Deep Q-learning with PCA and Prioritized Experience Replay for Trading in the Forex Market

João Manuel Guerreiro Barra Feijão

Instituto Superior Técnico,
Universidade de Lisboa.
joaomfeijao25@gmail.com

Abstract

This paper presents an approach combining Technical Analysis and Principal Component Analysis (PCA) with a deep Q-network (DQN) using Double Q-learning and Prioritized Experience Replay (PER) to generate trading decisions capable of achieving long-term financial gains with low financial risk. The proposed approach is tested with real data from five Forex markets with different characteristics from each other, using transaction costs. Two different state and reward functions are considered in the DQN algorithm and the most robust results are produced by a combination of a state representation using features produced by PCA and a reward function measuring the profit percentage obtained by the generated trading signal in a trading period. The system uses on every test the same network topology and system's hyperparameters values. The results achieved show that this approach outperforms the Buy and Hold (BnH) strategy in four of the five tested markets. In the EUR-USD market, this system achieves a rate of return of 19.21% while the BnH strategy only achieves 3.92%. Furthermore, it is concluded that the PCA technique, Prioritized Experience Replay and Double Q-learning are essential to the performance.

I. Introduction

One of the biggest challenges in our current society is building an intelligent system that can outperform financial markets by carefully produce timely trading suggestions for financial investments. Financial market trading is an application domain with large potential for machine learning because the existence of an enormous amount of historical data suggests that machine learning can provide a competitive advantage over human inspection of these data. However, this is very difficult because markets are affected by too many highly interrelated factors, such as economic, political and investors psychological factors. Also, they have noisy, non-stationary and non-linear characteristics [1].

Efficient Market Hypothesis [2], indicates that it is impossible to "beat the market", because market efficiency causes existing prices to always incorporate and reflect all relevant information. Therefore, outperforming the overall market throughout expert market timing is impossible, with the Buy and Hold (BnH) investing strategy being the best investment strategy available. Despite this theory, it is also believed that the markets are inefficient enabling opportunities to obtain gains above the BnH strategy. This is mainly due to the markets not responding immediately to newly released information and due to the investors psychological factors affecting the markets.

In this paper a system framework combining deep learning (DL) methods and reinforcement learning (RL), termed deep Q-network (DQN), capable of maximizing financial gains by timely selecting the best trading actions while trading in Forex markets is presented. The framework will use the same network topology, system's hyperparameters values and raw financial data parameters in every market. Double Q-learning and prioritized experience replay (PER) extensions to the DQN standard approach are also used. Technical analysis will be used to extract features from the raw financial data in order to mitigate data noise and uncertainty enhancing the learning of market patterns and trends. However, instead of solely relying on feature extraction, a feature learning method will be used to learn features automatically from the raw financial data and from the features extracted by the technical analysis. This method is the PCA technique, transforming the high dimensional extracted features to a lower dimensional features that are going to be the DQN algorithm inputs. The propose of the DQN algorithm is to use DL to learn abstract and robust market patterns from the input features and use Q-learning algorithm to predict the best trading actions for the market status.

This paper is organized as follows: in Section II the theoretical background and related work is discussed. Section III presents the architecture and describes the implemented system. In Section IV the case studies and results are presented and analyzed. Section V provides the conclusions obtained by the work developed.

II. Related Work

Technical analysis forecasts the future financial price movements based on an investigation of past price movements. Involves the analysis of the market's activity, such as price and volume, to identify patterns that can be used as a basis for investment decisions. This makes the technical analysis more suitable for short-term trading, as it is the case of the system proposed in this paper. Technical indicators are a fundamental part of technical analysis as they try to predict the future price or future trend based on their value.

Applying machine learning to financial markets is about learning from data and making predictions and/or decisions. Financial data are among the best application domains for intelligent processing because data have been recorded very accurately for very long periods of time, scales and different markets, providing a very rich environment for analysis and experimentation using advanced processing techniques [3]. Machine learning algorithms are capable of processing a big amount of past financial data suggesting that it can provide a competitive advantage over human inspection of the data, detecting patterns and predicting future outcomes of the market.

i. Feature Learning

Feature Learning is a set of methods that allows a machine to be fed with raw data and to automatically discover representations that preserve as much information about the original data as possible, and at the same time to keep the representation simpler or more accessible than the original data, with low-dimensional, sparse, and independent representations.

i.1 Deep Learning (DL)

DL allows computational models composed with more than one hidden layers between input and output layers to learn representations of data with multiple levels of abstraction [4]. It discovers intricate structure in large datasets by using the backpropagation algorithm to indicate how a model should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. A neural network (NN) with more than one hidden layers is termed a deep neural network (DNN) and it is considered a DL method. However, models may not necessarily be more effective when they become more complicated because of overfitting.

i.2 Principal Component Analysis (PCA)

The input data for many machine learning problems, including RL, is often very high-dimensional. Therefore, it is in most cases not feasible to apply learning algorithms directly on this input data as the amount of samples that is required to generalize accurately increases exponentially with its amount of dimensions (curse of dimensionality). This leads to excessive computation times and overfitting.

