Optimal action selection for the Foreign Exchange market using a Sarsa-Echo State Network

Adrian Millea, Damian Podareanu, and Marco Wiering

Abstract—This paper proposes a Sarsa-Echo State Network approach to Foreign Exchange market trading. This approach is based on using an Echo State Network to map a two dimensional time series, formed from the Bid and Ask prices to optimal Buy and Sell actions, which in turn, are learned by Reinforcement Learning. We propose a novel reward function for the learning agent that is continuous in the time domain. By employing Recursive Least Squares on the Echo State Network, mediated by Sarsa, we achieve a high performance with a simple trading strategy and moreover, using a reduced number of training steps. The system is tested on multiple currencies and periods to confirm its generalization ability.

Index Terms—Forex trading, Reinforcement Learning, Time-series, Recurrent Neural Networks, Sarsa

1 Introduction

Defined as a decentralized market for trading currencies, the Foreign Exchange Market (Forex) is an essential component of the global economy, determining the values of various currencies in respect to each other. This, in turn, facilitates international trades and investments, by enabling currency conversion. Due to the potentially unstable nature of this global financial context, studies have been carried out regarding the impact of interventions in the Forex market by various central banking institutions [5]. These have shown that the market itself is robust to unilateral central bank intervention, therefore not being influenced at a global scale by a single central bank [3]. One consequence of these interventions is an increased market volatility [4], which in turn is tightly linked with inflation [6] for emerging economies. The impact of macroeconomic news and central bank communication on the exchange rates has been recently studied [7], proving that the Forex market reacts in an intuitive manner that corresponds to exchange rate-related theories.

All this previously conducted economical research leads to the conclusion that although volatile and locally influenceable by single players, the Forex market has in fact a global component, that could be modeled in a statistical manner over time. More and more expert systems are enabled by cutting edge software and hardware solutions. Intelligent software is used for a multitude of tasks ranging from space and medical data analysis to daily human assistance tasks [9]. It is foreseeable that due to the faster and increased-capacity hardware available commercially nowadays, all context unaware legacy algorithmic solutions shall be enhanced by information obtained from the environment, leading to higher performance and interconnectivity. Furthermore, this is the stem that led to the birth of the big-data field, which is currently growing at an incredible pace. We propose in this paper a simple but efficient approach to trade on the Forex market, making use of new methods for learning.

Reviewing the numerous attempts to build automated trading systems, it is noticeable that researchers have approached various machine learning technologies in order to predict prices or find trading rules. Results point towards Artificial Neural Networks (ANN) as better performing than [12][14]. Furthermore, NN based solutions appear to be superior to a polynomial or radial basis function kernel Support Vector Machine [13]. Genetic algorithm (GA) attempts to discover optimal trading strategies [15][16] are thought to be particularly promising when compared to other machine learning techniques [17][18]. Hryshko et al.[19] also attempted to find optimal trading strategies by use of temporal difference learning. Latest research has proposed evolutionary algorithms for predicting prices [20].

Most of these approaches make use of several technical indicators. This use of historical prices or transaction volumes is in fact a separate abstraction layer. One important challenge in pattern analysis and learning is the feature extraction step - isolating the most significant characteristics of the mapped process, may it be for classification, regression, or behaviour mining. When dealing with time series, additional questions become important, such as: which sample rate should be used, what size of the time-sequence, what dependency is there between different time periods? However, the most important issue is the creation of some meaningful rich representation of the time-series without extensive computational cost which could aid in the final decision to be made. We investigate this problem and propose the use of an Echo State Network (ESN) [1] as a rich, but efficient mapping of a two dimensional time-series (representing the Bid and Ask price, details in Section 2) representing Forex. The next step in the learning process is the association of the system states to the desired actions, which in this case are Buy or Sell. We propose to do this by training two sets of read-out vectors using Recursive Least-Squares (RLS) mediated by Sarsa [2]. In fact one could see our system from two points of view, either as a Reinforcement Learning system using an ESN as a function approximator, or as a multiple-readouts ESN trained with Reinforcement Learning. Common to both is the fact that the ESN part offers a rich representation of the time-series and the Reinforcement Learning part constrains the learning in some meaningful way based on the rewards it gets. The reward giving mechanism is of course critical for fruitful learning. We hint that this biologically inspired learning method will be able to adapt to volatile market conditions, with reinforced actions yielding superior profit to naive predictions.

