Recipe For Quantitative Trading With Machine Learning

Daniel Bloch

8th of November 2018

Recipe For Quantitative Trading With Machine Learning

Daniel BLOCH [*]
QUANT FINANCE
Working Paper

8th of November 2018
Revised version : 1.2.4

Abstract

On one hand, financial time series are multifractal, thus exhibiting non-Gaussian distribution, the presence of extreme values (outliers), and long-range dependent dynamics. On the other hand, machine learning (ML) models are processes relying heavily on statistical models and methodologies, but treated as black box models due to their inability to explicitly know the relations established between explanatory variables (input) and dependent variables (output). However, when forecasting market returns, or generating autonomous patterns, it is crucial to know the statistical properties of the time series produced by the ML model under consideration.
Taking into consideration these statistical characteristics, we present a recipe using technical indicators to forecast both market returns and their directions. We choose to reverse the causality and propose a solution consisting in deciding upon the framework by defining how the model should be specified before beginning to analyse the actual data. We should define the framework by properly formulating model hypotheses which make financial or economic sense, and then carefully determining the number of dependent variables in a regression model, or the number of factors and components in a stochastic model.
Based on these observations, we use the multifractal formalism (MF) as a framework for testing the capacity of an ML model to reproduce some non-overlapping statistical properties of the time series. First we define a set of theoretical models with distinct statistical characteristics as well as a set of ML models with the ability to reproduce the properties of the models in the former set. We then train each individual ML model to replicate their respective theoretical model and use ensemble methods to combine these now calibrated ML models to form a meta-model. However, the weights of the latter are statistically computed as a function of the Hurst exponent. As a result, the meta-model is dynamically recombined based on the changing properties of the financial series over time. Once we have identified an ensemble of ML models with specific non-overlapping statistical properties, we can train each constituent model to learn a large number of patterns or technical indicators. We then devise a trading algorithm with strategies accounting for a specific level of Hurst exponent.

Machine Learning, Recurrent Neural Networks, Associative Reservoir Computing, Multifractal Formalism, Forecasting Returns And Directions, Trading Algorithm

1 Introduction

The accumulating evidence against the efficiency of the market (the efficient market hypothesis (EMH)) has caused a resurgence of interest in the claims of Technical Analysis (TA) (see Appendix (8)) as the belief that the distribution of price dynamics being totally random is now being questioned (see Appendix (7)). In general, the frequency distribution of returns is a fat-tailed, high-peaked distribution existing at many different investment horizons. Further, the standard

[*] dan@quantfin.eu

deviation of returns increases at a faster rate than the square root of time. As a result, correlations come and go, and volatility is highly unstable. Taking into consideration these observations, Mandelbrot [1960] [1963a] [1963] described the financial market as a system with fat tails, stable distributions and persistence. There exists a large body of work on the mathematical analysis of the behaviour of stock prices, stock markets and successful strategies for trading in these environments (see Section 2.1). While investing involves the study of the basic market fundamentals, which may take several years to be reflected in the market, trading involves the study of technical factors governing short-term market movements together with the behaviour of the market. As a result, trading is riskier than long-term investing, but it offers opportunities for greater profits (see Hill et al. [2000]).
Assuming that the historical data in the markets forms appropriate indications about the market future performance, TA relies on indicators designed to help a trader determine whether current behaviour is indicative of a particular trend, together with the timing of a potential future trade. However, there is no better indicator, so that the indicators should be combined in order to provide different perspectives. Further, a technical indicator always need to be applied to a time window, and the problem of determining the best time window can be the solution to an optimisation problem (see Fernandez-Blanco et al. [2008]), which requires sophisticated search and examination methods.

Machine learning is about programming computers to learn and to improve automatically with experience (see A. Samuel [1959]). While computers can not yet learn as well as people, algorithms have been devised that are effective for certain types of learning tasks, and a theoretical understanding of learning has emerged. Computer programs developed, exhibiting useful types of learning, especially in speech recognition and data mining (see Mitchell [1997]). Thus, new techniques combining elements of learning, evolution and adaptation from the field of Computational Intelligence developed, aiming at generating profitable portfolios by using technical analysis indicators in an automated way. In particular, subjects such as Neural Networks (see Appendix (11.1.2)), Swarm Intelligence, Fuzzy Systems and Evolutionary Computation can be applied to financial markets in a variety of ways such as predicting the future movement of stock's price or optimising a collection of investment assets (funds and portfolios). These techniques assume that there exist patterns in stock returns and that they can be exploited by analysis of the history of stock prices, returns, and other key indicators (see Schwager [1996]). With the fast increase of technology in computer science, new techniques can be applied to financial markets in view of developing applications capable of automatically manage a portfolio. Consequently, there is substantial interest and possible incentive in developing automated programs that would trade in the market much like a technical trader would, and have it relatively autonomous. Mechanical trading systems (MTS) founded on technical analysis developed. It is a mathematically defined algorithm designed to help the user make objective trading decisions based on historically reoccurring events.

If our goal is to forecast returns, or generate autonomous patterns, from a financial time series by learning some of its behaviour in the hope it repeats in the future, we should concentrate on series exhibiting long range dependence (LRD) rather than short time memory. Thus, it is crucial to understand the statistical properties of financial time series and concentrate on the latter. Further, machine learning (ML) is not a black box, but a process relying heavily on statistical models and methodologies (see Appendices (9) and (11.1.2)). It is inherently a multidisciplinary field drawing results from artificial intelligence, probability and statistics, computational complexity theory, control theory, information theory, philosophy, psychology, neurobiology, and other fields. In general, Recurrent Neural Networks (RNN) are designed to reproduce transient processes and must be modified (with feedback) to capture LRD properties. Hence, it would make sense to first design an ML model with the desired statistical properties, and then train it to recognise as many technical indicators (or patterns) as possible.

In Section (3.1), we test neural networks on the Mackey-Glass signal and several real-time financial time series in order to gain some insight on their capability of reproducing the dynamics of financial time series. We choose to model transient processes with Reservoir Computing (RC), which is an approach designed to train and analyse RNNs in a fast and efficient way (see Jaeger [2001]). There is no learning in the recurrent connections, only between the State units and the Output units, and adaptation is based on a simple associative learning mechanism (see Dominey et al. [2000]). RC models with output feedback were proposed to generate financial time series with interesting properties since feeding the output signal back into the reservoir allows to tackle prediction tasks. In order to improve

the accuracy of the forecast returns, we also consider an online algorithm behaving as an adaptive filter throughout the running.
In Section (3.2), we repeat the analysis using Associative Reservoir Computing. Given that an associative connection between two entities is bidirectional, if the forward relation is many-to-one, the reverse relation (one-to-many) is ambiguous. This is always the case if there exists several causes resulting in the same effect. More formally, if there is a non-injective relation between inputs and outputs, there is no unique solution to the inverse problem. This problem is typical of financial time series and traditional feed-forward models can not represent multiple solutions to a mapping without additional knowledge. In the case of persistent processes we consider solving the ambiguous problem displayed by market returns by using the Associative Reservoir Computing Theory (ARC) to shape multi-stable attractor dynamics that reflect the multiplicity of solutions (see Reinhart et al. [2009]). To do so, we overwrite the reservoir forward representation by a joint representation involving input and output. That is, to learn a mapping from output to input we rely on a feedback matrix to access the latter, and define a matrix mapping the reservoir states to the input layer. We then test the model on the Mackey-Glass signal and FX time series. We describe in Section (3.4) an algorithm to forecast the directions of the underlying asset which we enlarge to forecast both market returns and their directions. To conclude, we draw conclusions on the results and define a set of rules we want to follow when building our trading algorithm.

In Section (4), we use the multifractal formalism (MF) as a framework for testing the capacity of an ML model to reproduce some statistical properties of time series. We consider a set $\mathscr{T}$ of theoretical models $\mathcal{T}_m$, $m = 1, ..., M$, having non-overlapping pre-defined statistical properties. We also consider a set $\mathscr{M}$ of ML models $\mathcal{M}_m$, $m = 1, ..., M$, each one of them capable of reproducing one theoretical model from the set $\mathscr{T}$. We train each ML model $\mathcal{M}_m$ to approximate its associated theoretical model $\mathcal{T}_m$ (calibration), and use ensemble methods to combine the calibrated ML models $\mathcal{M}_{calib,m}$, obtaining a meta-model. However, the weights of the latter are statistically computed as a function of the Hurst exponent. As a result, the meta-model is dynamically recombined based on the changing properties of the time series over time. Once we have identified an ensemble of ML models with specific non-overlapping statistical properties we can train each model to learn a large number of patterns or technical indicators. We then devise a trading algorithm with strategies accounting for a specific level of Hurst exponent.

2 Presentation

Machine learning (ML) can give computers the ability to learn without being explicitly programmed (see A. Samuel [1959]). It focuses on designing, developing and analysing methods giving a computer the ability to follow a systematic process leading to producing results in difficult tasks that classical algorithms could not produce. As such, it is is a general method for solving optimisation problems (see Appendix (11.1.2)).
Asset returns and their properties are discussed in Appendix (6). We are going to consider forecasting asset returns as well as their directions with machine learning, and use the results to devise quantitative trading strategies. In the former, prediction task is directly applied to the market returns, while in the latter, data mining is performed on data sets and classification is privileged among the different possible tasks (see Appendix (9)). We are going to briefly introduce the prediction tasks and discuss training and exploitation of the model. We will then present some results on the Mackey-Glass signal and FX market returns.

2.1 Market returns

The stock market is an organised and regulated financial market where securities such as bonds, notes, and shares are bought and sold at prices governed by the forces of demand and supply. This demand coupled with advances in trading technology has opened up the markets for investors to perform quantitative trading with minimum risk. The price of an asset as a function of time constitutes a financial time series. High-frequency financial data are observations on financial variables taken daily or at a finer time scale, and are often irregularly spaced over time. We call the tick time scale the finest time scale and denote it t_{tick}. It corresponds to transaction-by-transaction data on trades and quotes.

These ticks are irregularly spaced time series with random daily numbers of observations. In general, for FX quotes data, the date and time are expressed in conventional format as $dd.mm.yyyy$ and $HH : MM : SS$, respectively. Consequently, a lot of attention had been devoted to the analysis and prediction of future values and trends of the financial stock markets (see Hargreaves et al. [2013]). We let the underlying process $(S_t)_{t \geqslant 0}$ be a one-dimensional Ito process valued in the open subset D. If the d-period gross return on a security is just $1 + R_{t-d,t}$, then the continuously compounded return, or logarithmic return, is defined in Equation (6.11). We are now going to discuss the properties of asset returns, test for trend and mean-reversion, and briefly introduce the multifractal market.

2.1.1 Assessing the properties of asset returns

2.1.1.1 Some empirical results With the availability of new sources of financial data, the 1990s saw a surge of interest in whether or not there is long-range dependence (LRD) in asset returns. The concepts of self-similarity, scaling, fractional processes were used to assess market returns. For instance, various authors used statistical methods of inference to estimate the tail index without assuming a particular shape of the entire distribution (Jansen et al. [1991], Loretan et al. [1994], Lux [1996], Dacorogna et al. [2001]). The tail index, α, was found to be in the range of 3 to 4, giving weight to the stability of the tail behaviour under time aggregation. As a result, it was then assumed that the unconditional distribution of returns converged toward the Gaussian distribution, but was distinctly different from it at the daily (and higher) frequencies. Hence, the non-normal shape of the distribution motivated the quest for the best non-stable characterisation at intermediate levels of aggregation. While the dependency of long lasting autocorrelation was subject to debate for raw (signed) returns, it was plainly visible in absolute returns, squared returns, or any other measure of the extent of fluctuations (volatility). Still using statistical tests, other authors studied long term dependence with more or less success on finding deviations from the null hypothesis of short memory for raw asset returns, but strongly significant evidence of long memory in squared or absolute returns (see Lo [1991], Ding et al. [1993], Longin [1996]).

Eveven though it has been assumed little evidence of fractional integration in stock returns, long memory has been identified in the first differences of many economic series a long time ago. For instance, Mandelbrot [1971] argued against martingale models and Maheswaran et al. [1993] suggested potential applications in finance for processes lying outside the class of semimartingales. Econophysics developed to study the herd behaviour of financial markets via return fluctuations, leading to a better understanding of the scaling properties based on methods and approaches in scientific fields. To measure the multifractals of dynamical dissipative systems, the generalised dimension and the spectrum have effectively been used to calculate the trajectory of chaotic attractors that may be classified by the type and number of the unstable periodic orbits. Even though a time series can be tested for correlation in many different ways (see Taqqu et al. [1995]), some attempts at computing these statistical quantities emerged from the box-counting method, while others extended the R/S analysis (see Mandelbrot et al. [1979]). The moment-scaling properties of financial returns have been the object of a growing physics literature confirming that multiscaling was exhibited by many financial time series (see Vandewalle et al. [1998], Schmitt et al. [1999], Pasquini et al. [2000]). Since scaling analysis and multifractal analysis developed, various authors performed empirical analysis to identify anomalous scaling in financial data (see Calvet et al. [2002]).

Since the absence of long range dependence (LRD) in returns is still compatible with its presence in absolute returns (or volatility), several authors suggested models such as the Fractionally Integrated GARCH models, where returns have no autocorrelation but their amplitudes have LRD (see Baillie [1996]). Many continuous multifractal models, such as the multifractal model of asset returns (MMAR), have been proposed to capture the thick tails and long-memory volatility persistence exhibited in the financial time series (see Mandelbrot et al. [1997]). Such models are consistent with economic equilibrium, implying uncorrelated returns and semimartingale prices, thus precluding arbitrage in a standard two-asset economy. Returns have a finite variance, and their highest finite moment can take any value greater than 2. However, the distribution does not need to converge to a Gaussian distribution at low frequencies and never converges to a Gaussian at high frequencies, thus capturing the distributional nonlinearities observed in financial series. These multifractal models have long memory in the absolute value of returns, but the returns themselves have a white spectrum. That is, there is long memory in volatility, but absence of correlation in returns. Subsequent literature moved from the more combinatorial style of the multifractal model of asset returns (MMAR) to iterative causal models of similar design principles, such as the Markov-switching multifractal (MSM) model proposed by Calvet et al. [2004] and the multifractal random walk (MRW) introduced by Bacry et al. [2001], constituting the second generation of multifractal models.

Examining the stability of the Hurst exponent, H, on financial time series on the basis of characteristic values, such as rescaled ranges or fluctuations analysis, some authors (see Vandewalle et al. [1998c], Costa et al. [2003], Cajueiro et al. [2004], Grech [2005]) observed that the values of H could be significantly higher or lower for a specific scale. Using detrended fluctuation analysis (DFA), or detrended moving average (DMA), to analyse asset returns on different markets, various authors observed that the Hurst exponent would change over time indicating multifractal process. They found that the exponent values and the range over which the power law holds varied drastically from one underlying asset to another one, obtaining three categories, the persistent behaviour, the antipersistent behaviour, and the strictly random one. In addition, they showed in some stocks and some exchange rate that the Hurst exponent was changing with time with successive persistent and antiperisistent sequences (see Costa et al. [2003], Kim et al. [2004]). Several authors proposed to use methods of long-range analysis such as DFA or DMA to determine the local correlation degree of the series by calculating the local scaling exponent over partially overlapping subsets of the analysed series (see Vandewalle et al. [1998c], Costa et al. [2003], Cajueiro et al. [2004], Carbone et al. [2004], Matos et al. [2008]).

Later, measuring multifractality with either DFA, DMA, or wavelet analysis, and computing the local Hurst exponent on sliding windows, a large number of studies confirmed multifractality in stock market indices, commodities and FX markets, such as Matia et al. [2003], Kim et al. [2004], Matos et al. [2004], Norouzzadeh et al. [2005]. Further studies confirmed multifractality in stock market indices such as Zunino et al. [2007] [2008], Yuan et al. [2009], Wang et al. [2009], Barunik et al. [2012], Lye et al. [2012], Kristoufek et al. [2013], Niere [2013], to name but a few. Other studies confirmed multifractality on exchange rates such as Norouzzadeh et al. [2006], Wang et al. [2011b], Barunik et al. [2012], Oh et al. [2012], while some confirmed multifractality on interest rates such as Cajueiro et al. [2007], Lye et al. [2012], as well as on commodity such as Matia et al. [2003], Wang et al. [2011].

Then methods for the multifractal characterisation of nonstationary time series were developed based on the generalisation of DFA, such as the MFDFA by Kantelhardt et al. [2002]. Consequently, the multifractal properties as a measure of efficiency (or inefficiency) of financial markets were extensively studied in stock market indices, foreign exchange, commodities, traded volume and interest rates (see Matia et al. [2003], Ho et al. [2004], Moyano et al. [2006], Zunino et al. [2008], Stosic et al. [2014]). For instance, Zunino et al. [2008] used MFDFA to analyse the multifractality degree of a collection of developed and emerging stock market indices. Gu et al. [2010] analysed the return time series of the Shanghai Stock Exchange Composite (SSEC) Index with the one-dimensional MFDMA model within the time period from January 2003 to April 2008, and confirmed that the series exhibits multifractal nature, not caused by fat-tailedness of the return distribution. Lye et al. [2012] used MFDFA coupled with the rolling window approach to scrutinise the dynamics of weak form efficiency of Malaysian sectoral stock market, and showed that it was adversly affected by both Asian and global financial crises.

It was also shown that observable in the dynamics of financial markets have a richer multifractality for emerging markets than mature one. As a rule, the presence of multifractality signalises time series exhibiting a complex behaviour with long-range time correlations manifested on different intrinsic time scales. Considering an artificial multifractal process and daily records of the $S\&P$ 500 index gathered over a period of 50 years, and using multifractal detrended fluctuation analysis (MFDFA) and multifractal diffusion entropy analysis (MFDEA), Jizba et al. [2012] showed that the latter posses highly nonlinear, and long-ranged, interactions which is the manifestation of a number of interlocked driving dynamics operating at different time scales each with its own scaling function. Such a behaviour typically points to the presence of recurrent economic cycles, crises, large fluctuations (spikes or sudden jumps), and other non-linear phenomena that are out of reach of more conventional multivariate methods (see Mantegna et al. [2000]).

2.1.1.2 The difficulties of measuring empirically LRD We saw in Section (2.1.1.1) that financial systems were assumed to display scaling properties similar to those of systems in statistical physics, such that the existence of long-term correlation could be empirically assessed using

1. fractal analysis
2. statistical self-similarity analysis (using various statistical methods)

However, these statistical approaches are based on the moment properties of stochastic processes and must be restricted to second-order stationary processes. In addition, fractal properties are only verified in the infinitesimal limit. Nonetheless, a large number of statistical studies on market prices checked for the existence of long-range and short-range power law correlation in financial data. Most of these studies did not find temporal correlations present in the system for price changes, but they did for absolute price changes, the average of absolute price changes, the square root of the variance, and the interquartile range of the distribution of price changes. It is well known that it is difficult to test for a particular parametric model, such as α-stable Levy processes or fractional Brownian motions, since self-similarity can have very different origins. For instance, studies estimating the tail exponent directly from the tail observations tends to overestimate the characteristic exponent α. Also, the true tail behaviour of L-stable laws is only visible for extremely large data sets, leading to strongly misleading results when the sample is not large enough, such as the rejection of the L-stable regime (see McCulloch [1997]). Thus, we should use high-frequency data and consider the most outlying observations when estimating the tail index. In addition, if the time series of asset returns possess the two features of heavy tails and long range dependence, then many standard statistical tests will fail to work (see Resnick et al. [1999]). Also, Resnick et al. [2000] give examples of such processes where sample autocorrelations converge to random values as the sample size grows. Further, we have seen that slow decay of sample autocorrelation functions could indicate non-stationarity, making it hard to distinguish between stationary long memory models and other non-stationary models. Mikosch et al. [2003] observed that the non-stationarity of returns may generate spurious effects easily mistaken for LRD in the volatility. Granger et al. [2004] considered the interaction of LRD with non-stationarity by combining an underlying long memory process with occasional structural breaks.
More generally:

- statistical analysis applies to processes with stationary increments
- estimators are vulnerable to trends in data, periodicity, large fluctuations, etc.
- estimators of Hurst exponent works for very long, or infinite, time series
- in the time domain, at high lags only a few samples are available
- in the frequency domain, at frequencies near zero, measurement errors are largest

Yet, stock returns and FX rates suffer from systematic effects mainly due to the periodicity of human activities, and can not be considered as processes with stationary increments. Clegg [2006] showed that LRD was a very difficult property to measure in real life because the data must be measured at high lags/low frequencies where fewer readings are available, and all estimators are vulnerable to trends in the data, periodicity and other sources of corruption.
When eliminating these problems, academics found correlations present in systems and highlighted the multifractal nature of financial time series. It was shown that self-similar models were too restrictive to explain financial time series, as they were unable to capture more fully the complex dynamics of the series. While all these models assume that the response times are trial-independent random variables, financial series exhibit multifractal scaling behaviour since a multitude of scaling exponents are necessary to fully describe the series. Simply put, financial time series exhibit

- non-Gaussian distribution,
- the presence of extreme values (outliers),

- long-range dependent dynamics.

These characteristics of financial time series indicate:

1. that the variations can not be exclusively described by the scaling of the variance alone, but that the scaling of higher order statistical moments must be considered.
2. intermittent changes in the magnitude of response time variation, which might be due to feedback effects, or, changes in investor's behaviour. These changes provide temporal modulation of both the width and shape of the response time distribution, and consequently, temporal modulation of the scaling exponent.

2.1.2 Testing for trend and mean-reversion

Bloch [2014] used the effective local Holder exponents (ELHE) (see Appendix (7.3.3)) to analyse markets for multi-fractality and obtained highly random local Holder exponents, emphasising the strong multifractal nature of financial data. We present some results on the equity and the FX markets.

2.1.2.1 The equity market We consider the price of "Google US" from August 22nd, 2012 to September 5th, 2014, and apply the same techniques as above to detect trends and mean reverting behaviours on a close-to-close strategy (we trade only at the close of the market everyday).

The Figure (1) represents Google prices from August 22nd, 2014 to September 5th, 2014. We have also plotted its trend computed using a wavelet denoising method. We observe that the stock prices and the trend coincides in most of the period, except in few time intervals, where it is strongly pushed away and then pulled back. For example, the prices fluctuates around the trend just before 2013, in January 2014 and more significantly between March and May 2014. This phenomena suggest to adapt a trend-following strategy when the price process coincides with the local trend, and on the contrary, we want to apply a mean reverting strategy when the price fluctuates around the trend. To do so, we need a indicator that allows us to take a decision systematically. We observe that the local Hurst exponent coincides exactly with the time interval where we observed strong fluctuations around the trend of the price process. Indeed, just before 2013, we observe a local minimum on the local Hurst exponent corresponding exactly with the fluctuations. The same observations may be made in January 2014. More significantly, between February and May 2014, the local Hurst exponent dropped from 0.55 to 0.35, and again, this phenomena exactly corresponds to the huge fluctuations interval observed in the price process. In addition, the global minimum of the local Hurst exponent occurs exactly when the price jumped from 450 to 500 in August 2013. In that period of time, the price is indeed highly mean reverting, since the difference between the trend (blue) and the actual process (black) is high. Hence, we can infer with a high probability that the process is more likely to come back to its trend. The autocorrelation is thus highly negative and the local Hurst exponent is subsequently close to zero.

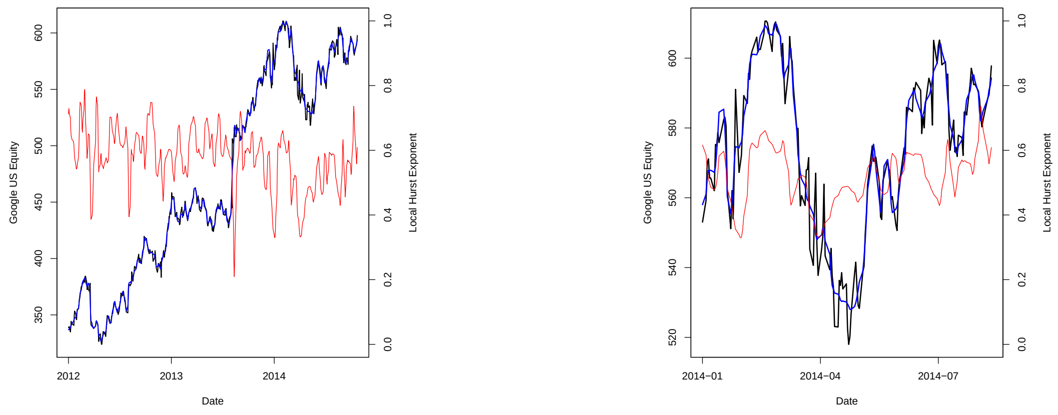


Figure 1: Google Equity Price from August 22nd, 2012 to September 5th, 2014 (black line) its trend (blue line) and its corresponding denoised Local Hurst exponent by MFDFA method (red line) and a zoom between January 2014 and August 2014

2.1.2.2 The FX market The high liquidity in the FX market allows us to define trading strategies at a high-frequency. As illustrated in the figure below, the data is very noisy on this time scale, making it highly difficult to trade directly on the currency pair. One solution could be to use wavelet analysis to extract the trend of the currency pair for different time scales and then build a strategy considering those market imperfections on each time scaling.

Figure (2) represents the EUR / GBP currency pair from August 31st to September 5th. We plotted its trend showing its moves of scale 4 (which represents 4 minutes) and its trend for the moves of scale 2 (which depicts 1 minute). While we observe that the two trends are roughly equal in most of the period, we can also see that the red line sometimes slightly fluctuates around the blue line. This is the case on the September 1st at around 10:00, the September 2nd at around 10:00, the September 3rd at around 10:00 and on the September 4th at around 14:00. In those periods, when the red line is below the blue line, the red line is most likely going to increase in the next ticks, and conversely, when the red line is above the blue line, it is most likely going to decrease in the next ticks. This phenomena exactly corresponds to a mean-reverting behaviour of the red process around its local mean, modelled by a longer trend (blue process). Further, this behaviour also coincides exactly with a drop in the local Hurst exponent. Therefore, we can propose a simple trading strategy:

1. When the local Hurst exponent is under a fixed threshold (typically 0.3 here), we compute the two trends (blue and red) of the EUR/GBP currency pair and we check the relative position of the blue line on the red line. If the red line is under the blue line, we buy Euros since its value will most likely increase. On the contrary, if the red line is above the blue line, we short sell Euros since its value will most likely decrease.

2. When the local Hurst exponent is over the fixed threshold, we buy and hold Euros if the trend is upward. On the contrary, we sell Euros if the trend is downward.

Obviously, to get an exact idea of the viability of this strategy in practice, many other factors have to be taken into account, such as taxes, transaction costs, technological means and the total traded amount, since the moves in the price are very low in this frequency (around 1/100 euros per tick).

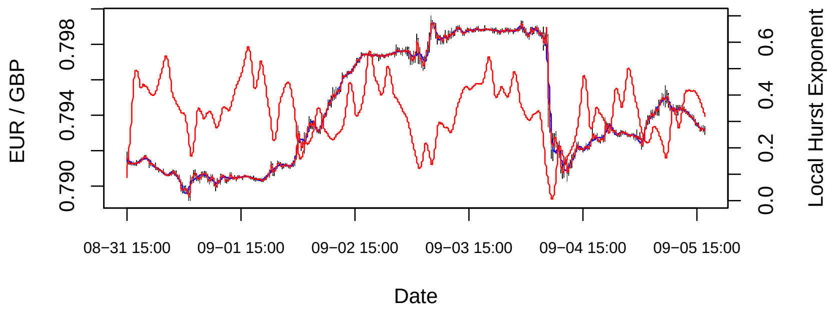


Figure 2: Tick-by-tick (20sec) EUR / GBP currency pair from August 31st, 2014 to September 5th, 2014 (grey line), its trend on different scale (red and blue lines) and its corresponding denoised Local Hurst exponent by MFDFA method (thin red line).

2.1.3 A multifractal market

We used in Section (2.1.2) the effective local Holder exponents (ELHE) (see Appendix (7.3.3)) to analyse markets for multifractality. Highly random local Holder exponents were obtained, emphasising the strong multifractal nature of financial data. Even the denoised effective exponents obtained with wavelet transform are random with abrupt change in values happening continuously, and extremely fast. That is, the denoised local Holder exponents oscillate around a level of Hurst exponent with a succession of small and large amplitudes (similar to spikes), and is capable of sudden, or abrupt, change to a different level of Hurst exponent in presence of large extreme price fluctuations related to market crashes. Various authors already suggested that financial markets could be described as a system of a number of coupled oscillators, subject to stochastic regulation (see Matia we al. [2003]). The source of multifractality of financial markets were shown to be due to nonlinear phase ordering as well as the essential contribution of extreme events resulting in fat tails of the event distribution. Even though the width of the multifractal spectra is capable of indicating the presence of large shocks, the oscillating nature of the local Holder exponent characterise the continuously changing dynamics of the response time distribution. Comparing the behaviour of the stock market to that of a healthy heartbeat dominated by two antagonistic systems (sympathetic and parasympathetic) the pessimists who sell stocks and the optimists who buy stocks, leading to a continuously varying number of active agents (degrees of freedom), Struzik [2003] showed that both systems were at work at any moment in time. Hence, the multifractal spectra can detect a change of level of the Hurst exponent (contribution to fat tails), but it can not be used to identify the intermittent changes in the magnitude of response time variation due to, for example, feedback effects. Similarly to outliers, the latter requires a methodology capable of determining the statistical nature of the non-stationary process both globally and locally, such as the effective local Holder exponent.

2.2 Neural networks

2.2.1 An overview

Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. The former is presented in Appendix (11.1.2) and the latter is introduced in Appendix (11.2). ANNs, formalised by McCulloch et al. [1943], are a class of generalised nonlinear nonparametric models inspired by studies of the human brain. They get their intelligence from learning process, giving them the capability of auto-adaptability, association, and memory to perform certain tasks. The backpropagation network, which is the most popular and the most widely implemented neural network in the financial industry, is based on a multi-layered feedforward topology with supervised learning (see Refenes et al. [1997]). The network is fully connected, with every node in the lower layer linked to every node in the next higher layer via weight values. The learning of the backpropagation neural network is based on an error minimisation procedure where the weights are modified according to an error function comparing the neural network output with the training targets. We evaluate the derivatives of the error function with respect to the weights which are used to compute the adjustment to be made to the weights. The simplest such techniques involves gradient descent. In the case of time series prediction, a set of input-target pairs is created, forming the training samples. Every time a new time series is generated, new observations are added to the set and the oldest ones are dropped out. Alternatively, artificial recurrent neural networks (RNNs) are a class of feedback artificial neural network architecture that uses iterative loops to store information, which is inspired by the cyclical connectivity of neurons in the brain. The existence of cycles allows RNNs to develop a self-sustained temporal activation dynamics along its recurrent connection pathways, even in the absence of input, making them a dynamical system. This feature of the RNNs make them attractive for processing serially correlated time series. Unlike the ANNs and traditional time series models, the RNNs are flexible enough, and avoids taking the size of sliding windows into account because they can decide what to store and what to ignore during the learning process.

In real-time data, training neural networks involves the optimisation of non-convex objective functions, which makes the computation of the gradient descent challenging and in turn leads to a costly or even infeasible learning process. One solution is to randomly assign a subset of the networks' weights, so that the resulting optimisation task can be formulated as a linear least-squares problem. Numerous experimental results indicate that such randomised models can match, or even improve, adaptable ones, with a number of favourable benefits, such as, simplicity of implementation, faster learning with less intervention from human beings, possibility of leveraging over all linear regression and classification algorithms. Such improvements in standard RNNs design were proposed independently by Maass et al. [2002] under the name of Liquid State Machines and Jaeger [2001] under the name of Echo State Networks (ESN). Over time, these types of models became known as Reservoir Computing (RC). RC assumes that supervised adaptation of all inter-connection weights are not necessary, and only training a memoryless supervised readout from it is sufficient to obtain good results. Thus, it avoids the shortcomings of the gradient-descent training of RNNs. RC is based on the computational separation between a dynamic reservoir and a recurrence-free readout (see details in Appendix (11.2.3)).

2.2.2 Modelling changing environment

2.2.2.1 Multiple characteristic timescales In general, the dynamics of RNN models evolve according to precise timing, while real-world time series are characterised by multiple seasonalities (or patterns). Substantial literature exists on modelling different temporal resolutions (multiple characteristic timescales), especially in presence of long-term relations, among which the Long Short Term Memory (LSTM) became a reference (see Hochreiter et al. [1997]). Several variants have been proposed to improve the simultaneous learning of long and short time relationships (see Cho et al. [2014]). Other methods developed based on the idea that temporal relationships are hierarchically structured, so that RNNs should be organised accordingly (see El Hihi et al. [1995]). It led some models to be implemented by stacking multiple recurrent layers (see Graves [2013], Chung et al. [2015]). In Reservoir Computing, characterised by fading memory preventing to model slow periodicities, solutions were proposed to slow down the dynamics of the reservoir by considering digital bandpass filters. The latter encourages the decoupling of the dynamics within a single reservoir by identifying particular frequency-domain characteristics. A large amount of filters are initialised with random cutoff frequencies to encode a wide range of timescales in the recurrent layer, providing the correct timings to capture the target dynamics. Alternatively, the filters can be manually tuned based on the information of the frequency spectrum of the desired response signal (see Siewert et al. [2007], Wyffels et al. [2008]).

An important development in machine learning was the use of ensemble models, which simultaneously combine different types of classifiers. The idea being that classifiers of different types and/or using different data exhibit distinct strengths and weaknesses. As a result, it has been assumed that combining multiple classifiers could improve prediction accuracy (see Ho et al. [1994]). Jaeger [2004] discussed the fact that a single reservoir could not train

a multiple superimposed oscillator (MSO). We know from classic kernel-based signal representation (Fourier and wavelet transform) that an individual kernel can represent an almost unique portion of the target signal. Further, a single reservoir can reproduce the behaviour of a sine-wave generator (see Jaeger [2001]). We can therefore construct an ESN with multiple reservoirs, each of which having a different configuration, to generate multiple sine waves of different frequencies. The problem being to define the grouping of neurons into several separate reservoir in a non-redundant manner.
Wiestra et al. [2005] explained that all the neurons in the same reservoir are coupled, when the task requires the simultaneous existence of multiple decoupled internal states. One solution was to consider an evolutionary-based reservoir design method, called Evolino. Xu et al. [2007] proposed the decoupled echo state network (DESN) involving the use of lateral inhibition. It is a structure making use of multiple reservoirs competing with each other through inhibitory connections and combining the internal states of all the reservoirs to form the output signal. Bianchi et al. [2017] proposed a network made of several recurrent groups of neurons trained to separately adapt to each timescale of the input signal. The recurrent layer of the network is randomly generated, but the connections must form a structure with K groups of bandpass neurons where each group $\mathcal{C}_k$ contains N neurons strongly connected to the other neurons in $\mathcal{C}_k$. Further, a small amount of connections between neurons belonging to different groups are also generated. The bandpass parameters are learned via backpropagation.

In most of these methods a large number of parameters must be learned in order to adapt the network to process the right time scales. It results in a long training time with the possibility of overfitting data. In general, these parameters are retrieved through an analysis in the frequency domain or by a priori knowledge of the input data.

