Deep Reinforcement Learning for Trading Commodities

Marius Dragic
CentraleSupélec
marius.dragic@student-cs.fr

Lancelin Poulet
CentraleSupélec
lancelin.poulet@student-cs.fr

Abstract

This project explores the application of Deep Reinforcement Learning to design trading strategies for commodity markets. In our work, we implement a Deep Q-Network (DQN) agent within a custom trading environment that incorporates realistic market dynamics and a discretized action space. We compare the performance of our DQN-based approach to a baseline model derived from Bollinger Bands, then we evaluate both methods using comprehensive financial metrics such as annualised expected trade return, volatility, downside risk, and risk-adjusted ratios. Our experiments on commodities demonstrate that the DQN agent effectively captures market dynamics and generates profitable trading signals, even in the presence of high uncertainty. Nevertheless, it's very complicated to converge learning to an optimal trading strategy, so we've simplified the problem by normalizing all prices between 0.1 and 1 so that our agent can trade on several commodities at the same time. This work highlights the ability to train an agent to trade on several commodities at the same time with very few computational resources required, although the problem has been simplified in view of the high complexity of financial market modelling.

1. Introduction

Financial trading, particularly in commodity markets, presents a dynamic and challenging environment where traditional forecasting methods often struggle to capture rapid fluctuations and inherent uncertainties. As part of this project, we've chosen to work on the commodities market, a lateral and extremely volatile market which, according to some experts, is even more difficult to model. The commodities market is said to be sideways, unlike other markets such as the S&P 500 or the CAC40, which are particularly bullish. Training an agent on this type of market is more complicated to achieve, but the potential of these agents on these markets is enormous, since long only strategies(buy and stay) strategies don't work well on so-called lateral markets.

In order to be able to trade on several commodities at the same time, we choose to train a Double DQN agent which is a Critic-only agent, i.e. an agent which attempts to approximate the State-Value function. These agents are particularly well-suited to discrete problems, but start to show shortcomings when too many discrete actions are possible. In order to better enable the agent to capture the temporality of states, we use an LSTM as a policy network to learn chronological sequences.

Our results demonstrate that the double DQN-based approach, despite its critic-only nature, is capable of effectively capturing market dynamics and generating profitable trading signals. If we had wanted to go further, we could also have trained agents such as policy gradient, A2C and TD3, which are sometimes more efficient, especially on continuous action spaces, although more complicated to implement from scratch. This work underscores the potential of RL in commodity trading and lays the foundation for further exploration of advanced RL architectures.

2. Trading Environment

2.1. State space

In the litterature, many different features are used to represent the state spaces. But using the past prices is a common practice, along with adding technical indicators. For our work, we took the past price of each commodities (Open, Close, High, Low), and technical indicators as listed below:

- **Volume** : Number of shares or contracts traded for a security over a given period.
- **Relative Strength Index (RSI)** : Momentum oscillator that measures the speed and change of price movements.

$$RSI = 100 - \frac{100}{1 + RS} \tag{1}$$

where

$$RS = \frac{\text{Average Gain over 14 periods}}{\text{Average Loss over 14 periods}}.$$

- **Moving Average Convergence Divergence (MACD)** : Trend-following momentum indicator that illustrates the relationship between two exponential moving averages (EMAs) of a security's price.

$$MACD = EMA_{12} - EMA_{26} \tag{2}$$

where EMA_n represents the Exponential Moving Average over n periods.
- **MACD Signal** : 9-period Exponential Moving Average (EMA) of the MACD line.

$$Signal = EMA_9(MACD) \tag{3}$$

- **Commodity Channel Index (CCI)** : Oscillator used to measure deviations of the price from its statistical mean.

$$CCI = \frac{TP - SMA_{20}(TP)}{0.015 \times \text{Mean Deviation}} \tag{4}$$

where the Typical Price (TP) is given by:

$$TP = \frac{High + Low + Close}{3}$$

- **Average Directional Index (ADX)** : Measure the strength of a trend

$$ADX = EMA_{14}(DX) \tag{5}$$

where

$$DX = 100 \times \frac{|\text{DI}^+ - \text{DI}^-|}{\text{DI}^+ + \text{DI}^-}$$