1.1 Contributions

Our contributions are related to the modeling of the process of making transactions on the Foreign Exchange market and the discovery of the tremendous power of the reinforcement learning paradigm in combination with the echo state network. We take into account the full spread (or transaction cost) exactly as it comes from the highly popular Forex trading platform MetaTrader 5, i.e. as two different price ratios, one for Buy and one for Sell, also known as Bid and Ask prices. So there is no additional cost involved. It is sensible to assume that if implementing this algorithm in a real life trading platform, such as MetaTrader, the same profits are to be expected as in our system. Moreover, the fact that we can use just 9 training steps to successfully transaction on 288 steps is remarkable. Lastly, although certainly not the least important, is the fact that the reward giving method is very different from the usual approach. The reward functions is continuous for the learning agent, in a time-dependent fashion, while in similar systems, the usual approach involves giving rewards just on specific actions or outcomes.

1.2 Outline

In the following sections we will first describe our learning system - ESN and SARSA, then explain the adaption to the Forex market for optimal action selection, afterwards present the experimental results and comparisons, and finish with a short discussion and future directions.

2 Learning systems

2.1 Echo state network

The echo state network is part of the Reservoir Computer paradigm [22] where a randomly connected recurrent neural network is driven by an input signal and outputs itself another signal, usually for prediction, but it can be modified to support classification too. The echo state network functions as a dynamical system, projecting the input signal into higher dimensional space (the dimension is the actual size of the reservoir, as each neuron activation can be seen as an individual signal). The neuron signals are highly redundant, so in theory there is a lot of room for improving on the size of an echo state network needed to perform a certain task. Sometimes, dependent on the task, the network is also fed with the output (desired) signal (if the output signal is different than the input signal) in the training phase. This is called teacher forcing in the literature and the motivation is quite intuitive, to provide output feedback to the network's internal states. The main equation of the echo state network, with input, and also with the output feedback, is:

$$\mathbf{x(t+1)} = f(W^{in} \cdot \mathbf{u(t)} + W \cdot \mathbf{x(t)} + W^{fb} \cdot \mathbf{y}(t)) \qquad (1)$$

where $x(t)$ is the vector containing all the reservoir states at time-step t, W is the reservoir matrix, where every entry W_{ij} corresponds to the connection between neuron i and j, $\mathbf{u(t)}$ is the input at time t, multiplied by the input vector W^{in}, W^{fb} is the feedback vector matrix, and $y(t)$ is the output at time t. This equation represents the initial driving phase of the network, where the input signal and the desired output are driving the dynamics of the network. The function f is usually chosen to be the hyperbolic tangent for the inner neurons ($tanh$) and the identity function for the output neuron(s). Some noise can also be inserted into the network update rule, which depending on the signal might be beneficial or not. The network is then let to run for a number of steps and the states are collected in a state matrix M which has on each row the state vector $x(t)$, at each time step t. On columns it has each neuron's state. Therefore it is a matrix of $training\_steps$ rows and $network\_size$ columns. We have to mention here that the first initial steps of the network are discarded when constructing the matrix M with the purpose of washing out the initial state, which is usually $[0, 0...0]_n$, with $n = network\_size$. The number of discarded steps usually depends on the nature of the time-series, as more chaotic ones tend to need more initial steps discarded than simpler functions. After collecting the states in all time steps, the usual procedure is performing a simple pseudo-inverse operation:

$$W^{out} = pinv(M) * T \qquad (2)$$

where W^{out} is the read-out vector, and T is the desired output vector (a 1-by-m vector, where m is the size of the training sequence). So, to sum it up: we have a set of m equations with n unknowns, where n is the number of neurons, the size of W^{out}, and the entries of W^{out} are the respective weightings of the neurons' states. The pseudo-inverse, or Moore-Penrose pseudo-inverse, is a generalization of a matrix inverse, but for matrices which are not rectangular. Following [10] we use an orthonormal (i.e. whose column vectors are linearly independent and have norm 1) weight matrix of 100 neurons with connectivity 0.7 and an input matrix (Win) with uniform weight values between -1 and 1. We arrived at this settings after extensive searching for good ESNs. We decided then that this is a good enough ESN and saved it to disk. Then we used the same loaded ESN for all experiments.

