Reinfocement learning for regime-aware pairs trading

Reinforcement Learning for Regime-Dependent Optimal Stopping in Pairs Trading

Mihail Bankov[1]

Supervisor(s): Prof.dr. Frans Oliehoek[1]**, Dr. Fenghui Yu**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Final project course: CSE3000 Research Project
Thesis committee: Prof.dr. Frans Oliehoek, Dr. Fenghui Yu, Dr. Neil Yorke-Smith

Abstract

Pairs trading is a strategy that utilises the mean-reverting spread between two correlated assets (stocks). An important factor in such strategies is the market regime, which captures characteristics like trend and volatility of the data, and can shift over time. This paper investigates whether incorporating regime awareness improves the performance of Reinforcement Learning agents for pairs trading entry and exit decisions. Three Double Deep-Q network variants are implemented and compared: a baseline DQN (Deep-Q network), a Recurrent DQN, and a Hidden Markov Model-based DQN that maintains a separate agent per inferred regime. The agents are evaluated on intraday Corn and Wheat futures data, as well as on single-regime generated daily data. Results show that the Recurrent DQN does not significantly improve over the baseline, suggesting it does not implicitly capture regime information. The Markov DQN outperforms the baseline on real data ($p = 0.033$), while performing worse on generated data, though between-run variance is high in both cases. This supports the hypothesis that explicit regime modelling can benefit pairs trading on real-world data.

1 Introduction

Pairs trading is the action of trading two historically correlated stocks. If the stocks diverge, and we expect them to converge, we can (short) sell the more expensive one and buy the cheaper one, to make a profit on their difference. We call this "entering" the position. We can enter the position by choosing which stock to buy and which to sell, depending on whether we expect the spread to increase or decrease. Similarly, exiting the positions means selling all assets we own. The behaviour of stocks, also known as the market regime, needs to be considered when deciding how to trade. The regime includes characteristics like trend and volatility. Regimes are usually modelled as Markov chains, which are essentially graphs where each node represents a regime, and each edge represents the probability of going from one regime to another. To make a profit, a good strategy that executes trades at the right times is required, and such a strategy could benefit from being aware of the current market regime. Pairs trading has been used for a long time, and has already been implemented by trading companies [1].

There are different ways of deciding when to execute the trades and what trades to execute. Most strategies model the difference between the two stocks - the spread - as a mean-reverting process. A mean-reverting process is a process that we expect to fluctuate around a specific mean value. One simple strategy, Bollinger Bands [5], enters the position whenever the deviation from this mean is significant, and exits when the deviation decreases. Since the problem of pairs trading involves choosing the right trades to maximise profit, this fits very naturally with Reinforcement learning, where our actions are the trades, and our reward is the profit.

Since pairs trading and optimal stopping are quite old, there is a considerable amount of literature related to them. Many analytical solutions exist for different formulations of the problem, like the ones by Leung and Li [6], as well as Das et al. [3]. Theoretical solutions like these assume known and constant parameters, which do not happen in the real world. This is where learning-based approaches come in - approaches like the ones by Macri et al. [8] and Cartea et al. [2], which use Reinforcement learning to perform the trades. Macri et al. deal with regime-aware portfolio optimisation rather than optimal stopping, and Cartea et al. [2] deal with currency triples rather than pairs of data, and do not do optimal stopping explicitly.

As far as we are aware, none of the learning-based papers model regimes explicitly to do learning-based optimal stopping. And so, the objective of this paper is to examine whether regime awareness enhances learning-based entry and exit times for pairs trading. This involves answering the following subquestions (SQs):

SQ1: How can we find and incorporate regimes into a Reinforcement Learning model?

SQ2: Does a regime-aware model perform better than a regime-agnostic model on data with regime switches?

SQ3: Does a regime-aware model perform similarly to a regime-agnostic model on data without regime switches?

Analytical solutions typically require stable conditions, known parameters, or assumptions [13], which is where the main contribution of this paper lies. The paper relies on experimentation and comparison of different models. For that, stock data that includes different regimes is utilised, including both generated and real stock data. This stock data will be used to train a baseline and regime-aware Reinforcement learning agents that enter and exit the pair trades. The regime-aware agents infer the regime parameters from the data and include them in their decisions. The agents are compared using metrics like profit and Sharpe ratio. They are also compared against a simple pairs trading method - Bollinger bands.

The paper is structured as follows. Section 2 expands on the relevant literature. In Section 3 the methodology of the experimental setup is shown. Section 4 presents in more detail how the regime-aware Reinforcement learning agents work. Section 5 depicts the setup of the agent comparison and the results. Section 6 reflects on the ethical aspects of the research and its reproducibility. Section 7 discusses the agent comparison and some of the downsides of the results. In section 8 concluding remarks and possible improvements are discussed.

2 Relevant literature

To gain a deeper understanding of the already existing work, a further analysis of relevant literature is presented. This includes both analytical solutions and learning-based solutions.

