# AUTOMATED FOREX TRADING USING RECURRENT REINFORCEMENT LEARNING

**Charles Ashby-Lepore**
Department of Computer Science
University of Montreal
charles.ashby-lepore@mail.mcgill.ca

## ABSTRACT

In this project, we replicated a paper by Dempster & Leemans (2006) who propose a fully automated trading algorithm based on Adaptive Reinforcement Learning (ARL). Their method relies on a layered structure composed of a Recurrent Reinforcement Learning (RRL) algorithm, a risk management layer that protects past gains and avoids losses by shutting down trading activity in times of high uncertainty and a dynamic utility optimization layer that can automatically re-calibrate the algorithms hyper-parameters.

## 1 INTRODUCTION

For many years, practitioners and academics have tried to build trading models to predict foreign exchange prices. More recently, a lot of attention has been given to reinforcement learning techniques. In this section, we review the most popular techniques that are being used and introduce the concept of recurrent reinforcement learning.

(Neuneier, 1996) trained a neural network using Q-learning to maximize cumulative profit. However, in his analysis, the trader is assumed to have no risk aversion, i.e. it can handle any amount of variance in it's returns. Following their work, (Gao & Chan, 2000) used an equivalent technique using Q-Learning by maximizing the Sharpe ratio instead of the profit to obtain a system that can effectively control it's risk-return trade-off.

(Kaelbling et al., 1996) shed light on the issue of scaling traditional reinforcement learning systems (TD-learning) to realistic environment such as financial trading due to rarely occurring states. (Dempster et al., 2001) compared searching value function space (using TD learning) versus searching policy space (using an evolutionary algorithm) and demonstrated that while both methods had complementary strengths, the TD learning approach showed classical signs of overfitting the in-sample data which they attributed to the rarely occurring states. This paper motivated (Dempster & Romahi, 2002) where they introduced an evolutionary reinforcement learning approach. They show that this hybrid algorithm limits overfitting and improves out of sample performance.

(Moody & Saffell, 1998) compared recurrent reinforcement learning (also known as direct reinforcement learning) with Q-learning, they showed that the RRL agent outperformed the QTrader and noted that the QTrader traded more frequently (which can be a cause of overfitting) and also found that Q-learning can suffer from the curse of dimensionality. For those reasons, they come to the conclusion that Q-learning is harder to use than the RRL approach.

While all these authors agrees that reinforcement learning systems have a lot of potential, practically, most of them can never be used in production. This is because they are still viewed as far too risky in comparison to the returns they are able to deliver. Even, if we can show that a system generates a reasonable risk-return profile on historical data, there is no reason why they would keep this profile (this level of utility based on the trader's risk aversion) in the future. To counter this problem, (Dempster & Leemans, 2006) introduce a layered system using adaptive reinforcement learning.

Their system is composed of a Recurrent Reinforcement Learning (RRL) algorithm, a risk management layer and a dynamic utility optimization layer that can automatically re-calibrate the algorithms hyper-parameters in terms of market condition in order to keep an optimal risk-return profile based on the trader's risk aversion. The combination of reinforcement learning and the automated hyper-parameter calibration is coined adaptive reinforcement learning. In this project, we replicate their work and show that their system still work in today's market environment, we also run experiments to compare their model using a single layer recurrent neural network with a long short-term memory network as (Lu, 2017) did.

Section 2 gives an overview of recurrent reinforcement learning. Section 3 introduce the layered system that we used and describes the data on which our experiments were ran. Section 4 presents our results and our methodology. Section 5 concludes.

## 2 Recurrent Reinforcement Learning

Recurrent Reinforcement Learning (RRL) was first applied for trading in Moody & Saffell (1999). The model is a single layer recurrent neural network that outputs the position to take at each time step, $sgn(F_t) \in \{-1, 0, 1\}$ (long, neutral, short):

$$F_t = \tanh(\Sigma_{i=1}^{M} w_i r_{t-i} + w_{M+1} F_{t-1} + b)$$

Where $w_i$ are the weight vectors, $b$ is the bias and $\{r_i\}_{i=1}^{M}$ are the last $M$ returns. The goal is to maximize the Sharpe ratio which is a function of the profit at time $t$, $S(t) = \frac{E[R_t]}{Var(R_t)} = \frac{A}{\sqrt{B-A^2}}$ where $R_t = F_{t-1}r_t - \delta|F_t - F_{t-1}|$, $\delta$ is the transaction cost, $A = \frac{1}{T}\Sigma_{t=1}^{T} R_t$ and $B = \frac{1}{T}\Sigma_{t=1}^{T} R_t^2$. The model is trained on a window of length $L_{train}$ for $n_{epoch}$ epochs, than, inference is made on the following $L_{test}$ time steps. Next, the training window is advanced $L_{test}$ time steps and the procedure is repeated. In our experiments, we use $L_{train}, L_{test}, n_{epoch} = 1000$, unless explicitly defined.