2.2.2.2 Long range dependence In general, memory or persistent internal states can not be modelled by fading memory systems (see Definition (11.1)). However, the latter can be enlarged through feedback from trained readouts. Maass et al. [2005a] [2005b] presented a computational theory characterising the gain in computational power achieved through feedback in dynamical systems with fading memory. As a result, anti-persistent systems using feedback can obtain universal computational capabilities similar to those of persistent systems. It leads to high-dimensional attractor-based models capable of capturing the characteristics of persistent signal response.

In the field of autonomous robot locomotion, stable pattern generation and robustness against perturbations is necessary. For instance, learning rhythmic stable motions in a controllable way is usually modelled with central patterns generators (CPG). In this framework, the output feedback can generate a suitable slow dynamics with high precision. As a result, RC models with output feedback were proposed to generate CPGs with interesting properties. In order to properly feedback the forecast and to modify appropriately the output weights if needed, one needs to implement an online algorithm. That is, when the reseroir is trained online, the model is adapted to recognising patterns and predicting chaotic time series (see Wyffels et al. [2009], Antonik et al. [2016]).

Reinhart et al. [2008] [2009] provided a new approach for training offline, or online, algorithms based on the transient/attractor concepts called Associative Reservoir Computing (ARC). As an example, an associative connection between two entities is bidirectional. The process denotes the recall of one entity from the other. Further, Reinhart [2011] introduced the concept of forward and inverse models in RC models. If the forward relation is many-to-one, the reverse relation becomes ambiguous, that is, one-to-many. Thus, there is a loss of information. One solution is to use additional information that is integrated over time. It can be modelled with an adaptive dynamical system to facilitate bidirectional and continuous association. The author pointed out the necessity of having a feedback in bidirectional association context. The main idea being to solve the ambiguous inverse problems by letting the model remain in an attractor network by teaching the reservoir states to stay stable. At every time step, the algorithm makes the entire network (input, reservoir and output) converge, thus eliminating transient effects. It is very important when one considers an output feedback matrix as it might add transient effect. Since feedback of erroneous outputs into the network can lead to error amplification, the concept of output feedback dynamics is formalised and an output feedback stability criterion is proposed.

2.2.2.3 The variable weight neural network In conventional neural networks, connection weights are fixed. That is, the characteristics of the neural network is invariant (the invariant type neural network). The adaptive ability of neural networks to an unknown environment depends on its generalisation capability. However, there exists a limit of generalisation ability in invariant type neural networks. Yasuda et al. [2006] proposed the neural network with variable connection weights, which changes its connection weights and its characteristic according to the changing environment. Analysing an obstacle avoidance problem for an electric-powered wheelchair, they split the weights in two terms where the first term are basic quantities giving the basic motion of obstacle avoidance and the second terms adjust the connection weights in order to adapt the neural network to the change of environment in the vicinity of the wheelchair. The latter are calculated by using another neural network, whose inputs are outputs of PSD sensors. As a result, the law producing avoidance orders, changes according to the current condition of the obstacle in the vicinity of the wheelchair. Thus, this mechanism adapts the characteristics of neural network to the changing environment at every moment and corrects the generation algorithm of obstacle avoidance operation appropriately.
Lam et al. [2014] presented the variable weight neural network (VWNN), allowing its weights to be changed in operation according to the characteristic of the network inputs. Analysing the problem of surface material recognition and epilepsy seizure phases recognition, they demonstrated its ability to adapt to different characteristics of input data resulting in better performance than with fixed weights neural networks. A VWNN consists of two traditional neural networks, namely tuned and tuning neural networks. The former is the one which actually classifies the input data, while the latter provides the weights to the tuned neural network according to the characteristic of the input data. The VWNN works on the principle that the connection weights are function to the external input signal. Thus, in theory, the VWNN can be viewed as an infinite number of traditional neural networks with fixed weights.
A description is given in Appendix (11.3.2) and an application to financial time series is presented.

2.2.3 Interpreting neural networks

Neural networks (NNs) are based on predefined equations or formulae (see Appendix (11.1)) and have the capability of modelling complex statistical interactions between features for automatic feature learning. In order to perform a task, data scientists create an architecture (topology) whose (meta)-parameters are able (or not) to provide correct answers when some new input data is presented. These parameters are the key to their knowledge (see Palmer et al. [2002]), which depends on their generalisation ability.
Given this singular way of learning, NNs have been treated as black box models for their lack of interpretability compared with classical statistical models. That is, their inability to know in an explicit way the relations established between explanatory variables (input) and dependent variables (output).
Since the recent applications of NNs in the industry are intended to make critical decisions, it has become paramount to understand how these models make predictions. To do so, academics focused on explaining individual feature importance and determining the contribution of explanatory variables. One need first to discover (detect) the interactions and then interpret them, which is not an easy task. The methods used for detecting interactions from NNs are usually statistical methods. In general, they rely on the assumption that the response signal is stationary with independent and identically distributed increments. Further, these tests rely on the idea that NNs are capable of successfully modelling complex statistical interactions.

2.2.3.1 Explaining the input feature importance Several approaches have been proposed to interpret NNs, classified as follows:

1. direct interpretation: explaining individual feature importance (Ross et al. [2017]), developing attention-based models, which illustrate where neural networks focus during inference, providing model-specific visualisations, such as feature map and gate activation visualisations (see Yosinski et al. [2015]).
2. indirect interpretation: post-hoc interpretation of feature importance (see Ribeiro et al. [2016]), knowledge distillation to simpler interpretable models.

The discovery of interactions (or statistical interaction detection) has become an essential part of the literature on neural networks. Two general approaches exist:

1. to conduct individual tests for each combination of features. Examples are ANOVA and Additive Groves (see Sorokina et al. [2008]). Both methods are computationally expensive in dimension higher than three.
2. to pre-specify all interaction forms of interest, then use Lasso to simultaneously select which are important. They are quick at selecting interactions but require specifying all interaction terms of interest (see Sun [1999]).

Interpreting NNs is far more difficult, but new methods have recently developed for traditional feedforward form and deep architectures. Methods such as feature map visualisation, de-convolution, saliency maps have been especially important to the vision community for understanding how convolutional networks represent images. In the case of long short-term memory networks (LSTMs), a research direction has studied multiplicative interactions in the unique gating equations of LSTMs to extract relationships between variables across a sequence (see Arras et al. [2017], Murdoch et al. [2018]). Tsang et al. [2018] proposed the Neural Interaction Detection (NID), which detects statistical interactions of any order or form captured by a feed-forward neural network, by examining its weight matrices. Top-K true interactions are determined from interaction rankings by using a special form of generalised additive model, which accounts for interactions of variable order.

2.2.3.2 Determining the contribution of explanatory variables Several methods have been proposed to assess the relative importance (contribution) of each explanatory variable: methods from the family of sensitivity analyses (SA) (perturb method, profile method) (see Cortez et al. [2013]), numeric sensitivity analyses (NSA) computing the slope between input and output (partial derivative method (PaD)) and even the curvature effects (second order derivatives), and methods specific to the topology of NNs (connection weights method). In general, an optimal NN architecture is selected and several variable contribution methods are applied. However, when applying several methods to optimal or suboptimal NN architecture, the importance ranking of variables differs from method to method (high variability), indicating their inherent instability. As a result, there is no consensus on which model is the best for determining the contribution of variables. The difficulty of choosing a single optimal NN model led some authors to consider a set of good-performing NN models called neural network committee (NNC) (see Cao et al. [2008]). To mitigate these variabilities, De Ona et al. [2014] suggested to apply the methods on a set of NNs with the same architecture and trained with identical learning algorithm, activation functions, momentum value and learning ratio. The calculation of the average values of the relative importance of variables was used to determine the contribution of variables. The variables Frequency, Speed, Punctuality and Proximity are classified as the most important by all methods.

2.3 Optimisation for machine learning

In machine learning, given the samples (x_i, y_i), $i = 1, 2, ..., m$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and output vectors $y_i \in \mathbb{R}^{N_o}$, a task consist in learning a functional relation between the input x_i and the desired output y_i, for the training set (see Appendix (11.1.1)).
The learning can be understood as finding a mapping $h(\cdot, w)$ such that $h(x_i, w) \approx y_i$ with w an N_w dimensional vector of parameters to be learned. Thus, the goal of machine learning is to minimise the expected loss (also called risk function)

$$\hat{L}(h) = E[L(h(X), Y)]$$

However, we do not know $P(X, Y)$ and can not estimate it. One solution is to consider Empirical Risk minimisation. In general, the relation between the input x_i and the desired output y_i can either be solved by a linear model as

$$\hat{y}_i = W^\top x_i$$

where $W \in \mathbb{R}^{N_i \times N_o}$ is a weight matrix, or by a nonlinear model

$$\hat{y}_i = f(W^\top x_i)$$

where $f(\cdot)$ is the activation function.
The training of neural networks (NNs) depends on the adaptation of free network parameters, the weight values. Thus, it is an optimisation task where the result is to find optimal weight set of the network that reduce an error function E. It can be formulated as the minimisation of an error function in the space of connection weights (see Appendix (13)). In the case of our loss function the minimisation problem is

$$\min_W \hat{L}(h, W)$$

In practice, the function f in the above minimisation problem is not given explicitly but only implicitly through some examples. While convex problems are theoretically and algorithmically much more tractable, the complexity of non-convex problems can grow enormously (NP-hard). Global optimisation addresses the problems of non-convex optimisation.

The methods for training neural networks (NNs) are backpropagation (BP), Levenberg-Marquadt (LM), Quasi-Newton (QN). In general, the error backpropagation method (EBP), based on gradient method, is preferred. However, the objective function describing the neural networks training problem is a multi-modal function, so that the algorithms based on gradient methods can easily be stuck in local extremes. To avoid this problem one can consider using a global search optimisation (see Appendix (15)).

Over time, the relationship between mathematical programming models (MP) and machine learning models have been increasingly coupled. Methods developed for introducing constraints into the learning model in view of adding domain knowledge to enforce non-negativity and sparsity in dimensionality reduction methods. Further, methods were developed for solving existing machine learning models more efficiently. As data set size grows, MP models became inadequate and methods were developed to exploit the properties of learning problems in machine learning. In turn, machine learning has motivated advances in mathematical programming: the optimisation problems arising from large scale machine learning and data mining far exceed the size of the problem typically reported in the mathematical programming literature.

Generally, an ANN model is specified by its topology, node characteristics and the training algorithms. There are a variety of ANNs with various topologies, node properties and training algorithms. See for instance Appendix (11.2.3) in the case of Reservoir Computing. These characteristics (or features) must be considered when designing a neural network model, leading to large families of models. The data scientist must select an appropriate family of models and massages the data into a format amenable to modelling. Then the model is typically trained (Training) by solving a core optimisation problem that optimises the variables (or parameters) of the model with respect to the selected loss function and possibly some regularisation function. The efficiency of an optimisation method often depends on the choosing of a number of behavioural parameters (meta parameters). In that case, the mapping becomes

$$h(x_i, w, \theta) \approx y_i$$

with θ an N_θ dimensional vector of parameters to be learned. Hence, in the process of model selection and validation (), the core optimisation problem may be solved many times. It led researchers to combine mathematical programming with machine learning. The interaction of state-of-the-art machine learning and mathematical programming has been called Large Scale Optimization and Machine Learning leading to improved machine learning models as well as new uses of mathematical programming in machine learning.

One solution consists in performing an additional optimisation aiming at minimising the RMSE on these meta parameters Θ. It can be done on the validation sample. In that case, the minimisation problem is

$$\min_{\Theta} \widehat{L}(h, W^*, \Theta)$$

Pedersen et al. [2008b] proposed an overlaying optimisation method known as meta-optimisation. The idea is to have an optimisation method act as an overlaying meta-optimiser, trying to find the best performing behavioural parameters for another optimisation method (see Pedersen et al. [2008a]).

2.4 The learning process

2.4.1 Prediction tasks

The strength of neural networks (NNs) compared to other techniques is their high capacity for classification, prediction and failure tolerance. In financial markets, NNs are generally applied for prediction tasks and for classification (see Appendix (9)). It can be summarised as follows:

1+ Chaotic series prediction: performing time series forecast.
In the academic literature, chaotic systems are generally described by Mackey-Glass equations (see Mackey et al. [1977]). It is generated from the following non-linear time delay differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^n(t-\tau)} - bx(t)$$

where a, b, n and τ are real numbers. τ is the delay parameter. Depending on the parameter set we choose, the Mackey-Glass signal exhibits various characteristics. Figure (3) below demonstrates a particular behaviour, which is at the same time very chaotic. Similarly to a financial time series we observe: an identified behaviour (trending or mean-reverting) together with an unidentified one (chaos). Examples are next period price change and next period actual price (opening, high, low and closing prices), which concerns the change in price with respect to the next period and the numerical value for the actual next period price, respectively.

2+ Autonomous pattern generation: classification problem.
A pattern is a short sequence of randomly chosen real numbers that is repeated periodically to form an infinite time series. In general, the length of the sequence may be set from 1 up to the number of neurons N. In finance patterns are distinctive formations of market prices (see Definition (8.1)). For example, the next period direction concerns the direction of the price movement with respect to the next period. In that case we classify each record as Up or Down. The results are generally obtained by using only historic prices and technical indicators (TIs), which are mathematical transformations of market returns (see Appendix (8)). The most common TIs are the simple moving average (SMA), exponential moving average (EMA), relative strength index (RSI), rate of change (ROC), moving average convergence / divergence (MACD), William's oscillator and average true range (ATR).

In general, academics and practitioners focus on case $(1+)$, which implies making a statement about the true dynamics (denoted f) followed by market returns. While in case $(2+)$ we only consider part of the probability space, this approach seems ignored by academics. From Computational Learning, an approximation model (the hypothesis, denoted h) must be devised and trained on the full measurement space (case $(1+)$) or part of the space (case $(2+)$). In order to assess if the function h is approximately correct, a measure of error must be considered (the cost function). This process requires sophisticated search and examination methods (optimisation algorithms) such as machine learning.

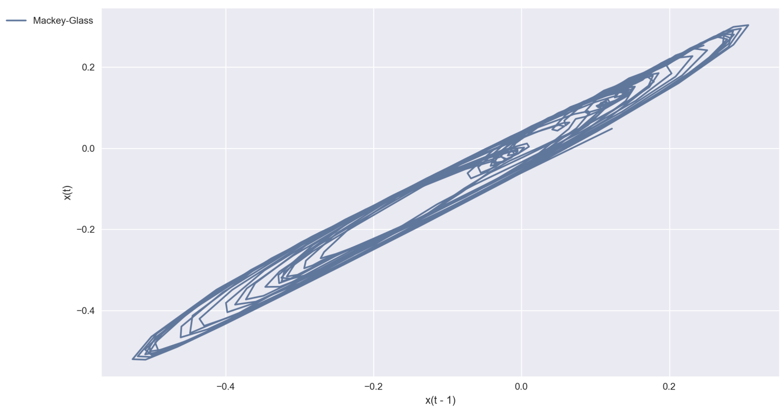


Figure 3: Mackey-Glass Behaviour for (a, b, n, τ, x_0) = (0.2, 0.01, 10, 17, $random$)

2.4.2 Training and exploitation of the model

When forecasting time series or, generating autonomous patterns, we consider the following three steps:

1. Training: it consists in determining the model parameters that minimise the cost function.
2. Optimisation, and
3. Adaptability.

We split the available data into 3 samples: Training, Validation and Testing. It is called sample hashing (see an example in Figure (4)). Optimisation is performed in the validation set. Testing is the sample where the calibrated model is assessed for generalisation. That is, the model is ran without any exhaustive filter nor complementary optimisation. The learning process is as follows:

- In the case of pattern generation: During the training phase, the reservoir computer receives the pattern signal as input and is trained to predict the next value of the pattern from the current one. The length of the training input sequence is measured in terms of the pattern length, that is, how many times the whole pattern is presented to the reservoir computer. The duration of the training process depends on the length of the pattern. For instance, short patterns are successfully learnt after 100 repetitions, while long patterns need to be shown up to $50k$ times. The RMSE is used to evaluate the training phase. After the training phase, the reservoir input is switched from the training sequence to the reservoir output signal, and the system is left running autonomously.
- In the case of series prediction, training and test phases are the same as for pattern generation. During training, the reservoir receives the time series and is trained to perform a one-step ahead prediction, while in the test phase the reservoir receives its own output and the system runs autonomously.

Based on the classification task chosen, several possibilities exist for training and exploiting models:

1* training on transient process $\rightarrow$ exploitation on transient process

2* training on attractor-based model $\rightarrow$ exploitation on attractor-based

3* training on attractor-based model $\rightarrow$ exploitation on transient process

4* training on attractor-based model → exploitation on transient process

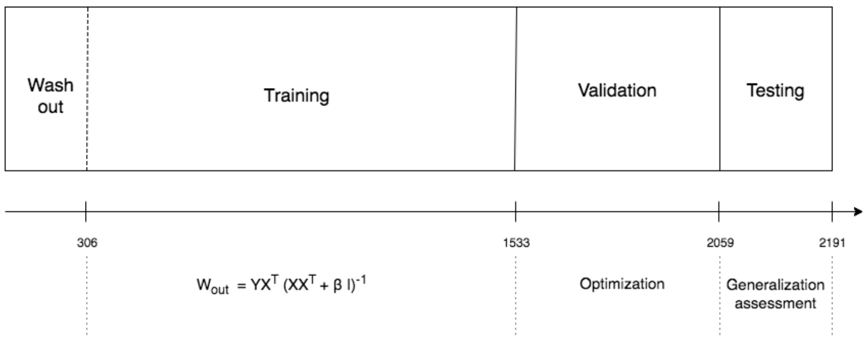


Figure 4: Sample hashing: Training sample is $washed - out$ at 20% meaning that the Ridge regression is only performed on the 80% remaining. By doing this, we get rid of the reservoir initial transient.

2.4.3 The framework

2.4.3.1 The input variables When selecting the right input variables, Krollner et al. [2010] showed that academics and practitioners rely on some form on lagged index data (case (1+) in Section (2.4)) rather than considering autonomous pattern generation (case (2+)). Examples of articles that focus on forecasting market returns are Zeng et al. [2006], Zhang et al. [2007], Zhu et al. [2008], Lu et al. [2009], Liao et al. [2010]. Based on some well known techniques combining forecasts (see Armstrong [2001]), several studies showed that combining forecasts (using NN models) could greatly enhance the forecast effectiveness (see Wang et al. [2008], Chitra et al. [2010]).
Alternatively, traders and academics have used technical analysis (see Appendix (8.2)) as well as certain fundamental valuation metrics to improve results (see Gencay [1997], Metghalchi et al. [2007]). Further, some statisticians showed that technical indicators have predictive power and found a good measure of support for their effectiveness (see Chen et al. [2009], Metghalchi et al. [2012] [2015]). As a result, these strategies have been incorporated in machine learning models to forecast future trend direction (case (2+)). The results obtained are achieved using only historic prices and technical indicators (see Kim rt al. [2006], Bekiros et al. [2008], Kara et al. [2011], Dingli et al. [2017]). Some authors also used technical indicators to forecast both future price levels and asset price directions (see Patel et al. [2015]). Note, one can directly use continuous technical indicators (actual time series) or one can choose to represent them as trend deterministic data (discrete in nature). Van den Poel et al. [2016] combined in some way continuous technical indicators (see Appendix (8.2)), technical trading rules (see Appendix (8.4.1)) and past returns to become input to predictor models. They proposed a predictive model to predict future stock price direction accurately.

2.4.3.2 The data The different forecasting outputs are computed for the following periods: daily, weekly, monthly, quarterly and yearly. However, an increasing number of articles consider intra-day high frequency data, which requires highly liquid markets. In that case the data comprises the date and time, open-high-low-close price levels and the volume traded for each underlying asset. One of the main reason for considering intraday trading horizon is the notion of self-destruction of predictable patterns in stock prices (see Timmermann et al. [2004]). For the data to be consistent, historical prices and/or volume are usually adjusted to reflect spin-offs, stock splits/consolidations, stock dividend/bonus and right offerings/entitlement.

2.4.3.3 Evaluation methods Most evaluation methods of the ML models are benchmark models where the proposed improved version is compared against the original one using a forecast error as an evaluation metric. Examples of benchmark models are buy and hold, random walk, statistical techniques and other machine learning techniques.

2.4.3.4 The measures Linear measures are measurements in one direction. For example length, width, or height of objects, and distance between two points are linear measures. Linearity also also express the idea that the model possesses the property of additivity and homogeneity, meaning that a change in one variable causes a proportional change in another variable. An example of a linear measure is the root-mean-square deviation (RMSD) or root-mean-square error (RMSE), which is used to measure the differences between values predicted by a model or an estimator and the values observed.
In mathematics and science, a nonlinear system is a system in which the change of the output is not proportional to the change of the input. Systems can be defined as nonlinear, regardless of whether known linear functions appear in the equations. Nonlinearity measures are a means of quantifying the size of nonlinearity in the Input/Output-behaviour of nonlinear systems (more than one dimension). Methods that efficiently compute nonlinearity measures are based on convex optimisation.
Note, linear measures do not account for the non-linearity of RNN models (see Bellgard et al. [1999]). One way around is to assess the performance of a model using trading simulation.

2.5 Using neural networks to model returns

2.5.1 An overview

Among the various techniques to classifying and forecasting financial time series, fundamental and technical analysis are the most popular ones (see Appendix (8)). Even though statistical procedures are widely used for patterns recognition, the effectiveness of these methods relies both on model's assumptions and prior knowledge on data properties. To remedy these pitfalls, several classifiers developed, using various data mining and computational intelligence methods such as rule induction, fuzzy rule induction, decision trees, neural networks etc. In 1990, Kimoto et al. [1990] applied a modular neural network machine learning algorithm to predict the movement of stock index of Tokyo Stock Exchange. Since ANN developed further, it has been widely applied to stock analysis. Nowadays, the best recognised tools in the currency markets is the artificial neural networks (ANNs), supported by numerous empirical studies (see Ahmed et al. [2010]). Similarly, Krollner et al. [2010] provided a survey on forecasting financial time series with machine learning and found that ANNs were the dominant technique in that field. The foremost reason for using ANNs is that there is some nonlinear aspect to the forecasting problem under consideration, taking the form of a complex nonlinear relationship between the independent and dependent variables. The characteristics of financial time series, such as equity stock or currency markets, are influenced by the psychology of traders (behavioural finance) and are strongly non-linear and hardly predictable (see Maknickiene et al. [2011]).

One advantage in using an ANNs is that the researcher does not need to know a priori the type of functional relationship existing between the independent and dependent variables (see Darbellay et al. [2000]). A vast literature demonstrated that neural network performs better than conventional statistic approaches in financial forecasting (see Refenes et al. [1994], Adya et al. [1998], Abu-Mostafa et al. [2001]). For many financial forecasting problems, classification models work better than point prediction. Further, Qi [2001] argued that due to the continually changing nature of financial relationships, ANNs are more likely to outperform traditional techniques when the input data is kept as current as possible. This is done by recursive modeling, where the researcher adds new observations and drops the oldest ones each time a new time series forecast is made (sliding window). Olson et al. [2003] compared neural network forecasts of one-year ahead Canadian stock returns with the forecasts obtained by using ordinary least squares (OLS) and logistic regression techniques and showed that the backpropagation algorithm outperformed the best regression alternatives for both point estimation and in classifying firms expected to have either high or low returns This superiority of the NNs translated into greater profitability using various trading rules. Using data from four major stock market indexes, Fok et al. [2008] compared linear regression and neural network backpropagation by testing their forecasting performances. They showed that the latter had better prediction accuracy.

2.5.2 Combining neural networks with evolutionary algorithms

Some of the disadvantages of NNs are long training time, unwanted convergence to local instead of global optimal solution, and large number of parameters (see Lee et al. [1991]). One way forward is to combine ANN with another algorithm such as evolutionary computing tools that can take care of a specific problem (see Appendix (15)). Many research papers have appeared in the literature using evolutionary computing tools, such as genetic algorithm (GA) (see Montana et al. [1989], Nair et al. [2011], Contras et al. [2016]), particle swarm optimization (PSO) (see Kennedy et al. [1995] Jordehi et al. [2013]), bacterial foraging optimization (BFO) (see Jhankal et al. [2011]), and Adaptive bacterial foraging optimization (ABFO) in developing forecasting models.
Inspired by the pattern exhibited by bacterial foraging behaviour, Helstrom et al. [1998] proposed an evolutionary computation technique called Bacterial foraging optimization (BFO). It analyses how the run-length unit parameter of BFO controls the exploration of the whole search space and the exploitation of the promising areas. Shin et al. [1998] proposed data mining approach using genetic algorithms (GA) to solve the knowledge acquisition problems that are inherent in constructing and maintaining rule-based applications for stock market. Kim et al. [2000] used GA to improve the learning algorithm, and also to reduce the complexity of the feature space. Kim et al. [2004b] developed a feature transformation method using genetic algorithms. It reduces the dimensionality of the feature space and removes irrelevant factors involved in stock price prediction. Jamous et al. [2016] considered particle swarm optimization with center mass (PSOCOM) technique to develop an efficient forecasting model. Rani et al. [2017] presented an overview on optimisation techniques and hybridisation to improve ANN performance (see details in Appendix (15)).

3 Quantitative trading

In order to gain some insight on the capability of neural networks to reproduce the dynamics of financial time series, we need to test some of the properties of recurrent neural networks introduced in Section (2.5) as well as some of their characteristic for modelling changing environment presented in Section (2.2.2). Following Section (2.4), we will test neural networks on the Mackey-Glass signal and several real-time financial time series.
Among the different ML models presented in Section (2.2), we are going to concentrate on random projection networks (RPN) with non-linear spatio-temporal projections generating transient and attractor dynamics. We will focus on the Associative Reservoir Computing (ARC) (see Reinhart et al. [2008] [2009b]) as it unifies the RPN models. The detailed implementation of the model is given in Appendix (12.1). The model will be tested both with offline and online algorithms. We will also test a method for teaching the reservoir how to remain consistently stable while feeding it with noisy input.
We will then describe an algorithm to forecast the directions of the underlying asset, which we will enlarge to forecast both market returns and their directions. To conclude, we will draw conclusions on the results and define a set of rules we want to follow when building our trading algorithm.

3.1 Forecasting returns

Using a particular impementation of the ARC (see Appendix (12.1)), we reproduce below some results on the Mackey-Glass signal and FX time series obtained by a student of ours who did his MSc thesis at QFL (see Bouizi [2017]).

3.1.1 Using Offline algorithms

3.1.1.1 The Mackey-Glass series In the academic literature, it is found that pattern generation can be easily implemented and produces very good results. However, the Mackey-Glass series prediction task is more complex and requires a large reservoir and more elaborate training algorithm (see Wyffels et al. [2009], Antonik et al. [2016]).

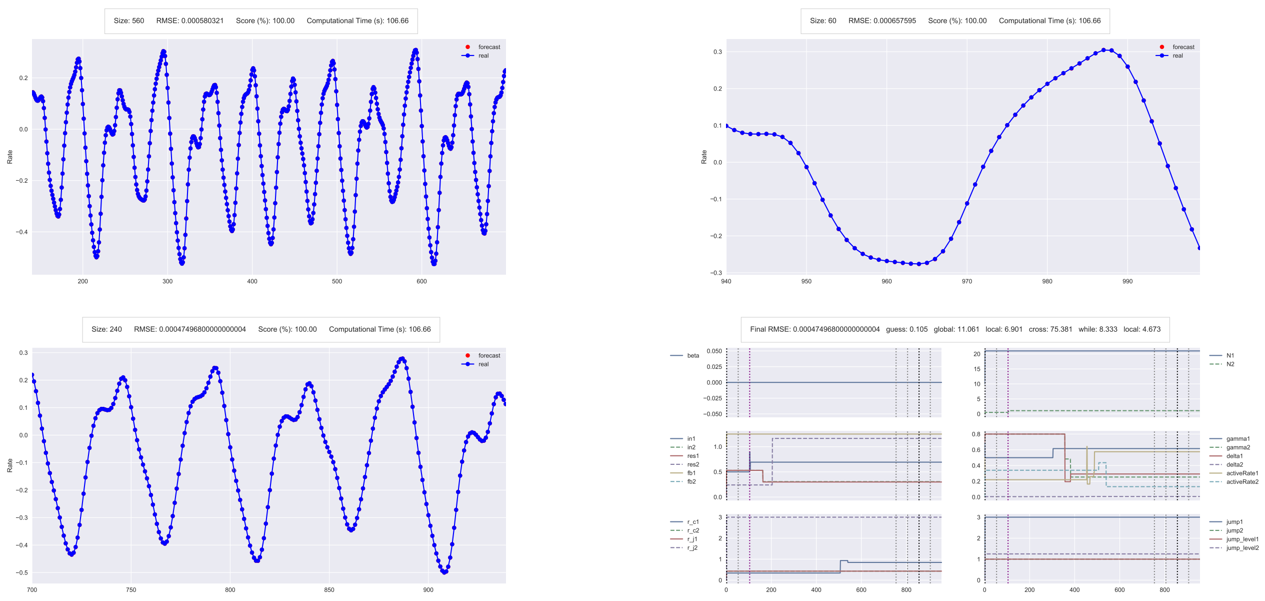


Figure 5: Mackey-Glass time series forecasting. On the left hand side from top to bottom: Training and Validation samples. On the right hand side from top to bottom: Testing and parameters optimisation through the GEN optimisation process. A black dotted vertical line indicates the beginning of a global or $while$ loop optimisation. A gray one corresponds to a local one and a purple one indicates a cross optimisation.

Method	RMSE	Ranking
Neural tree (see Chen et al. [2004])	0.0069	5
PNN (see Tan [1995])	0.0059	4
NN + Wavelet Denoising (see Tan [2009])	0.0028	3
NN + Wavelet Packet Denoising (see Tan [2009])	0.0024	2
ESN offline	$\underline{0.00065}$	1

Table 1: Comparison of our results with other research papers on the testing sample.

From Table (1) above, we can conclude that our method is very accurate at forecasting the Mackey-Glass time series and clearly outperforms classical methods. It is important to note that GEN optimisation method (see Appendix (12.1.3)) has reduced the RMSE from 0.00545518 to 0.000474968 on the validation sample. One of the reason for getting better results is that we optimise a large number of parameters in a proper manner without overfitting the data and in a relatively short period of time (approximately 00:01:36).

***Remark* 3.1** *It is not surprising to get good accuracy as this signal exhibits clear information without any noise. In this experiment, we only assess the ability of our neural network to incorporate its behaviour and to reproduce it step by step as redundant information comes in.*

3.1.1.2 Financial time series Following the same approach on financial time series, we reproduce some results on the FX markets. We perform a GEN optimisation (see Appendix (12.1.3)) with offline training using the following parameters: $(globalMax, crossMax, localMax, whileMax)$ = (20, 20, 20, 0). Considering Remark (12.2), we choose the following setting $whileMax = 0$. We study the specific FX pair: EURGBP from 2010-01-01 to 2017-05-01. From Figure (6) below, we observe good results on the training and validation samples with a relatively low RMSE and a good score (almost 60%). However, Figure (6) suggests that testing is not satisfying, with a high RMSE at 0.0440 and a score below 50%.

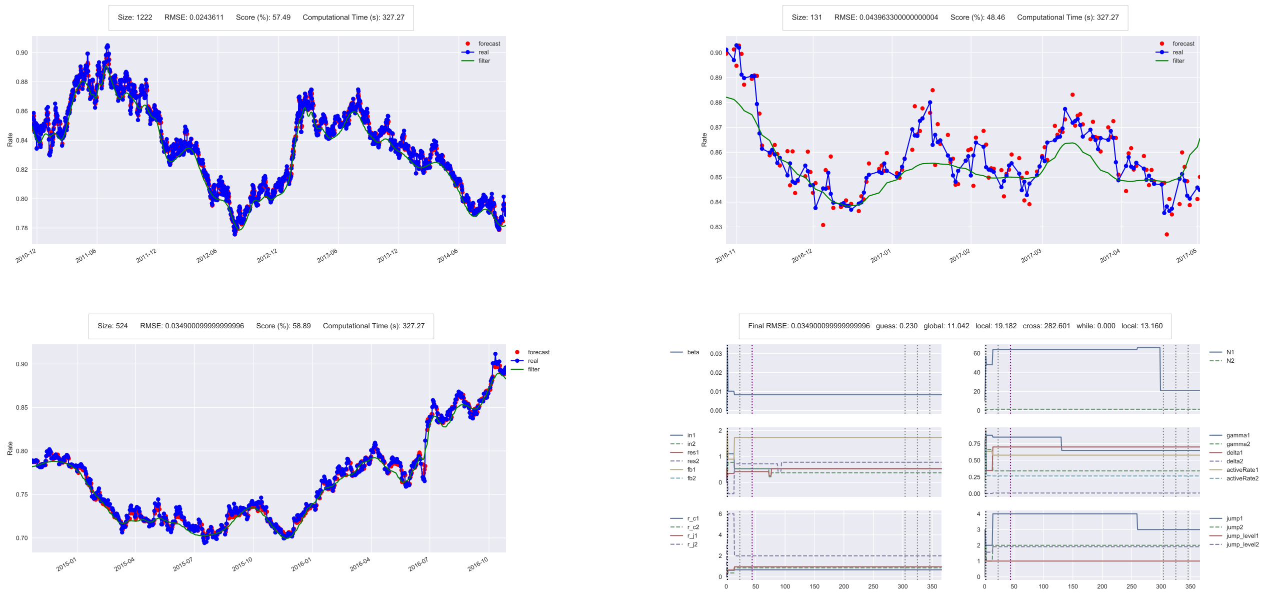


Figure 6: EURGBP time series forecasting. On the left hand side from top to bottom: Training and Validation samples. The filter (green curve) is used as input on these samples. On the right hand side from top to bottom: Testing and parameters optimisation through the GEN optimisation process. It is important to precise that the filter (green curve) is not used during testing and therefore is purely informative.

So far, we have worked with a single FX pair. In order to perform a broader study, we consider a pool of instruments from the same market and focus on two cases: one with filtered data and one with non-filtered data. We have gathered our results and computed the mean of the RMSE and Score for each sample.

	Filtered Input			Non-Filtered Input		
	Training	Validation	Testing	Training	Validation	Testing
RMSE	0.0199	0.0282	0.0290	0.0241	0.0336	0.0237
Score	0.6207	0.6692	0.5045	0.4860	0.5196	0.5126

Table 2: Comparison between filtered input and non-filtered input results on 71 FX pairs.

From Table (2), one can clearly observe that we get better learning when using filtered input data. However, we can not use a denoising technique when testing the data as it would result in data-snooping bias. Thus, we are left with using market data, which results in poor performance (slightly above 50%). Intuitively, we have trained a reservoir with filtered data. As long as the new input data is relatively smooth, the reservoir can produce good predictions. However, in presence of noise the reservoir state is too much disrupted and gets too far away from a stable state. When using non-filtered data, the table shows that there is no learning. It corresponds to case (3) above where the network is trained with an attractor-based model but exploitation is ran on a transient process.