2.1 Analytical solutions

Different analytical solutions exist for both optimal stopping and regime-aware optimal stopping. Leung and Li [6] achieve

an optimal stopping solution by modelling the spread as a mean-reverting Ornstein-Uhlenbeck (OU) process, and incorporating constraints like transaction costs and a limit to the maximal loss. They find expressions for the optimal spread levels to enter/exit the position. Their paper assumes known parameters, which is not the case in the real world, and the OU process assumption is also required and not always present in real data.

Das et al. [3] solve the regime-aware optimal stopping problem by modelling the different regimes as states in a Markov chain, and finding optimal entry and exit thresholds that depend on the regime. The paper also assumes a mean-reverting OU process, whose parameters are given by the state of the Markov model. The paper assumes a known Markov chain for the regimes and known stock parameters, which is also not practical in the real world, and may struggle with real stock data, even with inferred parameters.

2.2 Learning-based solutions

Different learning-based approaches also exist. Macri et al. [8] infer underlying information using a recurrent neural network to extract information from the trading signal, and then incorporate it in different ways into Policy Gradient-based algorithms. The paper deals with how much of the spread to hold at each point in time, rather than with optimal stopping directly. Cartea et al. [2] use double deep-Q learning (DQN) and Markov models to trade currency triplets, which are naturally cointegrated (e.g. EUR/GBP, GBP/USD, EUR/USD). The agent infers implicit regime parameters using reinforced deep Markov models and then uses discrete actions to trade the triplet. This problem is close to optimal stopping, with the difference that it chooses specific portfolio positions from a selection. Lu et al. [7] use intraday data to identify structural breaks - periods of time where the stock cointegration does not hold. Then, the probability of a structural break is given as an input into a DQN that decides trading entry/exit boundaries. The structural differences, though, are not necessarily different regimes, and the way the structural break probability is given to the DQN does not necessarily capture all structural break properties.

3 Methodology

To determine whether regime awareness improves the results of a Reinforcement Learning agent, baseline models for comparison are implemented, as well as regime-aware models. To train and test the models, a large quantity of historical data is needed.

3.1 Problem formulation

To understand the problem of optimal stopping and how to solve it, we will give a formulation. We denote the stock prices throughout time of a pair of stocks as A and B, and we denote the hedge ratio as β. For A and B to be cointegrated, it means that $X = A - \beta \cdot B$, which is called the "spread" of the two stocks, is not random noise, but follows some pattern. Usually, the spread is modelled as an autoregressive process: if X is AR(1), it means that, at time t, $X_t = \varphi X_{t-1} + c + \epsilon_t$, where φ and c are constants, and ϵ_t is white noise.

Figure 1: High-level data pipeline

We will allow an algorithm to trade with this spread, using a setup similar to the one used by Ngo and Pham [10]. At any point in time, an algorithm (agent) can choose whether to hold 1, -1 or 0 of the spread X - for example, holding 1 means we expect the spread to go up in value. The amount of spread we hold is called the position. To allow for short selling of stocks, there will be a collateral of 100%. This means that, to purchase 1 unit of the spread X at time t, we would need $A_t + |\beta|B_t$ units of money. This is done in the real world to prevent short-selling of too large quantities of stocks. The algorithm will always trade with all the money it has available - if it buys stocks, it will buy as much as possible, and vice versa for selling. Whenever the algorithm is not holding any stocks, its money will grow with a risk-free compound interest, like in a savings account.

3.2 Data pipeline

To get from stock to prediction, the following pipeline is needed (see Figure 1). Firstly, a suitable stock pair is chosen, so that each data point from the first stock corresponds in time to one in the other. Then, a suitable stock ratio based on the train data is chosen for the stocks, meaning that we can hold 0, -1 or 1 of $A - \beta B$. To infer the beta, there are different methods, like the Johansen method or the Ordinary Least Squares method. Since the Ordinary Least Squares(OLS) method is simple, and the coefficient will not introduce a significant difference in the comparison between different methods, since the focus is on optimal stopping, OLS is used for finding the hedge ratio. Afterwards, the stock prices and spreads are provided to the different algorithms to train and test on. The train data is earlier in time than the test data, to avoid overfitting by knowing future results. At the end, the different algorithms produce their strategies, which are executed on the test set to get the desired results.

3.3 Pair selection

Choosing the right pair of stocks is important, since not all pairs have a spread that reverts to a mean. To determine the right pair, a pool of stocks is taken, and all possible options are examined. For each option for a pair, different tests are run, and the results are filtered based on the results. The tests include correlation (>0.7), cointegration ($p<0.05$, augmented Engle-Granger two-step test), and Hurst exponent on the spread (<0.5). The resulting pairs of stocks can be used in data generation or directly in the models.

