# Knowledge discovery with Deep RL for selecting financial hedges

**Eric Benhamou**[1,2], **David Saltiel**[1,3], **Sandrine Ungari**[4],
**Abhishek Mukhopadhyay**[4], **Jamal Atif**[3], **Rida Laraki**[3]

[1] Ai For Alpha, France, {eric.benhamou,david.saltiel}@aiforalpha.com
[2]MILES, LAMSADE, Dauphine PSL, France {jamal.atif,rida.laraki}@lamsade.dauphine.fr
[3] LISIC, ULCO, France [4] Societe Generale, Cross Asset Quantitative Research, UK and France
{sandrine.ungari,abhishek.mukhopadhyay}@sgcib.com

**Abstract**

Can an asset manager gain knowledge from different data sources to select the right hedging strategy for his portfolio? We use Deep Reinforcement Learning (Deep RL or DRL) to extract information from not only past performances of the hedging strategies but also additional contextual information like risk aversion, correlation data, credit information and estimated earnings per shares. Our contributions are threefold: (i) the use of contextual information also referred to as augmented state in DRL, (ii) the impact of a one period lag between observations and actions that is more realistic for an asset management environment, (iii) the implementation of a new repetitive train test method called walk forward analysis, similar in spirit to cross validation for time series. Although our experiment is on trading bots, it can easily be translated to other bot environments that operate in sequential environment with regime changes and noisy data. Our experiment for an augmented asset manager interested in finding the best portfolio for hedging strategies achieves superior returns and lower risk.

## Introduction

Can an asset manager gain knowledge from different data sources to select the right hedging strategy for his portfolio with sequential observations? By knowledge, we mean capture time dependencies between some relevant features in order to time some hedging strategies. By sequential, we mean that chronological order matters and that observations are completely modified if we change order. To answer this question, we use a deep reinforcement learning method whose aim is to gain knowledge between a wide range of data (financial and non financial data that augment initial observations with contextual information) and the optimal investment asset allocation. Indeed, asset management is a well-suited industry to apply this new robotic machine learning: large amount of data available due to electronic tradings and strategic, use of systematic investment methods that do not give any hold on emotional and behavioral bias and scientific approach in exploring and using data for non conventional knowledge discovery. Surprisingly, machine learning is still not widely used in investment decision because of the complexity of the learning environment. Hence, asset managers are still largely relying on traditional methods, based on human decisions.

This is in sharp contrast with recent advances of deep reinforcement learning (DRL) on challenging tasks like game (Atari games from raw pixel inputs Mnih et al. (2013, 2015), Go Silver et al. (2016), StarCraft II Vinyals et al. (2019)), but also more robotic learning like advanced locomotion and manipulation skills from raw sensory inputs (Levine et al. (2015, 2016) Schulman et al. (2015, 2017), Lillicrap et al. (2015)), autonomous driving (Wang, Jia, and Weng (2018)) and general bot learning (Gu et al. (2017)).

We investigate if deep reinforcement learning can help creating an augmented asset manager when solving a classical portfolio allocation problem: finding hedging strategies to an existing portfolio, the MSCI World index in our example. The hedging strategies are different strategies operated by standard bots that have different logics and perform well in different market conditions. Knowing when to add and remove them and when to decrease or increase their asset under management is a fundamental but challenging question for an augmented asset manager.

### Related works

At first, reinforcement learning was not used in portfolio allocation. Initial works focused on trying to use deep networks to forecast next period prices, as presented in Freitas, De Souza, and Almeida (2009), Niaki and Hoseinzade (2013), Heaton, Polson, and Witte (2017). These models solved a supervised learning task akin to a regression, and tried to predict prices using past information only and compute portfolio allocations based on forecast. For asset managers, this initial usage of machine learning contains multiple problems. First, it does not ensure that the prediction is reliable in the near future: financial markets are well known to be non stationary and to present regime changes as illustrated in Salhi et al. (2015), Dias, Vermunt, and Ramos (2015). Second, this approach does not address the question of finding the optimal portfolio based on some reward metrics. Third, it does not adapt to changing environment and does not easily incorporate transaction costs.

A second stream of research around deep reinforcement learning has emerged to address those points: Yu et al. (2019); Wang and Zhou (2019); Liu et al. (2020); Ye et al.

(2020); Li et al. (2019). The dynamic nature of reinforcement learning makes it an obvious candidate for changing environment Benhamou et al. (2020a), Benhamou et al. (2020b). Transaction costs can be easily included in rules Yu et al. (2019); Wang and Zhou (2019); Liu et al. (2020); Ye et al. (2020); Yu et al. (2019). However, these works, except Ye et al. (2020) rely on time series of open high low close prices, which are known to be very noisy. Secondly, they all assume an immediate action after observing prices which is quite different from reality as asset managers need a one day turnaround to manage new portfolio positions. Thirdly, they mostly rely on a single reward function and do not measure the impact of the reward function. Last but not least, they only do one train and test period, and never test for model stability.