Here, DI^+ and DI^- represent smoothed directional movement indicators.
- **Bollinger Bands** : Volatility indicator consisting of three bands: a middle band representing a simple moving average (SMA), and an upper and lower band placed at two standard deviations above and below the middle band.

$$BB_{Mid} = SMA_{20}(Close) \tag{6}$$

$$BB_{Upper} = BB_{Mid} + 2 \times \sigma_{20} \tag{7}$$

$$BB_{Lower} = BB_{Mid} - 2 \times \sigma_{20} \tag{8}$$

where σ_{20} is the rolling standard deviation over 20 periods.

2.2. Action space

We study a discrete action space. A simple action can be set as {-1, 0, 1}, and each value represents the trading decision directly, i.e. -1 corresponds to sell, 0 to hold and 1 to buy. Another approach commonly used is based on the trade position, also known as target orders [1]. This simple discrete action space will be furthermore improved to become {-1, -0.75, -0.50, -0.25, 0, 0.25, 0.50, 0.75, 1}. This representation is useful to discretise the amount of shares to buy or sell. Indeed, in the first approach, each decision moves all the actions held, and this new strategy, moves all or a part of all the actions held. For example, if the share price is 0.5, the agent invests 50% of his available capital in the commodity in question. Discretizing the space over 3 assets at the same time explodes the number of possible actions, since the total number of possible actions is equal to N^3, i.e. in our case $9^3 = 729$. We see here the limit of space discretization, which encourages us to improve our modeling by using other types of agent that are capable of easily exploiting continuous spaces, such as A2C, PG or TD3.

As mentioned above, we've simplified the problem by normalizing all assets in $[0.1, 1]$. This simplification has one impact: the agent no longer trades on true market prices. Nevertheless, the 3 commodities gold, silver and natural gas now have the same weight, which helps the agent to trade. Before standardization, a unit of gold cost around $2,000, while a unit of natural gas cost $6, making it very difficult for the agent to divert his capital. What's more, the environment imposes on the agent an initial balance of 100 units of assets as starting capital, which means that at the outset the agent cannot buy more than 100 units of assets. In addition, to make the environment more realistic, we impose a 2% transaction fee on the agent, to force him not to overtrade by buying assets excessively without charge.

2.3. Reward function

The reward in our trading environment reflects the agent's ability to grow its net worth over time. At each timestep t, the reward r_t is defined as:

$$r_t = \frac{NW_t - NW_{t-1}}{NW_{t-1} + \epsilon} - \text{penalty}_t \tag{9}$$

Where :
- NW_t is the agent's net worth at time t
- NW_{t-1} is the agent's net worth at the previous timestep
- $\epsilon = 10^{-8}$ is a small constant to prevent division by zero
- We increment -0.001 to penalty_t each time the agent tries to sell an asset without having any unit of it.

This reward defined in the environment allows the agent to maximize his net capital over time, but it also penalizes the agent for performing actions that would be impossible in real life, and which would lead the agent to develop suboptimal trading strategies.

3. Agent

3.1. Deep Q-Networks

Deep Q-Network (DQN) is a neural network-based implementation of Q-learning, a reinforcement learning algorithm that estimates the action-value function $Q(s, a)$, i.e.,

the expected return of taking action a in state s and following the optimal policy thereafter [3].

Unlike methods that learn only the state-value function $V(s)$, DQN explicitly learns the value of each action in a given state, which is crucial when action selection has a strong impact on outcomes. It is particularly useful in large or high-dimensional state spaces, where table-based Q-learning becomes infeasible.

3.1.1. Q-Learning and Bellman Equation

The main goal of DQN is to learn an optimal policy by updating an action-value function based on the Bellman equation:

$$Q_{target}(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

where s_t is the current state, a_t is the action taken, r_t is the reward received, s_{t+1} is the next state, and γ is the discount factor.