3.1.2 Using online algorithms

3.1.2.1 The Mackey-Glass series In the case of the Mackey-Glass time series discussed in Section (3.1.1), we have shown that the offline algorithm was very good at calibrating and predicting this signal. However, the performance was downgraded when adding a noisy component to it (equivalent to financial time series).
Both academics and practitioners proposed to smooth financial data with filtering methods. For instance, wavelets were considered as they can represent any signal in a smooth manner and, as such, provide a certain stability (see Appendix (7.1)). Further, it was suggested that they give information about the past and future at any time t in the training and validation samples, teaching the model how to behave. A good overview of the application of wavelets in economics and finance is given by Ramsey [1999].
However, as explained by Bloch [2014], even though the discrete wavelet transform (DWT) developed by Mallat [1989] is ideal for data compression, it can not simply relate information at a given time point at the different scales. Further, it is not possible to have shift invariance (we cannot use information from the future). One can get around this problem by means of a redundant, or, non-decimated wavelet transform, such as the a trous algorithm.
Still, for simplicity of exposition, we consider the following framework:

$$\forall t \in \mathbb{T} \quad f(t) = f^*(t) + \epsilon(t) \tag{3.1}$$

where ϵ is a Gaussian white noise with an appropriate constant width. We also suppose that there exists a wavelet $\hat{f}*$ capable of approximating $f*$. We want to show that learning with an online algorithm produces better approximation results than with an offline algorithm. We consider the Mackey-Glass signal presented in Section (2.4.1). In the training and validation samples, $f*$ is used as target y in the Algorithm (12). Again, we cannot use it in the testing sample as it would result in data-snooping bias. Therefore, the green signal is only informative. The signal which is used in the testing sample by Algorithm (12) is the one coloured in magenta in Figure (9). It is the result of Algorithm (1) applied at every time t in the testing sample (from the beginning of validation).

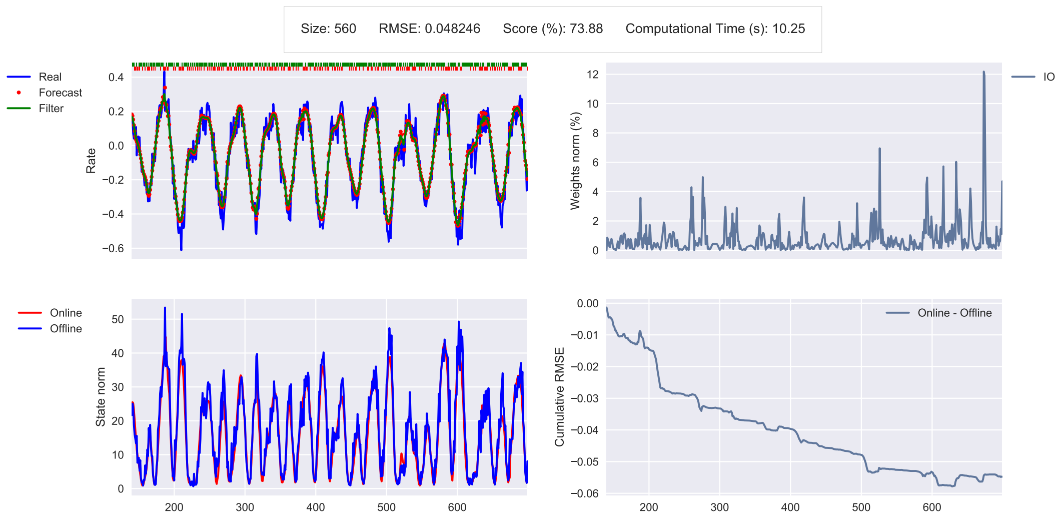


Figure 7: Noised Mackey-Glass online forecasting on the training sample. Top Left: Original, target filtered signals and forecasts. We have also plotted the local accuracy forecast (+1 if real slope is well forecast, -1 if not). Bottom Left: Reservoir states L2-norm from offline and online algorithms. Top Right: Normalised output weights evolution $\frac{||W_{out}(t)-W_{out}(t-1)||}{W_{out}(0)}$. where $W_{out}(0)$ is the offline matrix calibrated on the training sample. Bottom Right: error $cumulRMSE_{on} - cumulRMSE_{off}$.

From Figure (7) above, one can see that the denoised signal using Algorithm (1) is very smooth and has the same shape as the original Mack-Glass time series. Further, the learning Algorithm (12) leads to very good prediction of the green signal. This good learning is guaranteed by the combination of smooth states (plot on the bottom left, better visualisation on Figure (9)) and the adaptive filtering technique refining the output weights at each point in time (plot on the top right). This technique results in a very good approximation of the filtered signal and clearly outperforms the offline algorithm. Overall, we get a very good RMSE at 0.048246 and score at 73.88%. Note that we have used only a few iteration in this simulation: $(globalMax, crossMax, localMax) = (1, 1, 1)$. It indicates that a lot of reservoirs can be appropriate for our task, adding more flexibility. Also it is very computationally efficient as a good learning can be produced in less than 11 sec.

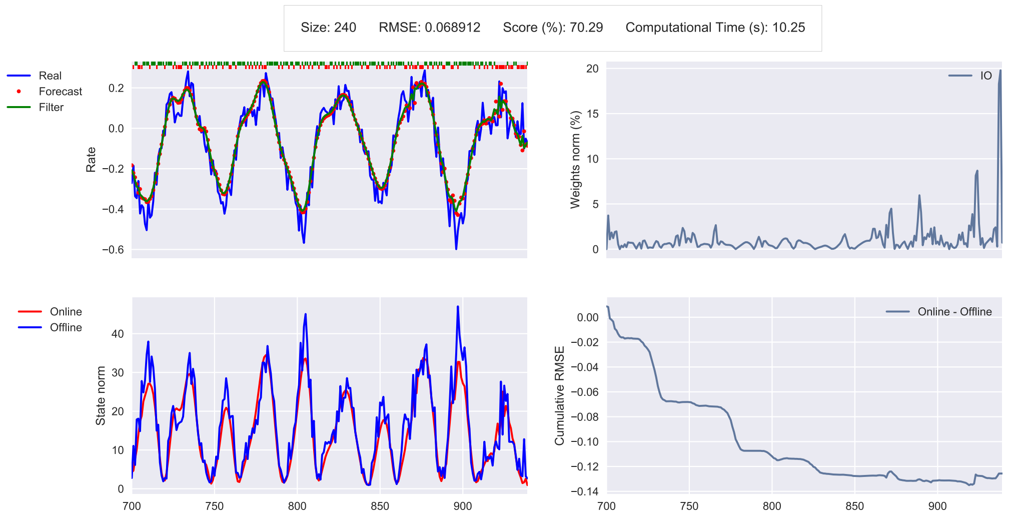


Figure 8: Noised Mackey-Glass online forecasting on the validation sample. See Figure (7) for an advanced description.

From Figure (8) above, we see that the good prediction performance is preserved on the validation sample and that the algorithm produces roughly the same results than the one obtained on the training sample.

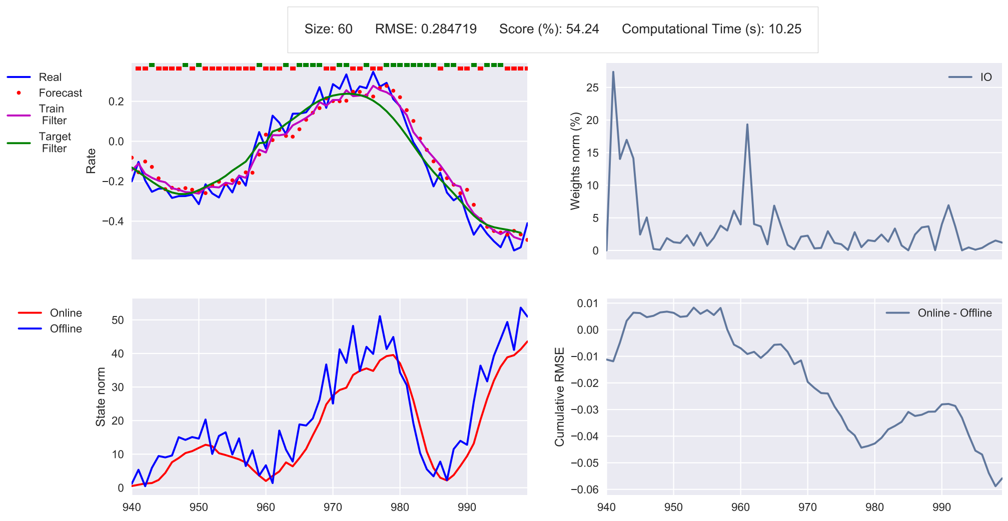


Figure 9: Noised Mackey-Glass online forecasting on the testing sample. See Figure (7) for an advanced description. We have added the filtered used by the algorithm in magenta. The green one is informative only.

Figure (9) shows that the magenta signal produces good results as the states remain stable. However, the results of the online algorithm against the offline one are not as good. Further, the output weights are still changing (from 5% to 25%), indicating that the online algorithm cannot forecast constant parameters in time. This is undesirable as we would like our reservoir to have good predictive properties when only feeding it with the new filtered input.

3.1.2.2 Financial time series We now assess the performance of the online algorithm on market data. Again, we use our pool of 71 FX pairs from 2010-01-01 to 2017-05-01.

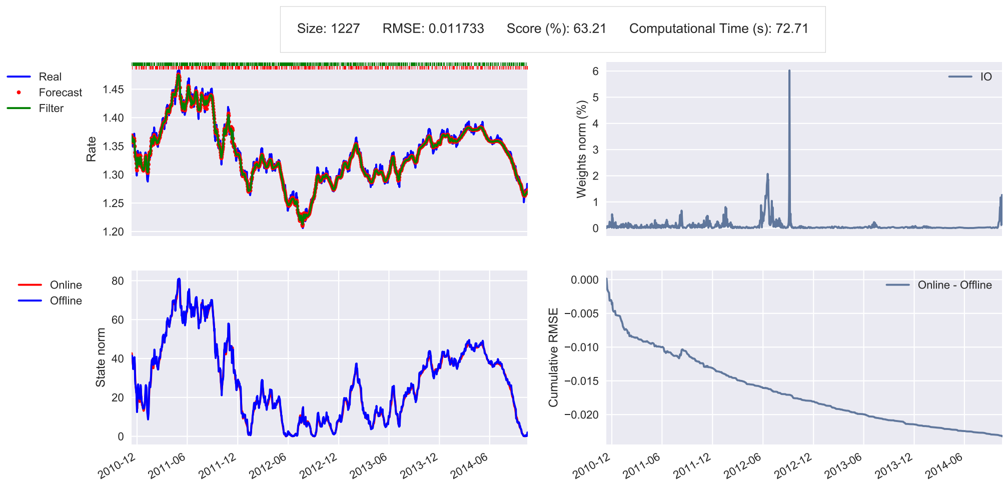


Figure 10: EURUSD online forecasting on the training sample.

Figure (10) above demonstrates good learning and good performance of the online algorithm compared to the offline one. However the output matrix is sometimes unstable. For instance, a peak of nearly 6% has occurred between 2012-06 and 2012-12, which is well above the average amplitude of the relative output matrix variation. It often happens during regime shift. Further, when we look at EURGBP on the 23^{rd} of June, 2016 (Brexit), we see that the adaptive filtering has produced a peak of 4%. Overall, there is good learning with a RMSE at 0.011733 and a Score at 63.21%. The algorithm ran in approximately one minute.

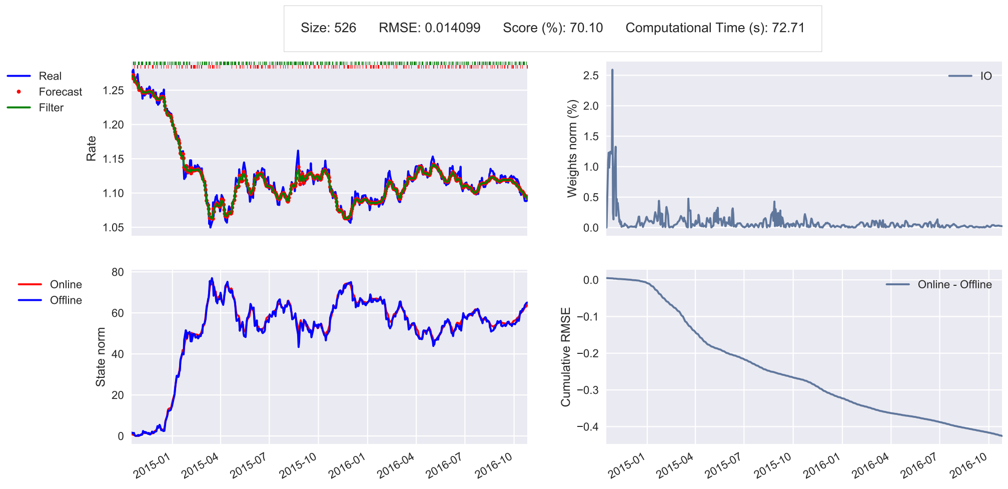


Figure 11: EURUSD online forecasting on the validation sample.

From Figure (11) above, we note that good performance is still maintained on the validation sample.

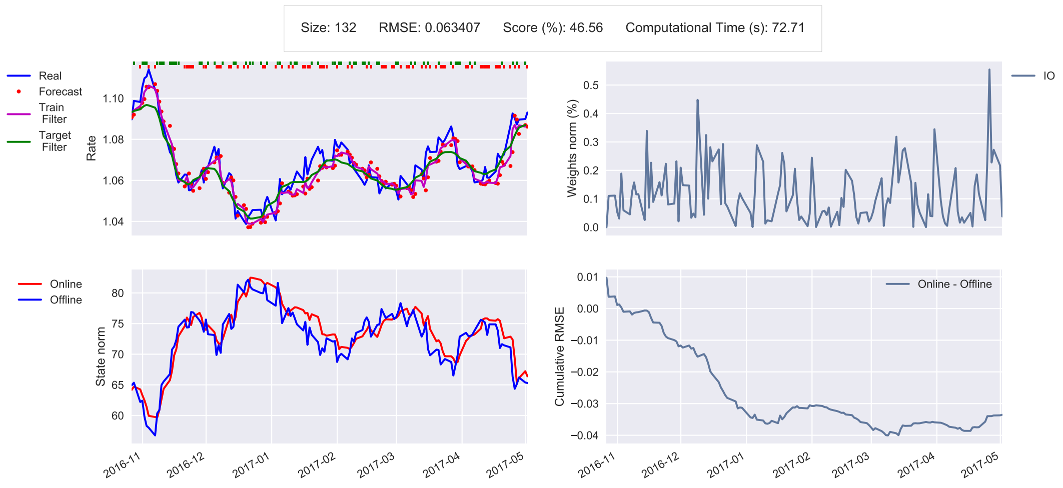


Figure 12: EURUSD online forecasting on the testing sample.

On the testing sample, the performance is degraded as the filter we used has no-longer information about the future. Therefore, it does not move like the green one. Nonetheless, the states norm is still smoother than the one obtained from the offline algorithm, which is desired when learning time series. Further, we observe that the output weights are varying a lot, although with small amplitude (most of the time below 0.5%). Thus, we conclude that the forecast has not reach the nearest stable point (which would correspond to the next value of the filtered series we use).

	Training	Validation	Testing
RMSE	0.0213	0.0314	0.0813
Score	0.6078	0.6694	0.4910

Table 3: Global assessment of the online algorithm on the pool of 71 FX pairs from 2015-01-01 to 2017-05-01.

In Table (3), we report the results on the same FX pool as the one used previously. Our results demonstrate good learning on the training and validation samples with low RMSEs and high scores. However, some issues remain on the testing sample. One could argue that remaining as close as possible to the filtered series (magenta) in the testing is crucial, while not interfering with the calibrated model by modifying some matrix (particularly the output weights).

3.2 Towards dynamically stable reservoirs

3.2.1 The proposed methods

So far, we have seen that Reservoir Computing (RC) models with offline training were very efficient for learning RNNs, since only the output weights were changed, making training extremely fast. However, the results provided in Section (3.1.1) are not satisfactory when exploiting the models with real data. In addition, even with an online algorithm, the results in Section (3.1.2) clearly indicate that the models fail to reproduce the patterns, or forecast chaotic series, in presence of noise. More generally, we have seen that the reservoir states must be smooth (in the sense of its norm) in order to learn the data and produce good predictions.

We presented in Appendix (12.4.2) two methods for teaching the reservoir how to remain consistently stable while feeding it with noisy input. The first one suppose we can observe stable states on the training sample and use it to obtain the output weight matrix in presence of noisy data. The second one consist in working only with filtered inputs and to directly teach the reservoir to predict the filtered data. Indeed, this filter can be seen as an indicator of the initial dataset. One could then argue that getting a good prediction of the filtered data could lead to good real time series forecasting if one can control the forecasting error.

3.2.2 Some results

To illustrate the second method (learning with filtered inputs), we consider the FX pairs previously introduced together with Algorithm (13). We observe that for some thresholds this algorithm does not converge as the states can settle in another attractor. In general it happens on the testing sample where W_{out} and W_{rec} are no-longer trained. We suspect that a more accurate choice of threshold is needed for some instruments. We have gathered the results of our experiment in the table below.

	Training	Validation	Testing
RMSE	0.0236	0.0340	0.0740
Score	0.5852	0.6205	0.5174

Table 4: Global assessment of the ARC algorithm on 71 FX pairs from 2015-01-01 to 2017-05-01.

RMSE are stable both on the training and validation samples. Regarding the score, it is also relatively stable on the training and validation samples and the hit ratio is around 52%. As a particular example, we consider the case of CADSGD presented below.

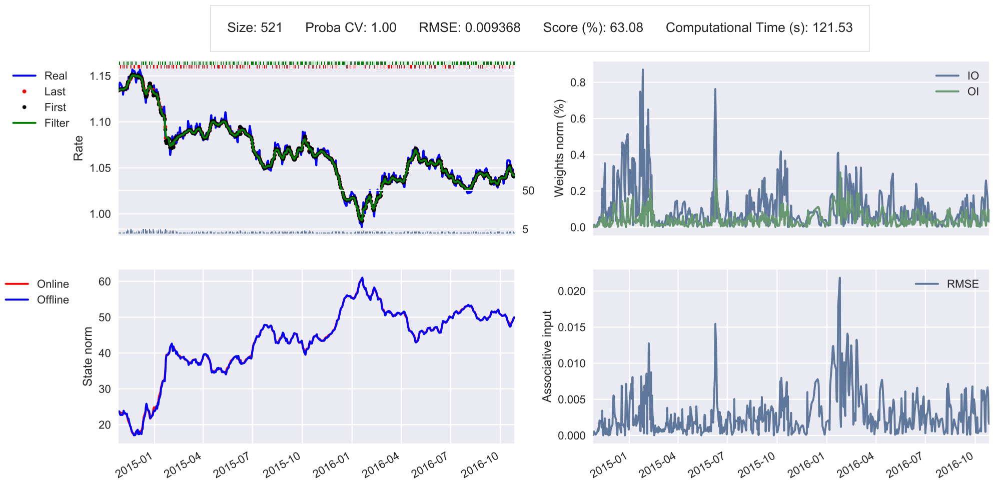


Figure 13: CADSGD online forecasting on the validation sample.

We note that the reservoir is learning very well to calibrate the filtered signal. There are very few iterations in Algorithm (13), which indicates reservoir stability. Furthermore, the weights norm graph shows that only small variation of W_{out} and W_{rec} are needed to learn to predict well (below 0.4% on the right half part of the graph). Also the associative input absolute error is very small at the end of the validation sample (below 0.007). This shows that the reservoir has properly learned the matrices W_{out} and W_{rec} to forecast the filtered signal, and therefore to predict well the slopes of the real financial time series. Consequently, it makes sense to suppose that our model is now constant in the testing sample (W_{out} and W_{rec} are no-longer updated with Algorithm (12)).

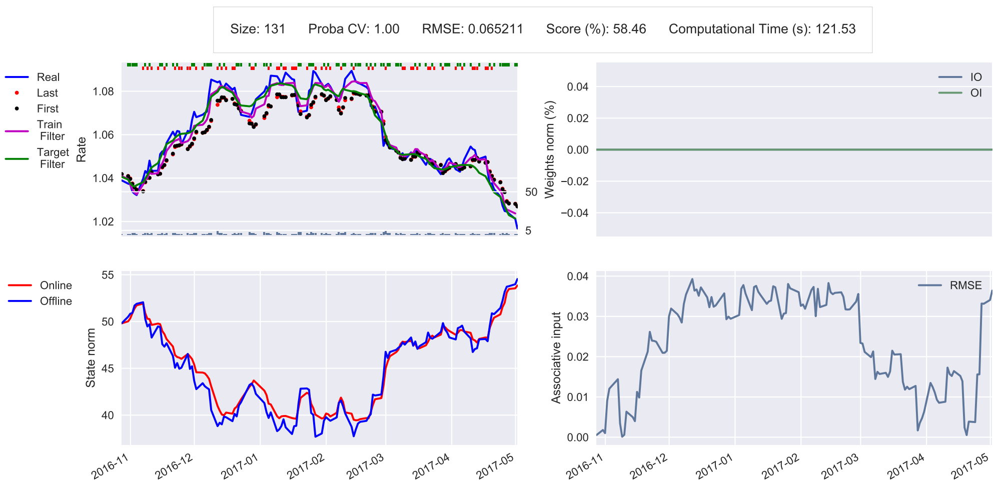


Figure 14: CADSGD online forecasting on the testing sample.

Note, it is important to recall that the model is constant in this sample, meaning that the output and recurrent matrices are not updated. That is, we let the model make prediction only by input feeding and use of Algorithm (13). The bottom bar plot on the top left graph indicates that the reservoir needs only a few iteration to converge to an attractor (always below 5), attesting prediction robustness. The reservoir reproduces well the shape of the train filter (magenta) and stay stable (bottom left figure) at the same time. Secondly, one can observe that the model produces good predictions at the beginning of the testing sample. It can be inferred as follows:

- The top bar plot on the left top graph.
- The associative input absolute error which is relatively small.

Further, by comparing the associative input with the real one at time t, we have some indications about the model convergence to the attractor it should predict without waiting for time $t + 1$.
Lastly, we note that the magenta series is very correlated with the green series, which explains why the forecasts are good. One solution could be to find appropriate filter thresholds to correlate them artificially on the validation sample and see if the correlation is maintained on the testing sample. It is important to point out that there is optimisation to be done for this issue as a too high or too low threshold would both lead to high correlation but poor performance. However, there may exist some thresholds leading to good learning and good enough correlation.

3.3 Forecasting directions

3.3.1 Introduction

3.3.1.1 An overview As discussed in Section (3.1), the prediction of financial time series is a highly challenging task since they are multifractal. It requires models to capture subtle functional relationships among the empirical data even though the underlying relationships are unknown or hard to describe. Such relationships are not required as far as predicting future trend is concerned. This is because the direction of an underlying asset refers to the movement of the asset price or the trend of fluctuation in the underlying asset in the future. Precise forecasting of the trends of the underlying asset can be extremely advantageous for investors. Over the last twenty years, traders and academics have used technical analysis (see Appendix (8.2)) as well as certain fundamental valuation metrics to improve results when

forecasting trend (see Gencay [1997], Metghalchi et al. [2007]). Further, some economists and statisticians showed that technical indicators have predictive power and found a good measure of support for their effectiveness (see Chen et al. [2009], Metghalchi et al. [2012] [2015]). These properties led some data scientists to incorporate such trading strategies in machine learning in order to forecast future price directions (see Tanaka-Yamawaki et al. [2007], Senol et al. [2008], Kara et al. [2011], Patel et al. [2015], Qiu et al. [2016]). There are two main approaches when using technical indicators, one can either directly use the continuous technical indicators (actual time series), or one can choose to represent them as trend deterministic data (discrete in nature) (see Appendix (8.4.3)).
We are going to present some of these methods to forecasting directions.

3.3.1.2 Notation We follow the notation in Appendix (10) and consider the data set $D = \{(x_1, y_1), ..., (x_l, y_l)\}$ for the instances x_i and $y_i = f(x_i)$, where f is the ground-truth target function. We consider binary classification task and let y_i takes values in the set $\{-1, +1\}$ representing down and up movement, respectively. We let the instance space $\mathcal{X}$ be the space of market returns and we let the label space $\mathcal{Y}$ be the space of market directions. Given the instance x, we suppose that the learner h makes the prediction $\widehat{f} = h(x) \in \{-1, +1\}$, which we call the signal. The model signal $\widehat{f}_k$ corresponds to a direction forecast of the market price observed at fixed time intervals $t_k = k \times \delta$, where $k = 1, ..., N_k$ is an integer and $\delta = \Delta t$ is a fixed fraction of time. We will suppose that the model forecast last for the period $\tau = n \times u$, where u is a fixed unit of time (in second) and $n = 1, 2, ...$ is an integer.
We are interested in the forecast for the learner h for a fixed period d. Thus, to validate the forecast, we are interested in the kth log-return $r_{L,d}(k) = r_L(k, k + d)$ for the d period defined in Equation (6.11). We have two time scales, the one given by the model signals and the market prices observed at tick time levels.

3.3.2 The predictive algorithm

The direction measures used to assess the predictive power of the algorithm are defined in Appendix (6.4.2).

3.3.2.1 Selecting the input variables When selecting the input variables of the predictive model one can either directly use continuous technical indicators (actual time series), or one can choose to represent them as trend deterministic data (discrete in nature). Kara et al. [2011] proposed a method in the case where the inputs are the continuous technical indicators. The values of all technical indicators are normalised in the range between $[-1, +1]$ so that larger value of one indicator do not overwhelm the smaller valued indicator. Given a (one hidden layer) neural network (NN), the output can be a single neuron with a sigmoid transfer function, which results in a continuous value output between 0 and 1 (see Figure (15)). A threshold of $\frac{1}{2}$ is used to determine the up and down direction prediction, where the former occurs for a value greater than $\frac{1}{2}$ and the latter occurs for a value smaller than $\frac{1}{2}$. In the case where we choose to represent the indicators as trend deterministic data (TDD), Patel et al. [2015] proposed the Trend Deterministic Data Preparation Layer (TDDPL) (see Figure (16)). It extracts trend related information from each of the technical indicators by converting continuous-valued technical parameters to discrete value (up $+1$, down -1), representing the trend (see details in Appendix (8.4.3)). The idea being that if a prediction model is given direct continuous-valued input, it will try to establish relation between the values at different time steps, which is not required as far as predicting future trend is considered. What matter is how its value (up $+1$, down -1) has changed with respect to previous time step rather than the absolute value of change. Each input parameters in its discrete form indicates a possible up or down trend determined based on its inherent property. Thus, when using TDDPL we are directly inputting the trend based such that prediction models only have to determine co-relation between the input trends and output trend (which is easier than determining relation within serially correlated continuous time series).

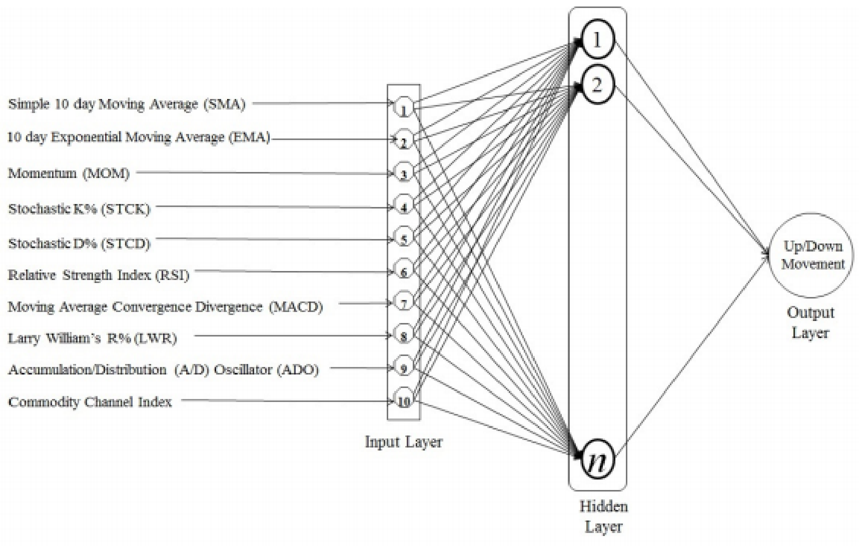

Simple 10 day Moving Average (SMA)
10 day Exponential Moving Average (EMA)
Momentum (MOM)
Stochastic K% (STCK)
Stochastic D% (STCD)
Relative Strength Index (RSI)
Moving Average Convergence Divergence (MACD)
Larry William's R% (LWR)
Accumulation/Distribution (A/D) Oscillator (ADO)
Commodity Channel Index
Input Layer
Hidden Layer
Up/Down Movement
Output Layer

Figure 15: A three-layer neural network with continuous indicators

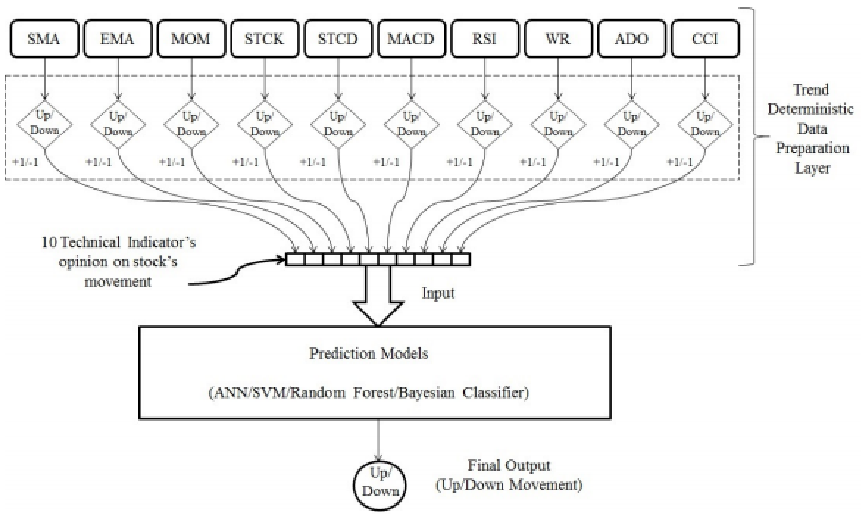

SMA
EMA
MOM
STCK
STCD
MACD
RSI
WR
ADO
CCI
Up/ Down
+1/-1
Trend Deterministic Data Preparation Layer
10 Technical Indicator's opinion on stock's movement
Input
Prediction Models
(ANN/SVM/Random Forest/Bayesian Classifier)
Up/ Down
Final Output (Up/Down Movement)

Figure 16: Predicting with trend deterministic data

3.3.2.2 The algorithm The predictive model proposed by Van den Poel et al. [2016] is made of two layers, namely the Feature Engineering (selection) layer and the Prediction layer. This is similar to stacking described in Appendix (10.5). Rather than using the Deterministic Data Preparation Layer (DDPL) proposed by Patel et al. [2015], the authors chose to combine the continuous indicators with some technical trading rules which give signals based on criteria for identifying market entry and exit points (see Appendix (8.4.1)). The Feature Selection layer devises the trading rules for each technical indicator. The input of the model are the return values corresponding to the dependent variable along with the continuous-valued technical indicators (see Appendix (8.2)), single technical trading rules and complex trading rules based on two-way and three-way combinations of individual indicators (see Appendix (8.3)), constituting the independent ones. Thus, the only difference with DDPL is the use of complex trading rules which greatly increases the number of predictors. First, using the generated data, they derive a set of simple and complex trading rules for each of the technical indicator employed, resulting in the discrete values $1, -1$, or 0, representing buy, sell or hold signals, respectively. We denote the direction triplet as $(1, -1, 0)$ for each predictor. These signals serve as inputs to the second layer, which employs Random Forest for feature learning and classification. The return values and the predictors are used for learning purposes (similar to Figure (16)). It produces an estimated direction triplet $(\widehat{1, -1}, 0)$.

3.3.2.3 Strategy and returns When testing their model, Van den Poel et al. [2016] considered one year of data sampled with a fixed period of time $\delta = 1$ minute. The training sample is eight-month long and the testing sample covers a four-month period. The trading horizon is a day and PnL is determined per minute ($d = 1$ minute) with no reinvestment. The strategy is as follows: when a buy signal is triggered the trader takes a long position in the underlying asset until a sell signal is generated. Then the trader unwind (sell) is position and goes short until the next buy signal or the end of day (if no position is held overnight). In case of two consecutive buy or sell signals, the first one is considered and the other ignored.
The returns are determined by applying the trading strategy to the generated signals. We get the mapping

$$(\widehat{1, -1}, 0) \Rightarrow \big((+R, -R), (+R, -R), 0\big) = \big((1, 0), (0, 1), 0\big)$$

where 1 indicates a positive return and 0 a negative return. The machine learning (ML) model is trained to recognise valuable signals from invaluable ones so that whenever the predictive probability on a positive return exceeds a cutoff value the trade is executed. The overall return generated by the model is then calculated by adding up the returns of the performed trades.

3.3.3 The evaluation metrics

Evaluation measures for classifiers are presented in Appendix (9.4). When it comes to forecasting directions, we need to express the evaluation metrics in terms of market returns and signals. To do so, we let $R_{k,up}$ and $R_{k,down}$ denote an up movement and down movement of the market return at time t_k, respectively. They are given by

$$R_{k,up} = I(r_{L,d}(k) > 0) \text{ and } R_{k,down} = I(r_{L,d}(k) < 0)$$

while a neutral position is $R_{i,neutral} = I(r_{L,d}(k) = 0)$. We also let $M_k = I(r_{L,d}(k) \neq 0)$ be a move and define the kth signal up and down as follows:

$$\underset{k,up}{\text{signal}} = I(\widehat{f}_k = +1) \text{ and } \underset{k,down}{\text{signal}} = I(\widehat{f}_k = -1)$$

The kth trade return $R_{k,trade}$ is given by

$$R_{k,trade} = \underset{k,up}{\text{signal}} \times r_{L,d}(k) - \underset{k,down}{\text{signal}} \times r_{L,d}(k)$$

A win trade is given by

$$win_k = I\big(\underset{k,up}{\text{signal}} \times R_{k,up} + \underset{k,down}{\text{signal}} \times R_{k,down}\big)$$

and a loose trade is given by

$$lose_k = I\big(\underset{k,up}{\text{signal}} \times R_{k,down} + \underset{k,down}{\text{signal}} \times R_{k,up}\big)$$

The union of both a win and loose trade is

$$wl_k = win_k \bigcup lose_k$$

We denote $N_{sig,up} = \sum_{k=1}^{N_k} \text{signal}_{k,up}$, $N_{sig,down} = \sum_{k=1}^{N_k} \text{signal}_{k,down}$ the number of signal up and down, respectively, and $N_{sig} = N_{sig,up} + N_{sig,down}$. We let R_{trade} be the vector of trade returns with kth element $R_{k,trade}$, and we denote R_{trade}^{win} the vector of trade returns for which all elements have $win = 1$ and R_{trade}^{loss} the vector of trade returns for which all elements have $loss = 1$. We also denote R_{trade}^{wl} the vector of trade returns for which all elements have $wl = 1$. We let N_h be the number of hours during the trading period and define the total number of possible forecast for the model as

$$N_f = N_h \times \frac{60 \times 60}{u}$$

where u is the unit of time. We can then present a few metrics

$$\begin{aligned}
\text{accuracy} &= \frac{\sum_{k=1}^{N_k} \left(\text{signal}_{k,up} \times R_{k,up} + \text{signal}_{k,down} \times R_{k,down}\right)}{\sum_{k=1}^{N_k} \left(\text{signal}_{k,up} \times M_k + \text{signal}_{k,down} \times M_k\right)} \\
\text{accuracy up} &= \frac{\sum_{k=1}^{N_k} \text{signal}_{k,up} \times R_{k,up}}{\sum_{k=1}^{N_k} \text{signal}_{k,up} \times M_k} \\
\text{accuracy down} &= \frac{\sum_{k=1}^{N_k} \text{signal}_{k,down} \times R_{k,down}}{\sum_{k=1}^{N_k} \text{signal}_{k,down} \times M_k} \\
\text{hit} &= \frac{1}{N_{sig}} \sum_{k=1}^{N_k} \left(\underset{k,up}{\text{signal}} \times R_{k,up} + \underset{k,down}{\text{signal}} \times R_{k,down}\right) \\
\text{hit up} &= \frac{1}{N_{sig,up}} \sum_{k=1}^{N_k} \underset{k,up}{\text{signal}} \times R_{k,up} \\
\text{hit down} &= \frac{1}{N_{sig,down}} \sum_{k=1}^{N_k} \underset{k,down}{\text{signal}} \times R_{k,down} \\
\text{coverage} &= \frac{1}{N_f} \sum_{k=1}^{N_k} \left(\underset{k,up}{\text{signal}} + \underset{k,down}{\text{signal}}\right) \\
\text{average win} &= \text{median}(R_{trade}^{win}) \\
\text{average loss} &= \text{median}(R_{trade}^{loss}) \\
\text{average total PnL} &= \text{ average win } \times \text{ accuracy } + \text{ average loss } \times (1 - \text{ accuracy }) \\
\text{standard dev win} &= \text{std}(R_{trade}^{win}) \\
\text{standard dev loss} &= \text{std}(R_{trade}^{loss}) \\
\text{mean total PnL} &= \text{mean}(R_{trade}^{wl}) \\
\text{standard dev total PnL} &= \text{std}(R_{trade}^{wl}) \\
\text{Sharpe ratio} &= \frac{\text{mean total PnL}}{\text{standard dev total PnL}}
\end{aligned}$$

3.4 Forecasting returns and directions

We are going to describe a framework to forecast both future price levels and asset price directions by using optimum combination of technical indicators (see Appendix (8.2)), trading rules and trading strategies (see Appendix (8.4)).

