Cambridge University Finance Seminar 17 March 2006

Design of an FX trading system using Adaptive Reinforcement Learning

M A H Dempster

Centre for Financial Research

Judge Institute of Management

University of Cambridge

&

Cambridge Systems Associates Limited

Coworker: V Leemans

Outline

- Introduction to trading systems
- RRL: The basic machine learning algorithm
- Improving RRL
- Adding risk and performance management
- Measuring the utility of trading performance
- Automatic tuning
- Evaluating performance

Introduction to trading systems

- Systems that are able to trade financial markets:
  - better than humans (profit, risk management)
  - fully autonomously
- Subject of extensive research among academics and professionals (hedge funds, banks)
- Replicating a real trader by transforming input signals to trade recommendations:
  - Inputs are often past returns but any information available can be fed in
  - Outputs are trade entry and exit signals

Introduction to trading systems

- Often the goal is maximisation of trading profits or the Sharpe ratio
- Many attempts in the past: systems based on fundamental analysis, technical analysis, econometric modelling and machine learning
- Few attempts successful because trading system needs to outperform the market out-of-sample consistently (danger of overfitting!)
- But market behaviour changes over time a system must learn from its own failures and successes in trading the market

Introduction to trading systems

- Since a high-frequency proprietary trader wants to exploit his superior knowledge instantaneously he needs to buy at ask and sell at bid

  $\Rightarrow$ trader faces transaction costs: bid-ask spread plus fees and slippage

- This not only decreases profits but also constrains the range of profitable strategies to those that avoid frequent position switching

  $\Rightarrow$ The model needs to find a trade-off between reacting as fast as possible to exploitable situations and reacting as infrequently as possible to keep costs low

Introduction to trading systems

- Another issue is strategy risk: a model could generate high expected returns but have an unacceptable risk profile that would lead e.g. to large draw-downs and margin calls making leveraged positions a guaranteed ticket to bankruptcy

- The lack of proper risk control is another reason why professional traders are generally not keen on artificial intelligence trading systems

Introduction to trading systems

Requirements for a good trading system

- ✓ Outperform the market consistently
- ✓ Continuously adapt to changing market behaviour
- ✓ Find a balance between frequent trading and low transaction costs
- ✓ Fully automated
- ✓ Integrated risk management possibly depending on user's personal risk preferences. Automatically adjusting trading activity in times of high uncertainty or unfavourable conditions and protecting profits

Outline

- Introduction to trading systems
- RRL: The basic machine learning algorithm
- Improving RRL
- Adding risk and performance management
- Measuring the utility of trading performance
- Automatic tuning
- Evaluating performance

The machine learning algorithm

- Core of the trading system is a machine-learning algorithm called recurrent reinforcement learning (RRL)
  Moody (1999)
- Later the RRL algorithm was successfully applied to FX trading
  Moody&Saffell (2001) Gold (2003)
- Relatively old datasets were used to train and test this algorithm
  Olsen (1996)
- Unfortunately application of Moody's algorithm to recent datasets shows that RRL no longer works well in today's FX markets

The machine learning algorithm

The trading model takes as inputs standardized past time series returns and outputs the preferred position (long or short). Further, the assumption is made that the model can be specified as a recurrent single layer neural network:

$$F_t \;=\; \text{sign}(\sum_{i=0}^{M} w_{i,t} r_{t-i} + w_{M+1,t} F_{t-1} + v_t)$$

where $F_t \in [-1, 1]$ is the position to take at time t, $\mathbf{w}_t$ and v_t are respectively the weight vector and threshold of the neural network at time t, and $r_t := p_t - p_{t-1}$ are the returns of the FX time series

The machine learning algorithm

At every trade exactly 1 unit of a certain currency is bought or sold and hence the profit at time T can be calculated as follows when risk free interest rates are ignored:

$$
\begin{aligned}
P_T &= \sum_{t=0}^{T} R_t \\
R_t &= F_{t-1} r_t - \delta |F_t - F_{t-1}|
\end{aligned}
$$

where δ is the transaction cost

The machine learning algorithm

For computational efficiency Moody chose the Differential Sharpe Ratio as the risk-adjusted return measure to be optimized. It is derived by considering a moving average version of the classic Sharpe Ratio:

$$\begin{aligned} \widehat{S}(t) &= \frac{A_t}{B_t} \\ A_t &= A_{t-1} + \eta(R_t - A_{t-1}) \\ B_t &= B_{t-1} + \eta(R_t^2 - B_{t-1}) \end{aligned}$$