PCA is a dimensionality reduction algorithm that can be used to significantly speed up feature learning. It focus on reducing the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the dataset. This is achieved by transforming the data using an orthogonal linear transformation into a new set of variables, the principal components, which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables. The obtained principal components are a linear combination of the original features that reflect as much as possible the original information.

The number of principal components is equal to the original data's number of dimensions but the goal of PCA is to represent the data with only some of the principal components, the ones that have the biggest original data variance, constructing a low representation of the data, with most of the original data variance retained. For example, if the original data contains 20 dimensions and the first 5 principal components can explain 95% of the original data variance, then PCA can be used to reduce the dimensionality of the data by retaining only the first 5 principal components.

ii. Deep Q-network (DQN)

DQN is a powerful method combining DL with the Q-learning algorithm. This section first briefly introduces the Q-learning algorithm and then extends to the deep Q-network approach.

ii.1 Q-learning

In RL approach, the learner and decision-maker is called the agent. The agent has to perform a task, without being told what to do. Interacts with the environment in the form of actions and in each interaction will cause the environment to change from a current state into a new state. For each of these transitions the agent will receive a reward which indicate the quality of the transition. So, the transitions and reward functions are outside of the agents control, all

it can influence is its actions. Its task is to learn a policy, for choosing actions that achieve its goals. The agent must perform a sequence of actions, observe their consequences, and learn the policy. The desired policy is one that, from any initial state, chooses actions that maximize the long-term cumulative reward.

Q-learning is a RL algorithm that learns an action-value function Q, or just Q-function, that aims to estimate how good it is for the agent to perform a given action a in a given state s. Formally, at the time step t, the value of the action-value function $Q(s,a)$ under a fixed policy π is given by the expected return starting in the given state s, taking the given action a and then following the policy π:

$$Q_\pi(s,a) = \mathbf{E}_\pi\left[G_t \mid S_t = s, A_t = a\right], \tag{1}$$

where G_t is the sequence of rewards R an agent expects to receive from a given time period t. It is given by:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2}$$

where γ is a discount factor that weighs the future reward. A reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately.

There will be some policies better than or equal to others in a way that its expected return is greater than or equal to all the the other policies. This implies that exists a policy π^*, denoted as optimal policy, that is better than or equal to all the other policies, assigning to state-action pair, the largest expected return achievable by any policy. So, the agent's learning task is to learn the optimal policy given by:

$$\pi^*(s) = \max_a Q_\pi(s,a) \tag{3}$$

In this case the action-value function is said to be the optimal action-value function, Q^*:

$$Q^*(s,a) = \max_\pi Q_\pi(s,a), \ (\forall s)(\forall a) \tag{4}$$

A fundamental property of the action-value function Q is that it satisfies an important recursive relationship for any policy π and any state s:

$$Q^*(s,a) = r(s,a,s') + \gamma \max_{a'} Q_\pi\left(s',a'\right), \tag{5}$$

where s' is the new state after taking action a in state s and r is the reward received by the agent by performing action a in state s and ending up in state s'. This property is known as the Bellman equation and is very important because it forms the basis to approximate and learn the action-value function.

Q-learning algorithm learns the Q-function by iteratively approximate at each state-action pair using the Bellman equation, and this iterative update will converge to the optimal Q-function independently of the policy being followed. The update rule that is used in every step of the algorithm is based on the temporal difference between the predicted Q-value and its current value and is given by:

$$Q_{t(s,a)} \leftarrow Q_t(s,a) + \alpha\left[r(s,a,s') + \gamma \max_{a'} Q_t\left(s',a'\right) - Q_t(s,a)\right] \tag{6}$$

where α is the learning rate and allows for the determination of how quickly the existing value will be adjusted regarding the significance of new information in relation to the existing value, with $\alpha \in [0,1]$.

ii.2 Q-networks

RL algorithms such as Q-learning can only operate in discrete state-action spaces. The state-action space of a financial market is continuous. To tackle this limitation, the Q-function is approximated using the function approximation technique that aims to generalize from examples of a function to construct an approximate of the entire function. This way, one can deal with a continuous or high-dimensional state-space by generalizing the Q-function based on the available samples. The most common function approximation technique used are NNs because they are good nonlinear function approximators, being a natural approach to consider with modeling time series which have nonlinear dependence on inputs [5]. The integration of RL and NNs is termed *Q-network* and has been used as a common approach [6].

