Reinforcement Learning Applied to Forex Trading

João Maria B. Carapuço
jmcarapuco@gmail.com

Instituto Superior Técnico, Lisboa, Portugal

June 2017

Abstract

This thesis describes a system that automatically trades in the foreign exchange market to profit from short-term price fluctuations. This system is a neural network with three hidden layers of 20 ReLU neurons each and an output layer of 3 linear neurons, trained to work under the Q-learning algorithm. Novel state and reward signals are introduced and a customizable simulated market environment framework was developed which improves training quality and testing accuracy. In the EUR/USD market from 2010 to 2017 the system yielded, over 10 tests with varying initial conditions, an average total profit of 114.0±19.6%, an yearly average of 16.3±2.8%.

Keywords: Machine Learning, Neural networks, Reinforcement Learning, Financial Trading, Foreign Exchange

1. Introduction

Reinforcement learning (RL), a sub-field of machine learning, has of late come into a sort of Renaissance that has made it very much cutting-edge for a variety of control problems. Some high-profile successes ushered in this new era of reinforcement learning. First, in 2013 a London-based artificial intelligence company (AI) called Deepmind, stunned the AI community with a computer program based on the RL paradigm that had taught itself to play 7 different Atari video-games, 3 of them at human-expert level, using simply pixel positions and game scores as input and without any changes of architecture or learning algorithm between games [1]. Deepmind was bought by Google, and by 2015 the system was achieving a performance comparable to professional human game testers in over 20 different Atari games [2]. Then, that same company achieved wide mainstream exposure when its Go-playing program AlphaGo, which uses a somewhat similar approach to the Atari playing system, beat the best Go player in the world in an event that reached peaks of 60 million viewers.

This was made possible by the use of techniques from another field of machine learning, neural networks. Neural networks consist of interconnected artificial neurons inspired by the biological brain, which provide computing power. This combination has been used before, but often proved unreliable, especially for more complex neural networks. Advances in the neural networks field such as the ReLU neuron and gradient descent algorithms with adaptive learning rate, along with contributions from Mnih et al. [1] aimed specifically at this mixed approach, have made it much more effective and allowed more powerful neural networks to work under the reinforcement learning algorithm Q-learning in what is knows as a Q-network. From the point of view of reinforcement learning, neural networks provide much needed computational power to find patterns for decision making that lead to greater reward. From the point of view of neural networks, reinforcement learning is useful because it automatically generates great amounts of labelled data, even if the data is more weakly labeled than having humans label everything, which is usually a limiting factor for neural networks [3].

We aim to adapt these new developments to financial speculation in the foreign exchange market: profiting from a currency's market value fluctuations. The challenges speculation trading present are completely different from those posed by environments such as Atari games or Go. Unlike these two, financial markets are not deterministic and the data on which the system bases its decisions is non-stationary and very noisy. This makes it more difficult to learn policies that consistently lead to positive outcomes. Taking into account the challenges this problem presents, we have the following objectives for our trading system:

- Create a Q-network that is able to stably, without diverging, learn and thus improve its financial performance on the dataset it is being trained on;
- Show that the Q-network's learning has potential to generalize to unseen data;
- Harness generalization capabilities to generate profitable training decisions on a realistic simulation of live trading.

2. Background

The problem in reinforcement learning is, to put it succinctly, that of learning from interaction how to achieve a certain goal. We frame this problem by identifying within it two distinct elements and detailing their interactions, as depicted in Figure 1. The elements are the learner/decision-maker which we call the agent, and that with which the agent interacts, known as the environment.

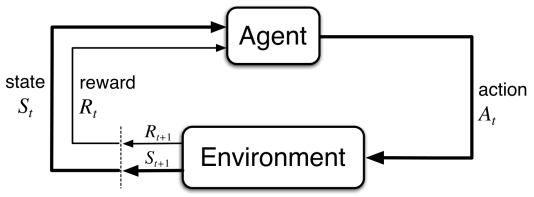

Figure 1: The RL learning framework. [4]

Considering discrete steps $t = 0, 1, 2, 3, ...$, at each t the agent receives some representation of the environment's state, $S_t \in \mathbb{S}$, where $\mathbb{S}$ is the set of possible states, and uses that information to select an action $A_t \in \mathbb{A}(S_t)$, where $\mathbb{A}(S_t)$ is the set of actions available in state S_t. The agent chooses an action according to its policy π_t, where $\pi_t(s)$ represents the action chosen if $S_t = s$ for a deterministic policy. On the next time step $t+1$, the agent receives its reward R_{t+1} as a consequence of action A_t, and information about the new state S_{t+1}. The goal is find a policy π_t that maximizes the sequence of rewards an agent will receive after step t:

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

where γ is a discount rate that allows for a prioritization of immediate versus future rewards. To relate this discounted sum G_t with a policy π, we use the action-value function q_π, which tells us how good we expect it is to perform a certain action a in a given state s and following policy π thereafter:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a].$$

The algorithm used in this work, Q-learning, works by trying to estimate the action-value function $q_*(s,a)$ associated with the optimal policy, the policy which maximizes expected G_t for any state. Q-learning takes advantage of an identity known as the Bellman optimality equation:

$$q_*(s,a) \doteq \mathbb{E}[R_t + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a],$$

by turning it into an update rule that can build an approximation of $q_*(s,a)$ iteratively:

$$\begin{aligned} q_{k+1}(s,a) &= (1-\alpha) q_k(s,a) \\ &\quad + \alpha[r + \gamma \max_{a'} q_k(s', a')], \end{aligned} \tag{1}$$

where r is the reward obtained in the transition from s to s' with action a.