To do so we will follow the predictive model proposed by Van den Poel et al. [2016], which we will enlarge by incorporating the trading strategies proposed by Metghalchi et al. [2018].

3.4.1 Improving the algorithm

3.4.1.1 Selection layer We are now going to explain how to enlarge the algorithm in Section (3.3.2) by incorporating the trading strategies proposed by Metghalchi et al. [2018]. At each time step we will select the most efficient trading strategy among the list of strategy given in Appendix (8.4.2). To do so, we propose the Meta Buy/Sell Signal (MBSS) inferred from all the predictors generated by the selection layer. We are going to build a level of confidence around the MBSS and use it to select the most efficient strategy. As discussed above, at a given time t, the inputs of the latter are all the independent variables and the output is the direction triplet $(1, -1, 0)$ for each independent variable. In order to get a meta buy/sell decision signal, we choose to combine all the direction triplets with a majority voting rule (or any other rule) (see Appendix (8.3)). We can then build an algorithm that estimates the likelihood of the majority voting rule and, accordingly, selects the best trading strategy. Since the learning process happens within the training sample (see Section (2.4.2)), everything is known and we can let the boolean $\alpha(t)$ be true (T) if the estimated direction is correct and false (F) otherwise. Assuming N predictors (or independent variables), we let $n(t)$ be the number of predictors that gives $\alpha(t) = T$. We can therefore estimate n^* as the minimum n that gives the maximum number of true flags over the training sample. Thus, when validating and testing the data at a given time t_+, we compute all direction triplets, apply the majority voting rule, and infer $n(t_+)$. Then, the trading rule is as follows:

$$\begin{cases} \text{if } n(t_+) << n^* \text{ use cash} \\ \text{if } n(t_+) < n^* \text{ use long} \setminus \text{cash} \\ \text{if } n(t_+) > n^* \text{ use long} \setminus \text{short} \\ \text{if } n(t_+) >> n^* \text{ use leverage} \setminus \text{short} \end{cases}$$

3.4.1.2 Prediction layer Since machine learning models can learn more than one task, we can use a neural network to learn both future price levels and asset price directions. It is a simple matter to modify the above algorithm to forecast price levels or returns. When discussing continuous technical indicators as input variables, Kara et al. [2011], Patel et al. [2015] among others, considered a three layer ANN model, which does not account for serially correlated time series. We have already presented a framework in Section (3.1) that forecast market returns and highlighted its good predictive power on denoised financial time series. Since the continuous technical indicators are mostly filtered financial time series we can use a recurrent neural network (RNN) model and consider continuous technical indicators to forecast future directions (see Figure (16)). The time dependent direction measures used to assess the predictive power of the algorithm are defined in Appendix (6.4.3).

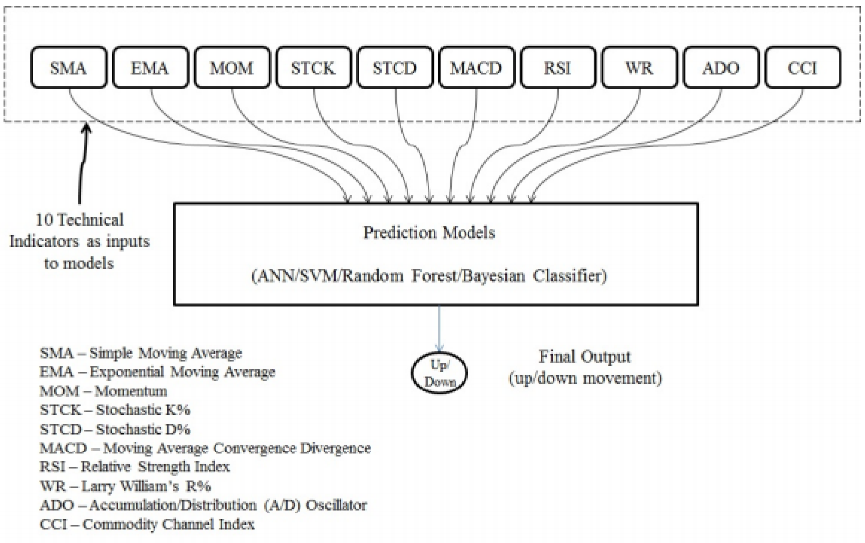

SMA
EMA
MOM
STCK
STCD
MACD
RSI
WR
ADO
CCI
10 Technical Indicators as inputs to models
Prediction Models
(ANN/SVM/Random Forest/Bayesian Classifier)
Up/ Down
Final Output (up/down movement)
SMA – Simple Moving Average
EMA – Exponential Moving Average
MOM – Momentum
STCK – Stochastic K%
STCD – Stochastic D%
MACD – Moving Average Convergence Divergence
RSI – Relative Strength Index
WR – Larry William's R%
ADO – Accumulation/Distribution (A/D) Oscillator
CCI – Commodity Channel Index

Figure 17: Predicting with trend continuous data

3.5 Conclusions

While a simplified representation of reality can either be descriptive or predictive in nature, or both, financial models are predictive to forecast unknown or future values based on current or known values, using mathematical equations or set of rules. However, the forecasting power of a model is limited by the appropriateness of the inputs and assumptions so that one must identify the sources of model risk to understand these limitations (see Appendix (8)). Forecasting financial time series is characterised by data intensity, noise, non-stationarity, unstructured nature, high degree of uncertainty, and hidden relationships. Thus, predicting their future values is a highly challenging task, which requires models capable of capturing subtle functional relationships between values at different time steps. Such relationships are not required as far as predicting future trend is concerned. Trend deterministic data is the statistical indication of whether the assets are over-bought or over-sold and as such is value independent. Thus, prediction models have to determine co-relation between the input trends and the output trend, which is a simpler task than forecasting market returns.

3.5.1 Failure to capture noisy data

When forcating price levels (case $(1+)$ in Section (2.4.1)), most articles do not report good results when testing data. This is because the financial time series are multi-fractal and transient. Thus, returns are filtered (denoised) and used to train ML models, which are then exploited on transient process (case $(3*)$ in Section (2.4.2)). A large literature has been written describing case $(3*)$. However, good results are reported on the training set, but when exploiting the data, the hit ratio collapses in the range $[48\%, 55\%]$ (see results in Section (3.1)). Unless the training set resemble the filtered data, the learning process is not preserved (see Lin et al. [2009]). Recognising that returns could not be directly forecast, Dablemont et al. [2003] first smoothed the rough data by projecting them onto a functional basis (for example cubic splines), and then forecast the future transactions splines with an empirical functional analysis of the past of series.
Consequently, ML models perform well on smoothed time series (with Hurst exponent $H > \frac{1}{2}$), but have great difficulties forecasting a series with $H \leqslant \frac{1}{2}$.

3.5.2 Overfitting the data

In general, academics and practitioners consider a single machine learning (ML) model to explain market returns from different asset classes and different underlying. Usually, different topologies are tried against the data and several sets of meta-data are calibrated to the returns and a score function is applied to select the set that best fit the data (see Lukosevicius et al. [2009]). Increasing the number of model parameters for the sake of fitting a finite sample is bound to overfitting the data, explaining the poor out-of-sample behaviour. As a result, the model lose its predictive power.

3.5.3 Common mistakes

Some of the mistakes commonly made by academics using ML in finance are:

- to ignore the properties of financial returns
- to ignore the statistical characteristic of the ML model considered
- to use the same ML model on all financial returns (including $H \leqslant \frac{1}{2}$)
- to train the model on a small data set (few thousands points)
- to trade at every sample point
- to ignore trading rules (See Appendix (8.4.1))
- to ignore trading strategies (See Appendix (8.4.2))
- to ignore risk management when validating and testing a model

3.5.4 Defining principles

We can infer some useful information from the results obtained in Sections (3.1) and (3.2.2):

- ML models can only reproduce certain stochastic processes (smoothed ones)
- it would make sense to use ML to forecasting time series when $H > \frac{1}{2}$
- when $H < \frac{1}{2}$, it would make sense to use ML
    1. for autonomous pattern generations by forecasting directions(see Section (3.3.2))
    2. to forecasting filtered time series, or equivalently technical indicators (see Appendix 8.4)

From these inferences, we can therefore the devise the following principles:

***Result* 1** *Principles*

1. *when $0 < H \leqslant \frac{1}{2}$ we cannot forecast market returns, but we can consider*
    (a) autonomous pattern generations (see Section (3.3.2))
    (b) forecasting technical indicators (see Appendix 8.4)
2. *when $H > \frac{1}{2}$ we can still consider autonomous pattern, but forecasting both returns and directions (see Section (3.4.1)).*
3. *when $H >> \frac{1}{2}$ we consider time series prediction.*

4 The recipe

4.1 Defining the framework

4.1.1 Problems and solutions

We have seen in Section (2.2.3) that neural networks (NNs) were treated as black box models due to their inability to explicitly know the relations established between explanatory variables (input) and dependent variables (output). While the discovery of interactions has become an essential part of the literature on neural networks, their interpretation is far more difficult. Some statistical methods have been proposed to solve the problem, but they rely on the existence of an optimal solution for the network and assume stationarity of the signal response.
Nowadays, the discovery of interactions and their interpretation is especially useful for scientific discoveries and hypothesis validation. For instance, doctors may want to know what interactions are accounted for in risk prediction models, in order to compare against known interactions from existing medical literature. In finance, we are interested in knowing the statistical properties of the time series produced by the ML model under consideration. This is because we need to make sure that the chosen ML model is capable of reproducing the characteristics of the desired series.

The results in Sections (3.1) and (3.2.2) suggest that a single RNN does not have the capacity to capture the information inherent to neither noisy chaotic series nor to financial data submitted during the training (and testing) process. These conclusions can be explained by the fact that financial time series are multifractal, thus exhibiting non-Gaussian distribution, the presence of extreme values (outliers), and long-range dependent dynamics (see Section (2.1)). Since financial markets can follow different behaviours over time, financial series should not be modelled by a single process, but preferably by a succession of different processes. For instance, Dablemont et al. [2003] assumed that a single model could not capture the dynamics of the whole series and split the historical series into clusters. They generated a specific local neural model for each of clusters, which were combined in a probabilistic way according to the distribution of the series in the past. Further, several studies showed that combining forecasts (using NN models) could greatly enhance the forecast effectiveness (see Wang et al. [2008], Chitra et al. [2010]).

Some academics showed that technical indicators with their associated trading rules have predictive power and found a good measure of support for their effectiveness (see Appendix (8.5)). As a result, some authors considered forecasting future trend direction (case $(2+)$) using classification (See Section (3.3.2)). The results obtained are achieved using only historic prices and technical indicators with their associated trading rules. Recently, some authors designed trading strategies to use in association with each trading rule. Accounting for risk and transaction costs, these strategies proved very profitable, beating the Buy and Hold strategy. Consequently, we should not only consider trading rules but go further and incorporate trading strategies in the learning process. To this end, we proposed the Meta Buy/Sell Signal (MBSS) in Section (3.4.1).

Rather than fitting an ML model to some external data and figuring out its statistical properties through some statistical interpretation methods, we inverse the causal relation by first generating large data from a priori theoretical model and then fitting an ML model to this data. This way we know the statistical properties of our ML model. We can then train a series of ML models to reproduce a set of theoretical models with distinct statistical characteristics and combine the trained (calibrated) models with ensemble methods.

As discussed in Section (2.2), in the case where real-world time series are characterised by multiple seasonalities, solutions were developed to train a multiple superimposed oscillator (MSO). Bandpass filters decoupling the signal in the frequency space were proposed, as well as networks made of multiple reservoirs trained to separately adapt to each timescale of the input signal. In each case, the parameters for adapting the network to the right time scales require analysis in the frequency domain or a priori knowledge of the input data.
While frequency analysis can be applied to decompose periodic time series, multifractal analysis must be used to decompose market returns (see Appendix (7)). Motivated by the recent research on MSOs, we propose to use multifractal

analysis to decompose the target signal with the aim of capturing its statistical properties. We build a general framework and define a meta-model with the desired statistical properties. We then train the model to recognise patterns, or technical indicators, and devise a trading algorithm.

4.1.2 The framework

We propose to decide upon the framework by defining how the model should be specified before beginning to analyse the actual data. We define the framework by properly formulating model hypotheses which make financial or economic sense, and then carefully determining the number of dependent variables in a regression model, or the number of factors and components in a stochastic model.

We consider the multifractal formalism (MF) which provides a scale invariant mechanism for the analysis and generation of complex signals that fit well with the observed experimental properties of fully developed turbulence (FDT) as well as other physical systems ranging from natural images to heartbeat dynamics or econometric signals. In addition, the formalism also allows to highlight relevant dynamical features of the systems under study, and theoretical models can be devised to fit the observed multifractal properties.
As discussed in Section (3.5), rather than using a single model (or blindly stacking models [1]) to fit a small sample of market returns, and risk overfitting the data, we want to find models generalising potential future data. Further, we saw in Appendix (11.3.2) that neural networks with fixed weights had great difficulty accounting for changing environment. We must therefore construct a framework where we can generate unlimited data from a priori distributions, formulate an hypothesis, build a model and test its adaptive ability to changing environments. In the case of poor performance, we would modify the hypothesis or formulate a new one, resulting in a modified (or new) model until a level of acceptance is reached.

Based on these observations, we use the multifractal formalism to devise a recipe for quantitative trading with machine learning which is independent from the choice of a model. Similarly to physics, we can use the multifractal formalism (MF) as a framework for testing the capacity of an ML model to reproduce some statistical properties of time series. To do so, we consider a set $\mathscr{T}$ of theoretical models $\mathcal{T}_m$, $m = 1, ..., M$, having non-overlapping pre-defined statistical properties. We also consider a set $\mathscr{M}$ of ML models $\mathcal{M}_m$, $m = 1, ..., M$, each one of them capable of reproducing one theoretical model from the set $\mathscr{T}$. We train each ML model $\mathcal{M}_m$ to approximate its associated theoretical model $\mathcal{T}_m$ (calibration), and use ensemble methods to combine the calibrated ML models $\mathcal{M}_{calib,m}$, obtaining a meta-model. However, the weights of the latter are statistically computed as a function of the Hurst exponent. As a result, the meta-model is dynamically recombined based on the changing properties of the time series over time.

Once we have identified an ensemble of ML models with specific non-overlapping statistical properties, we want to use the algorithm described Section (3.4) to forecast both the market returns and their directions. We still need to fit the external input signal and possibly learn some patterns, or, technical indicators with their associated trading rules. We consider a meta-set where the market returns are the dependent variables, while the continuous-valued technical indicators with single technical trading rules as well as complex trading rules are the independent variables. Since we can associate different trading strategies to each trading rules, we use the Meta Buy/Sell Signal (MBSS) presented in Section (3.4.1) to identify the best trading strategy under the constraints of risk and transaction costs.
Recall from Section (2), a reservoir can always be split into sub-reservoirs so that different tasks can be learned. Thus, we leave a number of variable neurons, or free sub-reservoirs $\mathcal{R}_{free}$, dedicated to fitting the external input signal (financial time series) as well as learning a large number of patterns or technical indicators (see Section (3.3.2)).

4.2 The algorithm

We are now going to build our ML meta-model to satisfies certain pre-defined statistical properties taken from the multifractal formalism and, following the principles in Result (1), we are going to devise our trading algorithm. For

[1] In general, stacking produces small gains with a lot of added complexity. The difficulty being to devise the combining function.

simplicity of exposition, we let the set of theoretical models $\mathscr{T}$ be generated from fractional Brownian motion (fBm) with successive Hurst exponent. In practice, they can be generated from any fractal or multifractal theoretical model. Further, we consider a time-dependent Hurst exponent, but we can also consider a time and scale Hurst exponent (see Appendix (7.4)).

4.2.1 Defining an ensemble ML models

Inspired by the ensemble methods (see Definition (10.1)), our approach consists in devising several ML models capable of reproducing some pre-defined statistical properties (such as the Hurst exponent H), training them, and recombining them. We consider M models $\mathcal{M}_1, ..., \mathcal{M}_M$ with corresponding predictions $\hat{f}_1(x), ..., \hat{f}_M(x)$ with instance x. For each model $\mathcal{M}_m$, $m = 1, ..., M$, we define its associated band of Hurst exponent $[h_{m-1}, h_m]$ where $h_0 = 0 < h_1 < \cdots < h_M = 1$ and set $\Delta_m = h_m - h_{m-1}$. We let $H_{mid,m} = \frac{\Delta_m}{2}$ be the mid Hurst exponent of the band value.
The method is as follows:

1. For each $m = 1, ..., M$, generate a large set of data from a theoretical model $\mathcal{T}_m$ in the set $\mathscr{T}$ having some pre-defined statistical properties (using the multifractal formalism). For instance, the data can be generated from a fractional Brownian motion(fBm) with $H_{mid,m}$.
2. Devise an ML model $\mathcal{M}_m$, $m = 1, ..., M$, capable of reproducing this theoretical model $\mathcal{T}_m$.
3. Train the model to get optimum network.
4. Repeat steps $(1 - 3)$, obtaining an ensemble of M ML models, each of them having non-overlapping statistical properties.

While there are several choices for building the ML model in step (2) above (see Section (2)), we will start by considering

- Reservoir computing (RC) to characterise transient processes
- Associative reservoir computing (ARC) to characterise persistent processes

Other choices are possible, as long as the models can reproduce transient and persistent processes.
We apply model stacking and combine the models to obtain a meta-model. While we could solve the minimisation problem in Equation (10.29) to find the weights of the meta-model, we feel that it would result in over-fitting the data. Alternatively, we are going to statistically compute the weights by making them a function of the Hurst exponent, $\omega_m = \varphi(H)$. The prediction of the meta-model is (see Equation (10.30)) defined by

$$\hat{f}_{avecv}(x) = \sum_{m=1}^{M} \omega_m \hat{f}_m^{(cv)}(x)$$

One solution for specifying the weight ω_m is given by

$$\omega_m = \begin{cases} 1 \text{ if } H \in [h_{m-1}, h_m] \\ 0 \text{ otherwise} \end{cases}$$

where $[h_{m-1}, h_m]$ is the band of Hurst exponent for model $\mathcal{M}_m$. This the indicatrice function I_A for the event $A = H \in [h_{m-1}, h_m]$. To handle events around the boundary of the band of Hurst exponent, we could also use a hat function. For instance, given the hat function centred around x_j and taking values in $[x_{j-1}, x_{j+1}]$,

$$\Lambda(x) = \begin{cases} \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \text{ if } x_{j-1} \leqslant x < x_j \\ \frac{(x_{j+1} - x)}{(x_{j+1} - x_j)} \text{ if } x_j \leqslant x < x_{j+1} \\ 0 \text{ otherwise} \end{cases}$$

we can set $x_{j-1} = h_{m-1}$, $x_{j+1} = h_m$ and $x_j = H_{mid,m}$ so that the hat function is centred around $H_{mid,m}$. Then,

$$\omega_m = \Lambda(H) \text{ and } \begin{cases} \omega_{m-1} = 1 - \omega_m \text{ if } m > 1 \text{ and } h_{m-1} \leqslant H < H_{mid,m} \\ \omega_{m+1} = 1 - \omega_m \text{ if } m < M \text{ and } H_{mid,m} \leqslant H < h_m \end{cases}$$

Note, more sophisticated function $\varphi(\cdot)$ can be used.

4.2.2 The learning algorithm

In the first part of the recipe we have identified an ensemble of ML models with specific non-overlapping statistical properties. We are still left with some free sub-reservoirs to fit the external input signal and learn a large number of patterns or technical indicators. The former is the basis of all learning algorithm and we refer the readers to Appendix (12) for more details in our chosen model.
We are now going to explain how to train the remaining sub-reservoirs to learn a large number of patterns or technical indicators (see Appendix (8)). We follow the predictive model described in Section (3.4.1), which is made of two layers, namely the Feature Engineering layer and the Prediction layer. First, using the generated data, we derive a set of simple and complex trading rules for each of the technical indicator employed, resulting in the discrete values $1, -1$, or 0, representing buy, sell or hold signals, respectively. These signals serve as inputs to the second layer, which employs the neural network for feature learning and classification.

We let $\mathcal{R}_{free,m}$ be a set of free reservoir for the m-th calibrated ML model. The meta-model is going to be combined on the fly as new external input reveals. Prior to this, we need to teach each set $\mathcal{R}_{free,m}$ to recognise patterns or technical indicators.
The method is as follows:

1. Pick a calibrated ML model $\mathcal{M}_{calib,m}$, $m = 1, ..., M$, and associated set $\mathcal{R}_{free,m}$, from our ensemble of models.
2. Teach the model (free sub-reservoirs) to identify:
   (a) either autonomous patterns corresponding to specific strategies (discrete technical indicators), or
   (b) technical indicators (continuous technical indicators).
3. Repeat steps $(1 - 2)$ with other calibrated models from the ensemble.

At the end of the learning process we have an ensemble of calibrated ML models $\mathcal{M}_{calib,m}$, $m = 1, ..., m$, with non-overlapping statistical properties and associated sub-reservoirs $\mathcal{R}_{calib,m}$, each of them capable recognising patterns or technical indicators. We are left with fitting the remaining free reservoir to external input data. At last, as time evolves, we statistically estimate the Hurst exponent of the desired time series and use it to combine our meta-model. The latter is used in the trading strategy of our choice.

4.2.3 The trading algorithm

Once we have an ensemble of calibrated ML models capable of recognising a large number of patterns, or forecasting technical indicators, we devise a trading algorithm with strategies accounting for a specific level of Hurst exponent. We can then validate and exploit our approach with real market data.
We let the coefficient H be a function of time and denote H_d the Hurst exponent computed at discrete time t_d. The trading algorithm is as follows:

1. At time t_d, statistically analyse the market returns, identifying the Hurst coefficient H_d, or time-space Hurst exponents. Build the meta-model by computing the weights ω_m as described above. Then proceed as follows:
   (a) if $H_d \leqslant \frac{1}{2}$, then:

- either do not trade, or
- forecast the asset directions using technical indicators. Then use the MBSS to identify the best trading strategy and execute it.

(b) if $H_d > \frac{1}{2}$, forecast both market returns and their directions. Then use the MBSS to identify the best trading strategy and execute it.

(c) if $H_d >> \frac{1}{2}$, then forecast directly the market returns.

2. Increment time and repeat step (1) to account for the fact that the statistical properties of the signal response might have been modified.

Annexes

5 Some random sampling

For details see text book by Rice [1995]. We assume that the population is of size N and that associated with each member of the population is a numerical value of interest denoted by $x_1, x_2, .., x_N$. The variable x_i may be a numerical variable such as age or weight, or it may take on the value 1 or 0 to denote the presence or absence of some characteristic. The latter is called the dichotomous case.

5.1 The sample moments

In general, the population variance of a finite population of size N is given by

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \text{ with } \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

where μ is the population mean. In the dichotomous case μ equals the proportion p of individuals in the population having the particular characteristic. The population total is

$$\tau = \sum_{i=1}^{N} x_i = N\mu$$

In many practical situations, the true variance of a population is not known a priori and must be computed somehow. When dealing with extremely large populations, it is not possible to count every object in the population and one must estimate the variance of a population from a sample. We take a sample with replacement of n values $X_1, ..., X_n$ from the population, where $n < N$ and such that X_i is a random variable. X_i is the value of the ith member of the sample, and x_i is that of the ith member of the population. The joint distribution of the X_i is determined by that of the x_i. We let $\xi_1, .., \xi_m$ be the elements of a vector corresponding to the possible values of x_i. Since each member of the population is equally likely to be in the sample, we get

$$P(X_i = \xi_j) = \frac{n_j}{N}$$

The sample mean is

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Theorem 5.1 *With simple random sampling* $E[\overline{X}] = \mu$

We say that an estimate is unbiased if its expectation equals the quantity we wish to estimate.

$$\text{Mean squared error } = \text{ variance } + (biased)^2$$

Since $\overline{X}$ is unbiased, its mean square error is equal to its variance.

Lemma 5.1 *With simple random sampling*

$$Cov(X_i, X_j) = \begin{cases} \sigma^2 \textit{ if } i = j \\ -\frac{\sigma^2}{(N-1)} \textit{ if } i \neq j \end{cases}$$

It shows that X_i and X_j are not independent of each other for $i \neq j$, but that the covariance is very small for large values of N.

Theorem 5.2 *With simple random sampling*

$$Var(\overline{X}) = \frac{\sigma^2}{n}\big(1 - \frac{n-1}{N-1}\big)$$

Note, the ratio $\frac{n}{N}$ is called the sampling fraction and

$$p_c = \big(1 - \frac{n-1}{N-1}\big)$$

is the finite population correction. If the sampling fraction is very small we get the approximation

$$\sigma_{\overline{X}} \approx \frac{\sigma}{\sqrt{n}}$$

We estimate the variance on the basis of this sample as

$$\hat{\sigma}^2 = \sigma_n^2 = \frac{1}{n}\sum_{i=1}^n (X_i - \overline{X})^2 \text{ with } \overline{X} = \frac{1}{n}\sum_{i=1}^n X_i$$

This is the biased sample variance. The unbiased sample variance is

$$\overline{\sigma}_n^2 = \frac{1}{n-1}\sum_{i=1}^n (X_i - \overline{X})^2 \text{ with } \overline{X} = \frac{1}{n}\sum_{i=1}^n X_i$$

While the first one may be seen as the variance of the sample considered as a population (over n), the second one is the unbiased estimator of the population variance (over N) where $(n-1)$ is the Bessel's correction. Put another way, we get

$$E[\sigma_n^2] = \frac{n-1}{n}\frac{N}{N-1}\sigma^2 \text{ and } E[\overline{\sigma}_n^2] = \sigma^2$$

That is, the unbiased estimate of σ^2 may be obtained by multiplying σ_n^2 by the factor $\frac{n}{n-1}\frac{N-1}{N}$. If the population is large relative to n, the dominant bias is due to the term $\frac{n-1}{n}$. In general one set n between 50 and 180 days. Being a function of random variables, the sample variance is itself a random variable, and it is natural to study its distribution. In the case that y_i are independent observations from a normal distribution, Cochran's theorem shows that $\overline{\sigma}_n^2$ follows a scaled chi-squared distribution

$$(n-1)\frac{\overline{\sigma}_n^2}{\sigma^2} \approx \chi_{n-1}^2$$

If the conditions of the law of large numbers hold for the squared observations, $\overline{\sigma}_n^2$ is a consistent estimator of σ^2 and the variance of the estimator tends asymptotically to zero. The obtained standard deviation $\tilde{\sigma}_N$ is an estimator of σ called the historical volatility.

***Corollary* 1** *An unbiased estimate of* $Var(\overline{X})$ *is*

$$s_{\overline{X}}^2 = \frac{\hat{\sigma}^2}{n}\frac{n}{n-1}\frac{N-1}{N}\frac{N-n}{N-1} = \frac{s^2}{n}(1 - \frac{n}{N})$$

where

$$s^2 = \frac{1}{n-1}\sum_{i=1}^n (X_i - \overline{X})^2$$

One can compute the sample mean and sample variance recursively as follow. Starting from the identities

$$(n+1)\overline{X}_{n+1} = n\overline{X}_n + X_{n+1}$$

and

$$(n+1)\big(\sigma^2_{n+1} + (\overline{X}_{n+1})^2\big) = n\big(\sigma^2_n + (\overline{X}_n)^2\big) + X^2_{n+1}$$

we get the recursive sample mean

$$\overline{X}_{n+1} = \overline{X}_n + \frac{X_{n+1} - \overline{X}_n}{n+1} \tag{5.2}$$

and the recursive sample variance

$$\sigma^2_{n+1} = \sigma^2_n + (\overline{X}_n)^2 - (\overline{X}_{n+1})^2 + \frac{X^2_{n+1} - \sigma^2_n - (\overline{X}_n)^2}{n+1} \tag{5.3}$$

such that σ^2_{n+1} only depends on σ^2_n, $\overline{X}_n$, $\overline{X}_{n+1}$ and X_{n+1}. If we normalise the data with a constant K such that the ith value is $\frac{X_i}{K}$, we let $\tilde{X} = \frac{1}{n}\sum_{i=1}^n \frac{X_i}{K}$ be the sample mean and $\tilde{\sigma}^2_n = \frac{1}{n}\sum_{i=1}^n (\frac{X_i}{K} - \tilde{X})^2$ be the sample variance. Then, the recursive sample mean becomes

$$\tilde{X}_{n+1} = \tilde{X}_n + \frac{1}{n+1}\big(\frac{X_{n+1}}{K} - \tilde{X}_n\big) \tag{5.4}$$

and the recursive sample variance becomes

$$\tilde{\sigma}^2_{n+1} = \tilde{\sigma}^2_n + (\tilde{X}_n)^2 - (\tilde{X}_{n+1})^2 + \frac{1}{n+1}\Big(\frac{X^2_{n+1}}{K^2} - \tilde{\sigma}^2_n - (\tilde{X}_n)^2\Big) \tag{5.5}$$

5.2 Estimation of a ratio

We consider the estimation of a ratio, and assume that for each member of a population, two values, x and y, may be measured. The ratio of interest is

$$r = \frac{\sum_{i=1}^N y_i}{\sum_{i=1}^N x_i} = \frac{\mu_y}{\mu_x}$$

Assuming that a sample is drawn consisting of the pairs (X_i, Y_i), the natural estimate of r is $R = \frac{\overline{Y}}{\overline{X}}$. Since R is a nonlinear function of the random variables $\overline{X}$ and $\overline{Y}$, there is no closed form for $E[R]$ and $Var(R)$ and we must approximate them by using $Var(\overline{X})$, $Var(\overline{Y})$, and $Cov(\overline{X}, \overline{Y})$. We define the the population covariance of x and y as

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)$$

One can show that

$$Cov(\overline{X}, \overline{Y}) = \frac{\sigma_{xy}}{n} p_c$$

Theorem 5.3 *With simple random sampling, the approximate variance of* $R = \frac{\overline{Y}}{\overline{X}}$ *is*

$$Var(R) \approx \frac{1}{\mu^2_x}(r^2\sigma^2_{\overline{X}} + \sigma^2_{\overline{Y}} - 2r\sigma_{\overline{XY}}) = \frac{1}{n}p_c\frac{1}{\mu^2_x}(r^2\sigma^2_x + \sigma^2_y - 2r\sigma_{xy})$$

The population correlation coefficient given by

$$\rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

is a measure of the strength of the linear relationship between the x and the y values in the population. We can express the variance in the above theorem in terms of ρ as

$$Var(R) \approx \frac{1}{n} p_c \frac{1}{\mu_x^2} (r^2 \sigma_x^2 + \sigma_y^2 - 2r\rho \sigma_x \sigma_y)$$

so that strong correlation of the same sign as r decreases the variance. Further, for small μ_x we get large variance, since small values of $\overline{X}$ in the ratio $R = \frac{\overline{Y}}{\overline{X}}$ cause R to fluctuate wildly.

Theorem 5.4 *With simple random sampling, the expectation of R is given approximately by*

$$E[R] \approx r + \frac{1}{n} p_c \frac{1}{\mu_x^2} (r^2 \sigma_x^2 - \rho \sigma_x \sigma_y)$$

so that strong correlation of the same sign as r decreases the bias, and the bias is large if μ_x is small. In addition, the bias is of the order $\frac{1}{n}$, so its contribution to the mean squared error is of the order $\frac{1}{n^2}$ while the contribution of the variance is of order $\frac{1}{n}$. Hence, for large sample, the bias is negligible compared to the standard error of the estimate. For large samples, truncating the Taylor Series after the linear term provides a good approximation, since the deviations $\overline{X} - \mu_x$ and $\overline{Y} - \mu_y$ are likely to be small. To this order of approximation, R is expressed as a linear combination of $\overline{X}$ and $\overline{Y}$, and an argument based on the central limit theorem can be used to show that R is approximately normally distributed, and confidence intervals can be formed for r by using the normal distribution. In order to estimate the standard error of R, we substitute R for r in the formula of the above theorem where the x and y population variances are estimated by s_x^2 and s_y^2. The population covariance is estimated by

$$s_{xy} = \frac{1}{n-1} \sum_{i=1}^n (X_i - \overline{X})(Y_i - \overline{Y}) = \frac{1}{n-1} \big(\sum_{i=1}^n X_i Y_i - n \overline{XY} \big)$$

and the population correlation is estimated by

$$\hat{\rho} = \frac{s_{xy}}{s_x s_y}$$

The estimated variance of R is thus

$$s_R^2 = \frac{1}{n} p_c \frac{1}{\overline{X}^2} (R^2 s_x^2 + s_y^2 - 2R s_{xy})$$

and the approximate $100(1-\alpha)\%$ confidence interval for r is $R \pm z(\frac{\alpha}{2}) s_R$.

5.3 Stratified random sampling

The population is partitioned into subpopulations, or strata, which are then independently sampled. We are interested in obtaining information about each of a number of natural subpopulations in addition to information about the population as a whole. It guarantees a prescribed number of observations from each subpopulation, whereas the use of a simple random sample can result in underrepresentation of some subpopulations. Further, the stratified sample mean can be considerably more precise than the mean of a simple random sample, especially if there is considerable variation between strata.