The machine learning algorithm

When $\widehat{S}(T)$ is expanded in the adaptation parameter η as a Taylor series the first derivative term D_t can be regarded as an instantaneous performance measure:

$$
\begin{aligned}
D_t &= \frac{d\widehat{S}(t)}{d\eta}\Big|_{\eta=0} \\
&= \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{\frac{3}{2}}}
\end{aligned}
$$

Increasing η from 0 means turning on the adaptation.

The machine learning algorithm

Because of its simplicity and tractability a simple gradient ascent is used as optimization algorithm to gradually improve the model parameters:

$$w_{i,t} = w_{i,t-1} + \rho \Delta w_{i,t}$$

Δw can then be approximated by an on-line update by only considering the term that depends on the most recent trading return R_t:

$$\begin{aligned} \Delta w_{i,t} &= \frac{dD_t}{dw_i} \\ &\approx \frac{dD_t}{dR_t}\left\{\frac{dR_t}{dF_t}\frac{dF_t}{dw_{i,t}} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{dw_{i,t-1}}\right\} \end{aligned}$$

The machine learning algorithm

Since the network is recurrent, an updating scheme similar to back-propagation through time can be applied:

$$\frac{dF_t}{dw_{i,t}} \approx \frac{\partial F_t}{\partial w_{i,t}} + \frac{\partial F_t}{\partial F_{t-1}} \frac{dF_{t-1}}{dw_{i,t-1}}$$

Together the previous form the set of equations necessary for updating the model weights This machine-learning technique was termed Recurrent Reinforcement Learning (RRL)

The machine learning algorithm

- RRL is suitable for a rolling-window approach
- Train the model on an initial set of the first L_{train} returns and test out-of-sample on the next L_{test} returns
- Then advance the window by L_{test} datapoints and repeat the procedure
- Out-of-sample performance of the model is the sum of the performance on non-overlapping individual test sets

Outline

- Introduction to trading systems
- RRL: The basic machine learning algorithm
- Improving RRL
- Adding risk and performance management
- Measuring the utility of trading performance
- Automatic tuning
- Evaluating performance

Improving RRL: Extension 1

- Extend the input space: also feed other signals into the trading model apart from past returns

- Tried feeding signals originating from 14 popular technical indicators Dempster&Jones (2001) Dempster et al. 2001)

- Performance did not improve except when only a low number M of past returns were fed into the system

- Pre-filtering of returns offered by technical indicators is unable to improve performance

⇒ RRL is able to efficiently exploit the information in past time series returns

Improving RRL: Extension 2

- In the standard RRL implementation the transaction cost factor δ is fixed to the actual bid-ask spread
- Instead let δ be a tuning parameter that can adopt values larger than the actual bid-ask spread of the FX-rate
- Setting a higher cost factor necessitates a higher expected raw profit before engaging in a trade. By consequence δ will influence the performance and risk profile of the resulting trading strategy

Improving RRL: Extension 3

- Large jumps in FX returns (e.g. caused by central bank interventions) can introduce instability in the underlying RRL algorithm

- After a weight has jumped to a large value it often starts drifting to even larger values. These large values cause numerical and model instabilities

- Prevent this behaviour by rescaling all weights by a factor f < 1 as soon as a threshold value has been exceeded. (Note that rescaling all the weights will not have any impact on the trading decisions.)

Improving RRL: Extension 4

Smarter position updating scheme to avoid indecisive switching between positions:

- In the standard RRL implementation weights are updated after the new inputs have been received and the trading signal based on those new inputs is calculated

- This trading signal is in fact calculated with the 'old' weights

- Thus it makes sense to re-evaluate the trading model a second time with the current (new) inputs as before but this time with the new weights. Only this final trading signal is used for decision making

Improving RRL

- Modifications 2, 3 and 4 to the standard RRL algorithm all improve its performance

- Only the first modification does not add any value when a sufficiently large number M of past return inputs are used

- The concept of this first modification remains interesting however. It demonstrates a possible way of incorporating any extra information a trader possesses into the model Bates et al. (2003)

Overview

- Introduction to trading systems
- RRL: The basic machine learning algorithm
- Improving RRL
- Adding risk and performance management
- Measuring the utility of trading performance
- Automatic tuning
- Evaluating performance

Risk and performance management

Why?