The weights are updated at each time step with standard gradient ascent using full derivatives of $\frac{\partial S_t}{\partial w}$ and learning rate $\rho$. We can show that:

$$\frac{\partial S_t}{\partial w} = \Sigma_{t=1}^{T}\left(\frac{dS_t}{dA}\frac{dA}{dR_t} + \frac{dS_t}{dB}\frac{dB}{dR_t}\right)\left(\frac{dR_t}{dF_t}\frac{dF_t}{dw} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{dw}\right)$$

$$\frac{dR_t}{dF_t} = -\delta sgn(F_t - F_{t-1}), \frac{dR_t}{dF_{t-1}} = r_t - \delta sgn(F_t - F_{t-1})$$

$$\frac{dF_t}{dw} = (1 - tanh(w^t x_t)^2)(x_t + w_{M+1}\frac{dF_{t-1}}{dw})$$

Since the derivatives of the Sharpe ratio with respect to the weights depends on the derivatives of all the previous time steps ($t \in [1, T]$), we can update the weights using the back propagation through time algorithm where they are updated at every time steps instead of taking an average at the end.

## 3 The Trading System

Figure 1 illustrates the architecture of the system that we reproduced from (Dempster & Leemans, 2006).

### 3.1 The Risk Management Layer

The RRL algorithm is trained to produce a trading signal $F_t$ at each time step that is fed to the risk management layer who will take the decision to enter a position depending on some key metrics. First of all, we will change a position if and only if the strength of the signal is stronger than a threshold $|F_t| > \gamma$.

Second, this layer builds a trailing stop-loss to protect its profits from sudden changes in market behaviour. To implement this stop-loss, we store the maximal cumulative return for the last trade

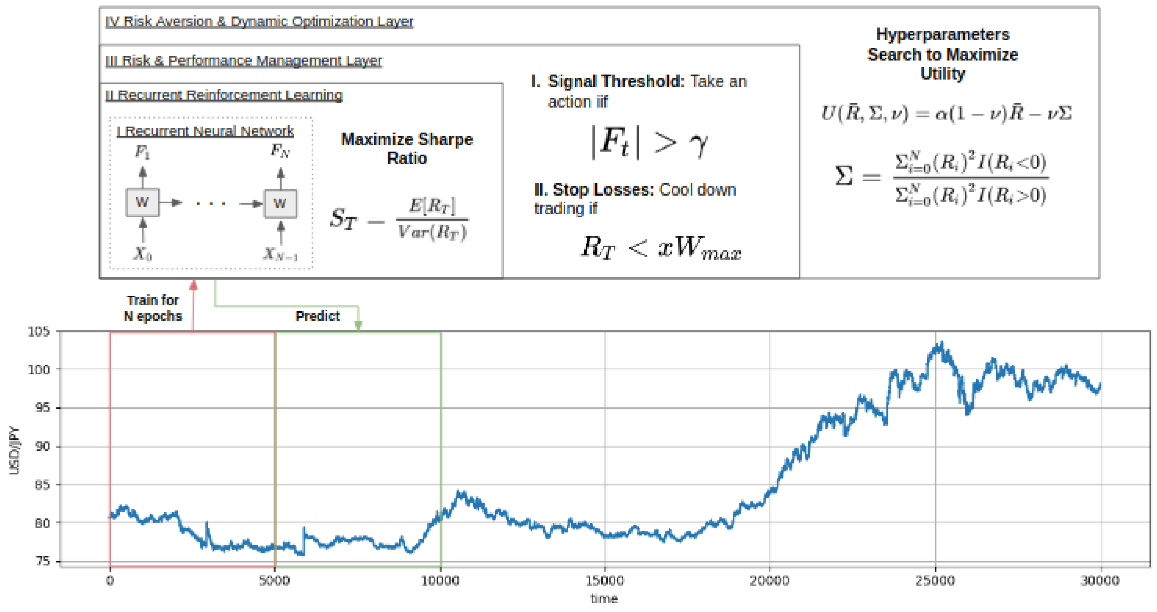
IV Risk Aversion & Dynamic Optimization Layer
III Risk & Performance Management Layer
II Recurrent Reinforcement Learning
I Recurrent Neural Network
$F_1$
$F_N$
W
W
$X_0$
$X_{N-1}$
Maximize Sharpe Ratio
$S_T = \frac{E[R_T]}{Var(R_T)}$
I. Signal Threshold: Take an action iif
$|F_t| > \gamma$
II. Stop Losses: Cool down trading if
$R_T < xW_{max}$
Hyperparameters Search to Maximize Utility
$U(\bar{R}, \Sigma, \nu) = \alpha(1-\nu)\bar{R} - \nu\Sigma$
$\Sigma = \frac{\Sigma_{i=0}^{N}(R_i)^2 I(R_i<0)}{\Sigma_{i=0}^{N}(R_i)^2 I(R_i>0)}$
Train for N epochs
Predict
USD/JPY
time