We will denote by N_l, where $l = 1, .., L$ the population sizes in the L strata such that $N_1 + N_2 + .. + N_L = N$ the total population size. The population mean and variance of the lth stratum are denoted by μ_l and σ_l^2 (unknown). The overall population mean can be expressed in terms of the μ_l as follows

$$\mu = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{N_l} x_{il} = \frac{1}{N}\sum_{l=1}^{L} N_l \mu_l = \sum_{l=1}^{L} W_l \mu_l$$

where x_{il} denotes the ith population value in the lth stratum and $W_l = \frac{N_l}{N}$ is the fraction of the population contained in the lth stratum.

Within each stratum a simple random sample of size n_l is taken. The sample mean in stratum l is denoted by

$$\overline{X}_l = \frac{1}{n_l}\sum_{i=1}^{n_l} X_{il}$$

where X_{il} denotes the ith observation in the lth stratum. By analogy with the previous calculation we get

$$\overline{X}_s = \sum_{l=1}^{L} \frac{N_l \overline{X}_l}{N} = \sum_{l=1}^{L} W_l \overline{X}_l$$

Theorem 5.5 *The stratisfied estimate, $\overline{X}_s$ of the population mean is unbiased. That is, $E[\overline{X}_s] = \mu$.*

Since we assume that the samples from different strata are independent of one another and that within each stratum a simple random sample is taken, the variance of $\overline{X}_s$ can easily be calculate.

Theorem 5.6 *The variance of the stratisfied sample mean is given by*

$$Var(\overline{X}_s) = \sum_{l=1}^{L} W_l^2 \frac{1}{n_l}(1 - \frac{n_l - 1}{N_l - 1})\sigma_l^2 \tag{5.6}$$

Note, $\frac{n_l-1}{N_l-1}$ represents the finite population corrections. If the sampling fractions within all strata are small ($\frac{n_l-1}{N_l-1} << 1$), we get the approximation

$$Var(\overline{X}_s) \approx \sum_{l=1}^{L} \frac{W_l^2}{n_l}\sigma_l^2 \tag{5.7}$$

The estimate of σ_l^2 is given by

$$S_l^2 = \frac{1}{n_l - 1}\sum_{i=1}^{n_l}(X_{il} - \overline{X}_l)^2$$

and $Var(\overline{X}_s)$ is estimated by

$$S_{\overline{X}_s}^2 = \sum_{l=1}^{L} W_l^2 \frac{1}{n_l}(1 - \frac{n_l - 1}{N_l - 1})S_l^2$$

The question that naturally arises is how to choose $n_1, .., n_L$ to minimise $Var(\overline{X}_s)$ subject to the constraint $n_1 + .. + n_L = n$. Ignoring the finite population correction within each stratum we get the Neyman allocation.

Theorem 5.7 *The sample sizes $n_1, .., n_L$ that minimise $Var(\overline{X}_s)$ subject to the constraint $n_1 + .. + n_L = n$ are given by*

$$n_l = n\frac{W_l \sigma_l}{\sum_{k=1}^{L} W_k \sigma_k}$$

where $l = 1, .., L$.

This theorem shows that those strata for which $W_l \sigma_l$ is large should be sampled heavily. If W_l is large, the stratum contains a large fraction of the population. If σ_l is large, the population values in the stratum are quite variable, and a relatively large sample size must be used. Substituting the optimal values of n_l into the Equation (5.6) we get the following corollary.

Corollary 2 *Denoting by $\overline{X}_{so}$ the stratified estimate using the optimal Neyman allocation and neglecting the finite population correction, we get*

$$Var(\overline{X}_{so}) = \frac{1}{n} \big(\sum_{l=1}^{L} W_l \sigma_l \big)^2$$

The optimal allocations depend on the individual variances of the strata, which generally will not be known. A simple alternative method of allocation is to use the same sampling fraction in each stratum

$$\frac{n_1}{N_1} = \frac{n_2}{N_2} = ... = \frac{n_L}{N_L}$$

which holds if

$$n_l = n \frac{N_L}{N} = n W_l \quad (5.8)$$

$l = 1, .., L$. This method is called the Proportional allocation. The estimate of the population mean based on proportional allocation is

$$\overline{X}_{sp} = \sum_{l=1}^{L} W_l \overline{X}_l = \frac{1}{n} \sum_{l=1}^{L} \sum_{i=1}^{n_l} X_{il}$$

since $\frac{W_l}{n_l} = \frac{1}{n}$. This estimate is simply the unweighted mean of the sample values.

Theorem 5.8 *With stratified sampling based on proportional allocation, ignoring the finite population correction, we get*

$$Var(\overline{X}_{sp}) = \frac{1}{n} \sum_{l=1}^{L} W_l \sigma_l^2$$

We can compare $Var(\overline{X}_{so})$ and $Var(\overline{X}_{sp})$ to define when optimal allocation is substantially better than proportional allocation.

Theorem 5.9 *With stratified random sampling, the difference between the variance of the estimate of the population mean based on proportional allocation and the variance of that estimate based on optimal allocation is, ignoring the finite population correction,*

$$Var(\overline{X}_{sp}) - Var(\overline{X}_{so}) = \frac{1}{n} \sum_{l=1}^{L} W_l (\sigma_l - \overline{\sigma})^2$$

where

$$\overline{\sigma} = \sum_{l=1}^{L} W_l \sigma_l$$

As a result, if the variances of the strata are all the same, proportional allocation yields the same results as optimal allocation. The more variable these variances are, the better it is to use optimal allocation.

We can also compare the variance under simple random sampling with the variance under proportional allocation. Neglecting the finite population correction, the variance under simple random sampling is $Var(\overline{X}) = \frac{\sigma^2}{n}$. We first need a relationship between the overall population variance σ^2 and the strata variances σ_l^2. The overall population variance may be expressed as

$$\sigma^2 = \frac{1}{N} \sum_{l=1}^{L} \sum_{i=1}^{N_L} (x_{il} - \mu)^2$$

Further, using the relation

$$(x_{il} - \mu)^2 = (x_{il} - \mu_l)^2 + 2(x_{il} - \mu_l)(\mu_l - \mu) + (\mu_l - \mu)^2$$

and realising that when both sides are summed over l, the middle term on the right-hand side becomes zero, we have

$$\sum_{i=1}^{N_L} (x_{il} - \mu)^2 = \sum_{i=1}^{N_L} (x_{il} - \mu_l)^2 + N_L(\mu_l - \mu)^2 = N_L \sigma_l^2 + N_L(\mu_l - \mu)^2$$

Dividing both sides by N and summing over l, we have

$$\sigma^2 = \sum_{l=1}^{L} W_l \sigma_l^2 + \sum_{l=1}^{L} W_l (\mu_l - \mu)^2$$

Substituting this expression for σ^2 into $Var(\overline{X})$ and using the formula for $Var(\overline{X}_{sp})$ completes the proof of the following theorem.

Theorem 5.10 *The difference between the variance of the mean of a simple random sample and the variance of the mean of a stratified random sample based on proportional allocation is, neglecting the finite population correction,*

$$Var(\overline{X}) - Var(\overline{X}_{sp}) = \frac{1}{n} \sum_{l=1}^{L} W_l (\mu_l - \mu)^2$$

Thus, stratified random sampling with proportional allocation always gives a smaller variance than does simple random sampling, providing that the finite population correction is ignored. Typically, stratified random sampling can result in substantial increases in precision for populations containing values that vary greatly in size.

In order to construct the optimal number of strata, the population values themselves (which are unknown) would have to be used. Stratification must therefore be done on the basis of some related variable that is known or on the results of earlier samples.

According to the Neyman-Pearson Paradigm, a decision as to weather or not to reject H_0 in favour of H_A is made on the basis of $T(X)$, where X denotes the sample values and $T(X)$ is a statistic.

The statistical properties of the methods are relevant if it is reasonable to model the data stochastically. There exists methods that are sample analogues of the cumulative distribution function of a random variable. It is useful in displaying the distribution of sample values.

Given $x_1, .., x_n$ a batch of numbers, the empirical cumulative distribution function (ecdf) is defined as

$$F_n(x) = \frac{1}{n}(nox_i \leqslant x)$$

where $F_n(x)$ gives the proportion of the data less than or equal to x. It is a step function with a jump of height $\frac{1}{n}$ at each point x_i. The ecdf is to a sample what the cumulative distribution is to a random variable. We now consider some of the elementary statistical properties of the ecdf when $X_1, .., X_n$ is a random sample from a continuous distribution function F. We choose to express F_n as

$$F_n(x) = \frac{1}{n} \sum_{i=1}^n I_{(-\infty,x]}(X_i)$$

The random variables $I_{(-\infty,x]}(X_i)$ are independent Bernoulli random variables

$$I_{(-\infty,x]}(X_i) = \begin{cases} 1 \text{ with probability } F(x) \\ 0 \text{ with probability } 1 - F(x) \end{cases}$$

Thus, $nF_n(x)$ is a binomial random variable with the first two moments being

$$\begin{aligned} E[F_n(x)] &= F(x) \\ Var(F_n(x)) &= \frac{1}{n} F(x)[1 - F(x)] \end{aligned}$$

5.4 Geometric mean

The use of a geometric mean normalises the ranges being averaged, so that no range dominates the weighting, and a given percentage change in any of the properties has the same effect on the geometric mean. The geometric mean applies only to positive numbers. It is also often used for a set of numbers whose values are meant to be multiplied together or are exponential in nature, such as data on the growth of the human population or interest rates of a financial investment. The geometric mean of a data set $\{a_1, a_2, .., a_n\}$ is given by

$$(\prod_{i=1}^n a_i)^{\frac{1}{n}}$$

The geometric mean of a data set is less than the data set's arithmetic mean unless all members of the data set are equal, in which case the geometric and arithmetic means are equal. By using logarithmic identities to transform the formula, the multiplications can be expressed as a sum and the power as a multiplication

$$(\prod_{i=1}^n a_i)^{\frac{1}{n}} = e^{\frac{1}{n} \sum_{i=1}^n \ln a_i}$$

This is sometimes called the log-average. It is simply computing the arithmetic mean of the logarithm-transformed values of a_i (i.e., the arithmetic mean on the log scale) and then using the exponentiation to return the computation to the original scale.

6 The importance of asset returns

It is important to understand the properties of the market returns in order to devise their dynamics. We are first going to define market returns and recall some statistical tools necessary to model asset returns.

6.1 Asset returns and their characteristics

6.1.1 Defining financial returns

We consider the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ where $\mathcal{F}_t$ is a right continuous filtration including all $\mathbb{P}$ negligible sets in $\mathcal{F}$. For simplicity, we let the market be complete and assume that there exists an equivalent martingale measure $\mathbb{Q}$ as defined in the diffusion model by Harrison et al. [1979] [1981]. In the presence of continuous dividend yield, that unique probability measure equivalent to $\mathbb{P}$ is such that the discounted stock price plus the cumulated dividends are martingale when the riskless asset is the numeraire. In a general setting, we let the underlying process $(S_t)_{t \geqslant 0}$ be a one-dimensional Ito process valued in the open subset D.

6.1.1.1 Asset returns Even though the price of an asset as a function of time constitutes a financial time series, it generally shows exponential behaviour in time, making it difficult to manipulate. Return series are easier to handle than price series due to their more attractive statistical properties and to the fact that they represent a complete and scale-free summary of the investment opportunity (see Campbell et al. [1997]).
Expected returns need to be viewed over some time horizon, in some base currency, and using one of many possible averaging and compounded methods. Holding the asset for one period of time, from date t to date $(t+1)$, would result in a simple gross return

$$1 + R_{t,t+1} = \frac{S_{t+1}}{S_t} \tag{6.9}$$

where the corresponding one-period simple net return, or simple return, $R_{t,t+1}$, is given by

$$R_{t,t+1} = \frac{S_{t+1}}{S_t} - 1 = \frac{S_{t+1} - S_t}{S_t}$$

More generally, we let

$$R_{t-d,t} = \frac{\nabla_d S_t + D_{t-d,t}}{S_{t-d}}$$

be the discrete return of the underlying process where $\nabla_d S_t = S_t - S_{t-d}$ with period d and where $D_{t-d,t}$ is the dividend over the period $[t-d, t]$. For simplicity we will only consider dividend-adjusted prices with discrete dividend-adjusted returns $R_{t-d,t} = \frac{\nabla_d S_t}{S_{t-d}}$. Hence, holding the asset for d periods, between the dates $t-d$ and t, gives a d-period simple gross return

$$\begin{aligned} 1 + R_{t-d,t} = \frac{S_t}{S_{t-d}} &= \frac{S_t}{S_{t-1}} \times \frac{S_{t-1}}{S_{t-2}} \times \cdots \times \frac{S_{t-d+1}}{S_{t-d}} \\ &= (1 + R_{t-1,t})(1 + R_{t-2,t-1}) \cdots (1 + R_{t-d,t-d+1}) = \prod_{j=1}^{d} (1 + R_{t-j,t-j+1}) \end{aligned} \tag{6.10}$$

so that the d-period simple gross return is just the product of the d one-period simple gross returns, which is called a compound return. In that setting, the price of the underlying asset becomes

$$S_t = S_{t-d} \prod_{j=1}^{d} (1 + R_{t-j,t-j+1})$$

Holding the asset for d years, we get the gross return

$$(1 + R^A_{t-d,t})^d = \frac{S_t}{S_{t-d}}$$

where $R^A_{t-d,t}$ is the annualised (average) return defined as

$$\begin{aligned} R^A_{t-d,t} &= \Big(\frac{S_t}{S_{t-d}}\Big)^{\frac{1}{d}} - 1 \\ &= \Big(\prod_{j=1}^{d}(1 + R_{t-j,t-j+1})\Big)^{\frac{1}{d}} - 1 \end{aligned}$$

It is the geometric mean of the d one-period simple gross returns involved, and can be computed (see Appendix (**??**)) by

$$R^A_{t-d,t} = e^{\frac{1}{d}\sum_{j=1}^{d}\ln(1+R_{t-j,t-j+1})} - 1$$

It is simply the arithmetic mean of the logarithm returns $\ln(1 + R_{t-j,t-j+1})$, for $j = 1, .., d$, which is then exponentiated to return the computation to the original scale. As it is easier to compute arithmetic average than geometric mean, and since the one-period returns tend to be small, one can use a first-order Taylor expansion [2] to approximate the annualised (average) return as

$$R^A_{t-d,t} \approx \frac{1}{d}\sum_{j=1}^{d} R_{t-j,t-j+1}$$

Note, the arithmetic mean of two successive returns of $+50\%$ and -50% is 0%, but the geometric mean is -13% since $[(1 + 0.5)(1 - 0.5)]^{\frac{1}{d}} = 0.87$ with $d = 2$ periods. While some financial theory requires arithmetic mean as inputs (single-period Markowitz or mean-variance optimisation, single-period CAPM), most investors are interested in wealth compounding which is better captured by geometric means.

We now introduce the continuously compounded interest rates. In general, the net asset value A of continuous compounding is

$$A = Ce^{r\times n}$$

where r is the interest rate per annum, C is the initial capital, and n is the number of years. We call $e^{r\times n}$ the compounding factor. Similarly,

$$C = Ae^{-r\times n}$$

is referred to as the present value of an asset that is worth A dollars n years from now, assuming that the continuously compounded interest rate is r per annum. We call $e^{-r\times n}$ the discount factor.
If the d-period gross return on a security is just $1 + R_{t-d,t}$, then the continuously compounded return, or logarithmic return, is

$$r_L(t-d,t) = \ln(1 + R_{t-d,t}) = L_t - L_{t-d} \tag{6.11}$$

where $L_t = \ln S_t$. Note, on a daily basis we get $R_t = R_{t-1,t}$ and $r_L(t) = \ln(1 + R_t)$.
The change in log price is the yield or return, with continuous compounding, from holding the security from trading day $t - 1$ to trading day t. As a result, the price of the underlying asset becomes

$$S_t = S_{t-1}e^{r_L(t)}$$

Further, the continuously compounded return $r_L(t)$ has the property that the log-return, between the price at time t_1 and at time t_n, is given by the sum of the $r_L(t)$ between t_1 and t_n, that is,

[2] since $\log(1 + x) \approx x$ for $|x| \leqslant 1$

$$\log \frac{S_{t_n}}{S_{t_1}} = \sum_{i=1}^{n} r_L(t_i)$$

which implies that

$$S_{t_n} = S_{t_1} e^{\sum_{i=1}^{n} r_L(t_i)}$$

As a result, if the $r_L(t)$ are independent random variables with finite mean and variance, the central limit theorem implies that for very large n, the summand in the above equation is normally distributed. Hence, we would get a log-normal distribution for S_{t_n} given S_{t_1}. In addition, the variability of simple price changes, for a given security, is an increasing function of the price level of the security, whereas this is not necessarily the case with the change in log price. Then, given Equation (6.10), the logarithmic returns for d-period, defined in Equation (6.11), become

$$r_L(t-d,t) = r_L(t) + r_L(t-1) + r_L(t-2) + \cdots + r_L(t-d+1) \tag{6.12}$$

so that the continuously compounded multiperiod return is simply the sum of continuously compounded one-period returns. Note, statistical properties of log returns are more tractable. Moreover, in the cross section approach aggregation is done across the individual returns.

***Remark* 6.1** *Simple returns $R_{t-d,t}$ are additive across assets but not over time (see Equation (6.10)), wheras continuously compounded returns $r_L(t-d,t)$ are additive over time but not across assets (see Equation (6.12)).*

6.1.1.2 The percent returns versus the logarithm returns In the financial industry, most measures of returns and indices use change of returns $R_t = R_{t-1,t}$ defined as $\frac{S_t - S_{t-1}}{S_{t-1}}$ where S_t is the price of a series at time t. However, some investors and researchers prefer to use returns based on logarithms of prices $r_t = \log \frac{S_t}{S_{t-1}}$ or compound returns. As discussed in Section (6.1.1.1), continuous time generalisations of discrete time results are easier, and returns over more than one day are simple functions of a single day return. In order to compare change and compound returns, Longerstaey et al. [1995a] compared kernel estimates of the probability density function for both returns. As opposed to the histogram of the data, this approach spreads the frequency represented by each observation along the horizontal axis according to a chosen distribution function, called the kernel, and chosen to be the normal distribution. They also compared daily volatility estimates for both types of returns based on an exponential weighting scheme. They concluded that the volatility forecasts were very similar. They used that methodology to compute the volatility for change returns and then replaced the change returns with logarithm returns. The same analysis was repeated on correlation by changing the inputs from change returns to logarithm returns. They also used monthly time series and found little difference between the two volatility and correlation series. Note, while the one month volatility and correlation estimators based on change and logarithm returns do not coincide, the difference between their point estimates is negligible.

6.1.2 Portfolio returns

We consider a portfolio consisting of N instruments, and let $r_L^i(t)$ and $R_i(t)$, for $i = 1, 2, .., N$, be respectively the continuously compounded and percent returns. We assign weights ω_i to the ith instrument in the portfolio together with a condition of no short sales $\sum_{i=1}^{N} \omega_i = 1$ (it is the percentage of the portfolio's value invested in that asset). We let P_0 be the initial value of the portfolio, and P_1 be the price after one period, then by using discrete compounding, we derive the usual expression for a portfolio return as

$$P_1 = \omega_1 P_0(1+R_1) + \omega_2 P_0(1+R_2) + \cdots + \omega_N P_0(1+R_N) = \sum_{i=1}^{N} \omega_i P_0(1+R_i)$$

We let $R_p(1) = \frac{P_1 - P_0}{P_0}$ be the return of the portfolio for the first period and replace P_1 with its value. We repeat the process at periods $t = 2, 3, ...$ to get the portfolio at time t as

$$R_p(t) = \omega_1 R_1(t) + \omega_2 R_2(t) + \cdots + \omega_N R_N(t) = \sum_{i=1}^{N} \omega_i R_i(t)$$

Hence, the simple net return of a portfolio consisting of N assets is a weighted average of the simple net returns of the assets involved. However, the continuously compounded returns of a portfolio do not have the above convenient property. The portfolio of returns satisfies

$$P_1 = \omega_1 P_0 e^{r_1} + \omega_2 P_0 e^{r_2} + \cdots + \omega_N P_0 e^{r_N}$$

and setting $r_p = \log \frac{P_1}{P_0}$, we get

$$r_p = \log \left(\omega_1 P_0 e^{r_1} + \omega_2 P_0 e^{r_2} + \cdots + \omega_N P_0 e^{r_N} \right)$$

Nonetheless, RiskMetrics (see Longerstaey et al. [1996]) uses logarithmic returns as the basis in all computations and the assumption that simple returns R_i are all small in magnitude. As a result, the portfolio return becomes a weighted average of logarithmic returns

$$r_p(t) \approx \sum_{i=1}^{N} \omega_i r_L^i(t)$$

since $\log(1 + x) \approx x$ for $|x| \leqslant 1$.

6.1.3 Modelling returns: The random walk

We are interested in characterising the future changes in the portfolio of returns, described in Section (6.1.2), by forecasting each component of the portfolio using only past changes of market prices. To do so, we need to model

1. the temporal dynamics of returns
2. the distribution of returns at any point in time

Traditionally, to get tractable statistical properties of asset returns, financial markets assume that simple returns $\{R_{it}|t = 1, .., T\}$ (see Equation (6.9)) are independently and identically distributed as normal with fixed mean and variance. However, while the lower bound of a simple return is -1, normal distribution may assume any value in the real line (no lower bound). Further, if we assume that R_{it} is normally distributed, then the multi-period simple return $R_{it}(k)$ is no-longer normally distributed. At last, the normality assumption is not supported by many empirical asset returns which tend to have positive excess kurtosis. Still, the random walk is one of the widely used class of models to characterise the development of price returns.
In order to guarantee non-negativity of prices, we model the logarithm price L_t as a random walk with independent and identically distributed (iid) normally distributed changes with mean μ and variance σ^2. It is given by

$$L_t = \mu + L_{t-1} + \sigma \epsilon_t \, , \, \epsilon_t \sim iidN(0, 1)$$

The use of logarithm prices, implies that the model has continuously compounded returns, that is,

$$r_t = L_t - L_{t-1} = \mu + \sigma \epsilon_t$$

with mean and variance

$$E[r_t] = \mu \, , Var(r_t) = \sigma^2$$

Hence, an expression for prices can be derived as

$$S_t = S_{t-1} e^{\mu + \sigma \epsilon_t}$$

and S_t follows the log-normal distribution. Hence, the mean and variance of simple returns become

$$E[R_t] = e^{\mu + \frac{1}{2}\sigma^2} - 1 \, , \, Var(R_t) = e^{2\mu + \sigma^2}(e^{\sigma^2} - 1)$$

which can be used in forecasting asset returns. There is no lower bound for r_t, and the lower bound for R_t is satisfied using $1 + R_t = e^{r_t}$. Assuming that logarithmic price changes are i.i.d. implies that

- at each point in time t the log price changes are identically distributed with mean μ and variance σ^2 implying homoskedasticity (unchanging prices over time).
- log price changes are statistically independent of each other over time (the values of returns sampled at different points are completely unrelated).

However, the lognormal assumption is not consistent with all the properties of historical stock returns. The above models assume a constant variance in price changes, which in practice is flawed in most financial time series data. We can relax this assumption to let the variance vary with time in the modified model

$$L_t = \mu + L_{t-1} + \sigma_t \epsilon_t \, , \, \epsilon_t \sim N(0,1)$$

These random walk models imply certain movement of financial prices over time.

6.2 An introduction to statistical models

6.2.1 The framework

In order to model returns with classic statistical models some statistical properties must be assumed such as stationarity or weak stationarity.
We let r_t be the log return of an asset at time t, and assume that $\{r_t\}$ is either serially uncorrelated or with minor lower order serial correlations, but it is dependent. We assume that r_t follows the dynamics

$$r_t = \mu_t + a_t \tag{6.13}$$

where a_t is the (stochastic) shock or mean-corrected return (or innovation) of an asset return [3]. The conditional mean and conditional variance of r_t, given the filtration $\mathcal{F}_{t-1}$, are defined by

$$\begin{aligned} \mu_t &= E[r_t|\mathcal{F}_{t-1}] = G(\mathcal{F}_{t-1}) \\ \sigma_t^2 &= Var(r_t|\mathcal{F}_{t-1}) = E[(r_t - \mu_t)^2|\mathcal{F}_{t-1}] = H(\mathcal{F}_{t-1}) \end{aligned}$$

where G and H are well defined functions with $H(\cdot) > 0$. The model for μ_t is the mean equation for r_t, and the model for σ_t^2 is the volatility equation for r_t.

***Remark* 6.2** *Some authors use h_t to denote the conditional variance of r_t, in which case the shock becomes $a_t = \sqrt{h_t}\epsilon_t$.*

[3] since $a_t = r_t - \mu_t$

Both the functions G and H could be non-linear. For simplicity of exposition we first consider the linear case. We will then briefly present some statistical models when H is non-linear. Since we assumed that the serial dependence of a stock return series was weak, if it exists at all, μ_t should be simple and we can assume that r_t follows a simple time series model such as a stationary $ARMA(p,q)$ model. In that case, the filtration is given by $\mathcal{F}_{t-1} = \{r_{t-1}, r_{t-2}, ..., r_{t-p}\}$ with p-lags and the drift satisfies

$$\mu_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} - \sum_{i=1}^{q} \theta_i a_{t-i} \tag{6.14}$$

where ϕ_i and θ_i are real numbers. The function H is constant, that is, $H(\mathcal{F}_{t-1}) = \sigma^2$. The parameters p and q are non-negative integers, and the order (p,q) of the ARMA model may depend on the frequency of the return series.

6.2.2 The need to forecast volatility

6.2.2.1 Presentation When visualising financial time series, one can observe heteroskedasticity [4], with periods of high volatility and periods of low volatility, corresponding to periods of high and low risks, respectively. We also observe returns having very high absolute value compared with their mean, suggesting fat tail distribution for returns, with large events having a larger probability to appear when compared to returns drawn from a Gaussian distribution. Hence, besides the return series, we must also consider the volatility process and the behaviour of extreme returns of an asset (the large positive or negative returns). The negative extreme returns are important in risk management, whereas positive extreme returns are critical to holding a short position. Volatility is important in risk management as it provides a simple approach to calculating the value at risk (VaR) of a financial position. Further, modelling the volatility of a time series can improve the efficiency in parameter estimation and the accuracy in interval forecast. As returns may vary substantially over time and appear in clusters, the volatility process is concerned with the evolution of conditional variance of the return over time. Users must make sure that volatilities and correlations are predictable and that their forecasts incorporate the most useful information available. As the forecasts are based on historical data, the estimators must be flexible enough to account for changing market conditions. One simple approach is to assume that returns are governed by the random walk model, and that the sample standard deviation $\hat{\sigma}_N$ or the sample variance $\hat{\sigma}_N^2$ of returns for N periods of time can be used as a simple forecast of volatility of returns, r_t, over the future period $[t+1, t+h]$ for some positive integer h. However, volatility has some specific characteristics such as

- volatility clusters: volatility may be high for certain time periods and low for other periods.
- continuity: volatility jumps are rare.
- mean-reversion: volatility does not diverge to infinity, it varies within some fixed range so that it is often stationary.
- volatility reacts differently to a big price increase or a big price drop.

These properties play an important role in the development of volatility models. As a result, there is a large literature on econometric models available for modelling the volatility of an asset return, called the conditional heteroscedastic (CH) models. Some univariate volatility models include the autoregressive conditional heteroscedastic (ARCH) model of Engle [1982], the generalised ARCH (GARCH) model of Bollerslev [1986], the exponential GARCH (EGARCH) of Nelson [1991], the stochastic volatility (SV) models and many more. Tsay [2002] discussed the advantages and weaknesses of each volatility model and showed some applications of the models. Unfortunately, stock volatility is not directly observable from returns as in the case of daily volatility where there is only one observation in a trading day. Even though one can use intraday data to estimate daily volatility, accuracy is difficult to obtain. The unobservability of volatility makes it difficult to evaluate the forecasting performance of CH models and heuristics must be developed to estimate volatility on small samples.

[4] Heteroskedastic means that a time series has a non-constant variance through time.

6.2.2.2 A first decomposition As risk is mainly given by the probability of large negative returns in the forthcoming period, risk evaluation is closely related to time series forecasts. The desired quantity is a forecast for the probability distribution (pdf) $\tilde{p}(r)$ of the possible returns r over the risk horizon ΔT. This problem is generally decomposed into forecasts for the mean and variance of the return probability distribution

$$r_{\Delta T} = \mu_{\Delta T} + a_{\Delta T}$$

with

$$a_{\Delta T} = \sigma_{\Delta T} \epsilon$$

where the return $r_{\Delta T}$ over the period ΔT is a random variable, $\mu_{\Delta T}$ is the forecast for the mean return, and $\sigma_{\Delta t}$ is the volatility forecast. The term $a_{\Delta T} = r_{\Delta T} - \mu_{\Delta T}$ is the mean-corrected asset return. The residual ϵ, which corresponds to the unpredictable part, is a random variable distributed according to a pdf $p_{\Delta T}(\epsilon)$. The standard assumption is to let $\epsilon(t)$ be an independent and identically distributed (iid) random variable. In general, a risk methodology will set the mean μ to zero and concentrate on σ and $p(\epsilon)$. To validate the methodology, we set

$$\epsilon = \frac{r - \mu}{\sigma}$$

compute the right hand side on historical data, and obtain a time series for the residual. We can then check that ϵ is independent and distributed according to $p(\epsilon)$. For instance, we can test that ϵ is uncorrelated, and that given a risk threshold α (say, 95%), the number of exceedance behaves as expected. However, when the horizon period ΔT increases, it becomes very difficult to perform back testing due to the lack of data. Alternatively, we can consider a process to model the returns with a time increment δt of one day, computing the forecasts using conditional averages. We can then relate daily data with forecasts at any time horizon, and the forecasts depend only on the process parameters, which are independent of ΔT and are consistent across risk horizon. The quality of the volatility forecasts is the major determinant factor for a risk methodology. The residuals can then be computed and their properties studied.

6.2.3 The structure of volatility models

6.2.3.1 Presentation The above heuristics being poor estimates of the future volatility, one must rely on proper volatility models such as the conditional heteroscedastic (CH) models. Since the early 80s, volatility clustering spawned a large literaturure on a new class of stochastic processes capturing the dependency of second moments in a phenomenological way. As the lognormal assumption is not consistent with all the properties of historical stock returns, Engle [1982] first introduced the autoregressive conditional heteroscedasticity model (ARCH) which has been generalised to GARCH by Bollerslev [1986].
We consider the log return r_t to follow the $ARMA(p,q)$ model described in the above Appendix. The excess kurtosis values, measuring deviation from the normality of the returns, are indicative of the long-tailed nature of the process. Hence, one can then compute and plot the autocorrelation functions for the returns process r_t as well as the autocorrelation functions for the squared returns r_t^2.

1. If the securities exhibit a significant positive autocorrelation at lag one and higher lags as well, then large (small) returns tend to be followed by large (small) returns of the same sign. That is, there are trends in the return series. This is evidence against the weakly efficient market hypothesis which asserts that all historical information is fully reflected in prices, implying that historical prices contain no information that could be used to earn a trading profit above that which could be attained with a naive buy-and-hold strategy which implies further that returns should be uncorrelated. In this case, the autocorrelation function would suggest that an autoregressive model should capture much of the behaviour of the returns.

2. The autocorrelation in the squared returns process would suggest that large (small) absolute returns tend to follow each other. That is, large (small) returns are followed by large (small) returns of unpredictable sign. It implies that the returns series exhibits volatility clustering where large (small) returns form clusters throughout

the series. As a result, the variance of a return conditioned on past returns is a monotonic function of the past returns, and hence the conditional variance is heteroskedastic and should be properly modelled.

The conditional heteroscedastic (CH) models are capable of dealing with this conditional heteroskedasticity. The variance in the model described in Equation (6.13) becomes

$$\sigma_t^2 = Var(r_t|\mathcal{F}_{t-1}) = Var(a_t|\mathcal{F}_{t-1})$$

Since the way in which σ_t^2 evolves over time differentiate one volatility model from another, the CH models are concerned with the evolution of the volatility. Hence, modelling conditional heteroscdasticity (CH) amounts to augmenting a dynamic equation to a time series model to govern the time evolution of the conditional variance of the shock. We distinguish two types or groups of CH models, the first one using an exact function to govern the evolution of σ_t^2, and the second one using a stochastic equation to describe σ_t^2. For instance, the (G)ARCH model belongs to the former, and the stochastic volatility (SV) model belongs to the latter. In general, we estimate the conditional mean and variance equations jointly in empirical studies.

6.2.3.2 Benchmark volatility models The ARCH model, which is the first model providing a systematic framework for volatility modelling, states that

1. the mean-corrected asset return a_t is serially uncorrelated, but dependent
2. the dependence of a_t can be described by a simple quadratic function of its lagged values.