2.2 Sarsa

We model the problem of trading on Forex as a control problem with 2 actions. For this we use an on-policy temporal difference based algorithm, namely Sarsa [9]. So we want to learn an action-value function rather than just the state-value function. For any on-policy method we have to estimate $Q^{\pi}(s,a)$ for the current policy π and for all the states and actions s and a. The transitions are from a state-action pair to another state-action pair (thus the name state-action reward state-action, Sarsa). Any approximation method will do here, even the basic tabular representation can be used, but we choose to use the ESN for the intrinsic temporal dynamics of the network. We show in Algorithm 1 the pseudo-code of the Sarsa algorithm as given in Sutton [9]:

Algorithm 1 Sarsa

```
Init $\theta$ arbitrarily
while i < number_of_episodes do
  $s, a \leftarrow$ initial state and action of episode
  $\mathbf{x} \leftarrow$ set of features present in s
  for j = 1 to steps_of_episode do
    Take action a, observe r, and next state s
    $\delta \leftarrow r - Q_a$
    With probability $1 - \epsilon$
      For all $a \in \mathcal{A}(s)$ :
        $\mathbf{x} \leftarrow$ set of features present in s
        $Q_{a_i} \leftarrow W_i^{out}\mathbf{x}$
      $a \leftarrow argmax_a Q_a$
    else
      $a \leftarrow$ a random action $\in \mathcal{A}(s)$
      $\mathbf{x} \leftarrow$ set of features present in s
      $Q_a \leftarrow W_a^{out}\mathbf{x}$
    $\delta \leftarrow \delta + \gamma Q_a$
    $W^{out} \leftarrow W^{out} + \alpha\delta\mathbf{x}$
  end for
end while
```

In our approach, following [8], we replace the linear function approximator (θ) with the ESN, which means that the Q function is estimated as:

$$Q(\mathbf{x}, a_i) = W_i^{out}\mathbf{x} \tag{3}$$

where $\mathbf{x}$ is the vector constituted of the network states (each element of the vector is a neuron's activation), a_i is the action with index i, (in our case i goes from 1 to 2, where 1 is a Sell and 2 is a Buy) and W_i^{out} is a read-out associated with action number i. We also benefit from the proof of convergence present in [8] assuring us that using Sarsa with the ESN as the function approximator will converge to a bounded region. The authors test the system on a simple maze problem and their results seem quite satisfactory, however we investigated further the maze problem (changing the goal state) and saw that the performance varies when changing the goal state, which is to be expected (the middle states are much worse defined than the corner states), but performance better than the tabular representation is always achieved. We show a depiction of Sarsa-ESN in Figure 1. We now turn to the problem of trading on Forex.

2.3 Sarsa-ESN on the FOReign EXchange

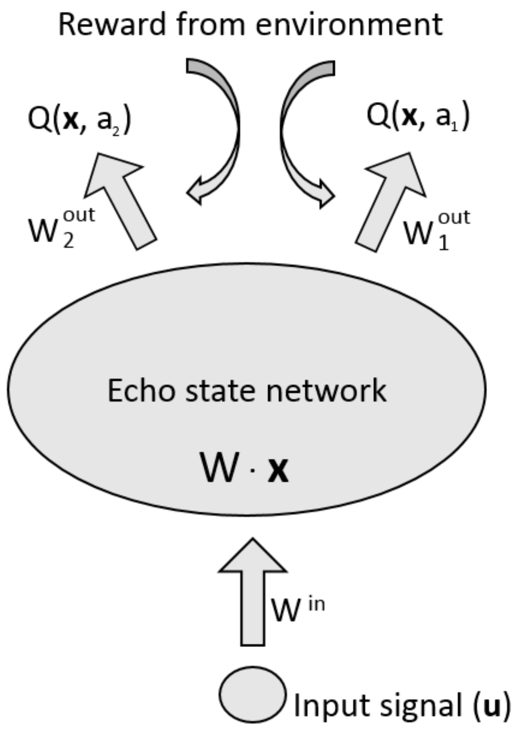

Fig. 1: A depiction of the Sarsa-ESN system.