Figure 1: Layered Recurrent Reinforcement Learning Architecture

$W^{p_t}_{max}$. Whenever a return is $x$ point of percentage lower than the maximal value of a position, we cool-down trading for a period of 100 time steps. i.e. when

$$R_t < xW^{p_t}_{max}$$

We also shut down trading until the next training episode ($L_{test} - t$ time steps) when a negative return is greater than $z$ percent of the total cumulative return. i.e. when

$$R_t > zW_t = z\Sigma_{i=1}^{t} R_i$$

### 3.2 The Dynamic Optimization layer

The recurrent reinforcement learning algorithm and the risk management layer introduce multiple hyper-parameters, it's the optimization layer's job to find a setting of these parameters that satisfy the trader's expectation in terms of risk and reward trade-off. To do so, the authors introduce the concept of utility $U = \alpha(1-\nu)R - \nu\Sigma$ where $\alpha$ is a constant, $\nu$ is the trader's risk aversion, $R$ is the average reward per time step and $\Sigma = \frac{\Sigma_{i=0}^{N}(R_i)^2 I(R_i<0)}{\Sigma_{i=0}^{N}(R_i)^2 I(R_i>0)}$. We find the optimal values for $\delta$, $\rho$, $x$, $z$, $\gamma$ and $n_{epoch}$ by solving

$$\max_{\delta}\max_{\rho}\max_{x}\max_{z}\max_{\gamma} U(R, \Sigma, \nu)$$

We implement a one-at-a-time random search optimization; we pick 15 values for each hyper-parameters and keep the one that maximize utility while keeping the other constant. This random search is than repeated using all the previous returns at constant intervals (e.g. At each 10,000 steps, find the new optimal hyper-parameters).

## 4 Experiments

Truefx.com provided us with tick data on the USD/JPY foreign market from january 2011 to december 2017. We preprocessed it to obtain high, low, volumes and bid-ask spread for each period