More generally, DRL has recently been applied to other problems than portfolio allocation and direct trading strategies Zhang, Zohren, and Roberts (2019), Huang (2018), Théate and Ernst (2020), Nan, Perumal, and Zaiane (2020), Wu et al. (2020) or to the case of multi agents Bao and yang Liu (2019) or to optimal execution Ning, Lin, and Jaimungal (2018).

## Contributions

Our contributions are threefold:

- **The addition of contextual information.** Using just past information is not sufficient for bot learning in a noisy and fast changing environment. The addition of contextual information improves results significantly. Technically, we create two sub-networks: one fed with direct observations (past prices and standard deviation) and another one with contextual information (level of risk aversion in financial markets, early warning indicators for future recession, corporate earnings...).
- **One day lag between price observation and action.** We assume that prices are observed at time $t$ but action only occurs at time $t+1$, to be consistent with reality. This one day lag makes the RL problem more realistic but also more challenging.
- **The walk-forward procedure.** Because of the non stationarity nature of time dependent data and especially financial data, it is crucial to test DRL models stability. We present a new methodology in DRL model evaluation referred to as walk forward analysis that iteratively trains and tests the model on extending data-set. This can be seen as the analogy of cross validation for time series. This allows to validate that selected hyper parameters work well over time and that resulting models are stable over time.

## Background and mathematical formulation

In standard bot reinforcement learning, models are based on Markov Decision Process (MDP) as in Sutton and Barto (2018). MDP assumes that the bot knows all the states of the environment and has all the information to make the optimal decision in every state. The Markov property in addition implies that knowing the current state is sufficient.

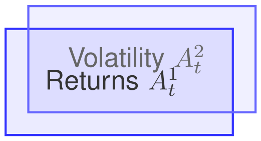
Yet, the traditional MDP framework is inappropriate here: noise may arise in financial market data due to unpredictable external events. We prefer to use Partially Observable Markov Decision Process (POMDP) as presented initially in Astrom (1969). In POMDP, only a subset of the information of a given state is available. The partially-informed agent cannot behave optimally. He uses a window of past observations to replace states as in a traditional MDP.

Mathematically, POMDP is a generalization of MDP. Recall that MDP assumes a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the state action to next state transition probability function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, and $\mathcal{R}$ is the immediate reward. The goal of the agent is to learn a policy that maps states to the optimal action $\mu : \mathcal{S} \rightarrow \mathcal{A}$ and that maximizes the expected discounted reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t]$. POMPD adds two more variables in the tuple, $\mathcal{O}$ and $\mathcal{Z}$ where $\mathcal{O}$ is the set of observations and $\mathcal{Z}$ is the observation transition function $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0, 1]$. At each time, the agent is asked to take an action $a_t \in \mathcal{A}$ in a particular environment state $s_t \in \mathcal{S}$, that is followed by the next state $s_{t+1}$ with transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$. The next state $s_{t+1}$ is not observed by the agent. It rather receives an observation $o_{t+1} \in \mathcal{O}$ on the state $s_{t+1}$ with probability $Z(o_{t+1}|s_{t+1}, a_t)$.

From a practical standpoint, the general RL setting is modified by taking a pseudo state formed with a set of past observations $(o_{t-n}, o_{t-n-1}, \ldots, o_{t-1}, o_t)$. In practice to avoid large dimension and the curse of dimension, it is useful to reduce this set and take only a subset of these past observations with $j < n$ past observations, such that $0 < i_1 < \ldots < i_j$ and $i_{k,1 \leq k \leq j} \in \mathbb{N}$ is an integer. The set $\delta_1 = (0, i_1, \ldots, i_j)$ is called the observation lags. In our experiment we typically use lag periods like (0, 1, 2, 3, 4, 20, 60) for daily data, where the tuple $(0, 1, 2, 3, 4)$ is indeed the last week observation, 20 is for the one-month ago observation (as there is approximately 20 business days in a month) and 60 the three-month ago observation.

## Observations

**Regular observations** There are two types of observations: regular and contextual information. Regular observations are data directly linked to the problem to solve. For a standard bot, these are observations from its environment like position of the arm, degree, etc. In the case of a trading bot, regular observations are past prices observed over a lag period $\delta = (0 < i_1 < \ldots < i_j)$. To re-normalize data, we rather use past returns computed as $r_t^k = \frac{p_t^k}{p_{t-1}^k} - 1$ where $p_t^k$ is the price at time $t$ of the asset $k$. For a financial asset $k$, to give information about regime changes, our trading bot receives also empirical standard deviation computed over a sliding estimation window denoted by $d$ as follows $\sigma_t^k = \sqrt{\frac{1}{d} \sum_{u=t-d+1}^{t} \left(r_u^k - \mu\right)^2}$, where the empirical mean $\mu^k$ is computed as $\mu^k = \frac{1}{d} \sum_{u=t-d+1}^{t} r_u^k$. Hence our regular observations is a three dimensional tensor represented as follows:



with $A_t^1 = \begin{pmatrix} r^1_{t-i_j} \dots r^1_t \\ \dots \dots \dots \\ r^m_{t-i_j} \dots\dots r^m_t \end{pmatrix}$, $A_t^2 = \begin{pmatrix} \sigma^1_{t-i_j} \dots \sigma^1_t \\ \dots \dots \dots \\ \sigma^m_{t-i_j} \dots\dots \sigma^m_t \end{pmatrix}$