Q-network parameterize an approximate value function $Q(s,a;\mathbb{W}_i)$ in which $\mathbb{W}_i$ are the weights of the Q-network at iteration i. The input to this network is a state representation and the output are the Q-values of the possible actions for that state. It can be trained by adjusting the weights $\mathbb{W}_i$ at iteration i to minimize the mean-squared error of the Q prediction error, the difference between the left and right side of the Bellmann equation:

$$Q(s,a;\mathbb{W}_i) = r + \gamma \max_{a'} Q(s',a';\mathbb{W}_i) \tag{7}$$

In other words, it can be formulated as the difference between the predicted $Q(s,a;\mathbb{W}_i)$ (left side) and the optimal target values y (right side), that in fact are approximate target values.

Thereby, the loss function of the Q-network is given by:

$$\mathbb{L}(\mathbb{W}_i) = \mathbf{E}_{(s,a,r,s')}\left[\left(y - Q(s,a;\mathbb{W}_i)\right)^2\right] \tag{8}$$

Recent achievements of DL has similarly accelerated progress in the combination of DNN and Q-learning, termed

deep Q-network (DQN). In 2015, a team of researchers at Google DeepMind developed a DQN system and made several contributions [7]. The remarkable thing that was shown in that work is that it showed how a single RL agent can achieve high levels of performance in many different problems without relying on different problem-specific feature sets.

III. Proposed approach

i. System Architecture

This paper presents a trading system that uses the DQN algorithm combined with recent training methodologies and the PCA technique for timely selecting the best trading actions in order to maximize the financial gains in Forex markets. The process of trading is well depicted as an online decision making problem, involving two critical steps of market condition summarization and optimal action execution.

The system was developed in Python programming language. System's architecture is presented in Figure 1 and is composed by three major layers: user layer, business logic layer and data layer.

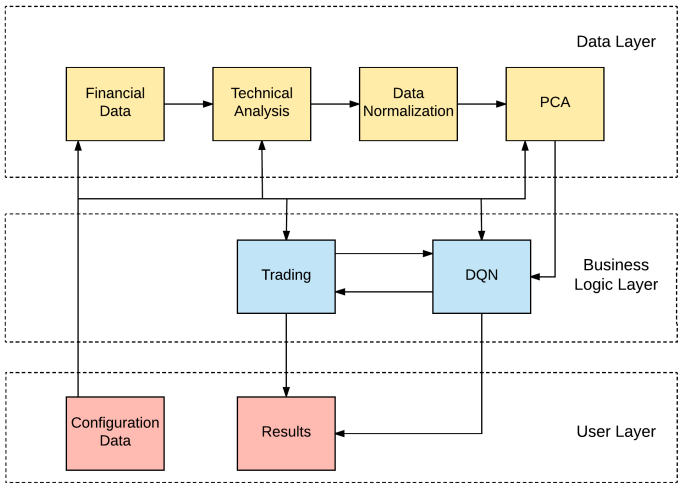

Figure 1: *System architecture.*

ii. Technical Analysis

Unlike other types of signals, such as images or speech, financial data contains high amounts of unpredictable uncertainty and noise. Therefore, reducing the uncertainties and noise in the raw financial data is an important approach to increase the robustness for financial signal mining. The quality of this data affects the system's learning performance and therefore needs to be preprocessed. Technical analysis has the capability of providing historical information in a compressed way, essential to machine learning problems because of the overfitting phenomenon, and filtering out the noise from random price fluctuations, making it much easier to understand the underlying trend. These properties may enhance the learning of market patterns and trends facilitating the system's prediction capability.

The technical analysis module receives the raw financial data and computes a set of 26 technical indicators. Then, these technical indicators together with the raw financial data (table 1) are fed into the next module, the data normalization module. The indicators are computed recurring to TA-Lib python library [8].

Tecnhical Indicators	Financial Data
SMA20,SMA50,SMA100	Hign
EMA20,EMA50,EMA100	Low
Bollinger Bands	Adj. Close
PSAR	Volume
ATR	
MACD and Signal line	
MACD Histogram	
PPO	
RSI	
ADX	
CCI	
Momentum	
Stochastic %D,%K	
Williams %R	
ROC	
OBV	
MFI	
Chaikin Oscillator	