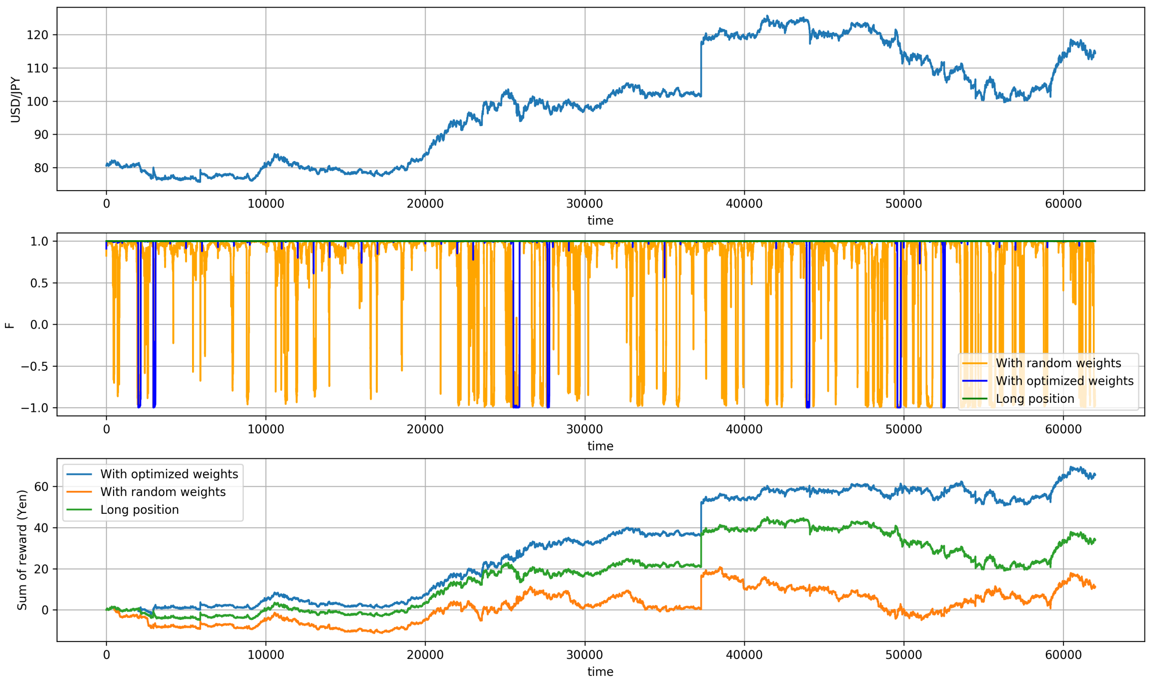

Figure 2: Cumulative returns for the layered recurrent reinforcement learning system. The first row plots the exchange rate of the USD/JPY during january 2011 to december 2017. The second row shows the trading signal for our algorithm (blue line), for a network with random weights (orange line) and for a long position (green). The last plot illustrates the return of the three strategies.

of 30 minutes during that interval. The returns that our model use are computed as $r_t = p_t - p_{t-1}$ where $p_t = \frac{high_t + low_t}{2}$. Periods when the market is down are removed from the dataset.

Figure 2 shows the total cumulative return for our algorithm with trained (blue) and random (orange) weights, as well as the return for a long position during a period of approximately 6 years (from march 2011 to march 2017). We can see that the model managed to get a return of 65.573 yens (around 81% return) which is comparable to the 26% return over 2 years on the USD/EUR market reported by the authors.

The second graph from figure 2 shows the trading signal outputted by the neural network at each time steps, as mentioned before, a signal of $+1$ is equivalent to taking a long position, $0$ is neutral and $-1$ is a short. Looking at this graph, we can see that the model has a long position almost everywhere, however, it still achieves a much greater return than the long position (green line). Therefore, it seems that it was able to benefit from short term movements in the prices without missing out on the bigger picture.

In our second experiment, we wanted to test the effect of using an LSTM instead of a simple recurrent neural network. We also wanted to see if using a deep neural network who takes as inputs the previous 100 returns to extract features that are fed to the RNN returns would help our algorithm. Finally, we compared training using gradient descent versus Adam.

Figure 3 illustrates the risk (horizontal axis) - return (vertical axis) profile of all the algorithms that we trained. The blue dots represents a RNN and the green dots represent an LSTM. On each graph we modified the transaction cost, the learning rate or the number of epochs and plotted the average return per time step versus the standard deviation that the algorithm obtained on a subset of 10,000 time steps.

We can see that it is not clear whether the RNN performs better or worse than the LSTM. Ideally, we would prefer algorithms with low variance and high return (top left), we can see that usually,