3.4 Data generation

Single-regime data is needed for our comparison, and since real stock data is highly irregular, single-regime data is generated. The parameters for this generation are inferred from real data. For our generation, we assume that the spread and one of the stocks' returns follow AR(1) processes, like the method used by Lin et al. [15]:

$$S_1(t) - \beta \cdot S_2(t) = \varepsilon_t$$

$$S_2(t) - S_2(t-1) = e_t$$

where ε_t and e_t are AR(1), and $S_{1,2}(t)$ is the respective stock price at time t. The hedge ratio and the autoregressive process variables are computed using ordinary least squares - they correspond to the maximum likelihood estimation. Using this method, data from a mean-reverting process that follows the same parameters can be generated.

The method in Lin et al. [15] has been used since there are only a few papers that use generated data, and most of them generate only the spread of the stocks. No papers generate two stocks based on real data, and include regime parameters or switches.

3.5 Chosen models

For a classical pairs trading baseline model, the Bollinger bands strategy [5] has been chosen, since it provides a good intuition of why pairs trading is a viable way to make money when the stocks are strongly mean-reverting. For the Reinforcement learning models, many different types of agents exist, for example, Proximal Policy Optimization, Q- and Deep-Q agents, Actor-critic models, and more. Optimal stopping deals only with whether we are in the position/out of the position. Since we have discrete states, the most suitable algorithm for that is a Deep-Q agent, which trains a Q function by representing it as a neural network, with input some information about the stock data and an output of the desired state (1/0/-1). From different variations of Deep-Q agents, the Double Deep-Q variation is chosen, since it avoids the overestimation caused by taking the maximum of the results of the next step [14], which is especially prominent in noisy data, like stocks. This is also the preferred method by other relevant papers which deal with learning-based optimal stopping.

3.6 Regime-aware models

For the regime-aware models, there are different ways to represent the regime. Two popular approaches used in relevant research are a (Hidden) Markov Model, which would infer the Markov chain on historical data and then use it when trading, and a Recurrent neural network that determines the current state of the market implicitly and uses the result, either in the same network or as an input into a different one. In this paper, both approaches are tried - the Markov model is simpler and explicitly regime-aware, whereas the Recurrent neural network approach is more complex and is not directly regime-aware, but requires no assumptions regarding the market state.

3.7 Model comparison

To compare how different models perform, they need to be trained on some data and then evaluated on data that is unknown to them. Naturally, stock data can be separated into three distinct time periods, where the oldest will be used for training, the middle one will be used for validation of the hyperparameters, and the most recent one for testing. The metric for training will be pure profit, and the models will be compared across multiple runs to account for randomness. The comparison metrics will be (raw) profit, as well as the Sharpe ratio. The Sharpe ratio is the risk-weighted profit, which is calculated in the following manner:

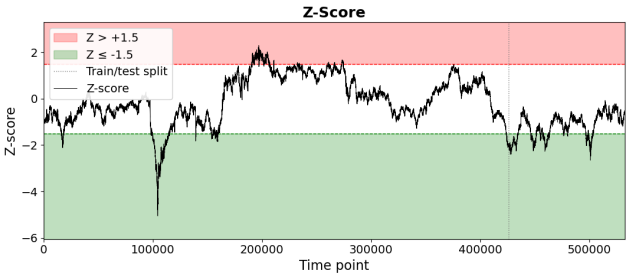

Figure 2: Bollinger bands strategy visualisation on the spread of real stock data. The position is entered whenever the spread diverges too much from the historical mean.

$$sharpe = \frac{\mu_p - r_f}{\sigma_p}$$

where μ_p is the average return of the algorithm for 1 time step, r_f is the risk-free rate (putting all money in the bank), and σ_p is the standard deviation of the algorithm returns throughout the whole time interval. In this paper, $r_f = 0$ is used for the Sharpe ratio calculation to allow for a better comparison with results that have negative returns.

In the comparison, it is possible to achieve different results in different runs, because of some randomness in the training. In this case, the average of the pure returns and Sharpe ratios will be looked at. Another way to compare the returns is to take their logarithm, since the returns are multiplicative with the amount of money owned (and the compound interest is linear in log-scale). However, the average of raw returns is used, since it makes more sense in a real-world context - if many agents are trained with different random seeds, giving them an equal amount of money and using all of them would act as if we have one agent that performs like the average of the many agents.

4 Model implementations

The baseline model, as well as the Reinforcement Learning agents, have been implemented specifically to answer our research question with the described setup. This section outlines how each model works.

4.1 Bollinger bands strategy

The Bollinger bands strategy has been implemented in the following way. On the train data, it figures out the mean and standard deviation of the spread. Then, a threshold T is chosen to maximise the profit on train data. Using the mean and standard deviation, the Z-score of each data point is computed. Then, when running the agent, if the Z-score of the spread at time t is more than our threshold ($|Z_t| > T$), then the strategy enters the position. If Z is negative, we expect the spread to go up in value, so it buys, otherwise it sells. The position is exited whenever the Z-score goes back to 0. An example of what this would look like can be found in Figure 2. After finding a threshold T that maximises profit on the

train data, the same procedure is done on the test data, using our learned threshold T.