Table 1: *List of all 30 variables outputted to the data preprocessing module.*

iii. Data normalization

The system divides the input provided by the technical analysis module into three distinct sets: training set, validation set and testing set. The training set is the set used to train the system - in-sample data. The validation and testing set are used to test the generalization capability of the system in out-of-sample data. Then, data normalization is applied to this data. The data normalizing technique applied to the data is the Min-Max normalization (equation 9),

which rescales the data values to [0,1] range.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{9}$$

iv. PCA

The PCA technique computes the principal components, the components that represent the directions of maximum variance, on the training dataset received from the normalization module. This process is known as fitting of the data. After this, the 30 principal components are ordered by the amount of variance they explain and the first principal components whose accumulated variance adds up to at least 95% are preserved. The data is then projected on the preserved principal components and this transformed low dimensional features are fed into the DQN module.

v. DQN module

The DQN module receives the preprocessed data from the PCA module, applies the DQN algorithm and outputs the trading action signal to the trading module, receiving back trading execution data and the reward signal from the trading module. The goal of the DQN agent is to interact with the trading module by selecting actions in a way that maximizes future rewards.

The DNN of the DQN system learns abstract and robust market patterns extracted from the preprocessed input financial data features (DL), forming a much better representation for the market status than just the input financial data features usually extracted in the most forecasting financial markets algorithms. At the same time, the DNN will also learn the Q-function $Q(s, a)$ to maximize the long-term accumulated profit in order to predict the best trading action for the market status learned.

To learn the Q-function, DQN agent interacts with the financial environment (trading module) in a sequence of actions, observations and rewards. At each time step the agent selects an action a_n, $n \in [0, 1, 2]$ that represent a neutral, long and short position respectively. The action is passed to the trading module where the action is executed and the reward is computed. Afterwards, DQN module receives the reward signal from the trading module and computes the next state with the information received from the trading module and with the features received from the PCA module.

v.1 Model Architecture

DQN model topology consists in receiving only the state representation as input, that is going to be addressed afterwards, and having three separate output units (neurons) for each possible action. The output units correspond to the predicted Q-values of the individual actions for the input state. In terms of network topology, a fully-connected feedforward network with three hidden layer with 200 neurons in each of the layers is used. Finally, all the neurons in the network have the same activation function, the ReLU activation function. To implement the NN, it was used a high-level NN API named Keras.

v.2 State

State representation is the available information about the market environment used by the agent to determine what happens next and it must contain enough information for the agent make a fully informed decision. Ideally should include immediate sensations and summarize past sensations compactly in a way all relevant information is retained. Two different state representations are going to be proposed:

- The first, is solely a features sample from the input features received from the PCA module. Thereby, from now on it is going to be denominated *features state*. Considering an example where the DQN algorithm receives from the PCA module a dataset with 8 features in every sample learned automatically by the PCA technique. Figure 2 presents the features state representation for the 3rd time step (s_3) that is equal to the third features sample. It can be noted that in this case the features sample has 8 dimensions and is represented as F_{tn}, with t representing the time step and n the dimension number.

Figure 2: *Example of the features state representation for a third trading time step.*

- In addition to the current market condition, the historical actions and the holding trading positions may be required to be explicitly modeled in the policy learning part. Therefore, the second state representation not only uses the features sample but also uses two components that will provide information about the holding trading positions and historic actions. Thereby, from

now on it is going to be denominated *historical state representation*. The first component is a integer scalar (A) with three discrete values in order to acknowledge the system if currently a trading position is open. If a short position is open $A_t = 0$, if long position is open $A_t = 1$ and if the agent is in a neutral position $A_t = 0.5$. The second component is a float scalar (P_t) in order to acknowledge the system how much unrealized profit percentage the currently trading position has made, in a normalized manner. Equation 10 presents how the second component is computed at time periods t before the normalization is applied.

$$P_t = \begin{cases} 0 & \text{if neutral} \\ \frac{current\ price - position\ open\ price}{position\ open\ price} & \text{if long} \\ \frac{position\ open\ price - current\ price}{position\ open\ price} & \text{if short} \end{cases} \quad (10)$$

Considering the previous example presented for features state, Figure 3 presents the historical state representation for the third time step (s_3) that consists in the two components (A_3, P_3) plus the third features sample (F_{tn}).

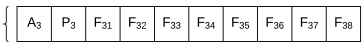

Figure 3: *Example of the historical state representation for the third trading time step.*

v.3 Reward

One of, if not the biggest and crucial challenge of the DQN algorithm applied to Forex markets is designing a reward signal that makes sure to accurately reflect the aims of the overall goal: financial gains.

The noisy characteristics of the price signal may not provide reliable supervision for the model training. An accurate reward signal should reflect price fluctuations in order to acknowledge profit possibilities in a way that prevent the agent from paying too many transaction costs that may lead to huge losses. Two reward functions are going to be proposed:

- *Period return* is the first reward function proposed. It consists on using the price difference percentage of a frequency period as reward.

$$R_{t+1} = a_t * \frac{p_{t+1} - p_t}{p_t} \quad (11)$$

Equation 11 presents how the period return reward is computed, where $a_t \in [short, neutral, long] = [-1, 0, 1]$ is the action executed and p_t is the close price at time step t,.