3.1.2. From DQN to Double DQN

A known limitation of DQN is to overestimate action values, due to using the same network to both select and evaluate actions when computing the target.

Double DQN addresses this by separating these two steps: the online network selects the action, while the target network evaluates it. The updated target becomes:

$$Q_{\text{target}}(s_t, a_t) = r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}))$$

where Q_θ is the online network, and Q_{θ^-} is the target network.

Thus the Double DQN reduces Q-value overestimation leading to more stable training.

3.1.3. Training the Q-Network

In Double DQN, the Q-network is trained to minimize the gap between its predictions and a more stable target. At each timestep t, the loss is defined as:

$$L(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}} \left[\left(y_t - Q_\theta(s_t, a_t) \right)^2 \right]$$

with the target value given by:

$$y_t = r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}))$$

Here, θ are the parameters of the current (online) network, and θ^- are the parameters of the target network, updated periodically for stability. The training data is sampled from a replay buffer $\mathcal{D}$, which stores past transitions to reduce temporal correlations during learning.

To better capture temporal dependencies in the environment we integrate a Long Short-Term Memory (LSTM) layer into the Q-network architecture, thus the output of the LSTM is then fed into fully connected layers to predict Q-values for all possible actions at time t. Combining Double DQN with an LSTM allows the agent to both stabilize learning and better exploit temporal information.

3.2. Other approaches

We have used a Deep Q-Network (DQN) for this reinforcement learning task, which is a Critic-Only method. However, we could have also explored other approaches such as Policy Gradient, which is an Actor-Only method, where the policy is directly optimized through gradient ascent. Alternatively, the Advantage Actor-Critic (A2C) method, which combines both an actor and a critic, could have been employed to benefit from the advantages of both value-based and policy-based methods.

Another interesting direction would have been to use Ensemble Learning, where multiple models, including Actor-Only, Critic-Only, and Actor-Critic methods, are combined to improve the overall performance and robustness of the reinforcement learning agent [4].

4. Experiments

4.1. Description of Dataset

We use data from 3 commodities : Gold, Silver and Natural Gas extracted from Yahoo Finance. Our dataset ranges from 2010 to 2022 and is split into training, validation and test. The training is done from 2010 to 2017 included; validation from 2018 to 2019 included; test from 2020 to 2022 included. Figure 1 shows the repartition of the dataset through training, validation and test for the 3 commodities open price.

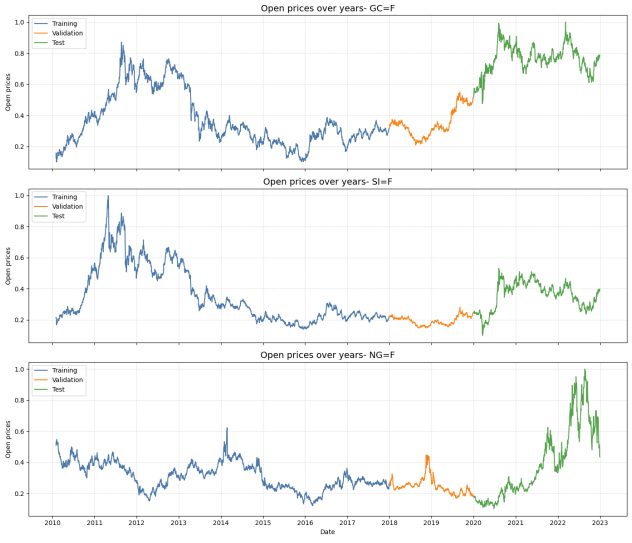


Figure 1. Open price evolution for Gold, Silver and Natural Gas

4.2. Baseline Algorithm

We compare our method to a Bollinger baseline model. It is a trading strategy that uses the middle Bollinger Band (see equations : 6, 7, 8) as a dynamic reference point to identify trends, with price movements above or below the baseline signaling potential buy or sell opportunities. Its action space is {-1, 0, 1}. Figure 2 shows the Bollinger computed curves for the three commodities.