This setting with two layers (past returns and past volatilities) is quite different from the one presented in Liang et al. (2018) that uses different layers representing open, high, low and close prices. There are various remarks to be made. First, high low information does not make sense for portfolio strategies that are only evaluated daily, which is the case of all the funds. Secondly, open high low prices tend to be highly correlated creating some noise in the inputs. Third, the concept of volatility is crucial to detect regime change and is surprisingly absent from these works as well as from other works like Yu et al. (2019); Wang and Zhou (2019); Liu et al. (2020); Ye et al. (2020); Li et al. (2019).

**Context observation** Contextual observations are additional information that provides intuition about current context. For our asset manager bot, they are other financial data not directly linked to its portfolio assumed to have some predictive power for portfolio assets. In the case of a financial portfolio, context information is typically modelled by a large range of features :

- the level of risk aversion in financial markets, or market sentiment, measured as an indicator varying between 0 for maximum risk aversion and 1 for maximum risk appetite,
- the bond/equity historical correlation, a classical ex-post measure of the diversification benefits of a duration hedge, measured on a 1-month, 3-month and 1-year rolling window,
- The credit spreads of global corporate - investment grade, high yield, in Europe and in the US - known to be an early indicator of potential economic tensions,
- The equity implied volatility, a measure if the 'fear factor' in financial market,
- The spread between the yield of Italian government bonds and the German government bond, a measure of potential tensions in the European Union,
- The US Treasury slope, a classical early indicator for US recession,
- And some more financial variables, often used as a gauge for global trade and activity: the dollar, the level of rates in the US, the estimated earnings per shares (EPS).

On top of these observations, we also include the maximum and minimum portfolio strategies return and the maximum portfolio strategies volatility. The latter information is like for regular observations motivated by the stylized fact that standard deviations are useful features to detect crisis.

Contextual observations are stored in a 2D matrix denoted by $C_t$ with stacked past $p$ individual contextual observations. The contextual state writes as $C_t = \begin{pmatrix} c^1_t \dots c^1_{t-i_k} \\ \dots \dots \dots \\ c^p_t \dots\dots c^p_{t-i_k} \end{pmatrix}$. The matrix nature of contextual states $C_t$ implies in particular that we will use 1D convolutions should we use convolutional layers. All in all, observations that are augmented observations, write as $O_t = [A_t, C_t]$, with $A_t = [A_t^1, A_t^2]$ that will feed the two sub-networks of our global network as presented in figure 1.

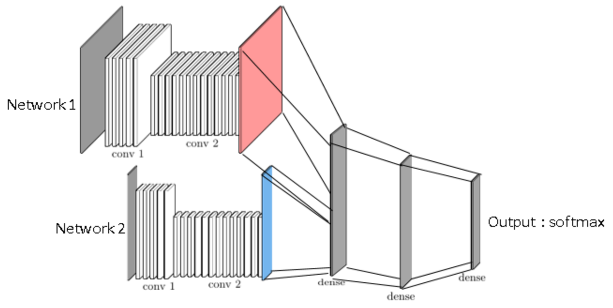


Figure 1: Network architecture

## Action

In our deep reinforcement learning the augmented asset manager trading bot needs to decide at each period in which hedging strategy it invests. The augmented asset manager can invest in $l$ strategies that can be simple strategies or strategies that are also done by asset management bots. To cope with reality, the bot will only be able to act after one period. This is because asset managers have a one day turn around to change their positions. We will see on experiments that this one day turnaround lag makes a big difference in results. As it has access to $l$ potential hedging strategies, the output is a $l$ dimension vector that provides how much it invests in each hedging strategy. For our deep network, this means that the last layer is a softmax layer to ensure that portfolio weights are between 0 and $100\%$ and sum to 1.

## Reward

There are multiple choices for our reward. A straightforward reward function is to compute the final net performance of the combination of our portfolio computed as the value of our portfolio at the last train date $t_T$ over the initial value of the portfolio $t_0$ minus one: $P_{t_T}/P_{t_0} - 1$. Another natural reward function is to compute the Sortino ratio, that is a variation of Sharpe ratio where risk is computed by the downside standard deviation (instead of regular standard deviation) whose definition is to compute the standard deviation only on negative daily returns denoted by $(\tilde{r}_t)_{t=0..T}$ . Hence the downside standard deviation is computed by $\sqrt{250} \times \text{StdDev}[(\tilde{r}_t)_{t=0..T}]$.