- Stop-loss due to real-world solvency constraints (cut the losses)

- A-priori evaluation of current trade quality allows to fit overall strategy risk-return profile to desired one

- Auto shut-down procedure to close system down when abnormal behaviour is detected

Risk and performance management

- Process of decision to enter a trade separated from the trade recommendation by the model   Veturi (2003)

  ⇒ Separate layers:
    - Core RRL trading model
    - Risk and performance management
- Why not incorporate both in 1 large model?
  - Model would become more complex and estimating the parameters by RRL will become much more difficult
  - Finding a profitable trading model is easier if there are no additional risk management constraints. Hence assume an idealized world and add additional features in a way that does not impact the underlying model structure

Stop-losses

- When a trade goes bad a psychological tendency exists to keep the position open in the hope that market will reverse itself and the trade will turn profitable again
- HOWEVER:
  - Often the market will not reverse itself
  - A gigantic loss could take you out of business for ever

- THEREFORE this behaviour needs to be systematically avoided by an automated stop-loss

Stop-losses

- The draw-down from the current position matters so implement trailing stop-loss Veturi (2002) Dempster & Romahi (2002)

$\Rightarrow$ A stop-loss is set and adjusted so that it is always x basis points under or above the best price ever reached during the life of the position

- At this point x is an unknown parameter that will influence the trading performance

- Numerical values for x will be obtained later

Stop-losses

- Stop-loss hit ⇔ RRL model currently has incorrect view
- From model's point of view the market behaves unexpectedly
- The model will not instantaneously change its recommendations in the current market situation because it is designed to avoid frequent position switching

⇒ Need for a 'cool-down' period after a stop-loss has been hit
Trading is halted during this short period

⇒ However the model continues to learn (real-time) from the current market environment during this period in order to optimally adapt itself to the new situation

A priori trade quality evaluation

- 'Strength' of the trading signal is given by the unthresholded output F of the trading model e.g. a very strong buy signal corresponds to an output close to +1
- Validate a trading signal only when the output exceeds a specified threshold $y \neq 0$ instead of just applying a sign-function
- At this point y is an unknown parameter that will influence the trading performance
- A numerical value for y will be obtained later

Auto shut-down

- Trading system may cease to be profitable due to:
  - Drastic sudden change in market behaviour that was never seen before
  - Software or system errors which lead to wrong updating and mess up the model
  - …
- In these cases the system needs to be shut down to protect profits

- Need to detect anomalous performance:
  - VaR based on fitted draw-down distribution
  - If amount lost over certain period > fixed amount z

Overview

- Introduction to trading systems
- RRL: The basic machine learning algorithm
- Improving RRL
- Adding risk and performance management
- Measuring the utility of trading performance
- Automatic tuning
- Evaluating performance

Measuring utility of performance

- How can different trading systems be compared in a well specified way?
- Define a utility function on out-of-sample trading performance
- Different utility functions possible with various requirements:
  - Monotonically increasing with average profit per unit time
  - Monotonically decreasing with strategy risk
  - Adaptable to a trader's risk aversion
- A rational quadratic risk sensitive utility function is used but others are possible

Measuring utility of performance

$$U(\overline{R}, \Sigma, \nu) \;:=\; a(1-\nu)\overline{R} \;-\; \nu\Sigma$$

Σ is a quadratic measure of strategy risk and $R_i := W_i - W_{i-1}$ (with W_i the wealth or cumulative profit at time i) is the strategy's raw return at time i. $\overline{R} := \frac{W_N}{N}$ is the average profit per frequency interval, ν is used to impose the trader's personal risk aversion, a is a scaling constant and N is the number of out-of-sample time intervals

Measuring utility of performance

How to define the strategy risk measure Σ ?