Specifically, setting $\mu_t = 0$ for simplicity, an $ARCH(p)$ model assumes that

$$r_t = h_t^{\frac{1}{2}} \epsilon_t \, , \, h_t = \alpha_0 + \alpha_1 r_{t-1}^2 + ... + \alpha_p r_{t-p}^2$$

where $\{\epsilon_t\}$ is a sequence of i.i.d. random variables with mean zero and variance 1, $\alpha_0 > 0$, and $\alpha_i \geqslant 0$ for $i > 0$. In practice, ϵ_t follows the standard normal or a standardised Student-t distribution. Generalising the ARCH model, the main idea behind (G)ARCH models is to consider asset returns as a mixture of normal distributions with the current variance being driven by a deterministic difference equation

$$r_t = h_t^{\frac{1}{2}} \epsilon_t \text{ with } \epsilon_t \sim N(0,1) \tag{6.15}$$

and

$$h_t = \alpha_0 + \sum_{i=1}^p \alpha_i r_{t-i}^2 + \sum_{j=1}^q \beta_j h_{t-j} \, , \, \alpha_0 > 0 \, , \, \alpha_i, \beta_j > 0 \tag{6.16}$$

where $\alpha_0 > 0$, $\alpha_i \geqslant 0$, $\beta_j \geqslant 0$, and $\sum_{i=1}^{\max(p,q)} (\alpha_i + \beta_i) < 1$. The latter constraint on $\alpha_i + \beta_i$ implies that the unconditional variance of r_t is finite, whereas its conditional variance h_t evolves over time. In general, empirical applications find the $GARCH(1,1)$ model with $p = q = 1$ to be sufficient to model financial time series

$$h_t = \alpha_0 + \alpha_1 r_{t-1}^2 + \beta_1 h_{t-1} \, , \, \alpha_0 > 0 \, , \, \alpha_1, \beta_1 > 0 \tag{6.17}$$

When estimated, the sum of the parameters $\alpha_1 + \beta_1$ turns out to be close to the non-stationary case, that is, mostly only a constraint on the parameters prevents them from exceeding 1 in their sum, which would lead to non-stationary behaviour. Different extensions of GARCH were developed in the literature with the objective of better capturing the financial stylised facts. Among them are the Exponential GARCH (EGARCH) model proposed by Nelson [1991] accounting for asymmetric behaviour of returns, the Threshold GARCH (TGARCH) model of Rabemananjara et al. [1993] taking into account the leverage effects, the regime switching GARCH (RS-GARCH) developed by Cai [1994], and the Integrated GARCH (IGARCH) introduced by Engle et al allowing for capturing high persistence observed in returns time series. Nelson [1990] showed that Ito diffusion or jump-diffusion processes could be obtained as a continuous time limit of discrete GARCH sequences. In order to capture stochastic shocks to the variance process, Taylor [1986] introduced the class of stochastic volatility (SV) models whose instantaneous variance is driven by

$$\ln(h_t) = k + \phi \ln(h_{t-1}) + \tau \xi_t \text{ with } \xi_t \sim N(0,1) \tag{6.18}$$

This approach has been refined and extended in many ways. The SV process is more flexible than the GARCH model, providing more mixing due to the co-existence of shocks to volatility and return innovations. However, one drawback of the GARCH models and extension to Equation (6.18) is their implied exponential decay of the autocorrelations of measures of volatility which is in contrast to the very long autocorrelation discussed in Appendix (7). Both the GARCH and the baseline SV model are only characterised by short-term rather than long-term dependence. In order to capture long memory effects, the GARCH and SV models were expanded by allowing for an infinite number of lagged volatility terms instead of the limited number of lags present in Equations (6.16) and (6.18). To obtain a compact characterisation of the long memory feature, a fractional differencing operator was used in both extensions, leading to the fractionally integrated GARCH (FIGARCH) model introduced by Baillie et al. [1996], and the long-memory stochastic volatility model of Breidt et al. [1998]. As an intermediate approach, Dacorogna et al. [1998] proposed the heterogeneous ARCH (HARCH) model, considering returns at different time aggregation levels as determinants of the dynamic law governing current volatility. In this model, we need to replace Equations (6.16) with

$$h_t = c_0 + \sum_{j=1}^{n} c_j r^2_{t,t-\Delta t_y}$$

where $r_{t,t-\Delta t_y} = \ln(p_t) - \ln(p_{t-\Delta t_j})$ are returns computed over different frequencies. This model was motivated by the finding that volatility on fine time scales can be explained to a larger extend by coarse-grained volatility than vice versa (see Muller et al. [1997]). Thus, the right hand side of the above equation covers local volatility at various lower frequencies than the time step of the underlying data ($\Delta t_j = 2, 3, ..$). Note, multifractal models have a closely related structure but model the hierarchy of volatility components in a multiplicative rather than additive format.

6.3 Generalised nonlinear nonparametric models

6.3.1 Presentation

Among the various techniques to forecast and classify financial time series, we saw in Appendix (8) that fundamental and technical analysis were the most popular ones. Even though statistical procedures were widely used for pattern recognition, the effectiveness of these methods relies both on model's assumptions and prior knowledge on data properties. To remedy these pitfalls, several classifiers developed, using various data mining and computational intelligence methods such as rule induction, fuzzy rule induction, decision trees, neural networks etc. For instance, the best recognised tools in the currency markets is the artificial neural networks (ANNs), supported by numerous empirical studies (see Ahmed et al. [2010]). Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. The former is presented in Section (11.1) and the latter is introduced in Section (11.2). The foremost reason for using ANNs is that there is some nonlinear aspect to the forecasting problem under consideration, taking the form of a complex nonlinear relationship between the independent and dependent variables. The characteristics of financial time series, such as equity stock or currency markets, are influenced by the psychology of traders (behavioural finance) and are strongly non-linear and hardly predictable (see Maknickiene et al. [2011]).

6.3.2 Describing the models

Given a time series $\{x_t, x_{t-1}, ...\}$, we aim to learn from the known data to predict future values. That is, we want to estimate x at some future time

$$\hat{x}_{t+h} = f(x_t, x_{t-1}, ..., x_{t-p}, z_1, z_2, .., z_q)$$

where h is the horizon prediction, and z_q is the qth other explanatory variable. Before training the network, the input data should be normalised into the interval $[b_L, b_H]$, with b_L and b_H being lower and upper bonds, by using the equation

$$\hat{x}_i = (b_H - b_L)\frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} + b_L \text{ for } i = 1, ..., N \tag{6.19}$$

where $\min(x_i)$ is the minimal value of all x_i, $\max(x_i)$ is the maximal value of all x_i, and $\hat{x}_i$ is the normalised value of x_i, respectively. Note, we sometime need to perform operations on the normalised data within a sample window. When comparing the results among different windows we must always convert them back into the original order of magnitude using

$$x_i = [\max(x_i) - \min(x_i)]\frac{\hat{x}_i - b_L}{(b_H - b_L)} + \min(x_i) \text{ for } i = 1, ..., N$$

As discussed in Appendix (6.2), the statistical approach to forecasting time series involves the construction of stochastic models to predict the value $\hat{x}_{t+h}$ given previous observations. One approach is to use an $ARMA(p,q)$ model, but it lacks the ability to capture the nonlinear features of financial time series. Another approach is to extend the linear models to nonlinear ones. For example, we can generalise an $AR(p)$ model to become a *nonlinear autoregressive* (NAR) model (see Connor et al. [1994])

$$x_t = h(x_{t-1}, x_{t-2}, \cdots, x_{t-p}) + e_t, \tag{6.20}$$

where h is an unknown smooth function. Assuming that $E(e_t|x_{t-1}, x_{t-2}, \cdots) = 0$ and e_t has finite second moment, then the minimum mean square error optimal predictor of x_t with the knowledge of $x_{t-1}, ..., x_{t-p}$ is the conditional mean:

$$\hat{x}_t = E(x_t|x_{t-1}, \cdots, x_{t-p}) = h(x_{t-1}, \cdots, x_{t-p}) \, , t > p$$

Alternatively, as discussed above we can use ANNs (see Appendix (11.1.2)). Lapedes [1987] showed that feedforward networks are NAR models for time series prediction. The predictor is given by the approximation of h as follow

$$\hat{x}_t = \hat{h}(x_{t-1}, \cdots, x_{t-p})$$

with

$$\hat{h}(x_{t-1}, \cdots, x_{t-p}) = \sum_{i=1}^{N_h} w_i f(\sum_{j=1}^{p} w_{ij} x_{t-j} + b_i) \tag{6.21}$$

Figure 18 shows the general architecture of the NAR as an extension of the $AR(p)$ in a feedforward network. The weights, $\{w_i\}$ and $\{w_{ij}\}$ in Equation (6.21) can be seen as knobs defining the input-output function, $\hat{h}$, of the network. N_h is the number of neurons in the second layer, f is a monotonic, smooth, and bounded function, b_i is a bias. The weights of the network, $\{w_i\}$ and $\{w_{ij}\}$ are estimated through supervised learning process given a time series sample $x_0, x_1, \cdots, x_n$, and the input-target pairs in the training set are $((x_{t-1}, x_{t-2}, \cdots, x_{t-p}), x_t)$, where $p > 0, t \geqslant p$.

More generally, to illustrate the nonlinearity of ANNs, we consider a multilayer network (see Appendix (11.1.6)) with $p = 1, .., P$ input-output training pairs where $p = P$ is the most recent observation, and assume a set of artificial neurons $\{f_{i,j,l}\}_{k=0}^{K}$ where the multilayer subscript $k = 0$ corresponds to the set of inputs $\{x_i\}_{i=1}^{N_0}$ and $k = K$ corresponds to the set of outputs $\{y_i\}_{i=1}^{N_K}$. For simplicity of exposition, we focus on a system with one hidden layer where the subscripts i and j represent the nodes on the input ($(K-2)$ layer) and the $(K-1)$th hidden layer, and the subscript s represent the nodes on the kth layer, respectively. In that setting, the output $O_{s,K}^{p}(x, w)$ can be expressed as

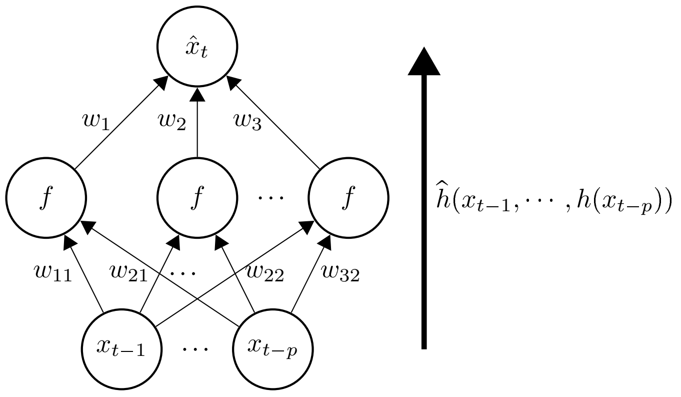


Figure 18: Feedforward network for nonlinear autoregressive model.

$$O^p_{s,K}(x,w) = g_{N_K}\Big(\sum_{j=1}^{N_{K-1}} g_{N_{K-1}}\big(\sum_{i=1}^{N_{K-2}} O^p_{i,K-2}(x,w)w^{k-2}_{i,j} + b_{j,K-1}\big)w^{k-1}_{j,s} + b_{s,K}\Big) \ , s = 1,..,N_K \tag{6.22}$$

where $O^p_{i,K-2}(x,w) = x_{p,i}$ is the ith element of the vector input, and $g_{N_K}(\cdot)$ and $g_{N_{K-1}}(\cdot)$ are two possibly different functions. In the special case where $g_{N_K}(\cdot)$ is a linear function and $g_{N_{K-1}}(\cdot)$ is a sigmoid function, the system simplifies to

$$O^p_{s,K}(x,w) = \sum_{j=1}^{N_{K-1}} g_{n_{K-1}}\big(\sum_{i=1}^{N_{K-2}} O^p_{i,K-2}(x,w)w^{k-2}_{i,j} + b_{j,K-1}\big)w^{k-1}_{j,s} + b_{s,K} \ , s = 1,..,N_K$$

which can be compared to the ordinary least squares (OLS) regression

$$y^p_s = \alpha + \sum_{i=1}^{N_{K-2}} \beta_i x_{p,i} + \epsilon_s \ , s = 1,..,N_K$$

Hence, we see that the OLS regressor variables y^p_s are put through a transformation, or squashing function, given by $g_{N_{K-1}}(\cdot)$. By iterating within sample and examining the error terms, the neural network modify the input weights $\{w^{k-2}_{i,j}\}$ and the output weights $\{w^{k-1}_{j,s}\}$ to minimise within-sample measure for point prediction models. It can also maximise the classification rate of correct predictions for classification models.
It is not difficult to see that Equation (6.22) corresponds to the rate dynamics in Equation (6.13) with nonlinear G function. That is,

$$y = g_K\Big(a + \sum_{i=1}^{p} \phi_i x_i + \sum_{j=1}^{q} \theta_j g_{K-1}\big(\alpha_j + \sum_{i\to j} \omega_{ij} x_i\big)\Big)$$

where $x_1,...,x_p$ are input values, y is the output, for the q-hidden nodes. If the activation function g_K is linear, the feedforward neural network (FFNN) becomes an autoregressive moving average model with nonlinear moving average part. Hence, the FFNN (with one hidden layer) is a generalisation of $ARMA(p,q)$ models.

6.4 Accounting for time and earning profits

In traditional time series forecasting, the criteria for assessing model performance are error functions based on the goodness of fit of target and predicted values. Since neural networks are similar to conventional regression estimators, except for their nonlinearity, error functions are also used to judge the goodness of fit of the model. However, normalised mean square error (NMSE) and other error functions are not the most appropriate measures in finance.

6.4.1 Time factor

For instance, weighting all data equally (ordinary leasy squares) are less accurate than discounted least squares, which weight the most recent data more heavily (see Makridakis et al. [1982]). Incorporating time factor to neural networks, Refenes et al. [1997] proposed the discounted least squares (DLS) neural network model. Assuming a feed-forward network with n input and a single output unit ($m = 1$) with k hidden layers, we let the ordinary least-squares criterion of the network be given by

$$E_{LS} = \frac{1}{2P} \sum_{p=1}^{P} (O_p - d_p)^2$$

where O_p and d_p denote the pth output and target value, respectively.

***Remark* 6.3** *In general, when considering a training set consisting of sequential data, or time-varying input, we represent them as $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_T, \overline{d}_T)\}$ with T ordered pairs. To be consistent with the articles described we keep the P ordered pairs, and we let $p = P$ be the most recent observation.*

Refenes et al. [1997] proposed a simple modification to the error backpropagation procedure taking into account gradually changing input-output pairs. The procedure is based on the principle of discounted least-squares whereby learning is biased towards more recent observations with long term effects experiencing exponential decay through time. The cumulative error calculated by the DLS procedure is given by

$$E_{DLS} = \frac{1}{2P} \sum_{p=1}^{P} \phi(p)(O_p - d_p)^2$$

where $\phi(p)$ is an adjustment of the contribution of observation p to the overall error. They chose the simple sigmoidal decay function

$$\phi_{DLS}(p) = \frac{1}{1 + e^{a-bp}}$$

where $b = \frac{2a}{P}$ and a is the discount rate. The learning rule is derived in the usual way by repeatedly changing the weights by an amount proportional to

$$\frac{\partial E_{DLS}}{\partial W} = \phi_{DLS}(p) \frac{\partial E_{LS}}{\partial W}$$

since the discount factor $\phi(p)$ is a function of the recency of the observation which is independent of the actual order of pattern presentation within the learning process. Hence, the algorithm is not affected by randomising the order in which patterns are presented and can be applied to both batch and stochastic update rules. Using a sine wave with changing amplitude and a non-trivial application in exposure analysis of stock returns to multiple factors, Refenes et al. [1997] compared the performance of the two cost functions on a backpropagation network with a single hidden layer of 12 units and batch update. They showed that DLS was a more efficient procedure for weakly non-stationary data series.

6.4.2 Direction measures

Further, we are usualy more interested in earning profits than in the quality of the forecast itself. When classifying returns, it is possible to use a direction measure to test the number of times a prediction neural network predicts the direction of the predicted return movement correctly. Merton [1981] proposed a modified direction given by

$$\text{Modified Direction} = \frac{\sharp \text{ of correct up predictions}}{\sharp \text{ of times index up}} + \frac{\sharp \text{ of correct down predictions}}{\sharp \text{ of times index down}} - 1$$

In general, direction is defined as the number of times the prediction followed the up and down movement of the underlying stock/index. Harvey et al. [2002] and Castiglione et al. [2001] proposed the following direction measure

$$\xi = \frac{1}{|T|} \sum_{t \in T} \mathcal{H}(R_t \hat{R}_t) + 1 - (|R_t| + |\hat{R}_t|)$$

where $R_t = \frac{P_t}{P_{t-1}}$ is the percentage return on the underlying at time step $t \in T$, $\hat{R}_t$ is the forecast percentage return, T is the number of days in the validation period, and $\mathcal{H}(\cdot)$ is the Heaviside function. Alternatively, we can write the direction measure as

$$\xi = \frac{1}{|T|} \sum_{t \in T} \mathcal{H}(\Delta P_t \Delta \hat{P}_t) + 1 - (|\Delta P_t| + |\Delta \hat{P}_{i+1}|)$$

where $\Delta P_t = P_t - P_{t-1}$ is the price change at step $t \in T$ and P_{t-1} is assumed to be known. These two direction measures provide a summary of how well the predicted time series and the actual ones move together at any point in time. However, we want to implement measurement errors taking into account the simultaneous behaviour of trend and magnitude within the trend. Hence, to reflect this point when evaluating the performance of a forecasting model, Yao et al. [1996] used the correctness of trend and a paper profit. They proposed a profit based adjusted weight factor for backpropagation network training by adding a factor containing profit, direction, and time information to the error function.

6.4.3 Time dependent direction profit

In order to take profit gain into account, Caldwell [1995] proposed a weighted directional symmetry (WDS) function which penalise more heavily the incorrectly predicted directions than the correct ones. The cumulative error function is given by

$$E_{WDS} = \frac{100}{P} \sum_{p=1}^{P} \phi(p) |O_p - d_p|$$

with

$$\phi(p) = \begin{cases} g \text{ if } \Delta d_p \times \Delta O_p \leqslant 0 \\ h \text{ otherwise} \end{cases}$$

where $\Delta d_p = d_p - d_{p-1}$ and $\Delta O_p = O_p - O_{p-1}$, and g and h are constants or some function of d_p. The values $g = 1.5$ and $h = 0.5$ were suggested. Yao et al. [1996] argued that the profit driven procedure was not only a function of the direction, but also of the amount of change. That is, the penalty on WDS weights should be increased in case of a wrongly forecasted direction for a big change of values, and the penalty should be further reduced if the direction is correctly forecasted for a big change in value. They proposed a modified directional profit (DP) adjustment factor defined as

$$\phi_{DP}(p) = \begin{cases} a_1 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } |\Delta d_p| \leqslant \sigma \\ a_2 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } |\Delta d_p| > \sigma \\ a_3 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } |\Delta d_p| \leqslant \sigma \\ a_4 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } |\Delta d_p| > \sigma \end{cases}$$

where σ is a threshold for the changes in sample values generally taken as the standard deviation of the training set

$$\sigma^2 = \frac{1}{P} \sum_{p=1}^{P} (d_p - \mu_d)^2$$

where μ_d is the mean of the target series. The values $a_1 = 0.5$, $a_2 = 0.8$, $a_3 = 1.2$, and $a_4 = 1.5$ were suggested. While the DLS model focus on a time factor $\phi_{DLS}(p)$ and the DP model focus on a profit factor $\phi_{DP}(p)$, Yao et al. [1998] [2000] combined the two approaches obtaining a time dependent directional profit (TDP) model given by

$$\phi_{TDP}(p) = \phi_{DLS}(p) \times \phi_{DP}(p)$$

The three models, DLS, DP, and TDP were tested and compared to the benchmarked OLS model on four major Asian stock indices and the US Dow Jones Industrials Index (DJ). They considered a one hidden layer network with learning rate $\eta = 0.25$ and momentum rate $\gamma = 0.9$. The structure of the neural network was $30 - 10 - 1$, that is, 30 input, 10 nodes in the hidden layer and one output. Thus, 30 consecutive days of data are fed to the network to forecast the index of the following day. Considering a paper profit measure for the model performance, they showed that the preference of DLS over OLS was of 60%, and that the preference of DP over OLS was of 80%. In this combined model, they obtained up to 27.6% excess annual return above the OLS model with at least 2.8% excess annual return in the worst case. Lu Dang Khoa et al. [2006] argued that big changes in prices between two consecutive periods are rare to occur, leading to change between two consecutive prices smaller than the standard deviation of the price. They replaced the inequality $|\Delta d_p| > \sigma$ with $\frac{\Delta d_p}{d_{p-1}} > a_5$ and obtained the following adjustment factor

$$\phi_{DPN}(p) = \begin{cases} a_1 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} < a_5 \\ a_2 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} > a_5 \\ a_3 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} < a_5 \\ a_4 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} > a_5 \end{cases}$$

with $a_5 = 0.05$. They used that model to predict the $S\&P$ 500 stock index one month in the future without much improvements compared to the traditional FFN algorithm. They considered a mixed of input based on fundamental factors such as the prices of oil, the interest rates, inflation, and technical indicators such as volatility, relative strength index, directional index etc.

7 Statistical properties of financial time series

7.1 Introduction to wavelet theory

Fitting financial time series is a difficult task as they are not stationary and contain noise. It is possible to get rid of this noise by using a filter. However, when denoising a signal we need to preserve the relevant multi-scaled information, which can be done with wavelets.
We present a short introduction to wavelets and provide some examples on denoising techniques.

7.1.1 Some definitions

Wavelet theory overcomes some problems linked to Fourier theory (such as stationarity) by generalising the decomposition to a time-frequency domain. The word wavelet (ondelette) were used by Morlet and Grossmann in the early 1980s. Started with the work of Haar in the early 20th century, Zweig discovered of the continuous wavelet transform in 1975, which were extended by Goupillaud, Grossmann and Morlet. Stromberg introduced discrete wavelets (1983) and Daubechies presented the orthogonal wavelets with compact support (1988). Wavelets were first used in physics to characterise the Brownian motion and have been widely used in diverse tasks related to financial time series analysis.

Definition 7.1 *Wavelet*
A wavelet is a square integrable function $\psi_{s,\tau} : t \in \mathbb{R} \to \mathbb{R}$*, where* $(s, \tau) \in \mathbb{R}^{*+} \times \mathbb{R}$*, such that* $\psi_{s,\tau} \in L^2(\mathbb{R})$ *and* $\int_{-\infty}^{+\infty} \psi_{s,\tau}(t)dt = 0$*.*

***Remark* 7.1** *Wavelets derived from a mother wavelet* Ψ *in the following way:* $\forall t \in \mathbb{R}, \psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\Psi(\frac{t-\tau}{s})$*. In that sense, a mother wavelet generates a subgroup* Λ *of the affine transformation group of* $\mathbb{R}^n$*.*

There are many wavelet families, each of them gathering wavelets having common properties.

7.1.1.1 Continuous Wavelet As in Fourier theory, both continuous and discrete wavelet transforms exist. We first define the continuous wavelet transform (CWT) of a signal f as follows:

$$\forall (s, \tau) \in \mathbb{R}^{*+} \times \mathbb{R} \qquad g(s, \tau) = \int_{-\infty}^{+\infty} f(t)\psi^*_{s,\tau}(t)dt$$

$*$ stands for the complex conjugate, τ is a translation factor and s is a dilatation factor. To recover f from g, the following equation is used:

$$\forall t \in \mathbb{R} \qquad f(t) = \frac{1}{C}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \frac{1}{|s|^2} g(s, \tau)\psi_{s,\tau}(t)dsd\tau$$

where $C = \int_{-\infty}^{+\infty} \frac{|\hat{\Psi}(x)|}{|x|}dx$, $\hat{\Psi}$ being the Fourier transform of the mother wavelet Ψ.

An example of such wavelet is the Mexican hat wavelet. The continuous wavelet transform defines a priori an infinite number of coefficients which cannot be used in practice. For that reason, we will only make use of discrete wavelet transform (DWT) introduced below.

7.1.1.2 Discrete Wavelet Consider a discrete time set $\mathbb{T} = [\![0, T-1]\!]$. We define a discrete wavelet on this set as follows:

$$\forall t \in \mathbb{T} \qquad \psi_{m,n}(t) = s_0^{-m/2}\psi(s_0^{-m}t - n\tau_0)$$

where s_0 and τ_0 are constant. Then, the scaling and wavelet functions of DWT are defined as follows:

$$\phi(2^m t) = \sum_{i=1}^{n} h_{m+1}(i)\phi(2^{m+1}t - i)$$

$$\psi(2^m t) = \sum_{i=1}^{n} g_{m+1}(i)\phi(2^{m+1}t - i)$$

where $(h_{m+1}(i))_{i\in[\![1,n]\!]}$ and $(g_{m+1}(i))_{i\in[\![1,n]\!]}$ are respectively the coefficient sequences of ϕ and ψ. From these functions, a discrete signal x can be written as:

$$x(t) = \sum_{i=1}^{n} \lambda_{m-1}(i)\phi(2^{m-1}t - i) + \sum_{i=1}^{n} \gamma_{m-1}(i)\psi(2^{m-1}t - i)$$

If $\{\psi_{m,n} : m, n \in \mathbb{Z}\}$ defines an orthonormal basis of $L^2(\mathbb{R})$, the following formula is used in order to recover the signal x.

$$\forall t \in \mathbb{T} \qquad x(t) = \sum_{m\in\mathbb{Z}}\sum_{n\in\mathbb{Z}} \langle x, \psi_{m,n}\rangle \cdot \psi_{m,n}(t)$$

Remark 7.2 *Orthonormality is an interesting property but not necessarily required for wavelets' construction. Compact support is also an useful property wavelets have as they can calibrate data efficiently, though it is not required neither.*

Remark 7.3 *Using a set of wavelets having the orthonormal property makes it easier to recover a discrete signal from its coefficients* $(\langle x, \psi_{m,n} \rangle)_{m,n}$*. This is the case of Haar wavelets for example.*

Wavelets take care of temporal characteristics of the regular factors described earlier as they are defined in a two dimensional domain: frequency and time resolution combinaison. Wavelets are indeed small waves located at different times and therefore give significant information about both time and frequency domains. This new degree of freedom provides the ability to capture trends in a local fashion.

7.1.2 Decomposition schemes

We introduce three decomposition schemes standing for different approaches of the wavelet transform introduced earlier.

7.1.2.1 Discrete Wavelet Transform DWT is the most basic wavelet decomposition, Figure (19) shows how it can be done. H, L and H', L' respectively stands for high-pass, low-pass filters for wavelet decomposition and reconstitution. In the decomposition phase, L and H remove respectively the higher frequencies and the lower frequencies to give an approximation and a detail of signal x. The result is then downsampled by 2. The right part of the figure denotes the reserve process and is known as the reconstruction phase.

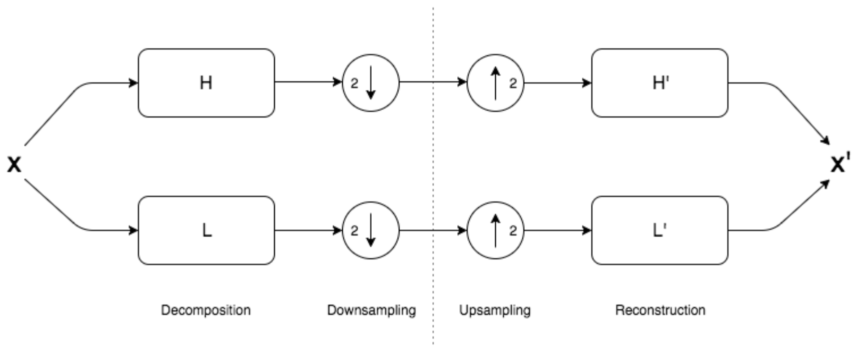


Figure 19: DWT Filter Scheme

More generally, we can repeat this process n times by a cascade algorithm which decomposes and downsamples the latter approximation. The following figure describes the process. A signal x can then be decomposed at level n in the following fashion:

$$x(t) = A_n(t) + \sum_{k=1}^{n} D_k(t) \tag{7.23}$$

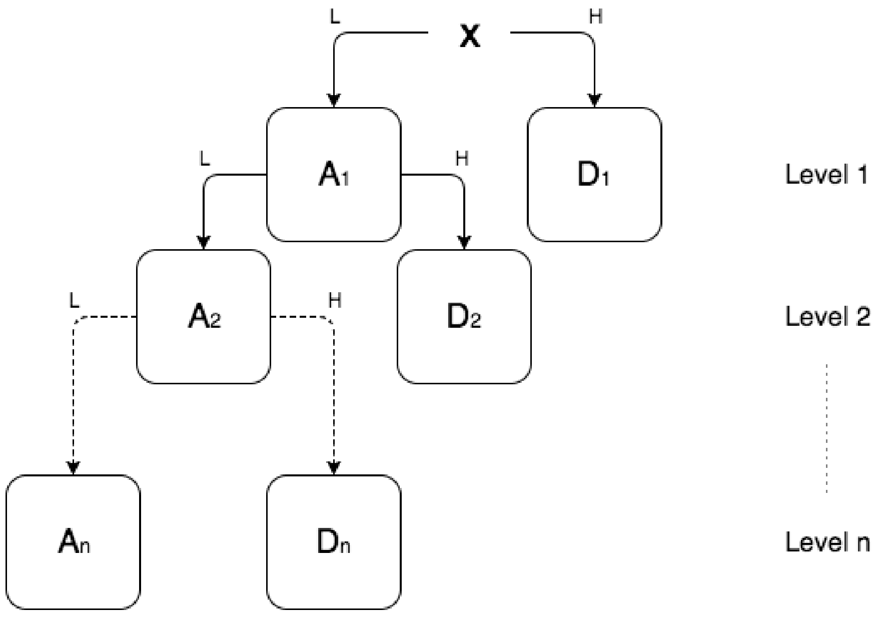


Figure 20: DWT Decomposition Tree

The main drawback of DWT is to be time-variant. For example, if we consider a periodic signal x with period τ, the DWT of x on $[\![t-\tau, t]\!]$ is in general not the DWT of x on $[\![t, t+\tau]\!]$ due to the downsampling. This property being desirable, researchers have developed a new type of wavelet transform called stationary wavelet transform (SWT) which guarantees time-invariance.

7.1.2.2 Stationary Wavelet Transform In the DWT framework the downsampling meant choosing even or odd indexed elements at every decomposition step. However, in general, we could choose between even and odd at every decomposition which would lead to many possible combinaisons. The cornerstone of SWT is to average these denoised signals. We choose 0 to be coinciding with an even downsampling and 1 with an odd one.

Definition 7.2 *ϵ-decimated DWT*
Let N be a decomposition level of a signal x and $\epsilon = (\epsilon_1, ..., \epsilon_N)$ a sequence of integers such that $\forall n \in [\![1, N]\!]$, $\epsilon_n \in \{0, 1\}$. We say that the DWT $\chi = (A_N, D_N, .., D_1)$ is ϵ-decimated DWT if $\forall n \in [\![1, N]\!]$, ϵ_n corresponds to the downsampling choice of χ at time n.

It is possible to calculate all the ϵ-decimated DWTs for a given finite signal x by computing χ for every possible sequence ϵ. However, doing so is very computationally inefficient as 2^N DWT need to be done for a given decomposition level N. Another way to do this is to avoid downsampling and perform an upsampling on the original filters H and L.

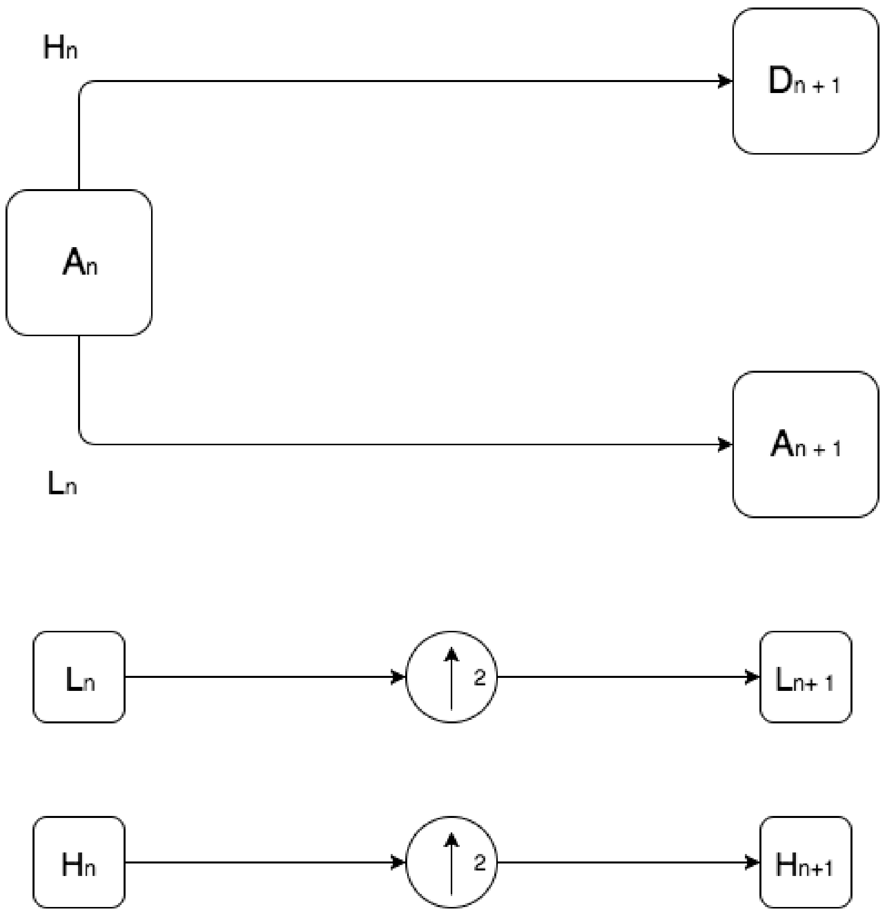


Figure 21: SWT calculation at level n

By following this process we end up with a time-invariant wavelet transform χ. In the next part, we describe a more general decomposition which is worth being investigated.

7.1.2.3 Wavelet Packet Transform WPT can be seen as an extension of DWT. The latter decomposes only the approximation coefficients while WPT decomposes both the approximation and detail coefficients. This leads to more complex and somehow more flexible decomposition. Figure (22) describes its process:

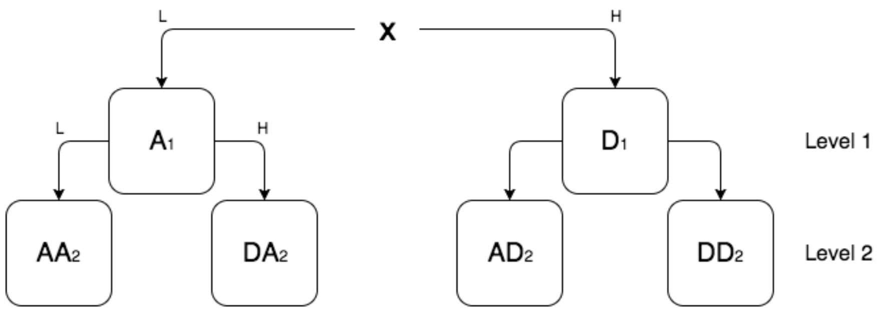


Figure 22: WPT Decomposition Tree

However, this process leads to 2^N coefficients for a given decomposition level N and thus can be more cumbersome in terms of storage and computation processing.

7.1.3 Denoising techniques

In this section we explain how DWT, SWT and WPT can be used to denoise a given signal. We say that a signal is noised when it suffers from unwanted or unknown modifications. Although we can have an intuitive understanding of noise, particularly in financial time series, there cannot be clear definition of it. Consequently, researchers often model it. A common model used in the research papers is the Gaussian white noise which has the characteristic to have the same power spectral density at every bandwidth frequency.

Definition 7.3 *Power spectral density*
Let x a signal and X its Fourier transform. We define the power spectral density (PSD) of the signal x as $\Gamma_x = |X|^2 * T$.

Definition 7.4 *Gaussian white noise*
A signal x is called a Gaussian white noise process if x is a stationary Gaussian process such that its mean $\mu_x = 0$ and its PSD is constant.