4.2 Deep-Q network

The regime-agnostic agent is a Double DQN [14]. This means that the goal of the agent is to optimize the following function, to reflect reality as much as possible:

$$Q_{policy}(s_t, a) \leftarrow r + \gamma \cdot Q_{target}(s_{t+1}, argmax Q_{policy}(s_{t+1}, a'))$$

where the policy function selects the action, and the target function evaluates it. The current state is represented by s_t, a and a' represent the actions, and r is the reward from the current action.

The target and policy functions are represented as neural networks. The neural networks are trained by choosing random windows of consecutive time points from the train data, and then choosing the agent actions using the ε-greedy strategy - choose the assumed optimal move with probability $1-\varepsilon$, or a random move with probability ε. If the whole train time interval is used in each train run, the ε-greedy strategy will lose all money very quickly in runs with transaction costs, which is why smaller random windows are used for the training. When testing, on each time step, the move that is assumed to give the best profit is chosen.

As input to the neural networks, a representation of the stock state is given. The representation includes the current position, as well as rolling Z-scores, the hold time, and the unrealized profit. How they are calculated can be seen in Table 1.

Input	Expression
Position	$1/0/-1$
Rolling Z-score	$\frac{X_t - average(X_{t-size+1} ... X_t)}{std(X_{t-size+1} ... X_t)}$
Hold time	$\log(t - t_{entry})$
Z-score profit	$\frac{position \cdot (X_t - X_{entry})}{std(X_{t-size+1} ... X_t)}$

Table 1: Statistics used as input into the DQN; $size$ is the size of the window for rolling statistics, $X_t = S_1(t) - \beta \cdot S_2(t)$ is the spread at time t, and X_{entry} is the spread at the time when the current position is entered.

The (rolling) statistics are necessary for the neural network, since it does not possess the ability to remember past data. Statistics are used instead of raw spread values, since they are less noisy and allow a simpler model that looks farther in the past.

4.3 Recurrent neural network

The Deep recurrent Q-network (DRQN), is an extension of the Deep-Q network, but with memory cells added. The goal for the memory cells is to provide the neural network with a way of representing the current state implicitly. The neural network input and output get extended with extra k nodes each. Initially, zeroes are given as input in those k nodes, after which the k new outputs are given as input to the neural network in the next time steps. To avoid those memory values from becoming really large, we take the hyperbolic tangent of them before feeding them into the next time step.

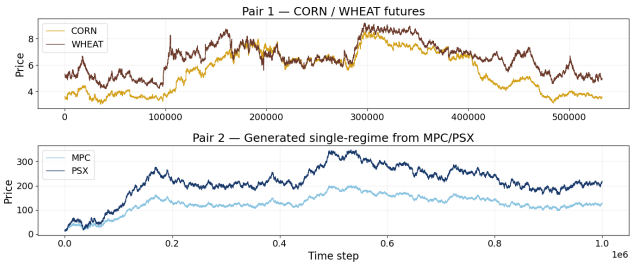

Figure 3: Stock data used in the experiments. Pair 1 is Wheat/Corn futures, and Pair 2 is generated data, using parameters from the real stocks MPC and PSX.

4.4 Hidden Markov model Deep-Q network

To explicitly represent each regime, a Hidden Markov model is utilised, which uses a different Deep-Q network for each regime. Firstly, each time point in the train data is assigned a regime by inferring a Hidden Markov model using the Expectation-Maximization [4] algorithm. We initialise as many Double Deep-Q agents as there are regimes, and we perform the learning algorithm. The difference here is that only the agent that corresponds to the specific regime at that point in time is trained on it, meaning that at the end, each agent is trained on its own regime. On test data, the agent corresponding to the regime with the highest conditional likelihood is chosen.

There are different ways of choosing what the Markov model represents. The most direct method with respect to the generated data is assuming each state represents an autoregressive model with different parameters per-regime, which is used to model the spread. Real stock data is quite irregular and difficult to model, meaning if a lot of parameters are added, there will not be any one specific Hidden Markov chain that the data would converge to.

To mitigate that, a simpler approach of assuming normally distributed parameters is used. This can be applied to, for example, the hedge ratio, or the standard deviation of the returns (representing periods of high/low volatility). After applying the assumptions, the Forward algorithm is run to find the most likely next state, based on the current assumed state and current observation (e.g. current standard deviation of returns).

5 Experimental Setup and Results

This section outlines the experimental setup, as well as the results. The algorithms explained in sections 3 and 4 have been implemented in python, using the libraries statsmodels [12] and scikit-learn [11].

5.1 Stock data

For the stock data, intraday real stock data, as well as generated daily data, have been used. The real stock data is taken from a Kaggle dataset [9]. Minute-level data is taken from Corn and Wheat futures and combined into a single file.

Points of time that exist in one stock but not the other are removed, so that only matching time points remain. At the end, a total of 532,635 data points remain, spanning from 20-02-2009 to 19-06-2015. From this data, only the column with closing prices is used. Plots of the data can be seen in Figure 3.

