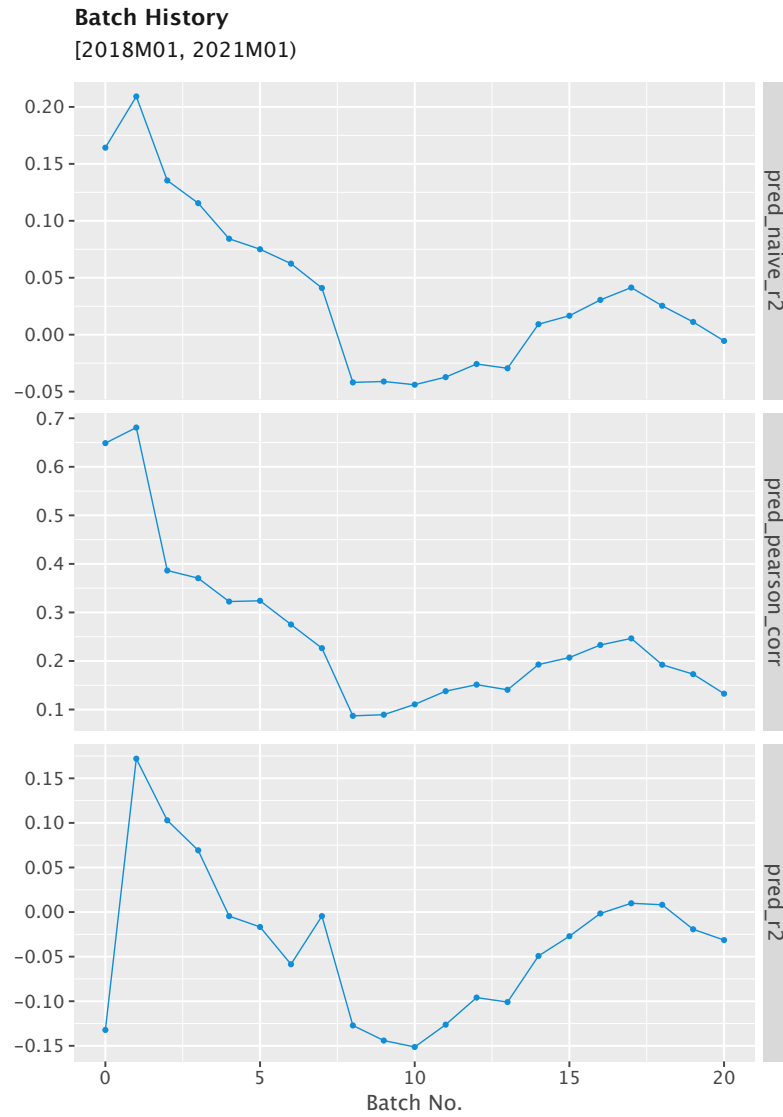


Fig. A.3.: Evaluation history (incremental metrics) – [2018M01, 2021M01]



Fig. A.4.: Detailed evaluation plot of the multi-agent stock environment (stock=PSA)



Fig. A.5.: Detailed evaluation plot of the single-agent stock environment (stock=PSA)

Tab. B.1.: Variance per sector of the S&P500 – [2016, 2021) – $\text{diff}(\log(\text{close}))$

Sector	Variance
Communication Services	0.00190845
Consumer Discretionary	0.00197811
Consumer Staples	0.00126824
Energy	0.00170938
Financials	0.00156497
Health Care	0.00123632
Industrials	0.00164144
Information Technology	0.00203079
Materials	0.00162716
Real Estate	0.00128567
Utilities	0.00149048

Tab. B.2.: Features ($n = 72$) – technical indicator details on mrjbq7.github.io/ta-lib/funcs.html

APO	BOP	CORREL
LINEARREG_SLOPE	PPO	STDDEV
VAR	diff(AD)	diff(ADOSC)
diff(ATR)	diff(AVGPRICE)	diff(BETA)
diff(CCI)	diff(CMO)	diff(DEMA)
diff(DX)	diff(EMA)	diff(HT_DCPERIOD)
diff(HT_DCPHASE)	diff(KAMA)	diff(LINEARREG)
diff(LINEARREG_ANGLE)	diff(LINEARREG_INTERCEPT)	diff(MA)
diff(MEDPRICE)	diff(MFI)	diff(MINUS_DI)
diff(MINUS_DM)	diff(MOM)	diff(NATR)
diff(OBV)	diff(PLUS_DI)	diff(PLUS_DM)
diff(ROC)	diff(ROCP)	diff(ROCR)
diff(RSI)	diff(TEMA)	diff(TRANGE)
diff(TRIMA)	diff(TYPPRICE)	diff(ULTOSC)
diff(WCLPRICE)	diff(WILLR)	diff(WMA)
diff(adjClose)	diff(close)	diff(diff(ADX))
diff(diff(ADXR))	diff(diff(HT_TRENDLINE))	diff(diff(SMA))
diff(diff(diff(T3)))	diff(diff(diff(TRIX)))	diff(high)
diff(log(adjClose))	diff(log(close))	diff(log(high))
diff(log(low))	diff(log(open))	diff(log(volume))
diff(log(vwap))	diff(low)	diff(open)
diff(volume)	diff(vwap)	pct_change(adjClose)
pct_change(close)	pct_change(high)	pct_change(low)
pct_change(open)	pct_change(volume)	pct_change(vwap)