## Adversarial Policy Gradient

A policy is a mapping from the observation space to the action space, $\pi : \mathcal{O} \rightarrow \mathcal{A}$. To achieve this, a policy is specified by a deep network with a set of parameters $\vec{\theta}$. The action is a vector function of the observation given the parameters: $\vec{a}_t = \pi_{\vec{\theta}}(\boldsymbol{o}_t)$. The performance metric of $\pi_{\vec{\theta}}$ for time interval $[0, t]$ is defined as the corresponding total reward function of the interval $J_{[0,t]}(\pi_{\vec{\theta}}) = R\left(\vec{o}_1, \pi_{\vec{\theta}}(o_1), \cdots, \vec{o}_t, \pi_{\vec{\theta}}(o_t), \vec{o}_{t+1}\right)$. After random initialization, the parameters are continuously updated along the gradient direction with a learning rate $\lambda$: $\vec{\theta} \longrightarrow \vec{\theta} + \lambda \nabla_{\vec{\theta}} J_{[0,t]}(\pi_{\vec{\theta}})$. The gradient ascent optimization is done with standard Adam (short term for Adaptive Moment Estimation) optimizer to have the benefit of adaptive gradient descent with root mean square propagation Kingma and Ba (2014). The whole process is summarized in algorithm 1, called adversarial policy gradient as we introduce randomisation both in the observations and the action (to have standard exploration exploitation). This two steps randomization ensures more robust training as we will see in the experiments. Noise in observations has already been suggested to improve training in Liang et al. (2018).

**Algorithm 1** Adversarial Policy Gradient

```
1:  Input: initial policy parameters $\theta$, empty replay buffer $\mathcal{D}$

2:  repeat
3:      reset replay buffer
4:      while not terminal do
5:          Observe observation $o$ and select action $a = \pi_\theta(o)$
            with probability $p$ and random action with probability $1 - p$,
6:          Execute $a$ in the environment
7:          Observe next observation $o'$, reward $r$, and done
            signal $d$ to indicate whether $o'$ is terminal
8:          apply noise to next observation $o'$
9:          store $(o, a, o')$ in replay buffer $\mathcal{D}$
10:         if Terminal then
11:             for however many updates in $\mathcal{D}$ do
12:                 compute final reward $R$
13:             end for
14:             update network parameter with Adam gradient
                ascent $\vec{\theta} \longrightarrow \vec{\theta} + \lambda \nabla_{\vec{\theta}} J_{[0,t]}(\pi_{\vec{\theta}})$
15:         end if
16:     end while
17: until convergence
```

## Walk forward analysis

In machine learning, the standard approach is to do $k$-fold cross validation as shown in figure 2. White rectangles represent training periods while grey rectangles testing periods. This approach breaks the chronology of data and potentially uses past data in the test set. Rather, we can take sliding test set and take past data as training data. We can either take the training data set with a fixed starting point and grow the training data set by adding more and more data, which is what we call extending walk forward as shown in figure 3 or take always the same amount of data and slide the training data (figure 4), hence the name of sliding walk forward. Extending walk forward tends to more stable models as we add incrementally new data, at each new training step, and share all past data. The negative effect of this is to adapt slowly to new information. On the opposite, sliding walk forward leads to more rapidly changing models as we progressively drop old data and hence give more weight to more recent data. To our experience, because we do not have so much data to train our DRL model, it is better to use extending walk forward. Last but not least, as the test set is always after the train set, walk forward analysis gives less steps compared to cross validation. In practice for our data set, we train our models from 2000 to end of 2006 (to have at least seven years of data) and use an extending test period of one year.

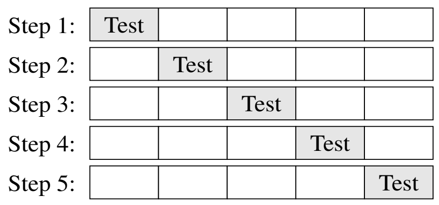


Figure 2: k-fold cross validation

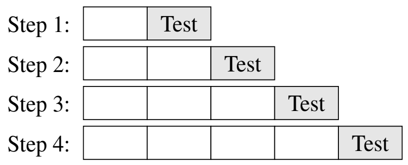


Figure 3: Extending Walk Forward

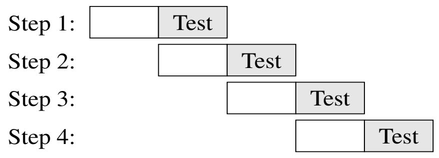


Figure 4: Sliding Walk Forward

## Experiments

### Goal of the experiment

We are interested in finding a hedging strategy for a risky asset. The experiment is using daily data from 01/05/2000 to 19/06/2020. The risky asset is the MSCI world index (see https://www.msci.com/ - data source: SG CIB). We choose the MSCI world index because it is a good proxy for a wide range of asset manager portfolios. The hedging strategies are 4 SG CIB proprietary systematic strategies (see https://sgi.sgmarkets.com/ - data source: SG CIB), computed and executed by trading bots and further described below. As we use extending walk forward traning, train set is initially from 2000 to 2006 with a test set in 2007, then training set is from 2000 to 2007, with a test set in 2008, and etc up to the last training set which is from 2000 to 2019 with a test set in 2020.