The generated data follows the method outlined in subsection 3.4. The daily tickers MPC and PSX have been chosen as a basis, as they pass all pair selection tests in section 3.3. From them, the last 3500 days have been taken, and data based on their extracted parameters has been generated, for 1,000,000 time intervals. Since the trend of the spread is daily, it has been scaled by a factor of 0.0035 to keep the data points from becoming very large.

5.2 Experimental setup

The experimental setup includes the baseline and the three DQN models - Double Deep-Q network, Markov Double Deep-Q network, and Recurrent Double Deep-Q network. All models use a 4% annual risk-free interest rate when their position is 0. They are trained and then tested on the Wheat and Corn futures data, and also on the generated MPC and PSX data.

The Deep-Q networks in all three models are set up in the same way. The train/test split is 80:20, where the train is further split into 80:20 for training and validation of the hyperparameters. Each training episode is 10000 data points long, and each training run includes 400 episodes, with ε decaying 0.994 multiplicatively per episode. The discount factor γ is 0.995, and each episode starts with 100 units of money. To avoid overfitting, a proportional transaction cost of 0.001 is used during training, and a transaction cost of 0.0001 is used during testing, which is on par with real-world transaction costs. The target and policy networks sync every 200 time steps. The Deep-Q network has two hidden layers, each with size 128, and a ReLU activation function. The input statistics include the position (-1/0/1), rolling Z-scores of size 100, 1000, 10000, and 40000, the hold time, and the Z-score profit with a window size of 100. Those specific sizes have been chosen so that they cover a wide range, while not being too small so as to avoid noise in the data. The output nodes are the three possible actions, which represent having 1/0/-1 of the spread. For the recurrent neural network, there are 4 extra memory nodes added in the input and output.

The Markov model uses the rolling beta (hedge ratio) of the last 50000 data points to determine the regime of the current time point. 50000 was chosen because it is close to a year's worth of data, since regimes are long-term. The number of assumed regimes is 3, meaning 3 agents that each correspond to one of the regimes are used. The training loop is the same as the regular Double Deep-Q network, with the difference that only the agent that corresponds to the most likely regime of the current point is trained. This is also done in testing - on each point, the model that corresponds to the most likely regime is used to perform an action. Each agent runs 5 times with different random seeds, which is due to the randomness caused by training with random windows, as well as randomness from the ε-greedy.

Figure 4: Training loss of the DQN agent across all training episodes. Note that each episode is on a different window of data.

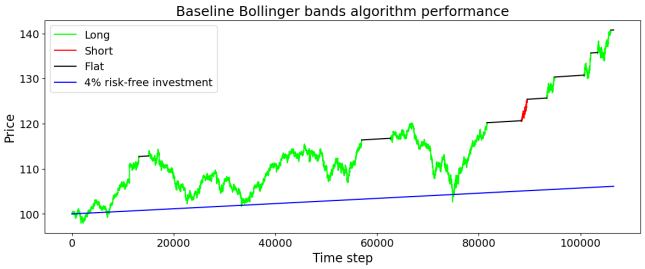

Figure 5: Bollinger bands algorithm performance on wheat and corn futures. The green periods are where the algorithm holds 1, the black periods are where it holds 0 and the red periods are where it holds -1 of the spread. It is compared to a 4% annual risk-free rate, in blue.

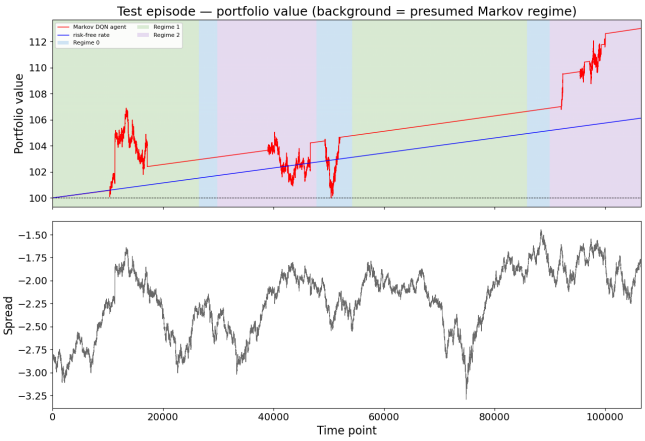

Figure 6: An example of a single run of the Markov DQN, with presumed 3 regimes. The upper plot shows the performance of the algorithm compared to the risk-free rate. Highlighted are the different regimes. For comparison, the spread is shown on the lower plot.