From theses definition we state our goal: from a noised signal x defined on a domain D and carrying a Gaussian white noise ϵ, we wish to find the true signal x^* such that:

$$\forall t \in D \qquad x(t) = x^*(t) + \epsilon(t) \tag{7.24}$$

In the previous section, we have introduced the concept of wavelet decomposition leading to a set of approximation and detail coefficients. As the noise in a signal is generally contained in the detail ones, a common procedure consists in setting the smallest coefficients to zero. Given a threshold $\lambda \in \mathbb{R}^+$, we sum up the many possibilities below.

soft	hard
$x \to \begin{cases} x - \lambda & \text{if} \quad x > \lambda \\ 0 & \text{if} \; \lvert x \rvert \leqslant \lambda \\ x + \lambda & \text{if} \quad x < -\lambda \end{cases}$	$x \to \begin{cases} 0 & \text{if} \; \lvert x \rvert < \lambda \\ x & \text{otherwise} \end{cases}$
greater	**less**
$x \to \begin{cases} 0 & \text{if} \; x < \lambda \\ x & \text{otherwise} \end{cases}$	$x \to \begin{cases} 0 & \text{if} \; x > \lambda \\ x & \text{otherwise} \end{cases}$

Table 5: Threshold Function Definitions

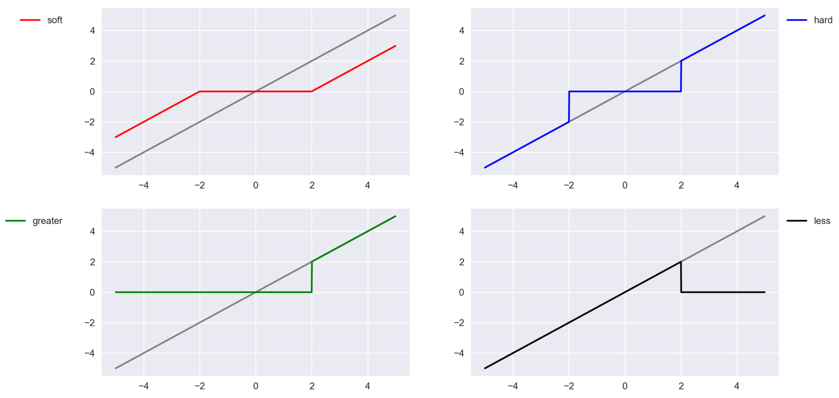


Figure 23: Thresholding functions ($\lambda = 2$)

It is a well-known fact that soft thresholding provides smoother results when comparing to other ones. However, hard thresholding provides better edge preservation.

Algorithm (1) defines our filtering process for Discrete Wavelet Transform and has been used in our experiments to denoise a signal.
Filtering processes for other wavelet decomposition methods are very similar, the general idea being thresholding detail coefficients.

Algorithm 1 DWT filtering process

```
Require: get noised signal $x$
Require: set wavelet name $wlt$
Require: set decomposition level $n$
Require: set thresholding function $f$ and level $\lambda$
1: $[A(n), D_n, D_{n-1}, .., D_1] \leftarrow$ decomposition($x$, $wlt$, $n$)
2: for $i$ = 1 to $n$ do
3:     $D_i \leftarrow f(D_i, \lambda)$
4: end for
5: $x^* \leftarrow$ reconstruction($[A, D]$, $wlt$)
6: return $x^*$
```

7.2 Fractal time series

A self-similar process is characterised by its Hurst coefficient H (see Hurst [1951]). The Brownian motion, the L-stable process (see Mandelbrot [1963]), and the fractional Brownian motion (fBm) are the main examples of self-affine processes in finance. The value of the Hurst Exponent varies between 0 and 1, with $H = \frac{1}{2}$ implying a random walk or an independent process (see Figure 24). Note, for $H \neq \frac{1}{2}$ we can observe two phenomenon

1. the Noah effect: rare events, fat tail distributions

2. the Joseph effect: observe long stretches of time when the process is above or below the mean

These phenomenon correspond respectively to

1. $0 < H < \frac{1}{2}$, short term memory: we have anti-persistence (or ergodicity) where the process covers less distance than a random walk. It is often referred as mean reverting, meaning that if the system has been up in the previous period, it is more likely to be down in the next period. The converse being true. Anti-persistent time series exhibit higher noise and more volatility than a random series because it consists of frequent reversals.

2. $\frac{1}{2} < H \leqslant 1$ long term memory, or long range dependence (LRD): we have persistence (or trend-reinforcing) where the process covers more distance than a random walk. A persistent series has long memory effects such that the trend at a particular point in time affects the remainder of the time series. If the series has been up (down) in the last period, then the chances are that it will continue to be positive (negative) in the next period.

Figure (25) illustrates the role of the Hurst exponent. If our goal is to forecast returns from a time series by learning some of its behaviour in the hope it repeats in the future, it does not make much sense to consider returns in the case (1). We are left with focusing on momentum (case (2)).

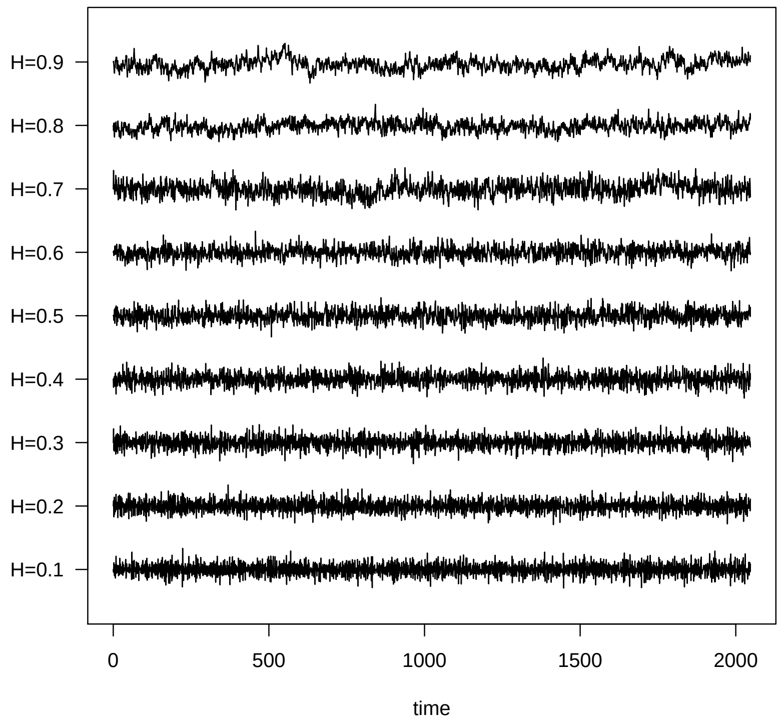
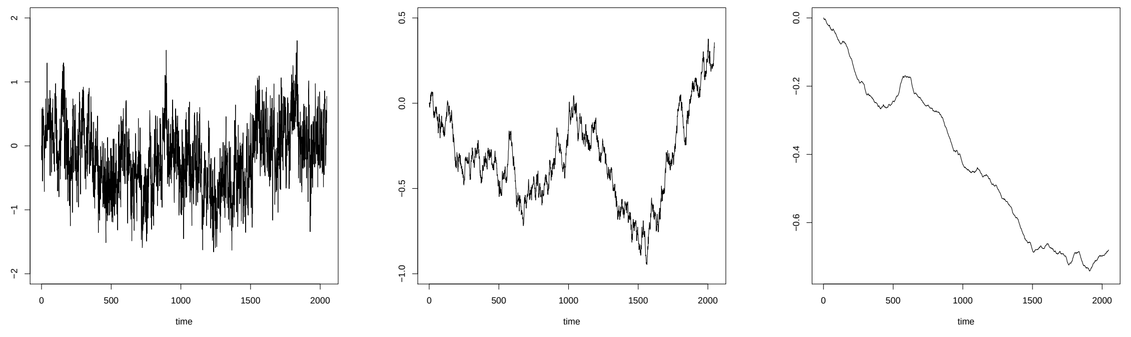


Figure 24: Range of Hurst exponents between 0.1 and 0.9



(a) $H = 0.1$ (b) $H = 0.5$ (c) $H = 0.9$

Figure 25: Cumulative observations for different values of H

The fractal dimension D of a time series measures how jagged the time series is. Mandelbrot showed that the Hurst Exponent, H, is related to the fractal dimension D for a self-similar surface in n-dimensional space by the relation

$$D = n + 1 - H \tag{7.25}$$

where $0 \leqslant H \leqslant 1$. While the fractal dimension of a line is 1.0 and the fractal dimension of a geometric plane is 2.0, the fractal dimension of a random walk is half way between the line and the plane at 1.5. A value of $\frac{1}{2} < H \leqslant 1$ results in a fractal dimension closer to a line, so that a persistent time series would result in a smoother, less jagged line than a random walk. Similarly, for $0 < H < \frac{1}{2}$, a time series is more jagged than a random series, or has more reversals.
The Hurst Exponent can be estimated by

$$H = \frac{\log\left(\frac{R}{S}\right)}{\log(T)}$$

where T is the duration of the sample data, and $\frac{R}{S}$ is the corresponding value of the rescaled range.
Persistent series are fractional Brownian motion, or biased random walk with the strength of the bias depending on how far H is above $\frac{1}{2}$. For $H \neq \frac{1}{2}$ each observation carry a long-term memory, theoretically lasting forever. A system exhibiting Hurst statistics is the result of a long stream of interconnected events where time is important as the future depends on the past.
Since the development of the rescaled range analysis (R/S), a large number of fractal quantification methods called Wavelet Analysis (WA) and Detrending Methods (DMs) have been proposed to accomplish accurate and fast estimates of the Hurst exponent H in order to investigate correlations at different scales in dimension one. While the R/S analysis makes the rough approximation that the trend is the average in each segment under consideration, the former uses spectral analysis, and the latter considers more advanced methods to detrend the series. Examples of the latter are detrended fluctuation analysis (DFA) (see Peng et al. [1994]) and detrending moving average analysis (DMA). There exists many more methods for estimating the Hurst exponent such as the aggregated variance, the differenced variance, absolute values, boxed periodigram, Higuchi, modified periodigram, Whittle and local Whittle methods (see Fox et al. [1986]).
Wavelet Transforms can be used to characterise the scaling properties of self-similar fractal and multifractal objects (see Muzy et al. [1991]). For instance, Abry et al. [1993] [1998] proposed a methodology for estimating the Hurst exponent.

7.3 Multifractal time series

7.3.1 Multifractal analysis

As explained by Kantelhardt et al. [2002], many series do not exhibit a simple monofractal scaling behaviour described with a single scaling exponent. Alternatively, the scaling behaviour may be more complicated, so that different scaling exponents are required for different part of the series. For instance, the scaling behaviour in the first half of the series differ from that in the second half. In complex system, such different scaling behaviour can be observed for many interwoven fractal subsets of the time series, in which case a multitude of scaling exponents is required for a full description of the scaling behaviour, and a multifractal analysis must be applied. Thus, multifractality, or anomalous scaling, allows for a richer variation of the behaviour of a process across different scales. This variability of scaling laws can be translated into the variability of the Hurst index, which is no-longer constant. Abry et al. [2002] and Wendt [2008] gave a synthetic overview of the concepts of scale invariance, scaling analysis, and multifractal analysis.
To do so, we need to adopt a more local viewpoint and examine the regularity of realised paths (of stochastic process X) around a given instant. Contrary to Scaling Analysis (SA), which concentrate on the scaling exponent $\tau(q)$, the multifractal analysis (MFA) studies how the (pointwise) local regularity of X fluctuates in time (or space). Local Holder regularity describes the regularity of sample paths of stochastic processes by means of a local comparison against a power law function and is therefore closely related to scaling in the limit of small scales. The exponent of this power law, $h(t)$, is called the (local) Holder exponent and depends on both time and the sample path of X.
Multifractal situations happen when the Holder exponent is no-longer unique, but vary from point to point. More precisely, when the regularity $h(t)$ is itself a highly irregular function of t, possibly even a random process rather than

a constant or a fixed deterministic function, the process X is said to be multifractal. The aim of multifractal analysis (MA) is to provide a description of the collection of Holder exponents h of the function f.
However, since the Holder exponent may jump from one point to another, it was understood that describing them for each time step was not of much importance. Hence, researchers focused on a global description of the regularity of the function of f in form of Multifractal Spectrum (also called the singularity spectrum) reflecting the size of the set of points for which the Holder exponent takes a certain value h. The measure of size most commonly used is the Hausdorff dimension which gives rise to the Hausdorff spectrum (called the multifractal spectrum). It describes the collection of Holder exponents $h(t)$ by mapping to each value of h the Hausdorff dimension $D(h)$ of the collection of points t_i at which $h_f(t_i) = h$. The main idea being that the relative frequency of the local exponents can be represented by a renormalised density called the multifractal spectrum.
These multifractal signals can only be described in terms of an infinite set of exponents and their density distribution. The characterisation of a multifractal process, or measure, by a distribution of local Holder exponents underlines its heterogeneous nature with alternating calm and turbulent phases.

Since scaling analysis and multifractal analysis developed, various authors performed empirical analysis to identify anomalous scaling in financial data. For instance, Calvet et al. [2002] investigated the MMAR model, which predicts that the moments of returns vary as a power law of the time horizon, and confirmed this property for Deutsche mark/US dollar exchange rates and several equity series. Kantelhardt et al. [2002] proposed the multifractal detrended fluctuation analysis (MF-DFA) as a generalisation of the detrended fluctuation analysis (DFA), with the advantage that it can be used for non-stationary multifractal data. Gu et al. [2010] extended the DMA method to multifractal detrending moving average (MFDMA) in order to analyse multifractal time series and multifractal surfaces.
Considering wavelet decomposition tools, Mallat et al. [1990], Muzy et al. [1991], and Arneodo et al. [1991] generalised the multifractal formalism to singular signals. DFA and its variants are methods used for analysing the behaviour of the average fluctuations of the data at different scales after removing the local trends. Further, wavelet transforms (WT) are well adapted to evaluate typical self-similarity properties. As a result, some authors (see Murguia et al. [2009], Manimaran et al. [2009]) merged the two approaches obtaining powerful tools for quantifying the scaling properties of the fluctuations. All multifractal analyses of a response time series follow a step-wise procedure, which was described by Ihlen [2013].

7.3.2 Interpretation

The generalised Hurst exponent $h(q)$ is a decreasing function of q for multifractal time series, and a constant for monofractal processes. The scaling exponent $h(q)$ is only one of several types of scaling exponents used to parametrise the multifractal structure of time series. Multifractal series are also described by the singularity spectrum $f(\alpha)$ through the Legendre transform, where $f(\alpha)$ denotes the fractal dimension of the series subset characterised by the singularity strength, or Holder exponent, α. The spectrum describes the distribution of the Holder exponents. For monofractal signals, the singularity spectrum produces a single point in the $f(\alpha)$ plane, whereas multifractal processes yield a humped function (see Figure 26). The plot of the singularity strength (exponent) $\alpha(q)$ versus the singularity spectrum $f(\alpha)$ is referred to as the Multifractal Spectrum. The multifractal time series has multifractal exponent $\tau(q)$ with a curved q-dependency, and a decreasing singularity exponent $\alpha(q)$. That is, the resulting multifractal spectrum is a large arc where the difference between the maximum and minimum $\alpha(q)$ are called the multifractal spectrum width (or degree). The Hurst exponent defined by the monofractal DFA represents the average fractal structure of the time series and is closely related to the central tendency of multifractal spectrum. The deviation from average fractal structure for segments with large and small fluctuations is represented by the multifractal spectrum width. Thus, each average fractal structure in the continuum of Hurst exponents has a new continuum of multifractal spectrum widths representing the deviations from the average fractal structure. Moreover, the shape of the multifractal spectrum does not have to be symmetric, it can have either a left or a right truncation originating from a levelling of the q-order Hurst exponent for positive or negative q's, respectively. The spectrum will have a long left tail when the time series have a multifractal structure being insensitive to the local fluctuations with small magnitudes, but it will have long right tail when the structure is insensitive to the local fluctuations with large magnitudes. Consequently, the width and shape of the multifractal spectrum can classify a wide range of different scale invariant structures of time series.

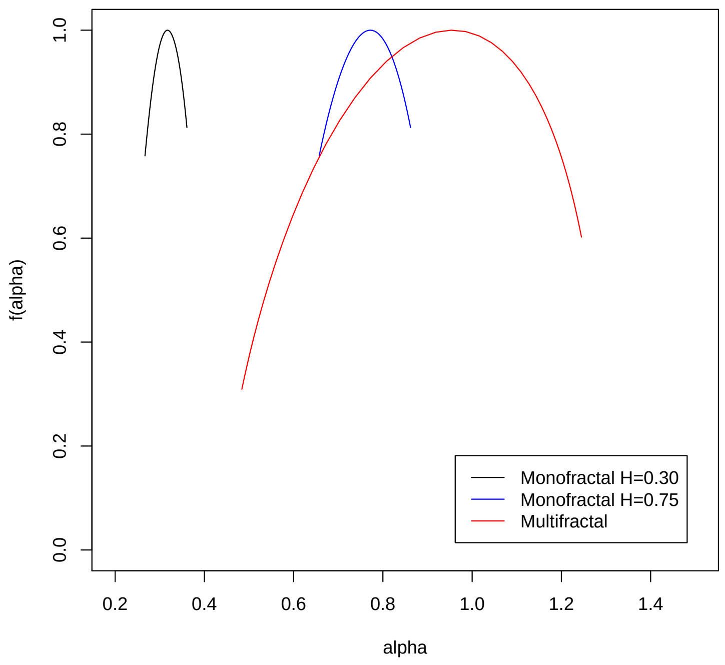


Figure 26: Multifractal spectrum $(\alpha(q), f(\alpha))$. The multifractral spectrum is the log-transformation of the normalised probability distribution of local Hurst exponents. Therefore, when the width of the spectrum is wide (red), the series is multifractal. On the contrary, when the width of the spectrum is small (black and blue), the series is monofractal. In this case, the peak is reached on the Hurst exponent.

7.3.3 Local Holder exponent estimation methods

There are cases when local information about scaling provides more relevant information than the global spectrum. It generally happens for time series where the scaling properties are non-stationary, due to intrinsic changes in the signal scaling characteristics, or even boundary effects. Until recently, the estimation of both local singularity and their spectra were very unstable and prone to gross numerical errors. However, using the wavelet transform multiscale decomposition, Struzik [1999] proposed stable procedures for both the local exponent and its global spectrum. This is the effective local Holder exponent (ELHE), which is a model-based approximation of the local scaling exponent. Later, Turiel et al. [2006] defined a scale-dependent measure with the help of a continuous wavelet transformation called the gradient modulus wavelet projection (GMWP).
We now briefly introduce the fluctuation analysis. We saw above that some researchers tried to estimate the local Hurst exponent at time t as the Hurst exponent computed on a sliding window of size n from $t-n+1$ till t. However, all methods estimating the Hurst exponent work well only in a large size window, and are therefore not capable of detecting local jumps. Following Struzik [2000], who combined a multiplicative cascade model with the wavelet transform modulus maxima (WTMM) method to estimate the local Hurst exponent, Ihlen [2012] detailed a method that retrieve the local Hurst exponent from a small window size using detrended fluctuation analysis (DFA). That is, rather than using the WTMM tree for defining the partition function-based multifractal formalism, he considered the

DFA model, obtaining a good estimate of the local Hurst exponent.
Bloch [2014] tested for the validity of the local Hurst exponent observed at time t and computed as the Hurst exponent on a sliding window against the one computed with the local detrended analysis (LDA). The former is called "Local" Hurst exponent with double quotes and the latter Local Hurst exponent without any quote. The inability for the "Local" Hurst exponent computed on a sliding window to detect large local changes of fractal structure was easily explained. Further, the equity and FX markets were successfully tested for multifractality.

7.4 Applications to finance

7.4.1 Introducing time-dependent Hurst exponent

Even though the Hurst exponent can only be estimated together with a confidence interval, the stability of the Hurst exponent is related to the fact that a process is self-similar only when the coefficient H is well defined. Examining the stability of H on financial time series on the basis of characteristic values, such as rescaled ranges or fluctuations analysis, some authors (see Vandewalle et al. [1998c], Costa et al. [2003], Cajueiro et al. [2004], Grech [2005]) observed that the values of H could be significantly higher or lower for a specific scale. For example, using the DFA method, Costa et al. [2003] analysed the behaviour of the Sao Paulo stock exchange Ibovespa index covering over 30 years of data from 1968 till 2001, amounting to $8,209$ trading days. For the complete time series they obtained $H = 0.6$, indicating persistence in the Ibovespa index. They also computed the local Hurst exponent with a three year sliding window, and found that $H(t)$ varied considerably over time. They found that the local exponent was always greater than $\frac{1}{2}$ before 1990, and then rapidly dropped towards $\frac{1}{2}$ and stayed there afterwards. They interpreted this phenomena as a consequence of the economic plan adopted in March 1990, corresponding to structural reforms in the Brazilian economy. Separate analysis of the two periods gave $H = 0.63$ for the former and $H \approx \frac{1}{2}$ for the latter, indicating multifractality of the process.

Several authors proposed to use methods of long-range analysis such as DFA or DMA to determine the local correlation degree of the series by calculating the local scaling exponent over partially overlapping subsets of the analysed series (see Vandewalle et al. [1998c], Costa et al. [2003], Cajueiro et al. [2004], Carbone et al. [2004], Matos et al. [2008]). The proposed technique examines the local Hurst exponent $H(t)$ around a given instant of time. The ability of DFA and DMA algorithm to perform such analysis relies on the local scaling properties of the scale-dependent measure. Following this method, we can calculate the exponent, $H(t)$, without any a priori assumption on the stochastic process and on the probability distribution function of the random variables entering the process. As a result, one can deduce trading strategies from these instabilities of the Hurst exponent H, such as optimal investment horizon (see Lo [1991]). However, the existence of such optimal investment horizons contradicts both EMH and FMH. In the former, it implies potential predictability of the market, while in the latter it implies that self-similar structure of the market breaks down, turning the market into a spiral. Consequently, these authors introduced time-dependent Hurst exponent for which the examination of different scales solely is not possible. In that setting, we can focus on significant changes in values of Hurst exponent as such changes imply significant shift of rescaled ranges or fluctuations at either low or high scales.

7.4.2 Time and scale Hurst exponent

We saw above that one way of generalising the global behaviour of the Hurst exponent was to estimate the local Hurst exponent $H(t)$ for fixed size widows, N_s, in view of studying its time dependency. Exploring the dichotomy between emerging and mature markets, Matos et al. [2008] considered the local Hurst exponent $H(t, N_s)$ as a function of both time t and scale N_s. To do so, they applied DFA on the interval $(t - \frac{N_s}{2}, t + \frac{N_s}{2})$ in order to recover a power of the scale inside each sub-interval. The technique called time and scale Hurst exponent (TSH) is performed with maximum scale $N_{s_{max}} = \frac{T}{4}$. In that framework $H(t, N_s)$ is the focus of the analysis, and the DFA or DMA are implementation details. Its goal is to provide a map of the evolution of markets to maturity, including significant episodes (major events affecting markets), but also gradual changes in response times. Considering daily data for a set of worldwide market indices from America, Asia, Africa, Europe and Oceania, they observed several notable features

- They can distinguish mature markets by the stability of H values around $\frac{1}{2}$, and emergent markets by the stability of H values above $\frac{1}{2}$.
- For some periods, a phase transition occurs, sometimes observable across all scales, sometimes across partial scales only. This is reflected by spikes occurring for both lower and larger scales.
- They expected smooth variations of H for large scales since it takes more data into account leading to greater robustness to sudden changes of the data. It was confirmed empirically.
- Some markets evolve in time, where they observed a shift from emergent to mature features. The other markets slowly decreased in the values of the Hurst exponent.
- Significant events, causing marked change in the Hurst exponent behaviour was seen in almost all markets.

We see that studying the Hurst behaviour for multiple time and scale intervals gives more refined details on the time series. Matos et al. proposed a refined classification of financial markets in three states

- mature markets: they have H around $\frac{1}{2}$. The presence of regions with higher values of H is limited to small periods and well defined both in time and scale.
- emergent markets: they have H well above $\frac{1}{2}$. The presence of regions with values of H around $\frac{1}{2}$ is well defined both in time and in scale.
- hybrid markets: unlike the two other cases, the distinction between the mature and emergent phases is not well determined, with behaviour seemingly mixing at all scales.

8 Introducing technical analysis

8.1 Defining technical analysis

8.1.1 Pattern analysis

As told by Mandelbrot [1982], the question of the predictability of financial markets is an old one, as financial newspapers have always presented analysis of charts claiming that they could predict the future from the geometry of those charts. It was the start of what was called pattern analysis (PA).

Definition 8.1 *Patterns*
Patterns are the distinctive formations created by the movements of security prices on a chart. A pattern is identified by a line that connects common price points, such as closing prices or highs or lows, during a specific period of time. Chartists seek to identify patterns as a way to anticipate the future direction of a security's price. Trading patterns can occur at any point or measure in time. While price patterns may be simple to detect in hindsight, spotting them in real time is a much larger challenge.

The simplest example of a pattern is the next period direction, which concerns the direction of the price movement with respect to the next period. In that case we classify each record as Up or Down. Patterns can be based on seconds, minutes, hours, days, months or even ticks and can be applied to bar, candlestick and line charts.
The most basic form of chart pattern is a trend line.

- Trend lines: A trend can often be found by establishing a line chart. A trend line is the line formed between a high and a low. If that line is going up, the trend is up. If the trend line is sloping downward, the trend is down. Trend lines are the foundation for most chart patterns. They are also useful for finding support and resistance levels, which can also be discovered through pattern recognition. A line of support is a level a stock price won't go under; a line of resistance is a point the stock won't go above.

- Pattern types: There are two basic types of patterns: continuation and reversal.
    1. Continuation patterns identify opportunities for traders to continue with the trend. These are retracements or temporary consolidation patterns. The most common continuation patterns include ascending and descending triangles, flag patterns, pennant patterns and symmetrical triangles.
    2. The opposite of continuation patterns is reversal patterns. These are employed to find favourable opportunities to base a trade on the reversal of a trend. In other words, reversal patterns seek to unearth where trends have ended. Common reversal patterns are double tops and bottoms, head-and-shoulders patterns and triple tops and bottoms.

Pattern analysis led to the development of technical indicators defined as mathematical transformations of market returns. Investopedia describes these indicators as any class of metrics whose value is derived from generic price activity. Each class of indicator focuses on a different aspect of market activity. Technical indicators evaluate price levels, volume, direction and momentum amongst others. The library TA-Lib, designed in the form of an open source API, provides over 200 technical indicators.

8.1.2 Technical analysis

The accumulating evidence against the efficiency of the market has caused a resurgence of interest in the claims of technical analysis as the belief that the distribution of price dynamics being totally random is now being questioned. Mandelbrot [1963] was the first to demonstrate that empirical data are distinctly non-Gaussian, exhibiting excess kurtosis and higher probability mass in the center and in their tails than the normal distribution. Since this seminal article, a large body of work developed on the mathematical analysis of the behaviour of stock prices, stock markets and successful strategies for trading in these environments. As a result, there are two primary types of stock analysis: fundamental and technical.

1. Fundamental analysis looks at the specifics of a company's business, conducting research on earnings projections, balance sheets, price-to-book ratios and much more.
2. Technical analysis is mostly involved with pattern recognition, regardless of performance. These patterns are then used to uncover pricing trends.

Fundamental analysis can help determine what to buy, while technical analysis can help determine when to buy. Technical analysts use chart patterns to find trends in the movement of a company's stock price. While investing involves the study of the basic market fundamentals, which may take several years to be reflected in the market, trading involves the study of technical factors governing short-term market movements together with the behaviour of the market. Consequently, trading is riskier than long-term investing, but it offers opportunities for greater profits (see Hill et al. [2000]). Recent research finds that the vast majority of fund managers use technical analysis and it is preferred to fundamental analysis (see Menkhoff [2010]).

Technical analysis is about market traders studying market price history with the view of predicting future price changes in order to enhance trading profitability (see Lo et al. [2010]). Technical trading rules involve the use of technical analysis to design indicators that help a trader determine whether current behaviour is indicative of a particular trend, together with the timing of a potential future trade. As a result, in order to apply Technical Analysis, which tries to analyse the securities past performance in view of evaluating possible future investments, we must assume that the historical data in the markets forms appropriate indications about the market future performance. Technical Analysis relies on three principles (see Murphy [1999])

1. market action discounts everything (all information available in the market)
2. prices move in trends or are contrarian (prices are not random)

3. history tends to repeat itself

Hence, by analysing financial data and studying charts, we can anticipate which way the market is most likely to go. That is, even though we do not know when we pick a specific stock if its price is going to rise or fall, we can use technical indicators to give us a future perspective on its behaviour in order to determine the best choice when building a portfolio. Technical indicators try to capture the behaviour and investment psychology in order to determine if a stock is under or overvalued. For instance, in order to classify each stock within the market, we can employ a set of rules based on technical indicators applied to the asset's prices, their volumes, and/or other financial factors. Based on entry/exit signals and other plot characteristics, we can define different rules allowing us to score the distinct stocks within the market and subsequently pick the best securities according to the indicator employed. However, there are several problems occurring when using technical indicators. There is no better indicator, so that the indicators should be combined in order to provide different perspectives. Further, a technical indicator always need to be applied to a time window, and determining the best time window is a complex task. For instance, the problem of determining the best time window can be the solution to an optimisation problem (see Fernandez-Blanco et al. [2008]).

8.1.3 Mechanical trading systems

New techniques combining elements of learning, evolution and adaptation from the field of Computational Intelligence developed, aiming at generating profitable portfolios by using technical analysis indicators in an automated way. In particular, subjects such as Neural Networks (28), Swarm Intelligence, Fuzzy Systems and Evolutionary Computation can be applied to financial markets in a variety of ways such as predicting the future movement of stock's price or optimising a collection of investment assets (funds and portfolios). These techniques assume that there exist patterns in stock returns and that they can be exploited by analysis of the history of stock prices, returns, and other key indicators (see Schwager [1996]). With the fast increase of technology in computer science, new techniques can be applied to financial markets in view of developing applications capable of automatically manage a portfolio. Consequently, there is substantial interest and possible incentive in developing automated programs that would trade in the market much like a technical trader would, and have it relatively autonomous. A mechanical trading systems (MTS), founded on technical analysis, is a mathematically defined algorithm designed to help the user make objective trading decisions based on historically reoccurring events. Some of the reasons why a trader should use a trading systems are

- continuous and simultaneous multimarket analysis
- elimination of human emotions
- back test and verification capabilities

Mechanical system traders assume that the trending nature of the markets can be understood through the use of mathematical formulas. For instance, properly filtering time series by removing the noise (congestion), they recover the trend which is analysed in view of inferring trading signals. Assuming that assets are in a continuous state of flux, a single system can profitably trade many markets allowing a trader to be exposed to different markets without fully understanding the nuances of all the individual markets. Since MTS can be verified and analysed with accuracy through back testing, they are very popular. Commodity Trading Advisors (CTA) use systems due to their ease of use, non-emotional factor, and their ability to be used as a foundation for an entire trading platform. Everything being mathematically defined, a CTA can demonstrate a hypothetical track record based on the different needs of his client and customise a specific trading plan. However, one has to make sure the the system is not over-fitting the data in the back test. A system must have robust parameters, that is, parameters not curve fitted to historical data. Hence, one must understand the logic and mathematics behind a system. Unlike actual performance records, simulated results do not represent actual trading. As a rule of thumb, one should expect only one half of the total profit and twice the maximum draw down of a hypothetical track record.

8.2 Presenting a few trading indicators

Any mechanical trading system (MTS) must have some consistent method or trigger for entering and exiting the market based on some type of indicator or mathematical statistics with price forecasting capability. Anything that indicates what the future may hold is an indicator. In general, time series analysis is used to examine data for repeating patterns by collecting data on a system over time in order to analyse a system or to predict future trends. For instance, researchers consider serial correlation and run-tests to test if there are significant price patterns (see Damodaran [2003]).
Most system traders and developers spend 90% of their time developing entry and exit technique, and the rest of their time is dedicated to the decision process determining profitability. Some of the most popular indicators include moving average (MA), weighted moving average (WMA), rate of change, momentum, stochastic (Lane 1950), Relative Strength Iindex (RSI) (see Wilder [1978]), moving average convergence divergence (MACD) (see Appel [1999]), Donchian breakout, Bollinger bands (BB) (see Bollinger [1992]), Keltner bands (see Keltner [1960]), Commodity Channel Index (CCI) (see Lambert [1980]), Chaikin accumulation/distribution (ChaikinAD), triple smoothed exponential oscillator (TRIX). However, indicators can not stand alone, and should be used in concert with other ideas and logic. There is a large list of indicators with price forecasting capability and we are now going to describe a few of them. For more details we refer the reader to Murphy [1999], Hill et al. [2000], Brabazon [2000], among other.

8.2.1 Stable levels

Technical Analysis tries to derive profitable buy and sell signals by isolating upward and downward price trends from oscillations around a stable level (see Schulmeister [2005]). We let C_t be the closing price at time t, O_t is the opening price, L_t is the low price, H_t the high price, and V_t is the volume of the trade at time t. LL_n is the lowest low and HH_n is the highest high in the look back period with n discrete events. Some of these levels are as follows:

- Momentum: $C_t - C_{t-n}$ where n is the number of discrete events in the look back period.
- Support and resistance: the former is a price level that a share has reached but not fallen below, the latter is a price level that the market has reached but has not risen above. A breach of either of these levels is sometimes considered to indicate a significant change in a share's price, with the level breached then forming a new support or resistance level respectively.
- Retracing: it assumes that prices trends will eventually tend to reverse to visit a support or resistance level. Thus, attempting to predict a future price.
- UpDown Length: It is the number of consecutive days that a security price has either closed up (higher than previous day) or closed down (lower than previous days). Closing up values represented in positive numbers and closing down is represented with negative numbers. If a security closes at the same price on back to back days, the UpDown Length is 0.
- Stochastic oscillators: In the 50s Lane introduced an oscillatoring type of indicator called Stochastics that compares the current market close to the range of prices over a specific time period, indicating when a market is overbought or oversold (see Lane [1984]). The stochastic oscillator is a momentum indicator that uses support and resistance levels. It is based on the assumption that when an up-trend/down-trend approaches a turning point, the closing prices start moving away from the high/low price of a specific range. The number generated by this indicator is a percentage in the range $[0, 100]$ such that for a reading of 70 or more it indicates that the close is near the high of the range. A reading of 30 or below indicates the close is near the low of the range. Note, in general the values of the indicator are smoothed with a moving average (MA).

  1. Stochastic $K\%$ oscillators: $\frac{C_t - LL_n}{HH_n - LL_n} \times 100$ where C_t is the current price, LL_n is the lowest price in the last n days and HH_n is the highest price in the last n days. A value nearer 0 is considered to indicate

a market which is oversold (which will tend to rise) and a value near 100 indicates a market which is overbought.

2. Stochastic $D\%$ oscillators: It is the moving average of the stochastic $K\%$ oscillators. The formula is $\frac{1}{n}\sum_{i=0}^{n-1} K_{t-i}\%$, where $K_t\%$ is the above oscillator.
3. Stochastic slow $D\%$ oscillators: $\frac{1}{n}\sum_{i=0}^{n-1} D_{t-i}\%$ where $D_t\%$ is the above oscillator.

- Larry William $R\%$ (LWR): $\frac{HH_n - C_t}{HH_n - LL_n} \times (-100)$
- Accumulation / Distribution (A/D) oscillator: $\frac{(H_t - O_t) + (C_t - L_t)}{2(H_t - L_t)} \times 100$
- Moving average indicator: the simple moving average (SMA), or just MA, is

$$MA_t(n) = \frac{1}{n}\sum_{i=0}^{n-1} C_{t-i}$$

  A buy signal is generated when a share's price exceeds the moving average and a sell signal when the moving average exceeds the share price. A variation of this indicator is the moving average convergence divergence (MACD) oscillator calculated by taking the difference of a short run and a long run moving averages of differing length. If the result is positive, this is taken as a signal that the market is trending upward.