- *Transaction profit* is a reward function that uses the profit percentage obtained from each transaction as reward, together with an instantaneous reward component that in fact is the period return reward signal.

$$R_{t+1} = \begin{cases} a_t * \frac{p_{t+1} - p_t}{p_t} & \text{if position is not closed} \\ a_t * \frac{p_{t+1} - p_o}{p_o} & \text{if position is closed} \end{cases} \quad (12)$$

Equation 12 presents how the transaction profit is calculated. If a open position is not closed, it is computed the same way as period return reward signal. If a open position is closed, p_o is the close price at the specific time step where the position was opened.

The reward signal for both proposed reward functions is clipped to the $[0, 1]$ range as proposed by [7] that concluded by clipping the rewards, error derivatives are limited.

v.4 Learning Algorithm

The basis of the implemented algorithm is the standard DQN algorithm proposed by [7]. However, recent improvements known as double Q-learning [9] and prioritized experience replay (PER) [10] were also implemented.

As proposed in the standard DQN approach, the algorithm uses two separate NNs, an online policy network Q and a separate target network $\widetilde{Q}$. To compute the optimal target values, y, $\widetilde{Q}$ is used. Initially it contains the weights of the network enacting the policy Q, but is kept frozen every C updates and only then the target network $\widetilde{Q}$ is updated to match the policy network Q.

Double Q-learning [9] was proposed to tackle an overestimate problem in Q-learning. To solve this problem it was proposed to evaluate the greedy policy according to the online policy network, but to use the target network to estimate its value:

$$y_j = r_j + \gamma Q(s_{j+1}, argmax_{a'} Q(s_{j+1}, a'; \mathbb{W}_i)\ ; \mathbb{W}_i^-) \quad (13)$$

Experience replay is a technique in which the agent's experiences are stored at each time step, $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$, in a data set $D_t = \{e_1, ...e_t\}$. DQN applies minibatch updates, to samples of experience, because it has many advantages [7].

This system uses the experience replay technique but instead of sampling the experience transitions in a uniformly manner from the replay memory D, PER [10] samples experience transitions with high expected learning progress more frequently. The magnitude of a transition's TD error

(δ) is the criterion used to evaluate the expected learning progress:

$$\delta_j = r_j + \gamma Q(s_{j+1}, argmax_{a'} Q(s_{j+1}, a'; \mathbb{W}_i)\ ; \mathbb{W}_i^-) - Q(s_j, a_j; \mathbb{W}_i) \tag{14}$$

The priority of a transition i is given by

$$p_i = (\ |\delta_i| + \epsilon\ )^{\alpha} \tag{15}$$

where ϵ is a small positive constant that guarantees that every transition may be replayed even if its TD error is zero, and $0 \leq \alpha \leq 1$ controls how much prioritization is used.

The probability of sampling a transition i is given by

$$P(i) = \frac{p_i}{p_{total}} \tag{16}$$

where $p_i > 0$ is the priority of transition i and p_{total} is the sum of all the priorities in D. This way, during each learning step, a batch will be sampled containing samples with this probability distribution.

To implement PER a sum-tree data structure was used. This structure is very similar to a binary heap but instead of the usual heap property, the value of a parent node is the sum of its children. Leaf nodes store the transition priorities and the internal nodes are intermediate sums, with the parent node containing the sum over all priorities p_{total}. To sample a minibatch of size k, the range [0, p_{total}] is divided equally into k ranges. Then, a value is uniformly sampled from each range and the transitions that correspond to each of the sampled values are retrieved from the tree. This provides a efficient way of calculating the cumulative sum of priorities, allowing O(log N) updates and sampling.

Therefore, with these improvements the loss function of the implemented DQN algorithm is given by

$$\mathbb{L}(\mathbb{W}_i) = \mathbf{E}\left[\left(r + \gamma Q(s', argmax_{a'} Q(s', a'; \mathbb{W}_i)\ ; \mathbb{W}_i^-) - Q(s, a; \mathbb{W}_i)\right)^2\right] \tag{17}$$

where $(s, a, r, s') \sim P(D)$ denotes the distribution of the transitions according to equation 16.