Requirements:

1. For a given total profit Σ needs to be large if there are many losing trades and small if there are many winning trades

   => punishment for deviations from monotonically upward sloping cumulative profits

2. For the same cumulative loss Σ should be large when a gigantic losing trade occurs and small when several losing trades occur

Measuring utility of performance

- Proposed strategy risk indicator Σ :

$$\Sigma := \frac{\sum_{i=0}^{N}(R_i)^2 I_{(R_i<0)}}{\sum_{i=0}^{N}(R_i)^2 I_{(R_i>0)}}$$

where $I_{(.)}$ is the indicator function

Measuring utility of performance

The utility of a trading strategy depends on the average profit $\overline{R}$ and on the risk measure Σ which in turn are controlled by the values of 5 parameters that were discussed before: trading costs δ, adaptation parameter η, learning rate ρ, stop-loss parameter x and trading threshold y.

$$U(\overline{R}, \Sigma, \nu) \;:=\; U(\delta, \eta, \rho, x, y, \nu)$$

Overview

- Introduction to trading systems
- RRL: The basic machine learning algorithm
- Improving RRL
- Adding risk and performance management
- Measuring the utility of trading performance
- Automatic tuning
- Evaluating performance

Automatic tuning

- Given that a utility function can be defined on a trading strategy this function can be maximized in the free parameters to obtain an optimal utility for a given risk-aversion setting

- This not only automatically calibrates the tuning parameters of the system but also tailors the system to the user's risk-return preferences

- Hence the problem of overfitting becomes less of an issue here as no system parameter except for user risk aversion is specified externally

Automatic tuning

- The optimization problem becomes:

$$\max_{\delta,\eta,\rho,x,y} U(\delta,\eta,\rho,x,y,\nu)$$

- There is no closed form relationship between average profits and the risk measure on one hand and the model parameters on the other hand

- However given the model parameters it is possible to evaluate the resulting utility

$\Rightarrow$ Optimization must be effected numerically

Automatic tuning

- The model needs to be optimized numerically with the following constraints applying:
  - Given the length of the high-frequency datasets the utility evaluations are rather costly in time with the current C++ implementation
  - The utility function is not known to be 'well-behaved' and thus convergence to the global optimum can not be guaranteed
- But empirically the utility function is reasonably smooth around the global optimum and we know how to find good starting values

Automatic tuning

- Therefore the choice was made to implement a one-at-a-time random search optimization in which each parameter is approximately optimized individually while keeping the others constant:

$$\max_{\delta}\max_{\eta}\max_{\rho}\max_{x}\max_{y}\; U(\delta,\eta,\rho,x,y,\nu)$$

- The individual optimisations are performed by evaluating 15 random starting values distributed normally around the starting value and picking the best value:
  - This saves evaluations
  - We do not have to rely on numerical derivatives which are not easily evaluated for our utility function
  - **Other more sophisticated optimization schemes can be applied**