The classic implementation is a table with an entry for each state-action pair $Q(s,a) \approx q_*(s,a)$ updated through Equation 1, and has been shown to converge to q_* with probability 1 under a few simple assumptions [5].

The tabular implementation is insufficient for complex problems such as financial trading. Firstly the state space is too large to be represented through a table and secondly since the action-value function is estimated separately for each sequence, there is no generalization. While trading we will never see the same exact state twice, so power of generalization is essential.

Neural networks acting as action value function approximators, Q-networks, are known for their capacity to process both linear and nonlinear relationships and for being the best available method for generalization power [3]. The Q-network computes a function $Q(s,a;\mathbb{W}) \approx q_*(s,a)$, where $\mathbb{W}$ is the set of parameters of the network. The approximation is improved when the Q-network acquires knowledge in the form of observed transitions $e = [s, a, r, s']$ and uses this knowledge to learn by adjusting its set of parameters $\mathbb{W}$ to minimize a sequence of cost functions, as described in the next section.

The concepts above can be applied to financial trading in the foreign exchange market by devising suitable reinforcement learning signals. The action and reward signals are somewhat straightforward. Reward should be either the financial profit directly or a quantity correlated with financial profit, so that the estimated action-values steer the system to profitable actions. The action signal should mimic the actions available to a foreign exchange trader, namely, opening and closing positions.

The state signal, is more elusive. To make consistent financial gains the system must base its decisions on information with potential for price prediction. There is no academic consensus about such information, but technical analysis, which focuses on finding patterns in historical market data - prices and trade volume - has a sizable amount of literature [6] supporting its effectiveness, and appears to be the most widely used and supported method for price prediction.

We use market data in the form of ticks, the smallest granularity available, which allows for a more accurate portrayal of the market state and to maximize the amount of data available to our Q-network. A tick T_i is put out whenever the market updates prices:

$$T_i = \begin{bmatrix} B_i & A_i & Vb_i & Va_i \end{bmatrix},$$

where B_i, A_i are the bid price and ask price at the time T_i was put out and Vb_i, Va_i were the volume of units traded at those respective prices.

3. Implementation

The system, a Q-network, interacts with a simulated market environment designed to train and test it in a manner consistent with trading in the real foreign exchange market. In this section we describe how this simulated market environment coordinates the flow of information that reaches the system, the implementation of the three RL signals communicated between environment and system, the Q-network's topology and learning mechanisms and hyper-parameter selection.

3.1. Market environment

The market simulation follows prices from a tick dataset $\mathbb{T} = \{T_0, .., T_K\}$, but the system is only allowed to make a decision every $time\_skip$ ticks otherwise trading frequency would be too high. We selected and optimized the system for $time\_skip = 5000$, which on average is somewhat less than two hours for 2013 EUR/USD tick data. This fits with our aim of a short-term speculator without being so high frequency as to be overly affected by practical concerns such as latency and lack of broker liquidity.

At each step[1] t the market environment is at the price in tick $T_{i=t\cdot time\_skip+b}$, where b is the chosen starting point for that pass over the data. A state S_t is sent to the system and a response in the form of an action signal A_t is received. The market environment goes to the price in T_{i+time_skip} and drafts a new state S_{t+1} and a scalar reward R_{t+1} for the action A_t, which are both sent to the system.

In training passes the actions selected by the system have an exploratory component, the system has a probability ϵ to select a random action rather than the one with highest estimated action value, and observed experiences are stored and used to update the network weights. In test passes the action with highest estimated action value is always chosen, no updates are performed to the Q-network and profit obtained over the pass is recorded as a measure of performance.

To train the network we repeatedly perform a training pass through a training dataset and two tests, which makes up an epoch. A test with the training dataset assesses learning progress and a test with a validation dataset, comprising of a period of time immediately following the training dataset, assesses ability to generalize to out-of-sample data. We rely on validation performance to tell us how many epochs to train for via early stopping. Each time the performance assessment on the validation reaches a new high the current model is saved as our candidate to final model and a counter is started. This counter is reset when a new candidate appears, but if it reaches a predefined value the training process interrupted and the current candidate is our final model. This is an inexpensive way to avoid strong overfitting even if the other hyper-parameters would yield to it [7]. The first few epochs are ignored in the early stop procedure to allow the Q-network to leave vicinity of weight hyperspace where it was randomly initialized.

Having a flexible starting point b is an important addition to our market environment. Changing b between each pass results in different paths through the dataset. We implemented this dynamic, defined with a nr_paths parameter, by taking advantage of the surplus of data generated by using tick data but only trading every $time\_skip$ ticks:

$$b \in \{\, x \cdot \frac{time\_skip}{nr\_paths} \mid x \in \mathbb{N}_0,\, x < nr\_paths \,\}.$$

We have found $nr\_paths = \frac{time\_skip}{500}$, meaning a distance of 500 ticks between paths, is a balanced value for both testing and training passes, and use it throughout this work. With this mechanism the network experiences greater variety of data when performing training passes, increasing learning quality. Also, while standard testing approaches deliver a path-dependent distribution of gains thus increasing the chances of a "lucky" trading strategy, by making each test an average over a number of test passes starting from all different initial ticks T_b we strongly mitigate this issue. The standard deviation of performance between these test passes also provides a measure of uncertainty of the assessment.