Tab. B.3.: Companies with symbols of sector *real estate* (S&P500)

Symbol	Company
ARE	Alexandria
AMT	American Tower
AVB	AvalonBay Communities
BXP	Boston Properties
CPT	Camden
CBRE	CBRE
CCI	Crown Castle
DLR	Digital Realty
DRE	Duke Realty
EQIX	Equinix
EQR	Equity Residential
ESS	Essex
EXR	Extra Space Storage
FRT	Federal Realty
PEAK	Healthpeak
HST	Host Hotels & Resorts
IRM	Iron Mountain
KIM	Kimco Realty
MAA	Mid-America Apartments
PLD	Prologis
PSA	Public Storage
O	Realty Income
REG	Regency Centers
SBAC	SBA Communications
SPG	Simon
UDR	UDR
VTR	Ventas
VNO	Vornado Realty Trust
WELL	Welltower
WY	Weyerhaeuser

Tab. B.4.: Variance per company of sector *real estate* of the S&P500 – [2016, 2021) – $\text{diff}(\log(\text{close}))$

Symbol	Variance
AMT	0.00145778
ARE	0.000523117
AVB	0.000247776
BXP	0.000331049
CBRE	0.00215936
CCI	0.000541385
DLR	0.000328824
DRE	0.00516576
EQIX	0.00144108
EQR	0.00370319
ESS	0.000241316
EXR	0.00269816
FRT	0.000704773
HST	0.00161174
IRM	0.000698569
KIM	0.000729981
MAA	0.00155199
O	0.0018667
PEAK	0.000772902
PLD	0.00170815
PSA	0.00021494
REG	0.00174317
SBAC	0.000495628
SPG	0.000465945
UDR	0.000579282
VNO	0.000996029
VTR	0.00119689
WELL	0.000567282
WY	0.00254938

Tab. B.5.: Hyper-Parameter spaces

Hyper-Parameter	Space
N_{stacks}	$\in [1, \dots, 6]$
ANN-type	$\in \{\text{LSTM}, \text{FNN}\}$
Loss-type	$\in \{\text{GaussianNLLLoss}, \text{MSELoss}, \text{HuberLoss}\}$
Optimizer-type	$\in \{\text{Adadelata}, \text{Adagrad}, \text{Adam}, \text{Rprop}\}$
Activation	$\in \{\text{SELU}, \text{Tanh}, \text{CELU}, \text{GELU}, \text{SiLU}, \text{Mish}, \text{Softplus}, \text{ELU}\}$
Units/Hidden size	$\in \{2^i \mid i \in [4, \dots, 10]\}$
Do regularization	$\in \{\text{True}, \text{False}\}$
Use normalization layer	$\in \{\text{True}, \text{False}\}$
Use Dropout1d	$\in \{\text{True}, \text{False}\}$
Dropout rate	$\in \{\frac{2^i}{100} \mid i \in [0, \dots, 6]\}$
BatchNorm1d lr	$\in \mathcal{U}[1\text{e-}6, 1\text{e-}4]$
BatchNorm1d eps	$\in \mathcal{U}[0.05, 0.2]$

Tab. B.6.: Hyper-Parameter spaces (optimizer)


Hyper-Parameter	Space
Adadelata lr	$\in \mathcal{U}[0.01, 1.0]$
Adadelata rho	$\in \mathcal{U}[0.001, 1.0]$
Adadelata eps	$\in \mathcal{U}[1\text{e-}07, 1\text{e-}05]$
Adagrad lr	$\in \mathcal{U}[0.001, 0.1]$
Adagrad lr_decay	$\in \mathcal{U}[0.001, 0.1]$
Adagrad weight_decay	$\in \mathcal{U}[0.0001, 0.001]$
Adam lr	$\in \mathcal{U}[0.001, 0.1]$
Adam eps	$\in \mathcal{U}[1\text{e-}9, 1\text{e-}7]$
Adam weight_decay	$\in \mathcal{U}[0.0001, 0.001]$
Adam beta1	$\in \mathcal{U}[0.85, 0.95]$
Adam beta2	$\in \mathcal{U}[0.95, 1.0]$
Adam weight_decay	$\in \mathcal{U}[0.0001, 0.001]$
Rprop lr	$\in \mathcal{U}[0.001, 0.1]$
Rprop eta1	$\in \mathcal{U}[0.4, 0.6]$
Rprop eta2	$\in \mathcal{U}[1.1, 1.3]$
Rprop step_size1	$\in \mathcal{U}[1\text{e-}08, 1\text{e-}04]$
Rprop step_size1	$\in \mathcal{U}[1.0, 50.0]$

Statutory declaration

Herewith I confirm that the submitted master thesis is my own independent work and that I only used sources and resources listed therein and I have not made use of any inadmissible help from any other third party. In particular, I have clearly identified matter from other works, cited verbatim or paraphrased, as such.

The submitted thesis of parts thereof have not been presented at any institution or higher education for the examination procedure.

Passau, September 2, 2022

A handwritten signature in black ink, appearing to read 'F. Peschke', written over a horizontal line.

Florian Peschke