Automatic tuning

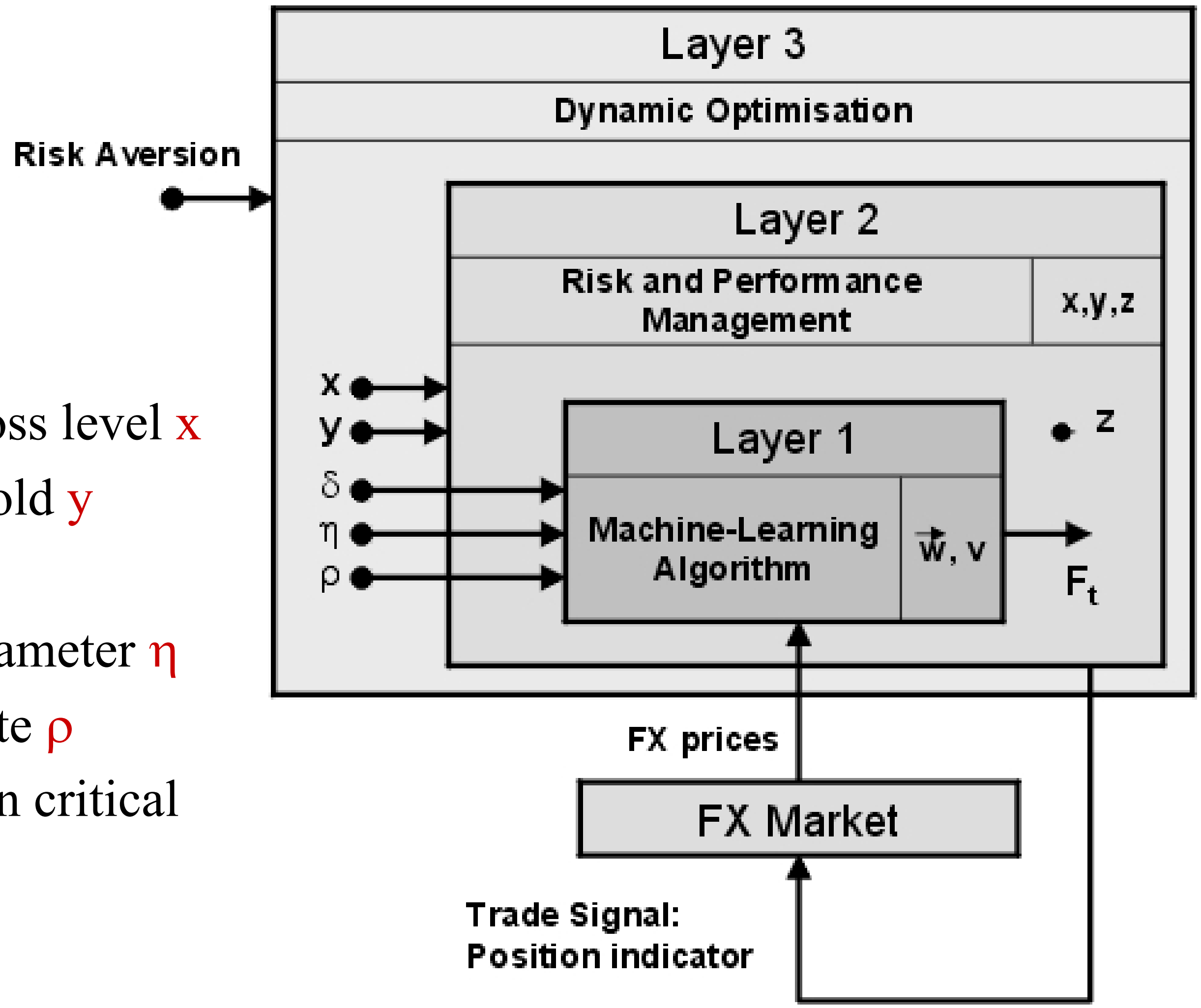
- **Note:** Care must be taken to properly schedule all multilayer processes and optimizations
- There are 2 optimization processes involved that are separated by the risk management layer:
  - At RRL level: advancing out-of-sample trading period periodically and then retraining the RRL model
  - At utility optimization level: periodically re-optimizing system parameters. The frequency of this optimization is lower than that of the first one:
    - For computational efficiency reasons
    - These hyper-parameters do not and should not change that frequently

Automatic tuning

- This methodology of periodically retuning the free system parameters that determine the resulting risk-return profile or govern the learning behaviour in a reinforcement learning framework was termed:

Adaptive Reinforcement Learning (ARL)

ARL System



- Layer 3 optimizes:
  - Trailing stop-loss level x
  - Trading threshold y
  - Trading cost δ
  - Adaptation parameter η
  - NN learning rate ρ
- The auto-shut-down critical loss z was fixed