2.3.1 Data

When considering that the Forex market data is a highly chaotic time-series, we are faced with a few choices, for example which period should we choose for the time-series which we will perform transactions on (1 minute, 1 hour, 1 day) ? After a few experiments, we can safely conclude that 1 minute data has too little variability for making significant profit and a lot of the money transactioned is lost into the spread (the difference between the Sell price and the Buy price). The spread can be seen as the fee of the broker for each transaction, or the transaction cost, as it is known in the literature. When testing on 1 hour data, the profits seem promising, but we think that by using 1 minute data for example, the overall profits can be increased, by making more transactions in the same time period. Ideally, we would like to have as many transactions as possible and exploit as much as possible the variability of the time-series. We use data from 1st of March 2014 to 23rd of June 2014 for our experiments. Each data point represents 1 hour, in total 1901 data points, excluding of course the time when the market is closed. We preprocess the data to obtain what is called in [11] the return and is defined as:

$$y_t = \frac{P_t}{P_{t-1}} - 1 \tag{4}$$

where P_t is the current price and P_{t-1} is the previous price. We perform this transformation for both Bid and Ask prices. We show in Figure 2 the preprocessed data, this is the data we feed to our system.

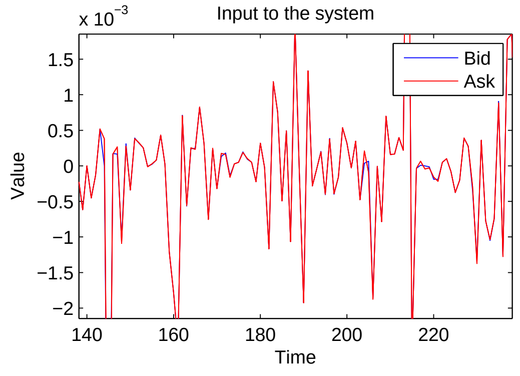

Fig. 2: Bid and Ask prices after transformation.

2.3.2 Reward

As we mentioned earlier how the agent gets the reward from the environment is critical for the ability of the system to learn.

2.3.3 Trading strategy

For the trading strategy we employ a very simple one ([11]). Remember that our system outputs an action at each time step. If the current action is different than the previous action, then an operation (an operation is Buy or Sell) is performed according to the current value of the action, otherwise if the previous action is the same as the current action then no operation is performed. We invest each time 1000 money units, with a leverage of 100. We use just one value of the lots at each operation, and even though this limits a lot the system, we will see that the performance is quite remarkable. So a direct way of improving the system is to customize also the number of lots, so basically each operation would have a specific weight based on how certain the system is of the success of this operation.

Algorithm 2 Reward routine

```
Get last two actions a(i-1),a(i)
while i < number_of_steps do
  if a(i-1)==buy then
    if a(i)==sell then
      difference = -Ask(i-1)+Bid(i);
    else
      difference = -Bid(i-1)+Bid(i);
    end if
  else
    if a(i-1)==sell then
      if a(i)==buy then
        difference = Bid(i-1)-Ask(i);
      else
        difference = Ask(i-1)-Ask(i);
      end if
    end if
  end if
  Reward = difference * constant;
end while
```

2.3.4 Experiments

When doing experiments, instead of the usual repetition of the experiment, we choose to change the data we feed for training, so that we know it generally works and it is not just a specific time-series that works. We choose for this 9 offsets, from 50 to 500 in increments of 50. We also vary the training size from 1 to 10 in increments of 1 and then from 10 to 100 in increments of 10. We do some experiments for 7 currencies: Euro - U.S. Dollar (EURUSD), Australian Dollar - Japanese Yen (AUDJPY), Australian Dollar - U.S. Dollar (AUDUSD), Swiss Franc - Japanese Yen (CHFJPY), Euro - Great British Pound (EURGBP), U.S. Dollar - Canadian Dollar (USDCAD) and Australian Dollar - Canadian Dollar (AUDCAD). We have to mention here that the testing set is 288 steps and it starts always from the next step after the last training step.