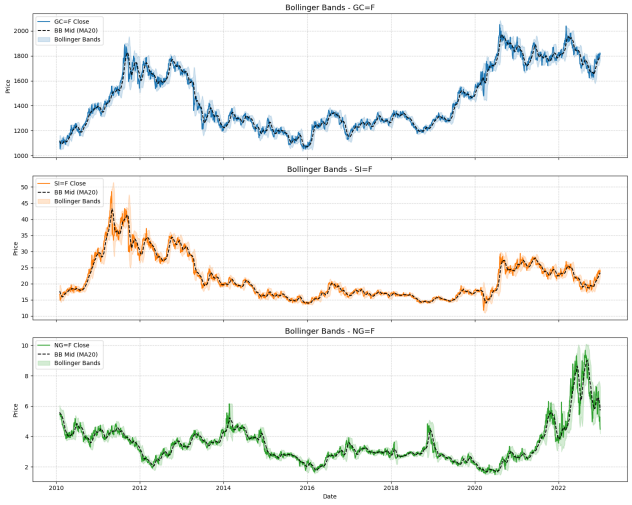


Figure 2. Bollinger curves for the three commodities

The following pseudo code is the deterministic trading algorithm we use, which bases its buys, sells and stays on whether the low or high bollinger bands have been breached in relation to the mean signal open price.

Algorithm 1 Deterministic Bollinger Band Trading Algorithm

Input: Price series P_t, window size n, coefficient k
Initialize: Action space $\mathcal{A} = \{-1, 0, 1\}$
for each time step $t = n, \ldots, T$ **do**
  Compute moving average: μ_t
  Compute standard deviation: σ_t
  Compute upper band: $UB_t = \mu_t + k \cdot \sigma_t$
  Compute lower band: $LB_t = \mu_t - k \cdot \sigma_t$
  if $P_t < LB_t$ **then**
    Action $a_t \leftarrow 1$ ▷ Buy signal
  else if $P_t > UB_t$ **then**
    Action $a_t \leftarrow -1$ ▷ Sell signal
  else
    Action $a_t \leftarrow 0$ ▷ Hold
  end if
  Execute a_t
end for

4.3. Training and hyperparameters

We train a Double DQN agent with a Long Short-Term Memory (LSTM) network as a Q-function approximator. To improve sample efficiency and stability, we use a replay buffer that stores past experiences and enables training on randomly sampled mini-batches. The target network is updated periodically to reduce overestimation bias. Each input to the agent is a sequence of 10 observations, allowing it to capture temporal patterns in financial time series.

Hyperparameter	**Value**
Q-network architecture	LSTM
Hidden layer dimension	128
Double Q-learning	Enabled
Replay buffer	Active
Sequence length	10
Learning rate (α)	10^{-4}
Batch size	32
Discount factor (γ)	0.99
Target network update	Hard update
Epsilon decay	Linear (by epoch)
Epochs	10
Optimizer	Adam

Table 1. Training hyperparameters for the Double DQN agent

4.4. Experimental Results

In order to compare the Double DQN agent we're training with other strategies, we've chosen to use the performance metrics listed below. We'll compare performance on the same test data set, from 2020 to 2022.

4.4.1. Performance Metrics [2]

These performance metrics listed below are widely adopted in financial literature:

1. **Expected Return** $E(R)$: Annualised average return per trade, reflecting overall profitability.
2. **Volatility** std(R): Annualised standard deviation of returns, measuring total risk or variability in performance.
3. **Downside Deviation (DD)**: Annualised standard deviation of negative returns only, quantifying downside risk.
4. **Sharpe Ratio**: A classical risk-adjusted performance metric, defined as:

$$\text{Sharpe} = \frac{E(R)}{\text{std}(R)}$$

5. **Sortino Ratio**: A variation of the Sharpe Ratio that penalizes only downside volatility:

$$\text{Sortino} = \frac{E(R)}{\text{Downside Deviation}}$$

6. **Maximum Drawdown (MDD)**: The largest observed loss from a peak to a trough in the equity curve, highlighting worst-case risk.
7. **Calmar Ratio**: Compares annualised return to maximum drawdown:

$$\text{Calmar} = \frac{E(R)}{\text{MDD}}$$

   Higher values indicate better risk-adjusted performance.
8. **Percentage of Positive Returns**: The proportion of trades that end with a profit, expressed as a percentage.
9. **Average Profit / Average Loss**: Ratio between the mean return of profitable trades and the mean return of losing trades.