3.2. State Signal

The main component of the state signal S_t are features extracted from market data. With the simulated market at tick T_i we divide the tick data preceding it into windows described by the parameter array:

$$TW = \begin{bmatrix} TW_0 = 0 & TW_1 & TW_2 & \dots & TW_N \end{bmatrix}$$

so that the n^{th} window, where $n \in [1, 2, ..., N]$, includes the ticks T_k with $k \in [i - TW_{n-1}, i - TW_n]$. From each window features F_i^n are extracted:

$$\left|\begin{array}{cc}
\underset{k\in[i-TW_{n-1},i-TW_n]}{\mu} B_k - B_i & \underset{k\in[i-TW_{n-1},i-TW_n]}{\mu} S_k \\
\underset{k\in[i-TW_{n-1},i-TW_n]}{\mu} Vb_k & \underset{k\in[i-TW_{n-1},i-TW_n]}{\mu} Va_k \\
\max_{k\in[i-TW_{n-1},i-TW_n]} B_k - B_i & \max_{k\in[i-TW_{n-1},i-TW_n]} S_k \\
\max_{k\in[i-TW_{n-1},i-TW_n]} Vb_k & \max_{k\in[i-TW_{n-1},i-TW_n]} Va_k \\
\underset{k\in[i-TW_{n-1},i-TW_n]}{\mathrm{std}} B_k & \underset{k\in[i-TW_{n-1},i-TW_n]}{\mathrm{std}} S_k \\
\underset{k\in[i-TW_{n-1},i-TW_n]}{\mathrm{std}} Vb_k & \underset{k\in[i-TW_{n-1},i-TW_n]}{\mathrm{std}} Va_k \\
\multicolumn{2}{c}{\min_{k\in[i-TW_{n-1},i-TW_n]} B_k - B_i}
\end{array}\right|$$

where μ, $\max$, std and $\min$ are the mean, maximum, standard deviation and minimum values in the interval. To remove outliers from these features we apply a simple percentile-based filter. The q^{th} percentile of feature x_j, $q^{th}(x_j)$, is computed as the value $\frac{q}{100}$ of the way from the minimum to the maximum of a sorted copy of

[1] Steps are counted from the system's perspective, which only happen every $time\_skip$ ticks.

an array containing all entries of the feature for a dataset. The filter rule used is:

$$x_{j,i} = \begin{cases} (1-q)^{th}(x_j), & \text{if } x_{j,i} < (1-q)^{th}(x_j) \\ q^{th}(x_j), & \text{if } x_{j,i} > q^{th}(x_j) \\ x_{j,i}, & \text{otherwise} \end{cases}$$

where $x_{j,i}$ is the j^{th} element of $F_i = F_i^1 \cup F_i^1 \cup \cdots \cup F_i^N$. With outliers removed, each feature is normalized to the range $[-1, 1]$ via min-max normalization.

The state signal includes more information besides the feature array. The market may have higher volume and volatility at certain hours of the day due to the working hours of the major trading hubs, thus we want to relay that information to the system to aid in its interpretation of the market. We use two input variables to encode the hour:

$$time_i^1 = \sin(2\pi \frac{seconds_i}{86400})$$
$$time_i^2 = \cos(2\pi \frac{seconds_i}{86400})$$

where $seconds_i$ is the time when T_i was put out, converted to seconds. This encoding method effectively conveys the cyclical nature of hours to the neural network. We also relay to the system if it currently has a position open and the value of that position using an integer scalar h_t:

$$h_t = \begin{cases} 1, & \text{if long position open} \\ 0, & \text{if no position open} \\ -1, & \text{if short position open} \end{cases}$$

and a float scalar v_t with the unrealized profit of the currently open position. v_t is normalized to the range $[-1, 1]$ through min-max normalization.

Finally a float scalar c_t is included which tells the system the current size of the account compared its initial size. c_t is normalized and clipped to the range $[-1, 1]$, where the maximum and minimum extremes represent, respectively, a $safe$ and $failure$ levels set by the user. Should the current account size reach the $failure$ threshold, the system flags it as a terminal state. The terminal state condition was added for its anchoring effect, by providing the system a state with an exact known action value, and to further punish strings of consecutive bad decisions, in an effort to speed up the training process. Empirically we verified that a failure threshold closer to 1, meaning more instances of terminal states, does increase learning speed, although if it is set too close to 1 leaning quality is disrupted.

3.3. Action Signal

We impose that the agent can only invest one unit of constant size $position\_size$ of the chosen asset at a time. Therefore the action signal can be simplified to $\mathbb{A}(s) = [a_0, a_1, a_2]\ \forall s \in \mathbb{S}$, where $[a_0, a_1, a_2]$ are interpreted by the environment as described in Table 1.

Signal	Position	Action
	Long	Hold
a_0	None	Nothing
	Short	Hold
	Long	Hold
a_1	Idle	Open Long
	Short	Close Short
	Long	Close Long
a_2	Idle	Open Short
	Short	Hold

Table 1: Interpretation of each value of the scalar action signal by the market environment.

3.4. Reward Signal

Actions to close positions are rewarded by the resulting profit. Actions to open positions or hold positions open, are rewarded with the variation of unrealized profit: the difference between unrealized profit in the state S_t in which they were taken and the unrealized profit in the state S_{t+1} to which they lead.