3 Results

3.1 Training size

Investigating the appropriate training size for our system lead to some surprising results. We use an epsilon of 0.2 for Sarsa which is slowly going to 0 in 2000 epochs. After 2000 epochs, we consider that the system has converged. Even with few data points, our system seems to yield adequate results. Increasing the number of data points,

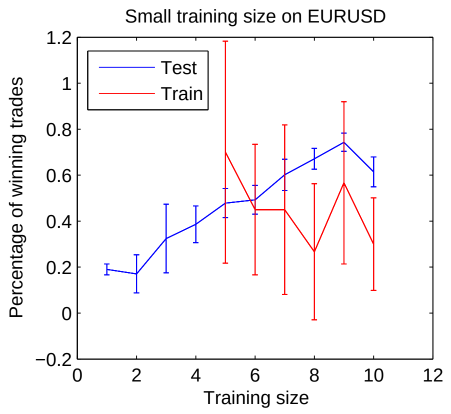

Fig. 3: Mean and standard deviation of winning trades for train and test sequences.

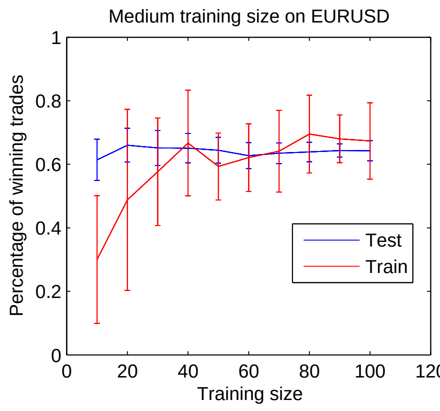

Fig. 4: Mean and standard deviation of winning trades for train and test sequences.

leads to more system constraints, due to the need to deal with more examples which, even though in theory makes the system more robust, in practice, it lowers performance compared to the fewer data points scenario. From another point of view, it becomes apparent that the state of the system at each epoch (the learning system is characterized just by the W_i^{out} with i taking values 1 and 2) is changed based on the data points encountered. However, at each epoch the system has a different initial value, the learned value from the previous epoch, therefore the data points have a different effect on it. Still, the results from Figures 34 are unexpected, however by testing on different offsets and with multiple currencies it proved to be consistent. It is predictable that for some different data the optimal training size would have a different value. This seems to be a feature of this specific choice of data, being a low complexity time-series, representing a limited number of 4 months worth of data, with each point being 1 hour. We show in Figure 3 one such experiment for EURUSD showing the mean and standard deviation for the 9 offsets and for 1 to 10 training steps. It is noticeable that there are no values for training performance from 1 to 4, this is because the system has too few points to make transactions, so a transaction is not performed in this small interval, thus the training performance being not defined. We wanted to be sure of our results, thus we performed the same experiment for all 7 currencies investigated. We show in Figure 5 the mean and standard deviation over 9 offsets for all currencies. We see that the results are very similar for all, the maximum performance is achieved with 9 training steps. We wanted to investigate what happens exactly with the system, so we show the actual transactions which the system performs on the training data in Figure 6, as well as a sample from the test data (the whole sequence of 288 steps would make the image too cluttered) in Figure 7. The red diamonds represent the Sell operations, the black diamonds are the Buy operations, while the green lines denote winning transactions and magenta lines denote losing transactions respectively.

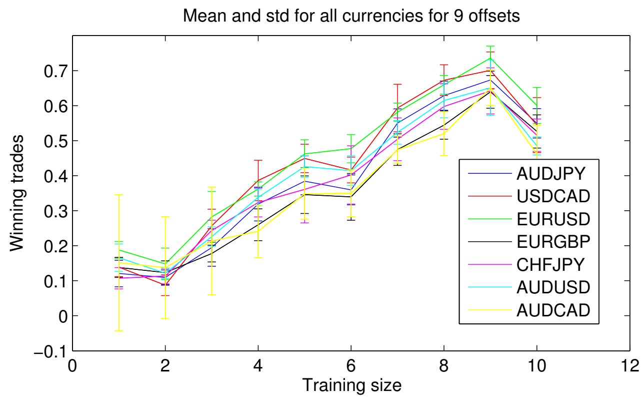

Fig. 5: Mean and standard deviation performance on the test set for all currencies.