4.4.2. General performances

The Table 2 presents the different results obtained for each model. From it, we can conclude that the DQN model performs better than the Bollinger strategy across most major performance indicators.

For example, the average return is greater (E(R) = 0.0022 compared to 0.0006) with a little more volatility, but it is compensated by a much better risk-adjusted returns based on the Sharpe Ratio (0.045 compared to 0.0191) and Sortino Ratio (0.0665 compared to 0.025).

Besides, DQN has the smallest maximum drawdown (MDD = 0.5266 vs. 0.6072) and a more excellent Calmar ratio, thus better preserving the capital in declining periods.

For the positive returns generated, they are almost similar at 52.86% vs. 53.25%, while a better average profit-to-loss ratio was reported for DQN (1.023 vs. 0.9301), hence implying more reliable and profitable trades.

Metric	DQN	Bollinger
Expected return $E(R)$	0.0022	0.0006
Volatility std(R)	0.048	0.0299
Daily drawdown (DD)	0.1601	0.1635
Sharpe Ratio	0.045	0.0191
Sortino Ratio	0.0665	0.025
Max Drawdown (MDD)	0.5266	0.6072
Calmar Ratio	0.0041	0.0009
% Positive Returns	52.86%	53.25%
Av. P /Av. L	1.023	0.9301

Table 2. Performance comparison between DQN and Bollinger strategy on the Test set

4.4.3. Double DQN agent performances

To adapt the penalty in the reward function and check performance distribution, we plotted Figure 3. This shows that our agent is well suited for his task as the reward's distribution is on average around 0 and more positive than negative.

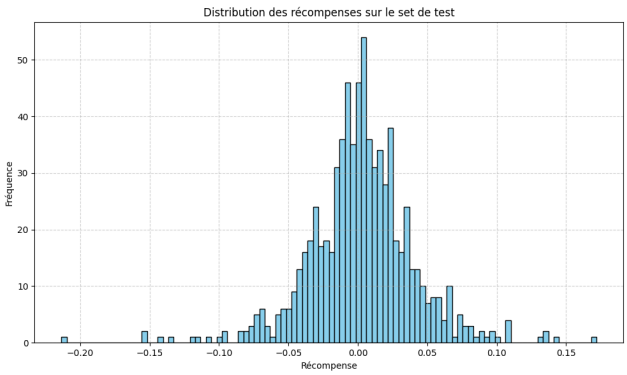


Figure 3. Distribution of test reward for DQN

In Figure 4 we see the net worth evolution for the DQN strategy over the Test set. At first, it had trouble making a profit, but after a year, it started increasing gradually. When it reaches the year 2022, it starts oscillating, this could be the consequences of multiple causes : Being stuck in a suboptimal solution, volatile market, overfitting to the past experiences.

Nevertheless, at the end, the agent is performant, generating a +150% net worth profit. This value could be criticized as in a real market environment with more parame-

ters and commodities, it may not reach this performance. Yet, with the simplification of this environment, the performances are acceptable.