The algorithm consists in at each time step t, for a given state s_t, the agent selects and executes actions a_t according to an ϵ-greedy policy based on Q. Then receives a reward r_{t+1} from the trading module and computes the new state representation s_{t+1}. The transition is stored in D with maximal priority. If gradient descent is going to be performed, a minibatch with k size is sampled, dividing p_{total} in k ranges and selecting a transition from every range. Then, for every transition j of the minibatch, the optimal target value y is computed and a gradient descent update is performed on all the minibatch. After the update, the new TD error δ is computed for every transition j and their priority is updated. Every C steps, the weights W^- of the target network $\widetilde{Q}$ are updated to the weights W of the policy network. At the end of a time step, new state is set to be the state of the next time step. This procedure is performed for every time step till the end of an episode.

v.5 Algorithm Procedure

To mitigate the disturbances of overfitting the training procedure is structured into epochs with two distinct phases: learning algorithm applied on training dataset episodes and test on the validation dataset. The algorithm will perform 100 epochs and the model chosen for the test experiment will be the model with the highest profit in validation test.

v.6 System Hyperparameter's

The hyperparameters were selected using an informal search on different datasets from the EUR-USD market, although it is conceivable that better results could be obtained by systematically tuning the hyperparameter values.

Table 2 presents the list of hyperparameters and their values. The system trading frequency is 2-hour interval. RMSProp algorithm is the iterative optimization algorithm chosen.

Hyperparameter	Value
minibatch_size	32
D	50000
C	5000
γ	0.99
learning_rate	0.00025
initial_exploration	1
final_exploration	0.1
epochs	100
ϵ	0.000001
α	0.6

Table 2: *List of hyperparameters and their values.*

vi. Trading

The trading module is responsible for emulating trading in Forex markets. It receives as input the trading signal outputted by the DQN module and simulates the execution of the action with the highest predicted Q-value from the trading signal in a Forex market. The trading signal outputted by the DQN module has 3 components that represent the estimated Q-function for neutral, long and short actions. Trading module verifies which of them has the highest value

and the action that it represents is the chosen action for the trading execution. Transaction costs are applied to simulate a real transaction according to the *Interactive Brokers* (largest electronic brokerage firm in the United States) low-cost online platform transaction costs and are given by:

$$Transaction\ Cost = 0.00002 * Traded\ Value \tag{18}$$

Figure 4 presents how the trading module of the system works.

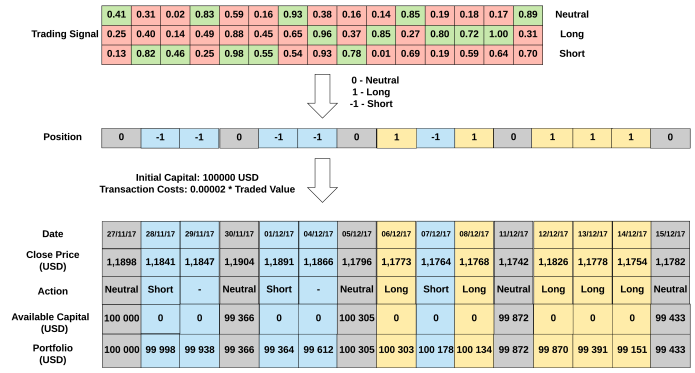

Figure 4: *Trading execution example in EUR-USD market from 27/11/17 to 15/12/17.*

IV. Results

To test the robustness of the proposed system in many Forex markets with different characteristics the system was applied to five markets. The financial data used to train and test the implemented system on different Forex markets was divided into 75% of the data to train, 15% to validation and 10% to test. The period of time of the experiments are: from 14/08/2003 to 15/11/2017 in EUR-USD, 15/05/2003 to 02/11/2017 in USD-JPY, 14/08/2003 to 02/11/2017 in AUD-USD, 15/05/2003 to 03/11/2017 in GBP-USD and 14/08/2003 to 15/11/2017 in USD-CAD.

The performance of the experiments is measured not only by the returns (ROR) obtained but also by the number of transactions, trading periods spent with capital invested in the market (Market periods), the ROR per trading period spent in the market (ROR/period), maximum drawdown (MDD) and risk-return ratio (RRR).

i. Case Study I - State and Reward Functions

The first experiment performed was doing combinations of the different proposed state and reward functions to test the quality of the system's results obtained by each combination. Features-period system uses features state and period return reward, Historical-period system uses historical state and period return reward, Features-transaction system uses features state and transaction profit reward and Historical-transaction system uses historical state and transaction profit reward.

The average results obtained by the Features-period and Historical-transaction system's combinations on the test data are presented in Table 3. The results of the BnH strategy are also presented for comparison.