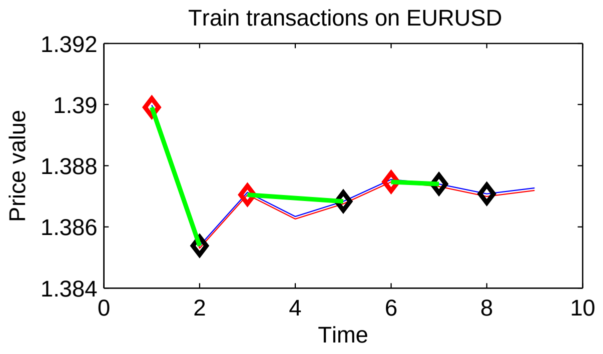

Fig. 6: Transactions on train data.

Figure 8 shows the actual cumulative profit for the 2000 epochs considered with 9 training steps. This constantly goes up, reaching a level of almost 800 euro in 288 hours! We consider this to be a remarkable performance with so little computational cost, with 2000 epochs and 100 neuron ESN. Figure 9 contains the reward for the 2000 epochs, which has a quite peculiar evolution in time. The explanation of this is that the system has few training points and correctly getting one transaction sends the system into one regime, while not getting that transaction sends the system into another regime. That

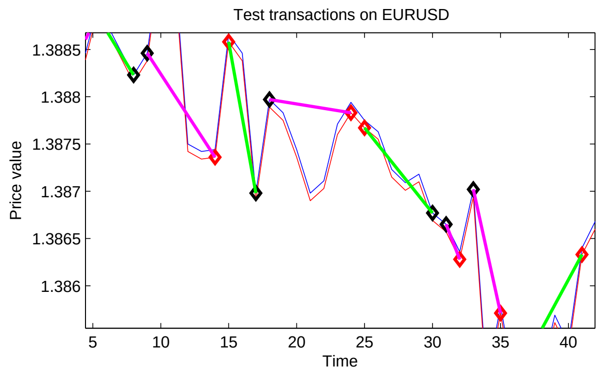

Fig. 7: Sample from the transactions on test data.

is why the oscillating behavior occurs with very few possible transactions to be done (for more training steps the reward plot has a different shape). This might also be the reason for the high performance when having a very small training size, the system tunes itself with high sensitivity to gain positive rewards, even though when presented with little data. We will see later that the system performs well for many training sizes, but we this case study shows an unexpected result after only 9 data points, so 9 hours.

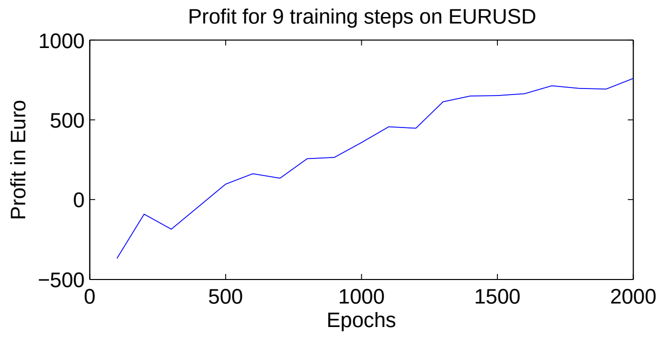

Fig. 8: Profit for 9 training points.

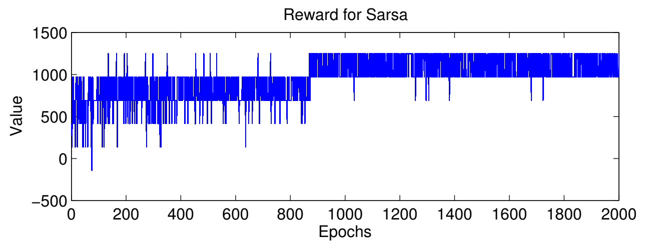

Fig. 9: Reward for 9 training points.