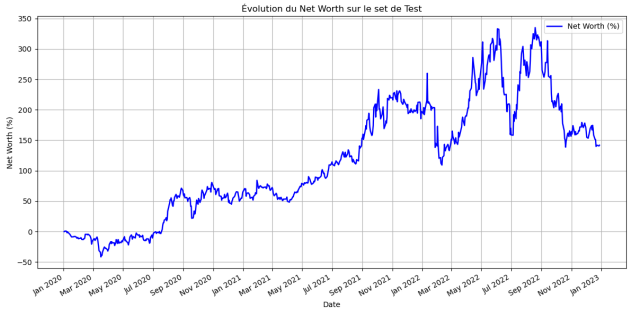


Figure 4. Net worth for Test set with DQN

In order to have a better understanding of how the agent takes its decision, we plotted the different strategies used in our commodities, shown in Figure 5. These graphs present firstly the close's price of the commodities on top with the *Buy* or *Sell* indicators. Under it, we have the action intensity, which is what discrete action has taken the agent at each step. Finally, the last graph in blue is the number of shares held at each step.

Thus, the combination of the three commodities, help us to understand how the net worth evolves. For example Gold is increasing during the first year, and then decreasing slowly, to make profit, the agent will buy before it gets high, and sell when it sees the first opportunity in July 2020. Then it will wait until a decrease to buy in November 2022.

Silver has a similar close price curve than gold, so the strategy will be similar. The difference is that the agent will wait for a certain decrease and not sell hastily until January 2022.

Finally for Natural Gas, the close price is mainly increasing, then the agent will principally buy and hold to make profit.

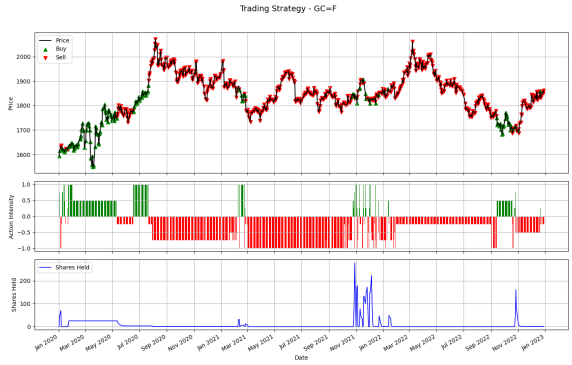
The three curves of shares held shows a noisy region around January 2022, this may be due to different causes : The agent is near a certain decision threshold and fluctuates a lot around it, The agent does a bad action approximation, the environment is too volatile.

Note that for the *Buy* or *Sell* indicators, if no shares are held or if we have reached the maximum of what we can buy, the agent doesn't act. It's a signal, but the action depends also on the context.

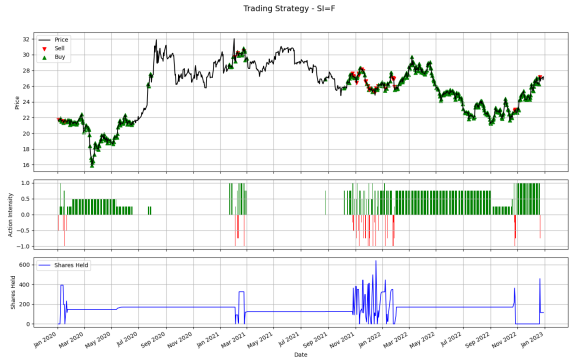
In general, the three commodities seem to be well managed by the agent in order to make profit, and it resulted in a high net worth pourcentage as seen in Figure 4.

4.4.4. Bollinger Bands agent performances

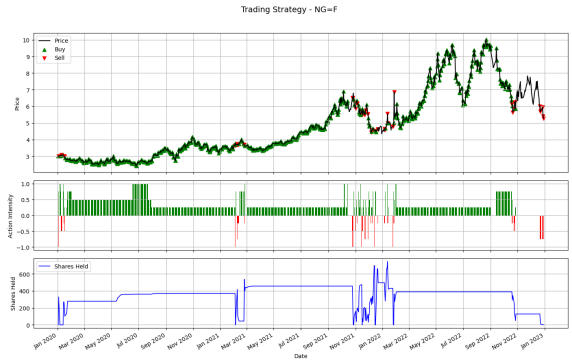
In Figure 6, we see the net worth evolution of the Bollinger strategy over the test set. It is less performant than the DQN



(a) Gold



(b) Silver



(c) Natural Gas

Figure 5. Trading strategy on the Test set for each commodity using DQN

strategy. Nevertheless, at the beginning, it had trouble making profit and fell to -60%, but afterward it got up to around 20%. At the end the profit falls a little bit, to end at 10%. This shows some bad performances, the agent didn't succeed to make a lot of profit, as the environment may be too volatile, or the strategy too simplistic.