	B&H	Features--Period	Historical--Transaction
EUR-USD			
Transactions	2	819.6	1395.2
ROR (%)	3.92	19.21	21.81
Market Periods	4460	2630.4	2764.2
ROR/period (%)	0.0009	0.0069	0.0076
MDD (%)	10.5	5.36	5.28
RRR	0.37	2.72	3.64
USD-JPY			
Transactions	2	566.2	442.0
ROR (%)	3.63	5.15	1.10
Market Periods	4540	1941.8	4045.3
ROR/period (%)	0.0008	0.0030	0.0001
MDD (%)	10.34	7.11	8.78
RRR	0.35	1.88	0.14
AUD-USD			
Transactions	2	951.8	1340.3
ROR (%)	4.84	14.94	12.88
Market Periods	4540	3507.1	4127.2
ROR/period (%)	0.0011	0.0044	0.0030
MDD (%)	6.05	4.79	5.90
RRR	0.8	3.56	2.77
GBP-USD			
Transactions	2	953.8	1350.8
ROR (%)	-12.83	5.59	10.15
Market Periods	4550	3507	3840.8
ROR/period (%)	-0.0028	0.0016	0.0025
MDD (%)	29.1	15.61	17.48
RRR	-0.44	0.62	0.89
USD-CAD			
Transactions	2	819.7	1100.8
ROR (%)	-0.29	6.79	5.97
Market Periods	4470	3025.1	3157.5
ROR/period (%)	-0.0001	0.0022	0.0015
MDD (%)	13.47	6.70	7.93
RRR	-0.02	1.42	1.09

Table 3: *Results of the B&H and the different state and reward functions combinations (average).*

Analyzing the results obtained, the following observations can be made:

- Features-period obtains the best results in the USD-JPY, AUD-USD and USD-CAD markets, outperforming the BnH strategy returns in all markets except GBP-USD, because a short and hold strategy would perform 12.83% returns, higher of what Features-period obtained. Obtains at least 5.15% in every market.
- Historical-transaction obtains the best results in the EUR-USD and GBP-USD markets, outperforming the BnH strategy in all markets except GBP-USD and USD-JPY. Obtains only at least 1.10% in every market.

This way, Features-period using features state and period return reward functions achieves better and more consistent overall results in maximizing financial gains while minimizing financial risk.

In Figure 5 it's presented the evolution of the returns obtained by the Buy and Hold strategy and the average returns obtained by the systems with the different state and reward functions combinations in the EUR-USD exchange market during the period of test.

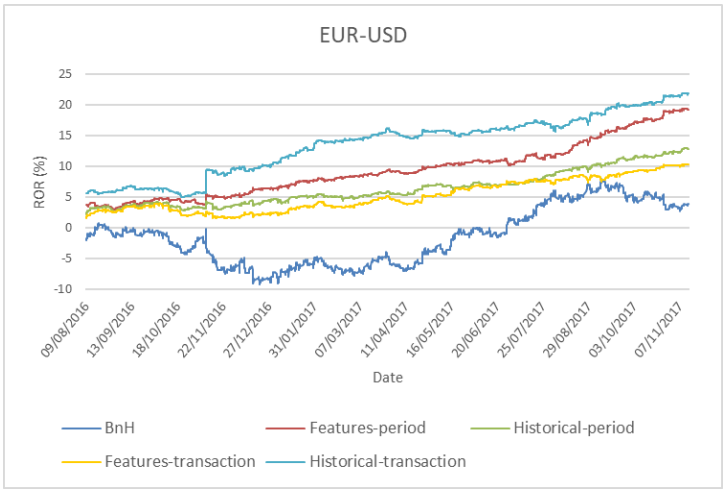

Figure 5: *Returns obtained by the systems with the different state and reward functions combinations and Buy and Hold strategy in the EUR-USD exchange market.*

ii. Case Study II

Having determined the more robust and suitable state and reward function combination to the application of this paper, in this case of study it is considered that the proposed final system uses Features-Period combination. The second case study of this paper tests and analyze the importance of PCA and PER in the performance of the system.

Table 4 presents the results of the final system without the PCA technique and the final system without PER. Each of the components importance is going to be analyzed in the next subsections.

	BnH	Final System	w/o PER
EUR-USD			
Transactions	2	819.6	461.3
ROR (%)	3.92	19.21	3.61
Market Periods	4460	2630.4	1842.4
ROR/period (%)	0.0009	0.0069	-0.0005
MDD (%)	10.5	5.36	6.28
RRR	0.37	2.72	10.67
USD-JPY			
Transactions	2	566.2	651.5
ROR (%)	3.63	5.15	1.41
Market Periods	4540	1941.8	1990.7
ROR/period (%)	0.0008	0.0030	0.0011
MDD (%)	10.34	7.11	8.40
RRR	0.35	1.88	0.69
AUD-USD			
Transactions	2	951.8	999.2
ROR (%)	4.84	14.94	9.04
Market Periods	4540	3507.1	2341.4
ROR/period (%)	0.0011	0.0044	0.0039
MDD (%)	6.05	4.79	4.95
RRR	0.8	3.56	2.38
GBP-USD			
Transactions	2	953.8	923
ROR (%)	-12.83	5.59	3.43
Market Periods	4550	3507	2499.46
ROR/period (%)	-0.0028	0.0016	0.0014
MDD (%)	29.1	15.61	13.02
RRR	-0.44	0.62	0.48
USD-CAD			
Transactions	2	6.8	846.1
ROR (%)	-0.29	6.79	4.75
Market Periods	4470	3025.1	2565
ROR/period (%)	-0.0001	0.0022	0.0019
MDD (%)	13.47	6.70	7.38
RRR	-0.02	1.42	0.98