## Data-set description

Systematic strategies are asset management bots that invest in financial markets according to adaptive and pre-defined trading rules. Here, we use 4 SG CIB proprietary 'hedging strategies', that tend to perform when stock markets are down:

- Directional hedges - react to small negative return in equities,
- Gap risk hedges - perform well in sudden market crashes,
- Proxy hedges - tend to perform in some market configurations, like for example when highly indebted stocks under-perform other stocks,
- Duration hedges - invest in bond market, a classical diversifier to equity risk in finance.

The underlying financial instruments vary from put options, listed futures, single stocks, to government bonds. Some of those strategies are akin to an insurance contract and bear a negative cost over the long run. The challenge consists in balancing cost versus benefits. In practice, asset managers have to decide how much of these hedging strategies are needed on top of an existing portfolio to achieve a better risk reward. The decision making process is often based on contextual information, such as the economic and geopolitical environment, the level of risk aversion among investors and other correlation regimes. A cross validation step selects the most relevant features contextual information. In the present case, the first three features are selected. The rebalancing of strategies in the portfolio comes with transaction costs, that can be quite high since hedges use options. Transactions costs are like frictions in physical systems. They are taken into account dynamically to penalise solutions with a high turnover rate.

## Evaluation metrics

Asset managers use a wide range of metrics to gauge the success of their investment decision. To keep things simple, we use the following metrics:

- annualized return defined as the average annualized compounded return,
- annualized daily based Sharpe ratio defined as the ratio of the annualized return over the annualized daily based volatility $\mu/\sigma$,
- Sortino ratio computed as the ratio of the annualized return overt the downside standard deviation,
- maximum drawdown denoted by max DD in table 4.

Let $P_T$ be the final value of the portfolio at time $T$ and $P_0$ its initial value at time $t = 0$. Let $\tau$ be the year fraction of the final time $T$. The annualized return is defined as $\mu = (P_T/P_0)^{1/\tau} - 1$. The maximum drawdown is computed as the maximum of all daily drawdowns. The daily drawdown is computed as the ratio of the difference between the running maximum of the portfolio value ($RM_T = \max_{t=0..T}(P_t)$ ) and the portfolio value over the running maximum of the portfolio value. Hence $DD_T = (RM_T - P_T)/RM_T$ and $MDD_T = \max_{t=0..T}(DD_t)$.

## Baseline

**Pure risky asset** This first evaluation is to compare our portfolio composed only of the risky asset (in our case, the MSCI world index) with the one augmented by the trading bot and composed of the risky asset and the hedging overlay. If our bot is successful in identifying good hedging strategies, it should improve the overall portfolio and have a better performance than the risky asset.

**Markowitz theory** The standard approach for portfolio allocation in finance is the Markowitz model (Markowitz (1952)). It computes the portfolio with minimum variance given an expected return which is taken in our experiment to be the average return of the hedging strategies over the last year. The intuition in Markowitz (or mean-variance portfolio) theory is that an investor wants to have the lowest risk for a given return. In practice, we solve a quadratic program that finds the minimum portfolio variance under the constraint that the expected return is greater or equal to the minimum return. In our baseline, Markowitz portfolio is recomputed every 6 months to have something dynamic to cope with regime changes.

**Follow the winner** This is a simple strategy that consists in selecting the hedging strategy that was the best performer in the past year. If there is some persistence over time of the hedging strategies' performance, this simple methodology should work well. It replicates standard investors behavior that tend to select strategies that performed well in the past.

**Follow the loser** Follow the loser is the opposite of follow the winner. It assumes that there is some mean reversion in strategies' performance, meaning that strategies tend to perform equally well on long term and mean revert around their trend. Hence if a strategy did not perform well in the past, and if there is mean reversion, there is a lot of chance that this strategy will recover with its pairs.

## Results and discussion

We compare the performance of the following 5 models: DRL model based on convolutional networks with contextual states (Sentiment indicator, 6 month correlation between equity and bonds and credit main index), same DRL model without contextual states, follow the winner, follow the loser and Markowitz portfolio. The resulting graphics are displayed in figure 5 with the risky asset position alone in blue and the models in orange. Out of these 5 models, only DRL and Follow the winner are able to provide significant net performance increase thanks to an efficient hedging strategy over the 2007 to 2020 period. The DRL model is in addition able to better adapt to the Covid crisis and to have better efficiency in net return but also Sharpe and Sortino ratios over 3 and 5 years as shown in table 4. In terms of the smallest maximum drawdown, the follow the loser model is able to significantly reduce maximum drawdown but at the price of a lower return, Sharpe and Sortino ratios. Removing contextual information deteriorates model performances significantly and is illustrated by the difference in term of return, Sharpe, Sortino ratio and maximum drawdown between the DRL and the DRL no context model. Last but not least, Markowitz model is not able to adapt to the new regime change of 2015 onwards despite its good performance from 2007 to 2015. It is the worst performer over the last 3 and 5 years because of this lack of adaptation. For all models, we use the walk forward analysis as described in the corresponding section. Hence, we start training the models from 2000 to end of 2006 and use the best model on the test set in 2007. We then train the model from 2000 to end of 2007 and use the best model on the test set in 2008. In total, we do 14 training (from 2007 to 2020). This process ensures that we detect models that are unstable over time and is similar in spirit to delayed online training.