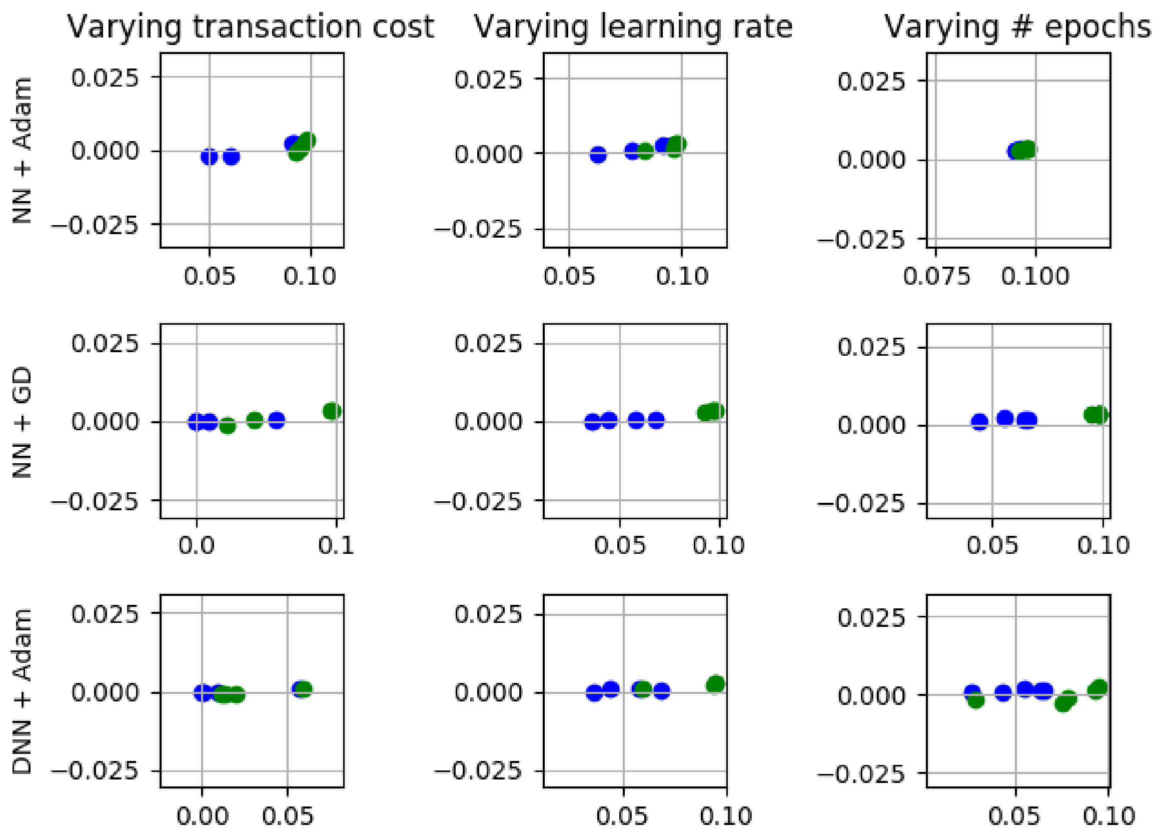

Figure 3: Hyper-parameter search on the RRL algorithm. The green dots represent an LSTM, blue dots are a simple RNN. The vertical axis is the average return per time steps on a period of 10,000 time steps, the horizontal line is the standard deviation. We vary transaction costs $c \in \{0.001, 0.0005, 0.0001, 0.00005\}$, learning rate $\rho \in \{0.001, 0.005, 0.01, 0.008\}$ and the number of epochs $\in \{1, 50, 100, 500, 1000\}$. First and last row are trained using Adam, second row is trained using standard gradient descent and last row uses a deep neural network whose output is fed to the RNN.

the LSTM (green dots) have higher variance than the RNN. However, they also tend to have slightly higher returns.

Finally, using a deep neural network to extract features typically produced lower returns. This is probably due to the fact that there are more weights to train, therefore, the algorithm requires more epochs to converge.

## 5 Conclusion

In this project we replicated the work by (Dempster & Leemans, 2006). We also tested the recurrent reinforcement learning algorithm with deeper architectures and LSTM as in (Lu, 2017). We found that the layered approach still worked in today's market even though the authors warned that they were becoming more efficient (i.e. arbitrage is hard). However, we found that it is not clear if deep neural networks and complex RNN architecture are helpful in trying to predict market behaviours.

## References

M. A. H. Dempster and Y. S. Romahi. Intraday fx trading: An evolutionary reinforcement learning approach. In Hujun Yin, Nigel Allinson, Richard Freeman, John Keane, and Simon Hubbard (eds.), *Intelligent Data Engineering and Automated Learning — IDEAL 2002*, pp. 347–358, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45675-9.

Michael Dempster and V Leemans. An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30:543–552, 04 2006. doi: 10.1016/j.eswa.2005.10.012.

Michael Alan Howarth Dempster, TW Payne, and Yazann S Romahi. *Intraday FX trading: Reinforcement vs evolutionary learning*. Judge Institute of Management Studies, 2001.

Xiu Gao and Laiwan Chan. An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. In *Proceedings of the international conference on neural information processing*, pp. 832–837, 2000.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996. URL `http://arxiv.org/abs/cs.AI/9605103`.

David W Lu. Agent inspired trading using recurrent reinforcement learning and lstm neural networks. *arXiv preprint arXiv:1707.07338*, 2017.

John Moody and Matthew Saffell. Reinforcement learning for trading. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pp. 917–923, Cambridge, MA, USA, 1999. MIT Press. ISBN 0-262-11245-0. URL `http://dl.acm.org/citation.cfm?id=340534.340841`.

John E. Moody and Matthew Saffell. Reinforcement learning for trading. In *NIPS*, 1998.

Ralph Neuneier. Optimal asset allocation using adaptive dynamic programming. In *in Advances in Neural Information Processing Systems*, pp. 952–958. MIT Press, 1996.