Overview

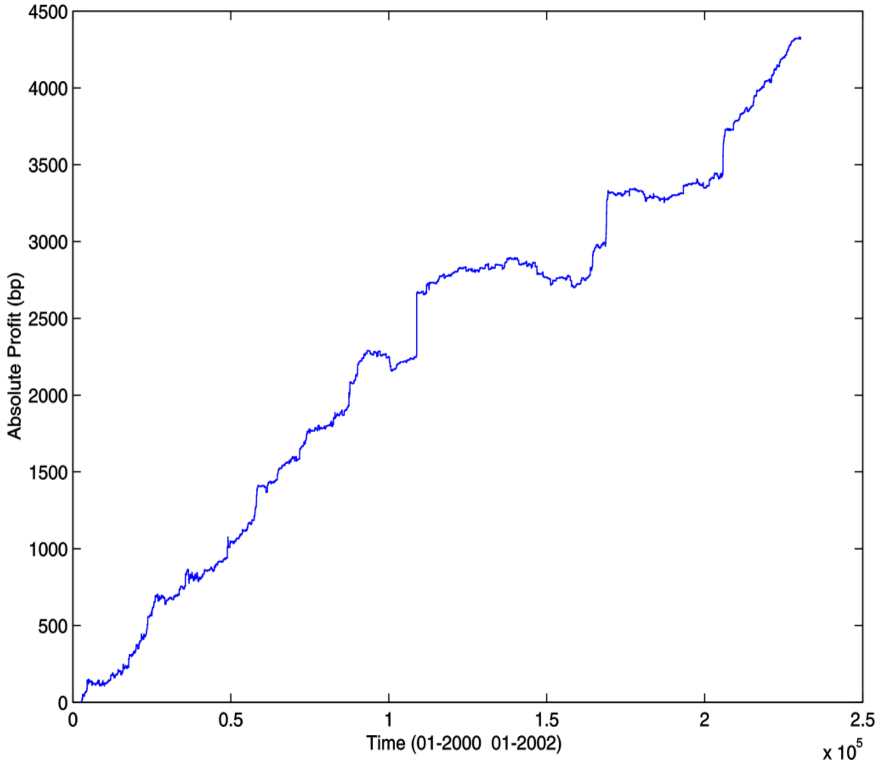
- Introduction to trading systems
- RRL: The basic machine learning algorithm
- Improving RRL
- Adding risk and performance management
- Measuring the utility of trading performance
- Automatic tuning
- Evaluating performance

Evaluating performance

- Testing the system on EUR-USD FX data:
  - 1 min frequency
  - 01-2000 until 01-2002
  - Inter-dealer spread of 2 bp during active hours
  - Only trade during active hours so that slippage is near zero
  - Credit lines of 1 EUR and 1 USD available
  - Zero leverage

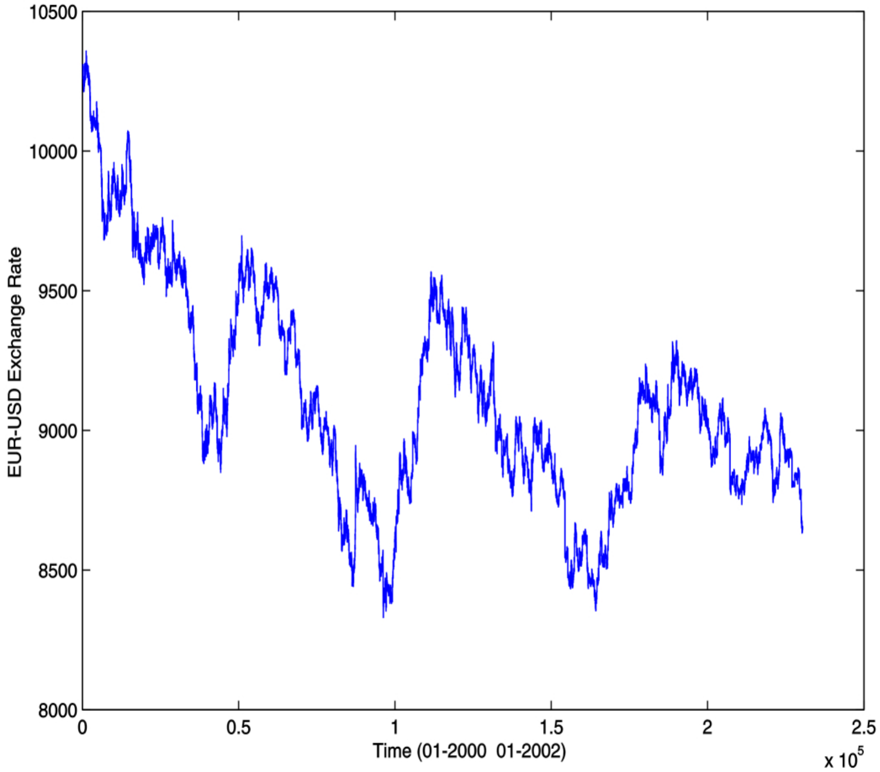
- Graph of out-of-sample (rolling window) ARL performance for a risk aversion factor of 0.5 compared to in-sample performance of system fitted to entire data set

Evaluating performance



ARL profits

+ 4328 pips or 21.6 % p.a.