## Impact of context

In table 1, we provide a list of 32 models based on the following choices: network architecture (LSTM or CNN), adversarial training with noise in data or not, use of contextual states, and reward function (net profit and Sortino), use of day lag between observations and actions. We see that the best DRL model with the day-lag turnover constraint is the one using convolutional networks, adversarial training, contextual states and net profit reward function. These 4 parameters are meaningful for our DRL model and change model performance substantially as illustrated by the table. We also compare the same model with and without contextual state and see in table 3 that the use of contextual state improves model performance substantially. This is quite intuitive as we provide more meaningful data to the model.

## Impact of one day lag

Reminding the fact that asset managers cannot immediately change their position at the close of the financial markets, modeling the one day lag turnover to account is also significant as shown in table 2. It is not surprising that a delayed action after observation makes the learning process more challenging for the DRL agent as influence of variables tends to decrease with time. Surprisingly, this salient modeling characteristic is ignored in existing literature Liang et al. (2018); Yu et al. (2019); Wang and Zhou (2019); Liu et al. (2020); Ye et al. (2020); Li et al. (2019).

$^*$: the number of iterations is at maximum 500 provided we do not stop because of early stop detection. We do early stop if on the train set, there is no improvement over the last 50 iterations.

## Conclusion

In this paper, we address the challenging task of learning in a noisy and self adapting environment with sequential, non-stationary and non-homogeneous observations for a bot and more specifically for a trading bot. Our approach is based on deep reinforcement learning using contextual information thanks to a second sub-network. We also show that the additional constraint of a delayed action following observations has a substantial impact that should not be overlooked. We introduce the novel concept of walk forward to test the robustness of the deep RL model. This is very important for regime changing environment that cannot be evaluated with a simple train validation test procedure, neither a $k$-fold

Table 1: Model comparison based on reward function, network (CNN or LSTM units) adversarial training (noise in data) and use of contextual state

| reward | network | adversarial training | contextual states | performance with 1 day lag | performance with 0 day lag |
|---|---|---|---|---|---|
| Net_Profit | CNN | Yes | Yes | 81.8% | 123.8% |
| Net_Profit | CNN | No | Yes | 75.2% | 112.3% |
| Net_Profit | LSTM | Yes | Yes | 65.9% | 98.8% |
| Net_Profit | LSTM | No | Yes | 64.5% | 98.5% |
| Sortino | LSTM | No | Yes | 61.8% | 87.4% |
| Net_Profit | LSTM | No | No | 56.6% | 59.8% |
| Sortino | LSTM | No | No | 48.5% | 51.4% |
| Net_Profit | LSTM | Yes | No | 47.5% | 50.8% |
| Sortino | LSTM | Yes | Yes | 29.6% | 47.6% |
| Sortino | LSTM | Yes | No | 28.4% | 47.0% |
| Sortino | CNN | No | Yes | 26.5% | 45.3% |
| Sortino | CNN | Yes | Yes | 26.3% | 29.3% |
| Sortino | CNN | Yes | No | -16.7% | 16.9% |
| Net_Profit | CNN | Yes | No | -29.5% | 13.9% |
| Sortino | CNN | No | No | -45.0% | 10.6% |
| Net_Profit | CNN | No | No | -47.7% | 8.6% |

Table 2: Impact of day lag

| reward | network | adversarial training | contextual states | day lag impact |
|---|---|---|---|---|
| Net_Profit | CNN | Yes | Yes | -42.0% |
| Net_Profit | CNN | No | Yes | -37.2% |
| Net_Profit | LSTM | Yes | Yes | -32.9% |
| Net_Profit | LSTM | No | Yes | -34.0% |
| Sortino | LSTM | No | Yes | -25.6% |
| Net_Profit | LSTM | No | No | -3.2% |
| Sortino | LSTM | No | No | -2.9% |
| Net_Profit | LSTM | Yes | No | -3.3% |
| Sortino | LSTM | Yes | Yes | -18.0% |
| Sortino | LSTM | Yes | No | -18.7% |
| Sortino | CNN | No | Yes | -18.8% |
| Sortino | CNN | Yes | Yes | -3.0% |
| Sortino | CNN | Yes | No | -33.6% |
| Net_Profit | CNN | Yes | No | -43.4% |
| Sortino | CNN | No | No | -55.6% |
| Net_Profit | CNN | No | No | -56.3% |