The most common methods to adjust profit for risk are the Sharpe ratio, which divides the profit by the standard deviation of unrealized profit, and its modification the Sortino ratio, which divides profits by the downside deviation of unrealized profit. Both aim to penalize large variations of unrealized profit, which are interpreted as risk. Our reward system is in essence very similar, especially to the Sortino ratio since positive volatility is not punished but rewarded. The difference is that rather than introduce the risk punishment/reward at the end of the transaction by adjusting the profit reward, it is spread out over its life cycle with the variation at each step, which has the empirically observed benefit of speeding up the learning process, possibly by alleviating the credit assignment problem.

The reward signal is normalized to the range $[-1, 1]$ with min-max normalization for gradient learning stability. In the case of the variation of unrealized profit, it is first subjected to a percentile filter to clip values above a percentile $return\_percentile$ of all possible unrealized profit variations in a given dataset. This prevents weekend-gap variations or other outliers from skewing the distribution and allows for control over how much importance the system gives to return rewards compared to profit rewards.

3.5. Q-network

The core of the system is a fully connected neural network, depicted schematically in Figure 2. This network is tasked with computing $Q(s; \mathbb{W}_k)$, where $\mathbb{W}_k$ is the set of weights and biases of the network at iteration k, an approximation of the action value function for the market environment.

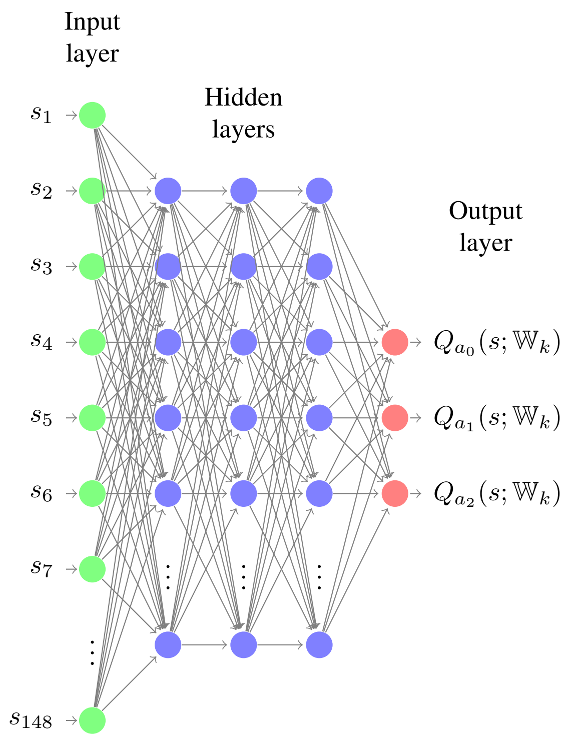

Figure 2: Schematic of the trading system's Q-network. s_n is the n^{th} element of the vector representing the state s. Hidden and output neurons have ReLU and Linear activations respectively.

The input layer has a number of neurons defined by the elements in our state signal, which per the hyper-parameter TW chosen in subsection 3.6 results in 148 input neurons. This input layer is followed by three hidden layers with 20 ReLU neurons each.

The ReLu activation function was chosen due to its documented superiority for training multi-layer neural networks [8] over other widely used activations. Future developments in the field of neural networks may bring more sophisticated methods, but currently the number of hidden neurons and layers has to be determined empirically ([7] for practical recommendations). We did so using the datasets used for all hyper-parameter selection (see subsection 3.6). This process was performed in tandem with the choice of parameter TW to make sure there was a balance between power of the network and number of input variables. We found this balance hard to achieve and very precarious, with the Q-network easily slipping into overfitting, not surprising given the noisy nature of financial data.

Tests with L1 regularization, L2 regularization and dropout regularization to prevent overfitting were not successful and no such methods were included in the final architecture. We found significant performance gain from the three hidden layer topology versus a two hidden layer or a single hidden layer with the same number of total neurons, although adding a fourth layer made training difficult and resulted in decayed performance.

The output layer has three neurons with linear activations to represent action values, which may take any real value. With this topology we have chosen to approximate the optimal action value function in its vectorial form:

$$Q(s; \mathbb{W}_k) \approx (q_*(s, a_0), q_*(s, a_1), q_*(s, a_2)),$$

rather than the arguably more intuitive:

$$Q(s, a_n; \mathbb{W}_k) \approx q_*(s, a_n), \tag{2}$$

as this second option would require a forward propagation to obtain the Q-value of each possible action, while this approach proposed by Mnih et al. [1], requires only a single forward propagation, saving computational resources.

The above-described Q-network is initialized with a random set of weights where each weight is drawn from an independent normal distribution with range $[-1, 1]$. As the network interacts with the market environment in a training pass, it collects and stores observed transitions in a sliding window buffer of size N. Every $update\_q$ steps a set of observed transitions, $\mathbb{S}_k = \{e_1, ..., e_B\}$ where $e_p = [s^p, a^p, r^p, s'^p]$, is randomly drawn from the buffer, a mechanism known as experience replay [9].