3.2 Comparisons

For completeness, we picked at random one of the previously trained strategie and tested how it works with the same magic number of training steps, i.e. 9, then how a random projection works (so no non-linearity and no time-dependency) and finally how a linear ESN performs in comparison with the non-linear version. We know from [10] that linear ESNs with orthonormal matrices are excellent when dealing with some time-series, especially for mapping (1-step prediction). In this paper's context, by a random strategy, we mean that the systems selects a ramdom action at each time-step. Random projection means that the actual time-series mapping becomes:

$$\mathbf{x} = f(W^{in} \cdot \mathbf{u}(\mathbf{t})) \tag{5}$$

where f is now the identity function and as there is no time dependency in the sense that there is no term involving the previous $\mathbf{x}$. Considering the fact that the weights W_i^{out} are learned, which then in turn are multiplied by $\mathbf{x}$, the state of the system, this is very similar to Sarsa using a linear function approximator for the Q function. Figure 10 contains the results on the 9 offsets tested, with winning trades percentage for all methods. We use for this experiment the EURGBP price. To see how the system performs with more training steps, which will probably be the case for a highly robust system, the same comparison is performed for 50 training steps (Figure 11). The highest performer is in this case the Sarsa with the non-linear ESN which was to be expected.

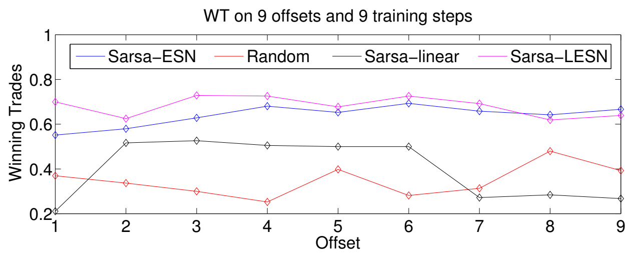

Fig. 10: Comparison with other systems.

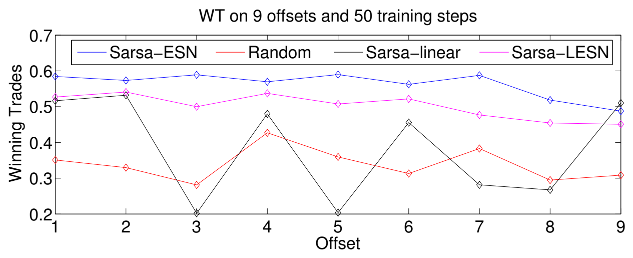

Fig. 11: Comparison with other systems.

4 Conclusion

4.1 Discussion

The most surprising result was the very reduced number of training steps needed for good performance. The performance curve goes steeply to high values from 1 to 9 steps. We do however hypothesize that this might be just a peculiarity of the time-series we used and not a general rule for any period, in this paper we use data from March to end of June 2014. This was tested in one last experiment using the same period of the year (1 March - 23 June), but selecting all years from 2007 to 2013 inclusively and using the EURUSD pair. Figure 12 contains these results, on the 9 offsets. It seems that indeed that for some years the performance is barely over 0.5, so the 9 training steps are a time-series specific value. It is remarkable though that the rest of the tested yers yield high performance. Therefore, it can be concluded that the training size is dependent on the specific time-series with some values (like 9) working better than other for many samples.

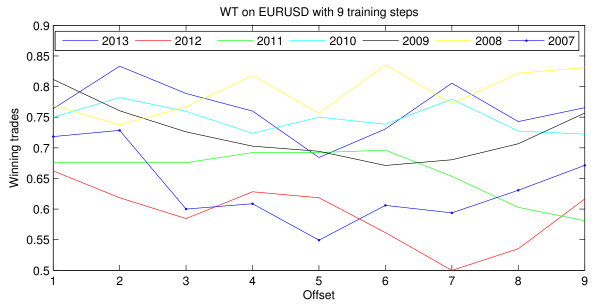

Fig. 12: EURUSD with 9 training steps in multiple years.

4.2 Future directions

We can envision many future directions to our initial developments. As previously mentioned, we could increase the number of actions from the system. Possibilities for this could be, for example, the amount invested in each operation (number of lots), so in fact this would lead to another 2 actions for investing 2 times more at each Buy/Sell. In this way the system should learn which operations are more certain and which are not, without any probabilistic modeling. Another interesting direction would be to extend our basic reward giving routine to a multi-objective reward, taking into account also the overall number of transactions, to increase or decrease their number (probably increasing them would be more beneficial). Finally to be able to have agents which are risk-averse or risk-seekers would be most desirable and this could be done easily by employing a risk term in the reward function.