Table 4: *Results of the final system, and without PCA and without PER (average).*

ii.1 Case Study II.a - Influence of the PCA technique

It can be concluded that the importance of the PCA technique in the final system is significant. This is attested by the

results showing that the system with PCA produces much higher returns in all markets, except in the USD-CAD where the system still produces slightly higher returns, when compared to the results produced by the system without PCA. In fact, without PCA the system would only outperform the BnH strategy in 2 markets while producing negative returns in also 3 markets.

ii.2 Case Study II.b - Influence of prioritized experience replay

It can be concluded that the importance of PER in the final system is also significant. This is attested by the results showing that the system with PER produces much higher returns in all markets when compared to the results produced by system without PER. The system without PER would also only outperform the BnH strategy in 2 markets.

Figure 6 present the evolution of the returns obtained by the BnH strategy and the average returns obtained by the final system, and without PCA, without PER and without double Q-learning in the EUR-USD market.

It can be observed that in the system without PCA is clearly outperformed by the system with PCA. The system without PER is also clearly, but not as much as the system without PCA, outperformed by the system using PER.

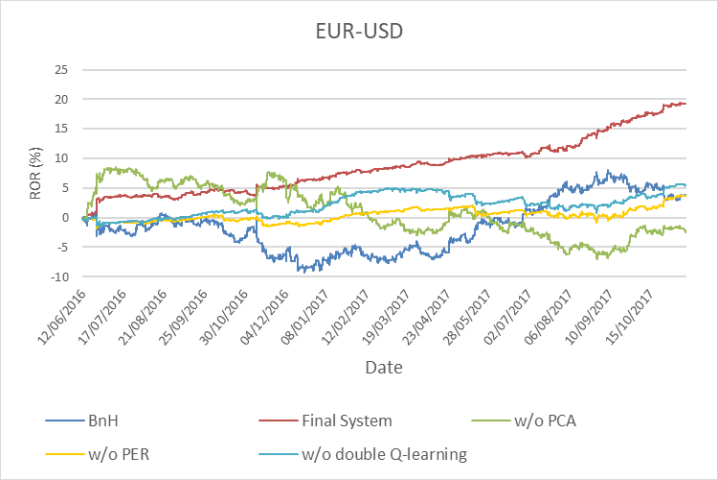

Figure 6: *Returns obtained by the BnH strategy and the average returns obtained by the final system, by the final system without PCA, by the final system without PER and by the final system without double Q-learning in the EUR-USD exchange market.*

V. Conclusions

This work proposes a DQN system approach using double Q-learning and PER that uses the same network topology, system's hyperparameters values and raw financial data parameters in every Forex market with the goal of maximizing financial returns while minimizing financial risk by timely selecting the best trading actions. The approach also uses technical analysis for feature extraction and the PCA technique for dimensionality reduction. This approach, with a features state and period return reward functions in the DQN algorithm, shows robust results over five different Forex markets. Also, it was concluded that PCA technique and PER are essential to the good performance of the system.

References

[1] Y. S. Abu-Mostafa and A. F. Atiya. Introduction to financial forecasting. *Applied Intelligence*, 6(3):205–213, July 1996.

[2] B. G. Malkiel. *Efficient Market Hypothesis. The world of Economics*. Palgrave Macmillan UK, 1991.

[3] M. Magdon-Ismil, A. Nicholson, and Y. S. Abdu-Mostafa. Financial markets: Very noisy information processing. *IEEE*, 86(11):765–769, Nov 1998.

[4] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, pages 426–444, 2015.

[5] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, page 359, 1989.

[6] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, pages 85–117, 2015.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Human-level control through deep reinforcement learning. *Nature*, pages 529–533, Feb 2015.

[8] M. Fortier. TA-Lib : Technical Analysis Library. `http://www.ta-lib.org/`, 2007. [Online; accessed 20-April-2017].

[9] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *CoRR*, 2016.

[10] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *ICLR conference*, 2016.