EUR-USD price

- 1636 pips or –8.2 % p.a.

Evaluating performance

- To illustrate the benefits of this structural approach and the top-level utility optimization a series of experiments were run to compare performance and utility with and without dynamic optimization

- When very low risk aversion parameters are used the impact of risk in the utility function is insufficient to give a less risky strategy

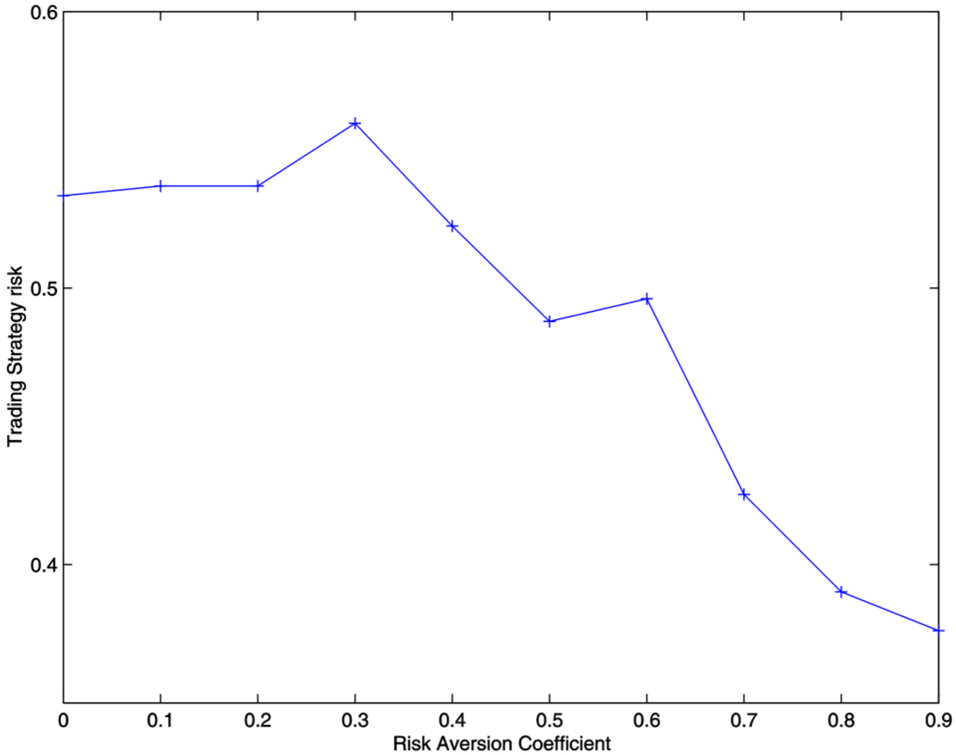
Evaluating performance

Risk Aversion	Avg. profit (bp)	Direction (%)	Cumul. Profit (bp)	Risk	Utility	Utility w/o layer 3
0	1.67	62%	4779	0.5334	2.07	1.92
0.1	1.65	62%	4717	0.5369	1.79	1.67
0.2	1.65	62%	4717	0.5369	1.53	1.43
0.3	1.55	62%	4663	0.5596	1.25	1.18
0.4	1.53	62%	5144	0.5225	1.13	0.93
0.5	1.53	62%	5104	0.4880	0.86	0.69
0.6	1.58	62%	5083	0.4962	0.58	0.44
0.7	1.53	61%	4780	0.4254	0.32	0.19
0.8	1.74	62%	4635	0.3901	0.09	-0.05
0.9	1.77	63%	4669	0.3761	-0.14	-0.30

- Out-of-sample statistics on the trading simulations
- Average net profit made per trade ranged from 1.53 pips to 1.77 pips depending on the trader's risk aversion