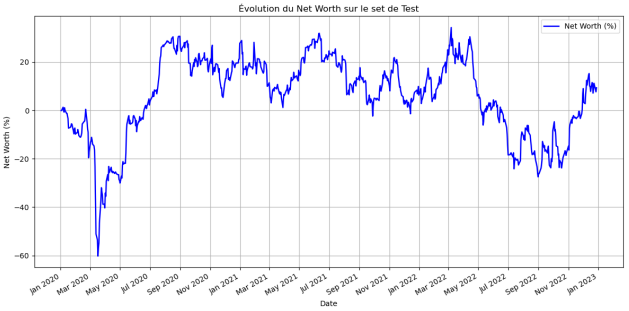


Figure 6. Net worth for Test set with Bollinger

The strategies of this algorithm are shown in Figure 7. On Gold, the agent seems to have buy and sell indicators at good strategic points, following the bollinger bands shown in Figure 2. But, it will buy a lot when the price is high, and lose profit as it will decrease through time.

For Silver, it started well as the agent bought a lot at the beginning, but it sold everything just before it rose. Then it buys gradually, but the price decreases as for Gold.

Natural Gas actions are not very bought overall, as Bollinger bands cross a lot of times and send sell signals. Thus it didn't generate profit.

4.4.5. Comparison

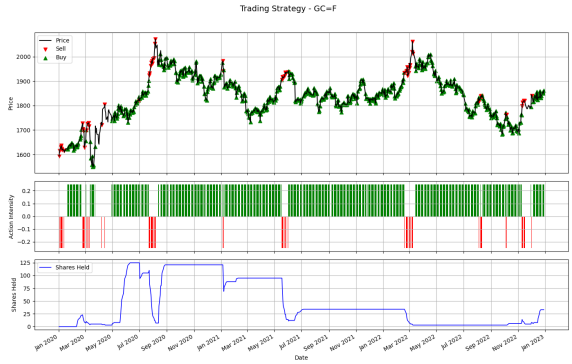
At the end, when we compare the different curves of net worth's evolution in Figure 8, we clearly see that the DQN strategy outperforms the Bollinger baseline. Indeed, at the beginning the two curves seem to be similar, but after one year, the Bollinger stays constant, while the DQN increases gradually. It shows that Bollinger didn't adapt to its environment as it is a deterministic algorithm. On the other hand, even in a volatile market, the DQN succeeded in making profit, through multiple years.

5. Discussion

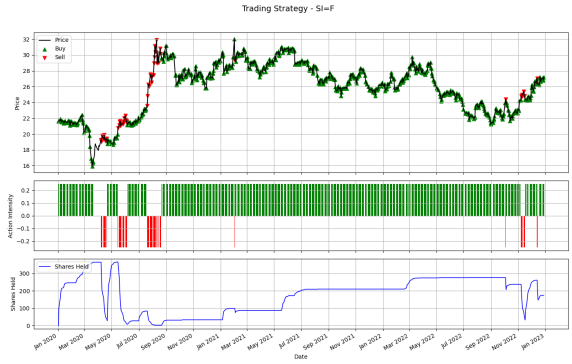
The project's findings present various interesting topics for further research. Investigating alternative reinforcement learning architectures could be one of them. For example, actor-critic methods like A2C and TD3, as well as actor-only methods like Policy Gradient, may provide better results, especially when working with continuous action spaces.

Improving temporal modeling is another research area. Financial data are by nature sequential, thus, using more temporal models may help to better capture dependencies. Methods like attention mechanisms or deeper LSTM networks may provide a better comprehension of the underlying market's dynamics.