Algorithm	Data	Returns, percentage	Sharpe Ratio
DRQN	Real	10.90 ± 12.43	0.63 ± 0.65
	Generated	488.47 ± 376.93	6.57 ± 2.77
Markov DQN	Real	8.71 ± 9.99	0.64 ± 0.68
	Generated	668.76 ± 238.84	6.47 ± 0.94
DQN	Real	8.03 ± 3.89	1.12 ± 1.01
	Generated	1775.85 ± 1388.81	8.78 ± 2.63
Bollinger bands	Real	40.77	1.69
	Generated	18.64	0.64
4% annual compound interest	Real	6.01	–
	Generated	11.80	–

Table 2: Experiment 1 - agent performance on real and generated data. The results are averages over 5 iterations, for each of the DQN strategies. Bollinger bands uses z=0.5 for real and z=0.2 for generated data.

5.3 Experiment results

The results from the agent runs can be seen in Table 2. Figure 4 shows the training loss on the DQN agent in one run. The Bollinger Bands performance on real data can be seen in Figure 5. An example run of the Markov DQN algorithm can be seen in Figure 6, along with the predicted regimes.

Looking at the results in Table 2, it is apparent that the algorithms have a big variance in how they perform, depending on the random seed. Thus, if the results of a comparison between two models are close, there would be insufficient evidence to support a meaningful difference.

Despite that, a few interesting results can be seen. Firstly, the baseline DQN outperforms both regime-aware agents with generated single-regime data. Its variance is also proportionally higher, but both average returns of the DRQN and the Markov DQN are a lot smaller than the average return of the baseline DQN (488.47% and 668.76% compared to 1775.85%). It can be seen that, on generated data, the DQN performs better than both the Markov DQN and the DRQN. All agents outperform both the Risk-free rate and the Bollinger bands strategy. The Sharpe ratio of the baseline DQN is bigger than that of the regime-aware agents. When comparing the ratio between the returns of different agents and their Sharpe ratios, we can see that the DQN has 2.66 times higher returns than the Markov DQN, but its Sharpe ratio is only 1.35 times higher. Since the Sharpe ratio difference is not as big as the one in returns, this means that the baseline DQN has a higher variance in its single-run tick-to-tick returns.

On real data, we can see that the returns are a lot lower, with a very big variance with respect to the mean and lower Sharpe ratios. Since the number of runs is only 5, no meaningful results can be extracted from comparing the baseline DQN with the regime-aware agents. The Bollinger bands strategy outperforms all agents. This is due to the fact that there is no trend in the chosen corn/wheat futures, meaning that the Bollinger bands cannot lose money since the spread will always revert to a global mean. If the data had a trend, like most stock data, then the Bollinger bands would not be viable. The Bollinger bands run on real data can be seen in Figure 5, and a Markov DQN run can be seen for comparison in Figure 6.

Because of the high variance on runs with real data, a second experiment is run to compare the DQN with the Markov DQN. Note that DRQN is not included, since it is not explicitly but implicitly regime-aware, and its variance is higher than the other models. The setup is the exact same as the previous experiment, but using 20 different random seeds per agent instead of 5, and only using real data. Because of the high computational time, the size of each hidden layer in each agent's network has been reduced to 64 from 128. The results of this experiment are shown in Table 3. It can be seen that the Markov DQN significantly outperforms the baseline DQN (8.74% compared to 0.80% average). The variance of the Markov DQN is higher than that of the DQN, though this is due to a single outlier - when removed, the Markov DQN variance becomes 9.3, which is close to that of the baseline DQN. To determine how significant the difference is between the two agents, we will use a single-tailed Welch's t-test on the returns to test the hypothesis of the Markov DQN performing better than the baseline DQN model. The test gives a t-value of 1.91, with 30 degrees of freedom that were calculated from the Welch–Satterthwaite equation. This corresponds to a p-value of 0.033, which is less than the threshold of 0.05 and is thus statistically significant. The conclusion is more fragile if the outlier value is removed (p-value of 0.052, check Appendix B).

Agent	Data	Returns, percentage	Sharpe Ratio
DQN	Real	0.80 ± 9.28	0.13 ± 0.52
Markov DQN	Real	8.74 ± 16.11	0.41 ± 0.60

Table 3: Experiment 2 - DQN compared to Markov DQN performance on real data. The results are averages over 20 iterations for each strategy.

6 Responsible Research

This section outlines relevant ethical considerations when conducting the research, as well as its reproducibility and LLM usage.

6.1 Ethical considerations

The actors related to trading and trading strategies include people who trade, trading companies, and trading providers. The people or companies who use such algorithms always run the risk of losing money due to market uncertainty, algorithm performance, or bugs in implementation. For employees using such algorithms in a trading company, their compensation and job security could be significantly affected by a big problem in strategies like the ones mentioned in this paper. For companies, using such trading strategies also runs the risk of financial losses, which could lead to bankruptcy of the company and, consequently, the layoff of a significant number of people. Usually, fail-safes are implemented to prevent significant losses.

For trading providers, trading algorithms always run the risk of overloading the systems, leading to a dysfunctional market. Overall, market events caused by rogue trading algorithms can have dire consequences for the wider population. For example, significant amounts of retirement savings

of people are in the stock market, or farmers who use futures contracts to avoid risk and secure financing for harvesting crops.