References

[1] Jaeger, H. (2001). The ”echo state” approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148, 34.

[2] Chen, S. L., & Wei, Y. M. (2008, October). Least-Squares SARSA (Lambda) Algorithms for Reinforcement Learning. In Natural Computation, 2008. ICNC'08. Fourth International Conference on (Vol. 2, pp. 632-636). IEEE.

[3] Beine, M., Bos, C. S., & Laurent, S. (2007). The impact of central bank FX interventions on currency components. Journal of Financial Econometrics, 5(1), 154-183.

[4] Beine, M., Laurent, S., & Palm, F. C. (2009). Central Bank forex interventions assessed using realized moments. Journal of International Financial Markets, Institutions and Money, 19(1), 112-127.

[5] Goodhart, C. A., & Hesse, T. (1993). Central Bank Forex internvention assessed in continous time. Journal of International Money and Finance, 12(4), 368-389.

[6] Berganza, J. C., & Broto, C. (2012). Flexible inflation targets, Forex interventions and exchange rate volatility in emerging countries. Journal of International Money and finance, 31(2), 428-444.

[7] Egert, B., & Kocenda, E. (2014). The impact of macro news and central bank communication on emerging European forex markets. Economic Systems, 38(1), 73-88.

[8] Szita, I., Gyenes, V., & Lorincz, A. (2006). Reinforcement learning with echo state networks. In Artificial Neural Networks-ICANN 2006 (pp. 830-839). Springer Berlin Heidelberg.

[9] Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning. MIT Press.

[10] Millea, A. (2014). Explorations in Echo State Networks. Unpublished Master's Thesis. University of Groningen, Groningen, The Netherlands.

[11] Maciel, L., Gomide, F., Santos, D., and Ballini, R. Exchange rate forecasting using echo state networks for trading strategies. Submitted to Computational Intelligence for Financial Engineering (CIFEr), 2014

[12] Kamruzzaman, J., Sarker, R. A., & Ahmad, I. (2003, November). SVM based models for predicting foreign currency exchange rates. In Data Mining, 2003. ICDM 2003. Third IEEE International Conference on (pp. 557-560). IEEE.

[13] Kamruzzaman, J., & Sarker, R. A. (2003, December). Forecasting of currency exchange rates using ANN: A case study. In Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on (Vol. 1, pp. 793-797). IEEE.

[14] Deng, S., & Sakurai, A. (2013, March). Foreign exchange trading rules using a single technical indicator from multiple timeframes. In Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on (pp. 207-212). IEEE.

[15] Bauer, R. J. (1994). Genetic algorithms and investment strategies (Vol. 19). John Wiley & Sons.

[16] Kim, K. J., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. Expert systems with Applications, 19(2), 125-132.

[17] Hirabayashi, A., Aranha, C., & Iba, H. (2009, July). Optimization of the trading rule in foreign exchange using genetic algorithm. In Proceedings of the 11th Annual conference on Genetic and evolutionary computation (pp. 1529-1536). ACM.

[18] Hryshko, A., & Downs, T. (2003, December). An implementation of genetic algorithms as a basis for a trading system on the foreign exchange market. In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on (Vol. 3, pp. 1695-1701). IEEE.

[19] Hryshko, A., & Downs, T. (2004). System for foreign exchange trading using genetic algorithms and reinforcement learning. International journal of systems science, 35(13-14), 763-774.

[20] Yaman, A., Lucci, S., & Gertner, I. (2014). Evolutionary Algorithm Based Approach for Modeling Autonomously Trading Agents. Intelligent Information Management, 2014.

[21] Villa, S., & Stella, F. (2014). A continuous time Bayesian network classifier for intraday FX prediction. Quantitative Finance, (ahead-of-print), 1-14.

[22] Schrauwen, B., Verstraeten, D., & Van Campenhout, J. (2007). An overview of reservoir computing: theory, applications and implementations. In Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007 (pp. 471-482).