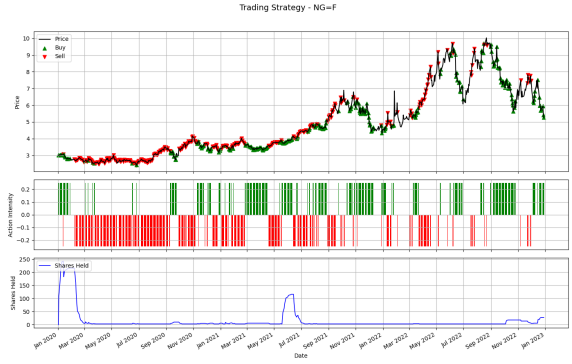
Capital allocation and risk management is also a crucial element in trading. By analyzing dynamic risk management and improving capital allocation, we may obtain a more



(a) Gold



(b) Silver



(c) Natural Gas

Figure 7. Trading strategy on the Test set for each commodity using Bollinger Bands

robust and successful trading strategy. The agent's performance might be further stabilized and improved under different market conditions by taking care of these factors.

Additionally, using multi-agent reinforcement learning or ensemble approaches may produce a more stable trading strategy [4]. Combining the advantages of several models could improve decision-making in a volatile market.

Lastly, it is important to extend the research with unnormalized's real-world commodity prices in order to confirm

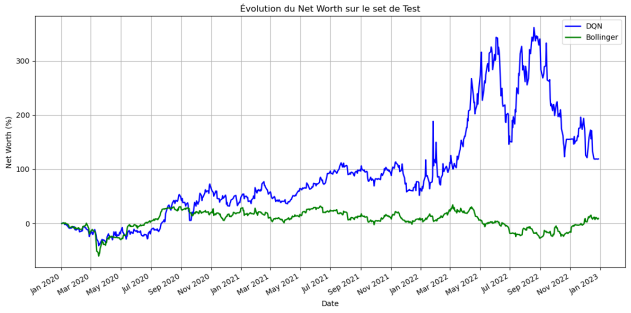


Figure 8. Benchmark of Net worth for Test set

the study's applicability. A more realistic tested with greater variety of transaction costs could help guaranteeing that the strategies created are strong enough to be applicable in real-world trading situations.

6. Conclusion

The aim of this project was to train a Deep Reinforcement Learning (DRL) model to trade intelligently on the commodities market, which is known to be a sideways market and therefore not bullish. We have specifically chosen to implement a Double DQN, with a fully customized environment that encourages the agent to maximize the net value of its portfolio (taking into account transaction costs) by trading on 3 commodities at the same time: gold, silver and natural gas. Given the complexity of this project, we have made several simplifying assumptions which mean that the final agent cannot be used in a real life environment trading:

- Price normalization in $[0.1, 1]$ to give each asset the same weighting (cf. 2.2).
- Discrete, non-continuous actions that limit the agent's portfolio allocation.
- Fixed transaction costs of 2%.

By applying these simplifying assumptions, we were able to train a Double DQN agent to perform very well, intelligently allocating his portfolio to all 3 assets at the same time. In 3 years, the agent achieves 100% net returns and reaches a maximum of 300% net returns before falling. We have also demonstrated that the agent outperforms the deterministic Bollinger bands strategy widely used in the trading world and ineffective on sideways markets. Studies to prove that the agent is really effective on these markets are obviously not effective enough. A more thorough comparison with other baseline strategies (e.g. MACD, Long Only) is necessary, and the average returns over several training sessions would give more interpretable performances. In fact, the agent can sometimes be trained to fall into sub-optimal strategies, requiring several training sessions before obtaining a model that works.

References

[1] Chien Yi Huang. Financial trading as a game: A deep reinforcement learning approach, 2018. 2

[2] Bryan Lim, Stefan Zohren, and Stephen Roberts. Enhancing time series momentum strategies using deep neural networks. 2020. 5

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. 3

[4] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. 2020. 3, 7

[5] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep reinforcement learning for trading. 2019.