The mean squared error between desired and actual output of the Q-network is computed for that set:

$$\begin{aligned} E(\mathbb{W}_k) &= \sum_{p=1}^{B} \frac{E_p(\mathbb{W}_k)}{B} \\ &= \sum_{p=1}^{B} \frac{(y_p - Q(s^p, a^p; \mathbb{W}_k))^2}{B}, \end{aligned}$$

where y_p is the target output. Adapting Q-learning, Equation 1, directly would yield $y_p = r^p + \max_a Q_a(s'^p, \mathbb{W}_k)$, however this implementation often causes learning issues with Q-networks. As proposed by Mnih et al. [1] we introduce an auxiliary Q-Network, $Q(s, \mathbb{W}^-)$, topologically identical to the original Q-Network, whose weights $\mathbb{W}^-$ are static and periodically copied from the original set $\mathbb{W}_k$ every $update\_q^-$ updates. This auxiliary Q-network is used to generate the targets for updates, $y_p = r^p + \max_a Q_a(s'^p, \mathbb{W}^-)$, to prevent targets from shifting in a correlated manner with the Q-value estimations, which can make learning more difficult and spiral out of control through feedback loops. With a further alteration known as double Q-learning implemented as suggested by van Hasselt et al. [10], $y_p = r^p + Q_a(s'^p, \mathbb{W}^-)$ with $a = \arg\max_n Q_n(s'^p, \mathbb{W}_k)$, the action choice is decoupled from the target Q-value generation, which is known to otherwise introduce a bias in the action value estimation resulting in poorer policies.

Backpropagation [11] provides an efficient way of finding $\nabla_{\mathbb{W}_k} E(\mathbb{W}_k)$ to be used with the gradient descent implementation known as RMSProp (Root Mean

Square Propagation) [12] in improving $\mathbb{W}_k$, an implementation that has been shown to work very well in a variety of benchmarks and practical applications [13, 14] and has been successfully utilized for Q-Networks [2]. For a given parameter $\theta_{i,k} \in \mathbb{W}_k$, the learning rate is adjusted with a exponentially decaying average of squared previous gradients v_k:

$$v_k = 0.9v_{k-1} + 0.1(\frac{\sum_{p=1}^{B} \nabla_{\theta_{i,k}} E_p(\mathbb{W}_k)}{B})^2$$

$$\theta_{i,k+1} = \theta_{i,k} - \frac{\alpha}{v_k} \frac{\sum_{p=1}^{B} \nabla_{\theta_{i,k}} E_p(\mathbb{W}_k)}{B}$$

where α is the general base learning rate.

3.6. Hyper-parameters

To assess each configuration of hyper-parameters it is necessary to fully train the network and test its performance, a time consuming process. Also, there is a somewhat large number hyper-parameters to tune and they are related to each other in complex ways: changing one of them means a number of others may no longer be optimal. This made it unfeasible to perform a systematic grid search for the optimal set of hyper-parameters with the available resources.

Instead, the hyper-parameters were selected by performing an informal search using three EUR/USD datasets with 12 months of training and 6 months validation, with the goal of obtaining the most stable learning curve with the highest generalization capability as measured by peak performance on the validation dataset. The hyper-parameters thus selected are detailed in Table 2, some of which merit some further discussion.

Hyperparameter	**Value**
q	99
$time\_skip$	5000
nr_paths	$\frac{time\_skip}{500}$
$return\_percentile$	90
$failure$ / $safe$	0.8 / 2.0
$update\_q$	8
$update\_q^-$	5000
B	60
N	60000
γ	0.99
α	0.001
ϵ_0 / ϵ_f	1 / 0.3
$init\_acc\_size$	10000
$position\_size$	10000

Table 2: Hyper-parameters chosen for the trading system.

The purpose of having distinct account size and position size parameters in the system is to allow for future implementation of a variable position size scheme to optimize profitability. Tests performed in section 4 do not include such optimization, thus we set $init\_acc\_size = position\_size$ so that profits can be easily interpreted in relation to the initial available capital, which was was given the placeholder value of 10,000. The RL discount rate parameter γ is close to 1, meaning future rewards are weighed heavily. This reflects the fact that a successful transaction requires future-oriented decision making. Parameter ϵ for action selection is made to decay over the training process from a initial value ϵ_0 and final value ϵ_f so that the system takes advantage of acquired knowledge proportionally to the quality of that knowledge.

As for the parameter array TW, whose elements define the tick windows from which to extract features, the following values were selected:

$$TW = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 60 \end{bmatrix} \cdot \frac{time\_skip}{2}$$

thus defined to allow a seamless transition to other $time\_skip$ values. Although, TW should ideally be re-optimized for different trading frequencies.

4. Testing

While performance on the validation dataset is a good measure of generalization power, it has a positive bias over performance on actual live trading. A test dataset is introduced to deliver an unbiased estimate of live trading performance by testing the model chosen through the training procedure on it.

The working premise is that a model that has learned to perform well on the training dataset and subsequently also performs well out-of-sample on a validation dataset, is more likely to then be profitable on the test dataset. However, markets are known to be non-stationary [15] and if market dynamics change too much from the training/validation period to the testing period this assumption will not hold. Therefore we perform the tests using a rolling window as depicted in Figure 3.

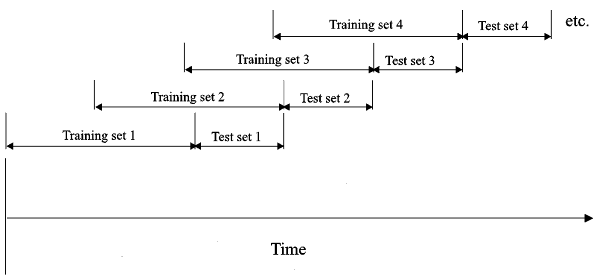

Figure 3: Rolling window approach to testing [15]. Training set includes both training and validation datasets.