Evaluating performance

Trading strategy risk Σ



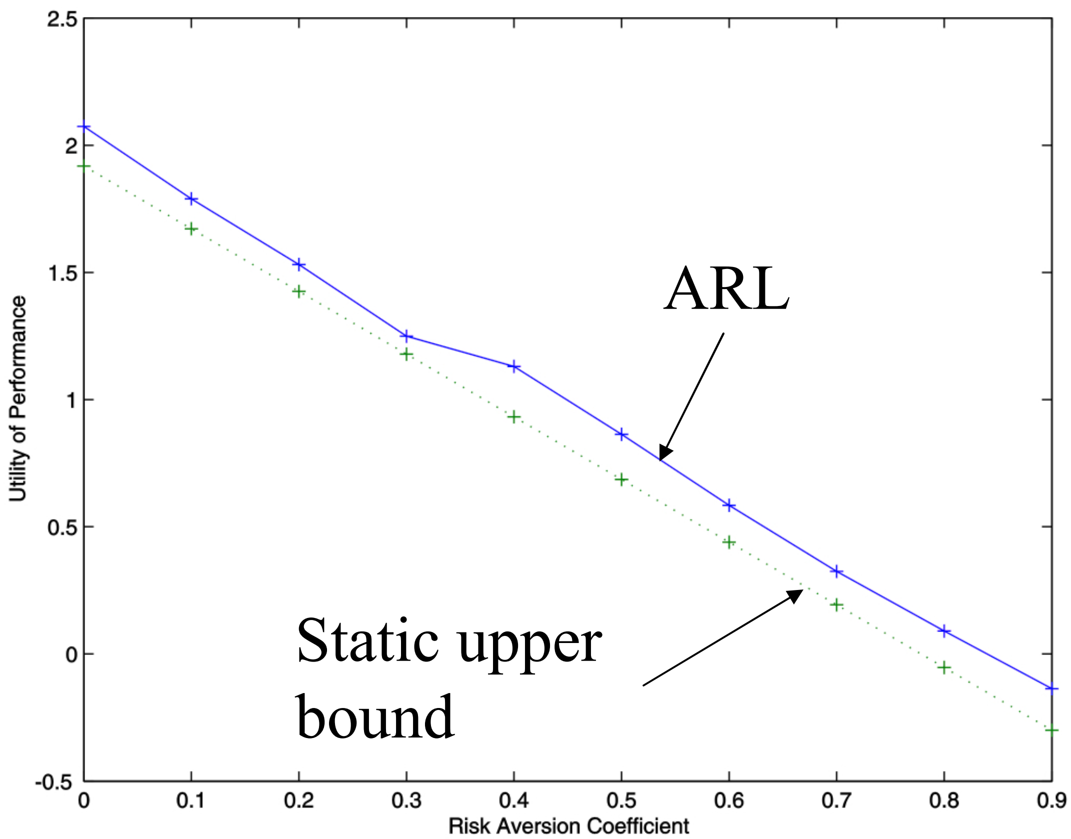
When a larger and more realistic risk aversion parameter is used the optimization layer makes sure that the resulting system is less risky

Evaluating performance

- Compared the utilities of performance for different risk aversions with the corresponding utilities when no dynamic optimization is performed
- In the second case the values of the 5 hyper-parameters were set so that they gave optimal performance on the whole dataset
- To assure that this static parameter setting is the best one possible and thus provide a reliable benchmark static optimal values were calculated based on the training as well as the test sets

⇒ Gives an **upper bound** on trading performance without optimization

Evaluating performance



- ARL performs consistently better than the benchmark

⇒ Optimization in layer 3 contributes significantly to better trading performance

Summary

- Developed an automated FX trading system based on dynamically optimizing parameters of an RRL-type system
- The parameters that govern the learning behaviour and influence the risk profile were optimized by maximizing a utility function at regular points in time (hence Adaptive RL)
- The risk-aversion setting allows control of the system's trade-off between risk and return
- Out-of-sample trading performance looks promising albeit only attainable by market makers and traders at large banks

Current and Future Research

- Can trading performance be improved by feeding the system with more (different) information e.g. order flow and limit order book? Bates et al. (2003) Leemans (2006)
- Extend risk management layer to control several FX trading models that trade different currencies in an attempt to diversify holdings
- Extend ideas from proprietary trading systems to market making systems Bank of America