Individuals or companies using such algorithms should always limit their risk, implement fail-safes, and be aware of their market impact.

6.2 Reproducibility

This paper utilises different models, implemented with many different hyperparameters, using different kinds of data, and using randomness in the model training. To make the results reproducible, the following measures have been taken:

- All relevant code, along with the code to train the agents, test the agents, generate the data, as well as the data itself, have been uploaded to GitHub[1] and the TU Delft repository.
- Whenever randomness is used for training models, like when choosing new random actions for the agent to try, a seed is set at the beginning of the program so that the output is always deterministic. This has been verified to work by trying multiple runs and getting the exact same results.
- The hyperparameters are set as constants at the top of the programme, and have been included in the paper wherever relevant to the results.
- Data clearing is done deterministically for the real data, and data generation also uses a random seed.

6.3 LLM disclosure

In the development of this paper, LLMs have been used for writing code, searching for relevant literature, and paper editing. Any code generated by an LLM has been checked for correctness and understanding before usage. In the paper, an LLM has been used only for grammatical errors and rewording.

7 Discussion

Trading strategies that beat the risk-free rate of 4% per year are considered good, and the ones with a Sharpe ratio of 2 are considered exceptionally good. The generated data is not representative of real-world data, but rather a controlled single-regime scenario, because of which it cannot be used to provide real-world context to the results. The results on real data perform better than the risk-free rate and have very realistic returns, considering that they are relatively simple strategies. Their variance is quite high, but that is to be expected considering the randomness from the ε-greedy training, the random windows, and the noisy stock data.

Despite this, when running the algorithms many times, the Markov DQN significantly outperforms the baseline DQN on real data, and the baseline DQN outperforms the regime-aware algorithms on generated single-regime data. This answers the research questions in the context of the Markov DQN - different patterns from different regimes can be inferred when training only on the specific regimes, which makes the regime-aware model perform better on real data. It cannot be said, however, that the regime-aware agents perform better on simulated data than the baseline DQN. This could be due to the Markov model finding patterns that do not exist, making the regime-specific agents overfit. It is also important to keep in mind that the second experiment, which compares only the baseline DQN with the Markov DQN, is run with a different network size due to computational constraints, which does not make it directly comparable to the first experiment.

With regards to the DRQN, it does not provide a meaningful advantage when compared to the regime-agnostic algorithm, especially because of its higher variance. It cannot be concluded that the DRQN implicitly learns the regime of the data - it likely overfits on the noise of the data, since its memory cells are updated at each time step, meaning it would be difficult to detect long-term patterns. This is consistent with other papers - Lu et al. [7] utilise a Long short-term memory(LSTM) neural network, which is a Recurrent neural network but specifically modified to have long-term memory, to find structural breaks in stock pair data, since it is less prone to overfitting on the noisy data.

Still, when concluding, we have to keep in mind the big run-to-run variance. The 0.80% average return of the DQN baseline in the second experiment shows exactly that - sometimes the agent learns a good strategy from the train data, and achieves good returns and Sharpe ratios, but other times it loses money, even when not trading, and using compound interest would provide a better result. The comparison results are only useful because the agents were trained and tested using the exact same conditions. It is possible that changing the conditions/training method would bring better results for either of the algorithms, but this has been tried, and is the best found performance for both of them on validation data (see Appendix A). The variance likely comes from the methods used.

8 Conclusions and Future Work

This paper attempts to answer whether regime awareness improves Reinforcement Learning agents when deciding whether to buy, sell, or take no action for a pair of stocks, in the cases of real-world data and generated data. Three Double Deep-Q network-based approaches have been implemented: a regular DQN, a recurrent DQN, and a Markov model-based DQN. The agents have been trained on random windows of real and generated data, and then tested on the same time series, but in the future. Although the results have very high variance, it can be concluded that there are no results to show that Recurrent DQN provides meaningful advantages when compared to the baseline DQN, and thus does not implicitly learn the regimes. It can also be concluded with a confidence value of p=0.033 that the Markov DQN captures regimes and performs better than the DQN on real data, while it performs worse on generated single-regime data. The comparison is done over many iterations, since the run-to-run variance is very high.

[1] https://github.com/mihailbankov/Regime-aware-RL-optimal-stopping

This answers our research questions: We can incorporate regimes into a Reinforcement learning model by training separate agents on data from different regimes. A regime-aware model performs better than a regime-agnostic model on data with regime switches. A regime-aware model performs worse than a regime-agnostic model on single-regime data.

Future improvements can include implementing an LSTM version of the recurrent DQN to study if an implicit regime can be determined. The DQN baseline can also be improved - more consistent methods of training could be found, which could achieve more significant results if compared with a Markov DQN. Possible improvements in the Markov DQN could include trying different parameters to model as a Markov chain - for example, rolling stock volatility, or including multiple parameters in multi-dimensional states in the Markov model. All models can be tested on more pairs of data from different sectors and different periods of time, to make the conclusions more concrete, or to show that assuming regimes in real data would not always lead to a better result.