Theoretically, smaller steps of the rolling window, meaning smaller test datasets, better tackle non-stationarity. However, this does entail a greater number of tests. Considering the running time of each test with our resources, we decided on a test dataset size of 4 months.

As for training and validation datasets, if they are smaller the data the network learns (training dataset) and the filter through which the model is selected (validation dataset) should be more similar to the test dataset and thus suffer less from non-stationarity. But since market data is remarkably noisy, if the training dataset is too small it will simply learn noise and lose generalization power, and if the validation dataset is too small, the overall performance on the dataset is more easily influenced by noisy unpredictable events and will tend to select models that randomly perform well on those events rather than those that truly generalize well. A training and validation dataset size of 24 and 6 months was chosen through informal testing with EUR/USD data from the year 2013.

Below we describe the results of testing the system on post-2008 crisis EUR/USD pair data: from 2010 to 2017. These tests are meant to assess the hypothetical validity of the trading system by confirming a positive expectation of gains, rather than attempt to accurately project its full revenue potential. Thus, execution is not optimized: position sizing was set a priori and kept constant and no stop-loss or take-profit orders were used. While bid-ask spread commission is included, the volume commission is not as the rates depend strongly on a specific trader's resources.

4.1. Results

We refer to each test by year followed by quadrimester. In Figure 4 a sample of the learning curves responsible for the selection of the final model for each of the tests is displayed.

For the three 2010 test datasets we curtailed the training and validation dataset size to 12/1, 12/1 and 18/3 months respectively, to avoid including data from the 2008 crisis. For the 2011 p.3 test, the standard training/validation dataset size resulted in learning curves from which no candidate could be selected as validation performance only worsened with training. Halving the validation to 3 months solved this issue, suggesting a portion of the removed 3 months was incompatible with the training dataset. The final exception was the 2016 p.2 test. There was a marked increase in tick density in the last two quadrimesters of 2016 from an average of 6,738,296 ± 2,339,367 in the remaining quadrimesters to 15,969,628 and 19.281.366. We could not ascertain the reason for this increase, most likely an internal change in the broker's method of producing tick data. We halved the validation dataset size for the 2016 p.2 test to include, proportionally, more data with the altered tick density so it would select a candidate that performs well in these new conditions.

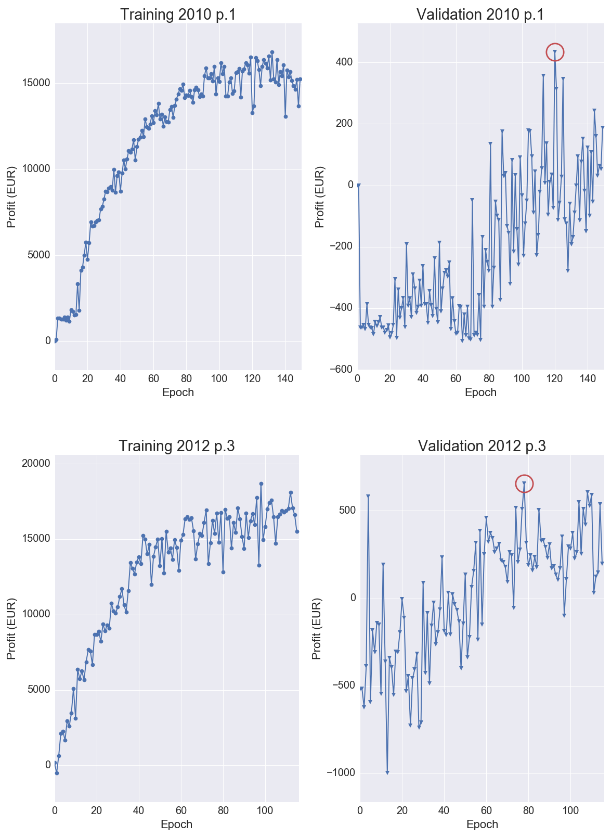

Figure 4: Sample of validation learning curves.

Table 3 details the test results in both absolute profit and profit relative to the initial account size. Note that results shown are an average of 10 test passes from different initial points following the nr_paths methodology, and their uncertainty is the standard deviation over those passes. The system is profitable in all but three tests, 2012 p.3, 2016 p.2 and 2016 p.3, although in two other cases, 2011 p.2 and 2012 p.1, it just about breaks even. This means it generated significant profit in roughly 75% of the tests, and significant losses in only 14%, a good indicator of its validity.

The simple and compounded total test results are concisely described in Table 4. We chose to use a fixed position size during each test and compound between tests, ie. change the fixed position size at the onset of a test by including the total profit of the previous tests in the account.

In Table 5 we look at the trades individually for a better insight into the behaviour of the system. It is apparent that the system generally relies on a large number of small trades, roughly 550 trades per year with 0.2% profit/loss per trade, which are profitable 53.5% of the time. There is a very slight tendency to favor the short position, especially in trades that end up being profitable. This can be explained the fact that the Euro has

Test Name	**Test Profit**	
	Abs. (EUR)	Rel. (%)
2010 p.1	605±312	6.1±3.1
2010 p.2	159±368	1.6±3.7
2010 p.3	322±377	3.2±3.8
2011 p.1	1011±75	10.1±0.8
2011 p.2	36±228	0.4±2.3
2011 p.3	320±343	3.2±3.4
2012 p.1	70±87	0.7±0.9
2012 p.2	323±218	3.2±2.2
2012 p.3	-327±78	-3.3±0.8
2013 p.1	388±231	3.9±2.3
2013 p.2	342±173	3.4±1.7
2013 p.3	434±69	4.3±0.7
2014 p.1	249±70	2.5±0.7
2014 p.2	296±141	3.0±1.4
2014 p.3	841±51	8.4±0.5
2015 p.1	767±36	7.7±0.4
2015 p.2	131±112	1.3±1.1
2015 p.3	328±227	3.3±2.3
2016 p.1	962±131	9.6±1.3
2016 p.2	-264±311	-2.6±3.1
2016 p.3	-212±222	-2.1±2.2

Table 3: Overview of the test results.