Table 3: Impact of contextual state

| reward | network | adversarial training | contextual states impact |
|---|---|---|---|
| Net_Profit | CNN | Yes | 111.4% |
| Net_Profit | CNN | No | 122.9% |
| Net_Profit | LSTM | Yes | 18.5% |
| Net_Profit | LSTM | No | 7.9% |
| Sortino | LSTM | No | 13.3% |
| Sortino | LSTM | Yes | 1.2% |
| Sortino | CNN | No | 71.5% |
| Sortino | CNN | Yes | 43.0% |

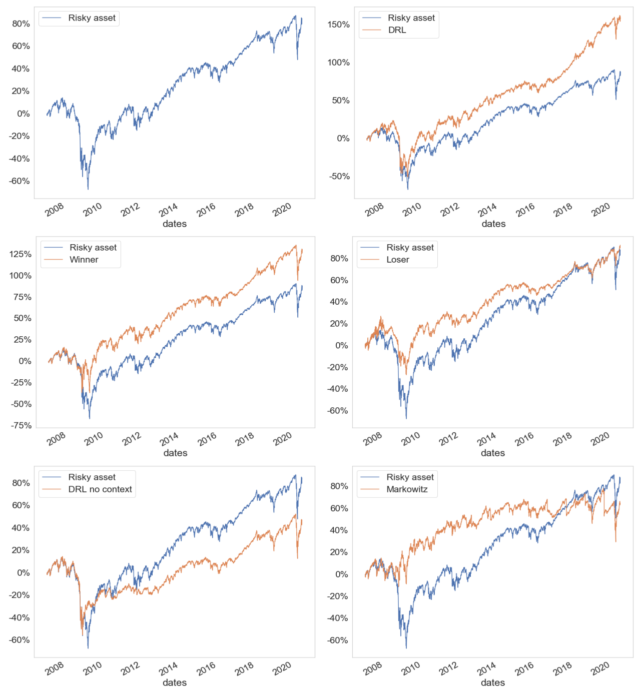


Figure 5: from left to right and from top to bottom, performance for risky asset, DRL, follow the winner, follow the loser, DRL without context and Markowitz models. Most models are not able to continuously adapt to regime changes and consequently under-perform compared to the standalone risky asset position on a long period like 2007 to 2020.

Table 4: Models comparison over 3 and 5 years

| 3 Years | return | Sortino | Sharpe | max DD |
|---|---|---|---|---|
| Risky asset | 10.27% | 0.34 | 0.38 | - 0.34 |
| DRL | **22.45%** | **1.18** | **1.17** | -0.27 |
| Winner | 13.19% | 0.66 | 0.72 | -0.35 |
| Loser | 9.30% | 0.89 | 0.89 | **-0.15** |
| DRL no context | 8.11% | 0.42 | 0.47 | -0.34 |
| Markowitz | -0.31% | -0.01 | -0.01 | -0.41 |

| 5 Years | return | Sortino | Sharpe | max DD |
|---|---|---|---|---|
| Risky asset | 9.16% | 0.54 | 0.57 | - 0.34 |
| DRL | **16.42%** | **0.98** | **0.96** | -0.27 |
| Winner | 10.84% | 0.65 | 0.68 | -0.35 |
| Loser | 7.04% | 0.78 | 0.76 | **-0.15** |
| DRL no context | 6.87% | 0.44 | 0.47 | -0.34 |
| Markowitz | -0.07% | -0.00 | -0.00 | -0.41 |

Table 5: Hyper parameters used

| hyper-parameters | value | description |
|---|---|---|
| batch size | 50 | mini-batch size during training |
| regularization coefficient | 1e-8 | $L_2$ regularization coefficient applied to network training |
| learning rate | 0.01 | Step size parameter in Adam |
| standard deviation period | 20 days | period for standard deviation in asset states |
| commission | 30 bps | commission rate |
| stride | 2,1 | stride in convolution networks |
| conv number 1 | 5,10 | number of convolutions in sub-network 1 |
| conv number 2 | 2 | number of convolutions in sub-network 2 |
| lag period 1 | [60,20,4,3,2,1,0] | lag period for asset states |
| lag period 2 | [60,20,4,3,2,1,0] | lag period for contextual states |
| noise | 0.002 | adversarial Gaussian standard deviation |
| max iterations * | 500 | maximum number of iterations |
| early stop iterations * | 50 | early stop criterion |
| random seed | 12345 | random seed |

cross validation that ignores the strong chronological feature of observations.

For our trading bots, we take not only past performances of portfolio strategies over different rolling period, but also standard deviation to provide predictive variables for regime changes. Augmented states with contextual information make a big difference in the model and help the bot learning more efficiently in a noisy environment. On experiment, contextual based approach over-performs baseline methods like Markowitz or naive follow the winner and follow the loser. Last but not least, it is quite important to fine tune the numerous hyper-parameters of the contextual based DRL model, namely the various lags (lags period for the sub network fed by portfolio strategies past returns, lags period for common contextual features referred to as the common features in the paper), standard deviation period, learning rate, etc...