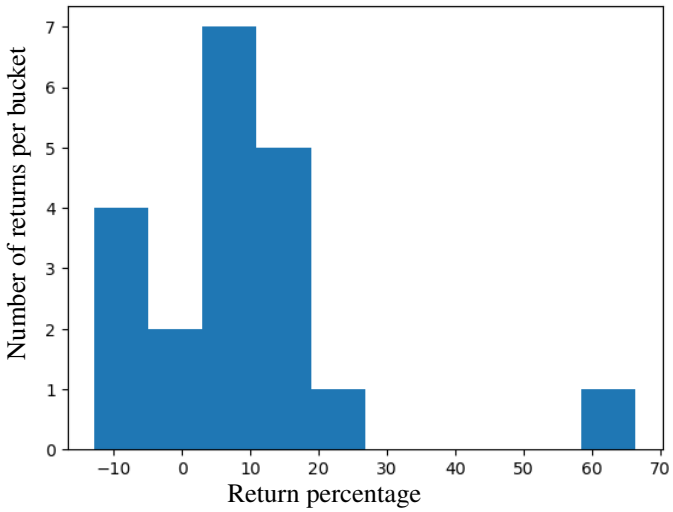

Figure 7: Histogram of the returns of experiment 2 using the Markov DQN model.

A Variance reduction attempts

Because of the high variance in the results, many different attempts have been made to get results that are more consistent. Training random windows of size 10, 50, 100, 1000, 10000, 50000, and the whole data have been tried, along with different window sizes for input statistics in the DQN. Other statistics as input, like a rolling average of the spread or its standard deviation, have also been tried. Different activation functions(e.g., Sigmoid), different network sizes (3x128, 2x128 internal nodes) have been tried, as well as normalizing the statistics before inputting them into the DQN. Different ε-decay constants have also been tried for the training, including anywhere from 10 to 4000 episodes. Another method that was tried is splitting the train set further into a second validation set, which is used to select the best performing state of the agent throughout training from the last 15% of training steps. None of the mentioned approaches has worked to reduce the variance or improve the strategy for all 3 types of DQN agents mentioned in the paper.

B Second experiment outlier

Because of the outlier in the second experiment, it is interesting to investigate the results without it. Without the outlier, the distribution is 5.7 ± 9.3. If the single-tailed Welch's t-test is run on this, a p-value of 0.052 is achieved, which is right on the significance threshold, and has to be kept in mind. The histogram of the returns of the experiment can be found in Figure 7.

References

[1] Richard Bookstaber. *A Demon of Our Own Design: Markets, Hedge Funds, and the Perils of Financial Innovation*. John Wiley & Sons, 2006. p. 186.

[2] Álvaro Cartea, Sebastian Jaimungal, and Leandro Sánchez-Betancourt. Deep reinforcement learning for algorithmic trading, 2021. doi:10.2139/ssrn.3812473.

[3] Emily Crawford Das, Phong Thanh Luu, Jingzhi Tie, and Qing Zhang. Pairs trading under a mean reversion model with regime switching, 2024. doi:10.3934/naco.2023023.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 09 1977. doi:10.1111/j.2517-6161.1977.tb01600.x.

[5] Mark Leeds. Bollinger bands thirty years later, 2013. arXiv:1212.4890.

[6] Tim Leung and Xin Li. Optimal mean reversion trading with transaction costs and stop-loss exit, 2015. arXiv:1411.5062.

[7] Jing-You Lu, Hsu-Chao Lai, Wen-Yueh Shih, Yi-Feng Chen, Shen-Hang Huang, Hao-Han Chang, Jun-Zhe Wang, Jiun-Long Huang, and Tian-Shyr Dai. Structural break-aware pairs trading strategy using deep reinforcement learning, 08 2021. doi:10.1007/s11227-021-04013-x.

[8] Andrea Macrì, Sebastian Jaimungal, and Fabrizio Lillo. Deep reinforcement learning for optimal trading with partial information, 2025. arXiv:2511.00190.

[9] Möbius. Huge stock price data: Intraday minute bar, 2021. Accessed: 2026-06-19. URL: https://www.kaggle.com/datasets/arashnic/stock-data-intraday-minute-bar.

[10] Minh-Man Ngo and Huyên Pham. Optimal switching for the pairs trading rule: A viscosity solutions approach. *Journal of Mathematical Analysis and Applications*, 441(1):403–425, 2016. doi:10.1016/j.jmaa.2016.03.060.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[13] Jingzhi Tie and Qing Zhang. Pairs trading: an optimal selling rule under a regime switching model. *Banach Center Publications*, 2020.

[14] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. arXiv:1509.06461.

[15] Lin Yan-xia, Michael McCrae, and Chandra Gulati. Loss protection in pairs trading through minimum profit bounds: A cointegration approach. *Journal of Applied Mathematics and Decision Sciences*, 2006, 08 2006. doi:10.1155/JAMDS/2006/73803.