Compounding frequency	Total Test Profit (%)	Yearly Avg. Test Profit (%)
None	67.8±9.8	9.7±1.4
Yearly	89.8±16.9	12.8±2.4
Triannual	92.5±18.4	13.2±2.6

Table 4: Simple and compounded total test profit.

on average been losing value relative to the Dollar since 2010. The duration that each position stays open is similar in both profitable and unprofitable trades, showing that the system has no problem cutting its losses and does not overly wait for rebounds.

4.2. Result Analysis

It was observed that validation profit is not a good predictor of test profit, with a Pearson correlation coefficient of -0.28 at p-value significance of 0.2 indicating no correlation between annualized validation and test profits. It is clear that the training process creates a positive expectation for profits in a test setting, but results are too dataset dependent to create an expectation on the magnitude of those profits.

	Profitable trades	Unprofitable trades
Nr.	21038	18249
Avg. profit (%)	0.215±0.337	-0.211±29.9
Avg. Duration	33679±96107	36066±77264
% Longs	48.8	49.6

Table 5: Trade-by-trade analysis. Note that trades from all 10 paths are included. Duration is in number of ticks.

On the other hand, the standard deviation between different paths obtained by the model in the validation dataset compared to that of the test dataset has a Pearson correlation coefficient of 0.73 with p-value significance 0.0002. This means that stabler candidates can be selected based on the validation process. Considering our total profit, with triannual compounding, has an uncertainty of almost 20%, our choice of candidate solely through peak validation performance was misguided, and exploring this correlation could be an important avenue for improving the system's performance.

We continue our result analysis by looking at equity growth trajectory. By equity we mean the current size of the account plus the value of any currently open positions, relative to the initial account size. Figure 5 shows the equity curve obtained by our system, with the bid price over the 7 years of testing for context.

The equity curve is somewhat regular, with no extreme drawdowns at any point. The maximum drawdown is -9.9±12.1%, without compounding, and -16.6±20.0% with triannual compounding. These are relatively low values, albeit with a large uncertainty due to how uncertainty is propagated, accumulating the uncertainty from both the peak and the drought. This amount of drawdown would allow for a comfortable use of leverage $L = 2$, which would have doubled our final profit.

Since a Q-network is a black box system and markets are a mostly inscrutable environment it is difficult to interpret the system's performance fluctuations over the datasets. However, a closer look at Figure 5 can provide some speculation, it seems there is a pattern of struggling when there is no overall trend to the price changes, when the market is sideways. This is clear during the year 2012, where equity barely rises. Also, in the years 2011 and 2015, while they are overall successful years, gains come from the beginning and the end of the year where there are clear trends while in the middle, where there is mostly sideways movement, there is little to no equity growth. Possibly, when the market has a clear trend there are distinguishing patterns in the input features that allow exploitation, while more indecisive