Despite the efficiency of contextual based DRL models, there is room for improvement. Other information like news could be incorporated to continue increasing model performance. For large stocks, like tech stocks, sentiment information based on social media activity could also be relevant.

## References

Astrom, K. 1969. Optimal control of Markov processes with incomplete state-information II. The convexity of the loss-function. *Journal of Mathematical Analysis and Applications* 26(2): 403–406.

Bao, W.; and yang Liu, X. 2019. Multi-Agent Deep Reinforcement Learning for Liquidation Strategy Analysis.

Benhamou, E.; Saltiel, D.; Ohana, J.-J.; and Atif, J. 2020a.

Detecting and adapting to crisis pattern with context based Deep Reinforcement Learning.

Benhamou, E.; Saltiel, D.; Ungari, S.; and Mukhopadhyay, A. 2020b. Time your hedge with Deep Reinforcement Learning.

Dias, J.; Vermunt, J.; and Ramos, S. 2015. Clustering financial time series: New insights from an extended hidden Markov model. *European Journal of Operational Research* 243: 852–864.

Freitas, F.; De Souza, A.; and Almeida, A. 2009. Prediction-based portfolio optimization model using neural networks. *Neurocomputing* 72: 2155–2170.

Gu, S.; Holly, E.; Lillicrap, T.; and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*, 3389–3396.

Heaton, J. B.; Polson, N. G.; and Witte, J. H. 2017. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry* 33(1): 3–12.

Huang, C. Y. 2018. Financial Trading as a Game: A Deep Reinforcement Learning Approach.

Kahneman, D. 2011. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux.

Kingma, D.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2015. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research* 17.

Levine, S.; Pastor, P.; Krizhevsky, A.; and Quillen, D. 2016. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *The International Journal of Robotics Research* .

Li, X.; Li, Y.; Zhan, Y.; and Liu, X.-Y. 2019. Optimistic Bull or Pessimistic Bear: Adaptive Deep Reinforcement Learning for Stock Portfolio Allocation. In *ICML*.

Liang et al. 2018. Adversarial Deep Reinforcement Learning in Portfolio Management.

Lillicrap, T.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *CoRR* .

Liu, Y.; Liu, Q.; Zhao, H.; Pan, Z.; and Liu, C. 2020. Adaptive Quantitative Trading: an Imitative Deep Reinforcement Learning Approach. In *AAAI*.

Markowitz, H. 1952. Portfolio Selection. *The Journal of Finance* 7(1): 77–91.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari With Deep Reinforcement Learning. *NIPS Deep Learning Workshop* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518: 529–33.

Nan, A.; Perumal, A.; and Zaiane, O. R. 2020. Sentiment and Knowledge Based Algorithmic Trading with Deep Reinforcement Learning.

Niaki, S.; and Hoseinzade, S. 2013. Forecasting S&P 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International* 9.

Ning, B.; Lin, F. H. T.; and Jaimungal, S. 2018. Double Deep Q-Learning for Optimal Execution.

Salhi, K.; Deaconu, M.; Lejay, A.; Champagnat, N.; and Navet, N. 2015. Regime switching model for financial data: empirical risk analysis. *Physica A: Statistical Mechanics and its Applications* 461.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *ICLR* .

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR* .

Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529: 484–489.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Théate, T.; and Ernst, D. 2020. Application of deep reinforcement learning in stock trading strategies and stock forecasting.

Vinyals, O.; Babuschkin, I.; Czarnecki, W.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.; Powell, R.; Ewalds, T.; Georgiev, P.; Oh, J.; Horgan, D.; Kroiss, M.; Danihelka, I.; Huang, A.; Sifre, L.; Cai, T.; Agapiou, J.; Jaderberg, M.; and Silver, D. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575.

Wang, H.; and Zhou, X. Y. 2019. Continuous-Time Mean-Variance Portfolio Selection: A Reinforcement Learning Framework. *arXiv e-prints* .

Wang, S.; Jia, D.; and Weng, X. 2018. Deep Reinforcement Learning for Autonomous Driving. *ArXiv* abs/1811.11329.

Wu, X.; Chen, H.; Wang, J.; Troiano, L.; Loia, V.; and Fujita, H. 2020. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences* 538: 142–158.

Ye, Y.; Pei, H.; Wang, B.; Chen, P.-Y.; Zhu, Y.; Xiao, J.; and Li, B. 2020. Reinforcement-Learning based Portfolio Management with Augmented Asset Movement Prediction States. In *AAAI*.

Yu, P.; Lee, J. S.; Kulyatin, I.; Shi, Z.; and Dasgupta, S. 2019. Model-based Deep Reinforcement Learning for Financial Portfolio Optimization. *RWSDM ICML Workshop* .

Zhang, Z.; Zohren, S.; and Roberts, S. 2019. Deep Reinforcement Learning for Trading.