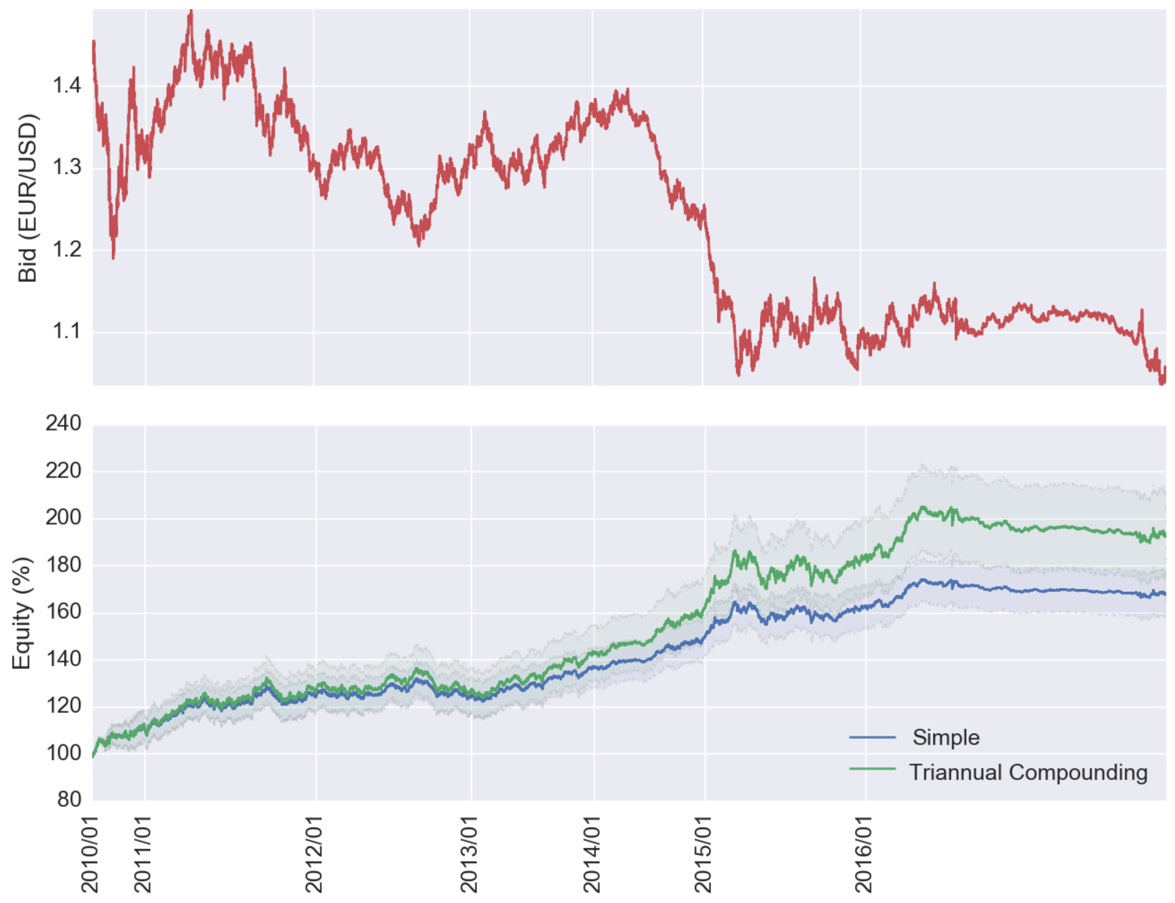

Figure 5: Bid prices for EUR/USD pair (top) and equity growth curve with and without compounding (bottom). Light area around equity curves represents their uncertainty. X-axis is in units of ticks.

periods result in noisier inputs.

This is exacerbated in the two final datasets, 2016 p.2 and 2016 p.3, which add to a mostly sideways moving market the sudden change in tick density by the broker. Due to how our system is designed, larger tick density leads to larger trading frequency. Thus, we have a system forced to trade much more frequently than intended in a market context where it has been shown to struggle, which unsurprisingly results in the biggest losses of the whole testing period.

An easy correction would be increasing the $time\_skip$ parameter to 10,000 once the larger tick frequency was detected. Doing so makes both those tests profitable, $2.5 \pm 2.2\%$ and $3.4 \pm 2.0\%$ respectively, and brings the results to those depicted in Table 6. We consider these results a realistic depiction of what the trading system designed in this thesis would have obtained as it is only natural that upon realizing the change in tracking tick data by the broker, $time\_skip$ would be accordingly adjusted to maintain a trading frequency that had so far yielded results.

Compounding frequency	Total Test Profit (%)	Yearly Avg. Test Profit (%)
None	78.5±9.5	11.2±1.4
Yearly	109.1±17.8	15.6±2.5
Triannual	114.0±19.6	16.3±2.8

Table 6: Simple and compounded total test profit for a mix of 5,000 and 10,000 $time\_skip$.

5. Conclusions

The main goals of this thesis were achieved. Learning in the training dataset is stable and it is apparent from a number of validation learning curves that the Q-network is indeed capable of finding relationships in financial data that translate to out-of-sample decision making. This speaks to the potential of the neural network / reinforcement learning combo: this noisy, non-stationary environment is a far cry from the deterministic Atari or Go environments, and still it was capable of learning performing policies. Furthermore, this learning capability was successfully leveraged to produce positive financial gain in a test setting.

Overall, it is my opinion that this approach has a great deal of potential to be explored. There are a number of probably sub-optimal parameters, chief among them the TW parameter controlling feature extraction. Furthermore there are design options, such as network topology, choice of cost function or activation function of the hidden layers, that could have been more fully explored with greater computational resources. These changes would certainly provide a performance boost without even changing the overall framework laid out in this thesis.

Besides the already discussed incomplete optimization, there are two main weaknesses in this trading system that should merit further work:

- Traning/validation/testing dataset size: a more interesting alternative based on identifying market regimes, for example, could be conjectured. Otherwise, our fixed size approach could be improved by exhaustive search rather than our limited informal testing.
- Model candidate selection: our approach to choose the model with highest validation performance was already shown to be overly simplistic when it became clear that the uncertainty the model presents in the validation dataset should have been taken into account. But more complex selection methods could be tested. For example, rather than using just one candidate the trading account could be split among a number of candidates which would help dilute the risk through diversification.

Other than changes to the system itself, some possible future work could focus on improving the financial facets of this work such as the use of other types of financial data and optimizing order execution.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *Computing Research Repository*, Apr. 2013.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, Feb. 2015.

[3] M. Krakovsky. Reinforcement renaissance. *Communications of the ACM*, 56(8):12–14, Aug. 2016.

[4] L. Liu. Reinforcement learning, 2012. URL `http://cse-wiki.unl.edu/wiki/index.php/Reinforcement_Learning`.

[5] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction*. The MIT Press, 1998.

[6] C.-H. Park and S. H. Irwin. What do we know about the profitability of technical analysis? *Journal of Economicl Surveys*, 21(4):786–826, July 2007.

[7] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 437–478. Springer, 2012.

[8] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. J. Gordon and D. B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[9] L.-J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Pittsburgh, PA, USA, 1992. UMI Order No. GAX93-22750.

[10] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, 1988.

[12] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[13] T. Schaul, I. Antonoglou, and D. Silver. Unit tests for stochastic optimization. *CoRR*, abs/1312.6055, 2013.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] C. L. Giles, S. Lawrence, and A. C. Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine Learning*, 44(1):161–183, July 2001.