- Disparity in n days: $\frac{C_t}{MA(n)} \times 100$
- Bias in n days: $BIAS(n) = \frac{C_t - MA(n)}{MA(n)} \times 100$
- OSCP price oscillator: $\frac{MA(short) - MA(long)}{MA(short)}$
- Average return in the last n days: $ASY(n) = \frac{1}{n}\sum_{i=1}^{n} SY_{t-i+1}$, where $SY_t = \ln \frac{C_t}{C_{t-1}} \times 100$
- Ratio of the number of rising periods over the n day period: $PSY(n) = \frac{1}{n}A \times 100$, where A is the number of rising days in the last n days.
- Exponential moving average (EMA) indicator:

$$EMA(k)_t = EMA(k)_{t-1} + \alpha \times \big(C_t - EMA(k)_{t-1}\big) \tag{8.26}$$

  where $\alpha = \frac{2}{k+1}$ is a smoothing factor and k is the time period of k-day EMA.

- On Balance Volume (OBV) indicator: it is based on a cumulative total volume and is intended to relate price and volume in the stock market (see Granville [1976]). The idea is that volume is higher on days where the price move is in the dominant direction, for example in a strong uptrend there is more volume on up days than down days. The formula is as follows:

$$OBV(k)_t = OBV(k)_{t-1} + \theta \times V_t \tag{8.27}$$

  where

$$\theta = \begin{cases} +1 \text{ if } C_t \geqslant C_{t-1} \\ -1 \text{ if } C_t < C_{t-1} \end{cases}$$

8.2.2 Technical indicators

We now describe more complex technical indicators:

- The Chande momentum oscillator (CMO) is a technical momentum indicator (see Chande et al. [1994]). It measures momentum on both up and down days. It is created by calculating the difference between the sum of all recent gains and the sum of all recent losses and then dividing the result by the sum of all price movement over the period. We consider $U_t(n) = \sum_{i=0}^{n_u-1} UP_{t-i}$ (UP_t is upward price changes) where n_u is the number of up periods characterised by the close being higher than the previous close, that is, $UP_t = C_t - C_{t-1} > 0$. We also consider $D_t(n) = \sum_{i=0}^{n_d-1} DW_{t-i}$ (DW_t is downward price changes) where n_d is the number of down periods characterised by the close being lower than the previous close, that is, $DW_t = C_{t-1} - C_t > 0$. The formula is given by

$$CMO = \frac{U_t(n) - D_t(n)}{U_t(n) + D_t(n)} \times 100$$

  Note, $U_t(n) + D_t(n)$ is the sum of all price movement over a given time period. The oscillator is in the range $[-100, +100]$.

- Commodity channel index (CCI): It is an oscillator originally introduced by Donald Lambert [1980] used for identifying cyclical trends. CCI measures a security's variation from the statistical mean, fitting into the momentum category of oscillators. It is calculated as the difference between the typical price of the underlying and its simple moving average, divided by the mean absolute deviation of the typical price. It is an unbounded indicator so that the index is usually scaled by an inverse factor of 0.015 to ensure that approximately 70 to 80 percent of CCI values would fall between -100 and $+100$. The four steps to compute the formula are as follow:

  1. Compute the Typical Price
  $$M_t = \frac{H_t + L_t + C_t}{3}$$
  2. Compute the simple moving average (SMA) price of the last n Typical prices
  $$SM_t(n) = \frac{1}{n} \sum_{i=1}^{n} M_{t-i+1}$$
  3. Compute the mean absolute deviation of the last n Typical prices
  $$D_t(n) = \frac{1}{n} \sum_{i=1}^{n} |M_{t-i+1} - SM_t(n)|$$
  4. Compute the CCI
  $$CCI = \frac{M_t - SM_t(n)}{0.015 \times D_t(n)}$$

  The percentage of CCI values falling in the range $[-100, +100]$ will depend on the number of periods used. A shorter CCI will be more volatile with a smaller percentage of values in that range. Conversely, the more periods used to calculate the CCI, the higher the percentage of values in that range.

- Wilder [1978] developed an indicator that measures the velocity of directional movement, making it a momentum [5] oscillator. It provides a relative evaluation of the strength of a security's recent price performance. The

[5] Momentum is the rate of the rise or fall in price.

Relative Strength Index (RSI) is a ratio of the upward price movement to the total price movement over a given period of days. The average gain (AU) and loss (AD) are calculated using an n-period smoothed or modified moving average (SMMA or MMA,) which is an exponentially smoothed Moving Average with $\alpha = \frac{1}{\text{period}}$. One can also use a standard exponential moving average (EMA) as the average instead of the SMMA. Default smoothing period is 14 periods (or days), $\alpha = \frac{1}{14}$. It can be lowered to increase sensitivity or raised to decrease sensitivity. Wilder originally formulated the calculation of the moving average as:

$$\text{newval } = \frac{1}{\text{period}}\big(\text{ prevval } \times (\text{ period } - 1) + \text{ newdata }\big)$$

A variation called Cutler's RSI is based on a simple moving average of U and D, instead of the exponential average above. Cutler found that since Wilder used a smoothed moving average to calculate RSI, the value of the RSI depended upon where in the data file his calculations started. Cutler termed this Data Length Dependency. His RSI is not data length dependent, and returns consistent results regardless of the length of, or the starting point within a data file. Losses are expressed as positive values, not negative values. The formula is given by

$$RSI = \frac{AU(n)}{AU(n) + AD(n)} \times 100$$

where

- $AU(n) = \frac{1}{n_u}\sum_{i=0}^{n_u-1} UP_{t-i}$ (UP_t is upward price changes) is the average of n_u days up closes. Up periods are characterised by the close being higher than the previous close. That is, $UP_t = C_t - C_{t-1} > 0$.
- $AD(n) = \frac{1}{n_d}\sum_{i=0}^{n_d-1} DW_{t-i}$ (DW_t is downward price changes) is the average of n_d days down closes. Down periods are characterised by the close being lower than the previous close. That is, $DW_t = C_{t-1} - C_t > 0$.

The ratio takes values in the range $[0, 100]$. When $AD = 0$ the RSI is 100 (only going up) and when $AU = 0$ the RSI is 0.
The formula is sometime written as

$$RSI = 100 - \frac{100}{(1 + RS(n))}$$

where $RS(n) = \frac{AU(n)}{AD(n)} = \frac{SMA(U,n)}{SMA(D,n)}$, the average gain over the average loss. In general the formula will need at least 250 data points. A a plot of RS looks exactly the same as a plot of RSI. The normalization step makes it easier to identify extremes because RSI is range bound.

- The Donchian channel (or breakout) is an envelope indicator involving two lines that are plotted above and below the market. The top line represents the highest high of n days back (the last n days) (or weeks), and conversely the bottom line represents the lowest low of n days back (the last n days). This centreline is simply the mean of the upper and lower values. While it is an indicator of volatility of a market price, it is used for providing signals for long and short positions. If a security trades above its highest n periods high, then a long is established, otherwise if it trades below the lowest n periods, a short is established. Alternatively, when the day's high penetrates the highest high of four weeks back and selling when the day's low penetrates the lowest low of four weeks back. The channels are wider when there are heavy price fluctuations and narrow when prices are relatively flat.

- The Moving Average Crossover involves two or more moving averages usually consisting of a longer-term and shorter-term average. When the short-term MA crosses from below the long-term MA, it usually indicates a buying opportunity, and selling opportunities occur when the shorter-term MA crosses from above the longer-term MA. Moving averages can be calculated as simple, exponential, and weighted average. Exponential and weighted MAs tend to skew the moving averages toward the most recent prices, increasing volatility.

- In the 70s Appel [1974] developed another price oscillator called the moving average convergence divergence (MACD). It is calculated by subtracting the value of a 26-period exponential moving average from a 12-period exponential MA. The formula is given by

$$DIFF_t = EMA(12)_t - EMA(26)_t$$

where $EMA(k)_t$ is given in Equation (8.26).
Variants exist. For instance, it can be derived from three different exponentially smoothed moving averages (see Appel et al. [2008]). It is plotted as two different lines, the first line (MACD line) being the difference between the two MAs (long-term and short-term MAs), and the second line (signal or trigger line) being an exponentially smoothed MA of the MACD line. The formula is given by

$$MACD(n)_t = MACD(n)_{t-1} + \frac{2}{n+1}\big(DIFF_t - MACD(n)_{t-1}\big)$$

Note, the difference (or divergence) between the MACD line and the signal line is shown as a bar graph. The purpose being to try to eliminate the lag associated with MA type systems. This is done by anticipating the MA crossover and taking action before the actual crossover. The system buys when the MACD line crosses the signal line from below and sells when the MACD line crosses the signal line from above.

- The rate of change (ROC) momentum involves the analysis of the rate of price change rather than the price level. The speed of price movement and the rate at which prices are moving up or down provide clues to the amount of strength the bulls or bears have at a given point in time. It is calculated by dividing the change in the days' closing price C_t, over n periods, by the closing price n number of days (or weeks) ago C_{t-n}, and then multiplying the ratio by 100. The formula is

$$\begin{aligned} ROC &= \Big(\frac{\text{Today's close} - \text{close n periods ago}}{\text{close n periods ago}}\Big) \times 100 \\ &= \frac{C_t - C_{t-n}}{C_{t-n}} \times 100 \end{aligned}$$

It is sometimes given by the difference in prices

$$ROC = \text{Today's close} - \text{close n periods ago}$$

Generally, the ROC is calculated based on 14-periods for input n, but can be modified to any trader preferred period. When prices are rising or advancing, ROC values remain above the Zero Line (positive) and when they are falling or declining, they remain below the Zero Line (negative). Note, it is not bounded to a set range since there is no limit to how far a security can advance in price even if there is a limit to how far it can decline.

- Connors RSI (CRSI): The indicator is a composite of three separate components where Wilder's RSI is used in two of the indicator's three components. The RSI, UpDown Length, and Rate-of-Change (ROC), combine to form a momentum oscillator. CRSI outputs a value in the range $[0, 100]$, which is then used to identify short-term overbought and oversold conditions. A value over 90 should be considered overbought and a value under 10 should be considered oversold. Note, have a tendency to produce false signals.

- As an example of channel trading, Keltner [1960] proposed a system called the 10-day moving average rule using a constant width channel to time buy-sell signals with the following rules

  1. compute the daily average price $\frac{high+low+close}{3}$.
  2. compute a 10-day average of the daily average price.

3. compute a 10-day average of the daily range.
4. add and subtract the daily average range from the 10-day moving average to form a band or channel.
5. buy when the market penetrates the upper band and sell when it breaks the lower band.

While this system is buying on the strength and selling on weakness, some practitioners have modified the rules as follow

1. instead of buying at the upper band, you sell and vice versa.
2. the number of days are changed to a three-day average with bands around that average.

- The Bollinger bands (BB), also called alpha-beta bands, usually uses 20 or more days in its calculations and does not oscillate around a fixed point (see Bollinger [1992]). It consist of three lines (bands), where the middle line is a simple moving average and the outside lines are plus or minus two (that number can vary) standard deviations above and below the MA. Note, the first two steps are the same as in the CCI. The formula is as follows:

  1. Compute the Typical Price
  $$M_t = \frac{H_t + L_t + C_t}{3}$$
  2. Compute the simple moving average (SMA) price of the last n Typical prices
  $$SM_t(n) = \frac{1}{n}\sum_{i=1}^{n} M_{t-i+1}$$
  3. Compute the standard deviation of the last n Typical prices
  $$SD_t(n) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(M_{t-i+1} - SM_t(n)\big)^2}$$
  4. Calculate the upper band
  $$\text{TopBand } = SM_t(n) + 2 \times SD_t(n)$$
  5. Calculate the middle band
  $$\text{MidBand } = SM_t(n)$$
  6. Calculate the bottom band
  $$\text{BotBand } = SM_t(n) - 2 \times SD_t(n)$$

Note, one can use a 20-day MA with one and two standard deviations above and below the MA. In the Bollinger bands (BB), looking at the chart we can deduce trend, volatility, and overbought/oversold conditions. A market above one standard deviation is overbought, and it becomes extremely overbought above two standard deviation. That is, most underlyings will pullback to the average even in strongly trending markets. Further, with narrow bands we should buy volatility (calls and puts) and when the bands are widening we should sell volatility.
Bollinger [1992] also described two other indicators, the $B\%$ (in relation to the stochastic oscillator $K\%$). The former tells where the prices are within the bands, and it can be above or below one and zero. The formula is

$$B\% = \frac{C_t - \text{ BotBand}}{\text{TopBand } - \text{ BotBand}}$$

Harrington [2005] normalised this equation to map the bands to $[0, 100]$ instead of $[0, 1]$

$$NB\% = 200 \times B\% - 100$$

In the $B\%$, the 100% line is the TopBand and the -100% line is the BotBand of the Bollinger bands. Recall, the bandwidth measures the width of the band as a percentage of the moving average

$$BW = \frac{\text{TopBand } - \text{ BotBand}}{\text{MidBand}}$$

When the bands narrow drastically, a sharp expansion in volatility usually occurs in the very near future.

8.2.3 Visualising trends

Geometrical tools have been proposed to visualise and summarise the information produced by the technical indicators described above. Some examples are as follows:

- Considering projected charts to map future market activity, Drummond et al. [1999] introduced the Drummond Geometry (DG) which is both a trend-following and a congestion-action methodology, leading rather than lagging the market. It tries to foretell the most likely scenario that shows the highest probability of occurring in the immediate future and can be custom fitted to one's personality and trading style. The key elements of DG include a combination of the following three basic categories of trading tools and techniques
    1. a series of short-term moving averages
    2. short-term trend lines
    3. multiple time-period overlays
- The concept of Point and Line (PL) reflecting how all of life moves from one extreme to another, flowing back and forth in a cyclical or wave-like manner, is applied to the market. The PLdot (average of the high, low, and close of the last three bars) is a short-term MA based on three bars (or time periods) of data capturing the trend/nontrend activity of the time frame that is being charted. It represent the center of all market activity. The PLdot is very sensitive to trending markets, it is also very quick at registering the change of a market out of congestion (noise) into trend, and it is sensitive to ending trend.

8.3 The limitation of indicators: combining indicators

In general, momentum based indicators fail most of the time because

- none of them is a pure momentum oscillator that measures momentum directly.
- the time period of the calculations is fixed, giving a different picture of market action for different time periods.
- they all mirror the price pattern, so that it may be better trading prices themselves.
- they do not consistently show extremes in prices because they use a constant time period.
- the smoothing mechanism introduces lags and obscures short-term price extremes that are valuable for trading.

As a result, technical analysts do not rely on a single indicator for detecting trend reversals but rather combine the outcome of various technical indicators. Since each indicator has a significant failure rate, the random nature of price change being one reason why indicators fail, Chande et al. [1994] explained how most traders developed several indicators to analyse prices. Traders use multiple indicators to confirm the signal of one indicator with respect to another one, believing that the consensus is more likely to be correct. For instance, one can generates complex trading rules based on two-way and three-way indicator combinations. The number of combinations is given by the binomial coefficient $\binom{n}{r}$ where r is the number of objects that can be drawn from a set containing n distinguishable objects. In order to get a single buy/sell decision, one must combine the different indicators according to some defined decision rules. For example, one can give a larger weight to trade signals than to hold ones. In case of multiple trade signals one solution is to employ majority voting for deriving the final outcome (see Appendix (10)). In the case of three-way indicator, two sell signal and one buy signal results in a sell signal.
However, this may not a viable approach due to the strong similarities existing between price based momentum indicators.

8.4 Trading with technical indicator

8.4.1 Trading rules

A trader must decide on a specific entry technique, stop point, and take profit. For instance, whenever a buy signal is triggered, the investor takes a long position in the respective share until a sell signal is generated, at which point the trader unwind his position (sell). He could have simultaneously unwind his position and go short. This position is held until the next buy signal. These are trading strategies discussed in the following section. For now we focus on the entry point and stopping point. When trading on a daily basis at market close, it is assumed that one can estimate the asset price that would trigger the buy and sell signal just before the day's close and initiate a conditional limit order at the close of the market to follow various trading rules. In the case of high frequency trading, buying and selling occur until the end of the trading day, since no positions are held overnight.
Given the technical indicators defined in Appendix (8.2), their associated trading rules are as follows:

- In the case of a simple moving average (SMA) and exponential moving average (EMA), a buy signal $(+1)$ is emitted when the asset price moves from below to above the SMA/EMA considered, and a sell signal (-1) is emitted when the asset price penetrates the SMA/EMA from above to below.

- In the case of stochastic oscillators, when the indicator is increasing, the asset prices are likely to go up and vice versa. In the case of an up-trend, prices tend to make higher highs, and the settlement price usually tends to be in the upper end of that time period's trading range. When the momentum starts to slow, the settlement prices will start to retreat from the upper boundaries of the range, causing the stochastic indicator to turn down at or before the final price high. A value nearer 0 is considered to indicate a market which is oversold (which will tend to rise) and a value near 100 indicates a market which is overbought. An alert or set-up is present when the $D\%$ line is in an extreme area and diverging from the price action. The actual signal takes place when the faster $K\%$ line crosses the $D\%$ line.

- In the case of two moving averages, say a 10 days and 20 days MA, when the 10D-MA moves up through the 20D- MA a buy sign is generated and when the 10D-MA moves down through the 20D-MA a sell sign is generated. The logic being that the price will continue to move in that direction so that the value of shares continues to increase or decrease.

- In the case of a moving average convergence divergence (MACD), then $MACD > 0$ represents the rule such that if MACD is greater than zero we choose to be in the market, otherwise we stay out of the market.

- For variants of MACD, we have $MACD >$ signal line is the trading rule such that if MACD is above the signal line we choose to be in the market, otherwise we stay out of the market.

- Traders and investors use the commodity channel index (CCI) to help identify price reversals, price extremes and trend strength. It is often used for detecting divergences from price trends as an overbought/oversold indicator, and to draw patterns on it and trade according to those patterns. Readings above $+100$ imply an overbought condition (showing strength), while readings below -100 imply an oversold condition (showing weakness). It means that there is a large probability that the price will correct to more representative levels. Thus, assuming momentum, when movements are above $+100$ we buy $(+1)$ and when they are below -100 we sell (-1). It becomes a coincident indicator. The position should be closed when the CCI moves back below $+100$, in the former. The position should be closed when the CCI moves back above -100, in the latter. Since about 70 to 80 percent of the CCI values are between $+100$ and -100, a buy or sell signal will be in force only 20 to 30 percent of the time. Alternatively, traders have also found the CCI valuable for identifying reversals and divergences. Assuming a mean-reverting market, when price movements are above $+100$ we sell (-1) and when they are below -100 we buy $(+1)$. It becomes a leading indicator, where the trader look for overbought or oversold conditions.

- Traditional interpretation and usage of the RSI is that RSI values of 70 or above indicate that a security is becoming overbought or overvalued, and therefore, may be primed for a trend reversal or corrective pull-back in price (-1). An RSI reading of 30 or below is commonly interpreted as indicating an oversold or undervalued condition that may signal a trend change or corrective price reversal to the upside $(+1)$. Sudden large price movements can create false buy or sell signals in the RSI. It is, therefore, best used with refinements to its application or in conjunction with other, confirming technical indicators.
In addition to Wilder's original theories of RSI interpretation, Cardwell and Brown developed several new interpretations of RSI to help determine and confirm trend. Brown [1999] suggested that oscillators do not travel between 0 and 100 and identified a bull market range and a bear market for RSI. RSI tends to fluctuate between 40 and 90 in a bull market (uptrend) with the $40-50$ zones acting as support. These ranges may vary depending on RSI parameters, strength of trend and volatility of the underlying security. RSI tends to fluctuate between 10 and 60 in a bear market (downtrend) with the $50-60$ zone acting as resistance. Andrew Cardwell developed positive and negative reversals for RSI, which are the opposite of bearish and bullish divergences. A positive reversal forms when RSI forges a lower low and the security forms a higher low. A negative reversal is the opposite of a positive reversal. RSI forms a higher high, but the security forms a lower high.

- The Rate of Change indicator (ROC) might be used to confirm price moves or detect divergences and might be used as a guide for determining overbought and oversold conditions. If it is positive we choose to be in the market (buy signal $+1$) and, if it is negative, then it is a sell signal (-1). It applies to both ROC formula. For instance we can consider $ROC(30)$, the 30-day rate of change of the asset price.

- Donchian channels (DC) are mainly used to identify the breakout of a stock or any traded entity enabling traders to take either long or short positions. Traders can take a long position $(+1)$, if the stock is trading higher than the Donchian channels n period and book their profits/short the stock (-1) if it is trading below the DC channels n period.
The middle band is the average of the upper and lower bands. The middle band in Donchian channels could also be used as a breakout indicator. If the stock rises above the middle band of the DCs, then you can open a long position $(+1)$. On the contrary, if the stock is trading below the middle band of the DCs, then a trader can open a short position (-1).

- A security is deemed to be overbought when the Chande momentum oscillator (CMO) is above $+50$ and oversold when it is below -50. Trend strength can also be measured using the Chande momentum oscillator. The higher the oscillator's value, the stronger the trend, the lower the value, the weaker the trend.
Traders can use the Chande momentum oscillator to spot positive and negative price divergence between the indicator and the underlying security. A negative divergence occurs if the underlying security is trending upward and the Chande momentum oscillator is moving downwards. A positive divergence occurs if price is declining, but the oscillator is rising.

Many technical traders add a 10-period moving average to this oscillator to act as a signal line. The oscillator generates a bullish signal $(+1)$ when it crosses above the moving average, and a bearish signal (-1) when it moves below the moving average.

- In the Bollinger bands (BB), a typical system buys $(+1)$ when price reaches the bottom (BotBand) and liquidates (-1) as the price moves up past the MA. The sell side is simply the opposite. It is assumed that when a price goes beyond two standard deviations it should revert to the MA. Note, some practitioners revert the logic and sell (-1) rather than buy when prices reach the lower band (BotBand), and vice versa with the upper band (TopBand).

8.4.2 Trading strategies

Accounting for the predictive power of trading rules, Metghalchi et al. [2015] proposed some trading strategies to beat the profitability of the buy and hold (B&H) strategy. Contrary to the B&H strategy, which is only long, the trader can still invest in cash or short the market. The authors detail some actions to be taken when the trader is out of the market. The strategies are as follows:

1= the trader will be in the market when a trading rule emits buy signals and be in the money market (cash) when a trading rule emits sell signals (long/money).

2= the trader will be in the market when a trading rule emits buy signals and would short the market when the rule emits sell signals (long/short).

3= the trader will borrow at the money market rate (cash rate) and double stock investment when a trading rule emits buy signals and be in the money market when the rule emits sell signals (leverage/ money).

4= the trader will borrow at the money market rate and double stock investment when a trading rule emits buy signals and would short the market when the rule emits sell signals (leverage/short).

In case of leverage, the total return on buy days is estimated as twice the market return minus the daily money market return.

8.4.3 Discretising technical indicators

Note, one can directly use continuous technical indicators (actual time series) or one can choose to represent them as trend deterministic data (discrete in nature). In the former, for comparison purposes, the values of all technical indicators are normalised in the range $[-1,+1]$ (see Kara et al. [2011]). Since each technical indicator has its own inherent property used by traders to predict up movement or down movement, a mapping function can be used to convert the continuous values to discrete values representing the trend (see Patel et al. [2015]). That is, $+1$ indicates an up movement and -1 shows a down movement. To do so, we use the trading rules in Appendix (8.4.1). For example, when SMA/EMA at time t is greater than at time $t-1$, it suggests an up movement $(+1)$ and vice versa for a down movement (-1). In the case of stochastic oscillators, they are clear trend indicators for any asset. MACD, RSI, CCI and A/D oscillators also follow the asset trend.

8.4.4 Trading methodology

Since pattern is defined as a predictable route or movement, all trading systems are pattern recognition systems. For instance, a long-term moving average cross over system uses pattern recognition, the crossover, in its decision to buy or sell. Similarly, an open range breakout is pattern recognition given by the movement from the open to the breakout point. All systems look for some type of reoccurring event and try to capitalise on it. Hill demonstrated the success of pattern recognition when used as a filter. The system was developed around the idea of a pattern consisting of the last four days' closing prices. A buy or sell signal is not generated unless the range of the past four days' closes is less than the 30-day average true range, indicating that the market has reached a state of rest and any movement, up or down, from this state will most likely result in a significant move.

Eventually, we want to develop an approach into a comprehensive, effective, trading methodology that combines analytical sophistication with tradable rules and principles. One way forward is to consider a multiple time frame approach. A time frame is any regular sampling of prices in a time series, from the smallest such as one minute up to the longest capped out at ten year. The multiple time frame approach has proven to be a fundamental advance in the field of TA allowing for significant improvement in trading results. For instance, if market analysis is coordinated to show the interaction of these time frames, then the trader can monitor what happens when the support and resistance lines of the different time frames coincide. Assuming we are interested in analysing the potential of the integration of time frames, then we need to look at both a higher time frame and a lower time frame.

8.5 Analysing the trading rules

We have defined the moving average indicator in Appendix (8.2). More formally, we let N_S be the short period, N_L the long period, X_b a percentage band, and $P(t)$ the price at time t. The MA rule is (N_S, N_L, X_b) and some popular ones are $(1, 50, 0)$, $(1, 100, 0)$. The buy signal is given by

$$\frac{1}{N_S}\sum_{i=1}^{N_S} P(t-(i-1)) > \frac{1}{N_L}\sum_{i=1}^{N_L} P(t-(i-1)) + X_b \rightarrow \text{ buy at time } t$$

and the sell signal is reversed, as follows

$$\frac{1}{N_S}\sum_{i=1}^{N_S} P(t-(i-1)) < \frac{1}{N_L}\sum_{i=1}^{N_L} P(t-(i-1)) - X_b \rightarrow \text{ sell at time } t$$

Sweeney [1986] applied filter rules on ten currencies, and Lukac et al. [1988] applied moving average and other technical rules to twelve US futures markets. They both showed that trading rules have predictive power. Brock et al. [1992] analysed the Dow Jones industrial index for a period of 89 years and reported consistent and positive results about the forecasting power of technical trading rules and in particular the Moving Average (MA) rules.
Since these seminal studies, numerous studies have examined MA rules in various world markets and generally found a good measure of support for their effectiveness (see Chen et al. [2009], Metghalchi et al. [2012]). However, it was observed that the rules tend to vary in their effectiveness over time (see Shynkevich et al. [2012]). Further, adding trading costs and other market frictions several articles found that these adjustments eliminate risk-adjusted excess profits (see Bessembinder et al. [1998], Day et al. [2002]). Some new insights were added to the literature that do not necessarily depend on the ability to make direct trading profits. For instance, Han et al. [2013] showed that applying a moving average strategy to portfolios sorted by volatility substantially outperforms a buy-and-hold strategy.

It is not clear as to why MA rules can predict future price moves. Some explanations considered short term market behaviour while other focused on the long term behaviour. It can be summarised as follows:

1. short term trends occur because of
   (a) rational factors: the existence of short term market frictions
   (b) behavioural factors: behavioural biases
2. the existence of long term effects tend to depend on behavioural biases such as under-reaction to news over a substantial period of time or the existence of extended periods where a particular type of market sentiment dominates.

Since the moving average rules draw on relatively long term data to make predictions, it is clear whether short term or long term models are more appropriate. In any case, the poor performance of conventional econometric models (random walk, $AR(1)$, GARCH-M and EGARCH) are not supportive of short term rational factors. Note that an MA rule can be explained by an almost infinite variety of underlying processes.

Hudson et al. [2017] proposed to model MA rule simple trend following rules such as a basic Markov chain. They showed that the ability of the rules to predict market direction is largely based on the existence of very short term trends in the data in which positive/negative daily returns are more likely to be followed by positive/negative returns. They concluded that short term market frictions were the main explanation for the success of the MA rules.
Metghalchi et al. [2018] tested the profitability of four trading rules (MA, RSI, MACD, ROC momentum) (see Appendix (8.4.1)) in the case of the OMX Iceland All-Share Index (market close from 1999 till 2016) and showed that they had predictive power. Then, they designed four trading strategies (see Appendix (8.4.2)) for each trading rule (a total of ten: 5 MA, 1 RSI, 2 MACD, 2 ROC). They obtained very good results for strategy (1) where the average of all 10 trading rules have a mean daily return of 0.0067 and annual return of 16.03%, compared with 0.00001 and 0.37% of the Buy and Hold (B&H) strategy. Strategy (2) has higher daily and annual returns than Strategy (1), but it also has higher risk. For Strategy (3) the rules have higher daily and annual returns than Strategy (1), implying that leveraging on buy days improves returns, but at higher risk (almost the double of Strategy (1)). However, these trading rules imply many times in and out of the market resulting in transaction costs and reduced net returns. In the case of Strategies (2) and (4) they estimated the one-way break-even transaction costs (BEC). The best rules are those with high BEC and low risk. Considering risk and transaction costs, they showed that three trading rules, namely MA200, MA150, and MACD, could beat the B&H strategy for the entire period.

8.6 Risk management

8.6.1 The risk of overfitting

While a simplified representation of reality can either be descriptive or predictive in nature, or both, financial models are predictive to forecast unknown, or future, values based on current or known values using mathematical equations or set of rules. However, the forecasting power of a model is limited by the appropriateness of the inputs and assumptions so that one must identify the sources of model risk to understand these limitations. Model risk generally occurs as a result of incorrect assumptions, model identification or specification errors, inappropriate estimation procedures, or in models used without satisfactory out-of-sample testing. For instance, some models can be very sensitive to small changes in inputs, resulting in big changes in outputs. Further, a model may be overfitted, meaning that it captures the underlying structure or the dynamics in the data as well as random noise. This generally occurs when too many model parameters are used, restricting the degrees of freedom relative to the size of the sample data. It often results in good in-sample fit but poor out-of-sample behaviour. Hence, while an incorrect or misspecified model can be made to fit the available data by systematically searching the parameter space, it does not have a descriptive or predictive power. Familiar examples of such problems include the spurious correlations popularised in the media, where over the past 30 years when the winner of the Super Bowl championship in American football is from a particular league, a leading stock market index historically goes up in the following months. Similar examples are plentiful in economics and the social sciences, where data are often relatively sparse but models and theories to fit the data are relatively prolific. In economic time series prediction, there may be a relatively short time-span of historical data available in conjunction with a large number of economic indicators. One particularly humorous example of this type of prediction was provided by Leinweber who achieved almost perfect prediction of annual values of the $S\&P$ 500 financial index as a function of annual values from previous years for butter production, cheese production, and sheep populations in Bangladesh and the United States. There are no easy technical solutions to this problem, even though various strategies have been developed. In order to avoid model overfitting and data snooping, one should decide upon the framework by defining how the model should be specified before beginning to analyse the actual data. First, by properly formulating model hypothesis making financial or economic sense, and then carefully determining the number of dependent variables in a regression model, or the number of factors and components in a stochastic model one can expect avoiding or reducing storytelling and data mining. To increase confidence in a model, true out-of-sample studies of model performance should be conducted after the model has been fully developed. We should also be more comfortable with a model working cross-sectionally and producing similar results in different countries. Note, in a general setting, sampling bootstraping, and randomisation techniques can be used to evaluate whether a given model has predictive power over a benchmark model (see White [2000]). All forecasting models should be monitored and compared on a regular basis, and deteriorating results from a model or variable should be investigated and understood.

8.6.2 Evaluating trading system performance

We define a successful strategy to be one that maximise the number of profitable days, as well as positive average profits over a substantial period of time, coupled with reasonably consistent behaviour. As a result, while we must look at profit when evaluating trading system performance, we must also look at other statistics such as

- maximum drawdown: the highest point in equity to the subsequent lowest point in equity. It is the largest amount of money the system lost before it recovered.
- longest flat time: the amount of time the system went without making money.
- average drawdown: maximum drawdown is one time occurrence, but the average drawdown takes all of the yearly drawdowns into consideration.
- profit to loss ratio: it represents the magnitude of winning trade dollars to the magnitude of losing trade dollars. As it tells us the ratio of wins to losses, the higher the ratio the better.
- average trade: the amount of profit or loss we can expect on any given trade.
- profit to drawdown ratio: risk in this statistic comes in the form of drawdown, whereas reward is in the form of profit.
- outlier adjusted profit: the probability of monstrous wins and/or losses reoccurring being extremely slim, it should not be included in an overall track record.
- most consecutive losses: it is the total number of losses that occurred consecutively. It gives the user an idea of how many losing trades one may have to go through before a winner occurs.
- Sharpe ratio: it indicates the smoothness of the equity curve. The ratio is calculated by dividing the average monthly or yearly return by the standard deviation of those returns.
- long and short net profit: as a robust system would split the profits between the long trades and the short trades, we need to make sure that the money made is well balanced between both sides.
- percent winning months: it checks the number of winning month out of one year.

9 Introduction to data mining

9.1 From data to information

We refer the readers to textbooks by Hand et al. [2001], Han et al. [2006], among others.

9.1.1 Data

9.1.1.1 Definitions Data is a set of values of qualitative or quantitative variables. In computing, data is information that has been translated into a form that is efficient for movement or processing. A digital computer represents a piece of data as a sequence of symbols drawn from a fixed alphabet. The most common digital computers use a binary alphabet, that is, an alphabet of two characters, typically denoted 0 and 1. Thus, data is information converted into binary digital form. It is acceptable for data to be used as a singular subject or a plural subject. More generally, data, information, knowledge and wisdom are closely related concepts, but each has its own role in relation to the other, and each term has its own meaning. Data is collected and analysed; data only becomes information suitable for making decisions once it has been analysed in some fashion. The concept of data in the context of computing was introduced by Claude Shannon who ushered in binary digital concepts based on applying two-value Boolean logic to electronic circuits. The amount of information content in a data stream may be characterized by its Shannon entropy. Data is often assumed to be the least abstract concept, information the next least, and knowledge the most abstract. Thus, data becomes information by interpretation (see Ilkka [2000]). Generally, the concept of information is closely related to notions of constraint, communication, control, data, form, instruction, knowledge, meaning, mental stimulus, pattern, perception, and representation (see Beynon-Davies [2009]). This leads to the concept of data mining.

9.1.1.2 Data set A data set consists of feature vectors, where each feature vector is a description of an object by using a set of features. For example, in a three-Gaussian data set, a feature vector can be represented by $(value1, value2, cross)$ or $(value1, value2, circle)$, where $(value1, value2)$ are (x, y)-coordinates and $cross$ and $circle$ are labels. The number of features of a data set is called dimension. Features are also called attributes, a feature vector is also called an instance, and a data is sometime called a sample. Labelled data is a group of samples that have been tagged with one or more labels. Labelling typically takes a set of unlabelled data and augments each piece of that unlabelled data with meaningful tags that are informative. A data class refers to a class that contains only fields and crude methods for accessing them (getters and setters). These are simply containers for data used by other classes.

9.1.2 Data mining

Data mining is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems. It is an interdisciplinary subfield of computer science with an overall goal of extracting information (with intelligent method) from a data set and transform the information into a comprehensible structure for further use (see Hastie et al. [2009], Kamber et al. [2011]). The rapid growth and integration of databases provided scientists, engineers, and business people with a vast new resource that can be analysed to make scientific discoveries, optimise industrial systems, and uncover financially valuable patterns. New methods targeted at large data mining problems have been developed. Hand et al. [2001]) defined Data Mining (DM) as follows:

Definition 9.1 *Data Mining is the analysis of (large) observational data sets to find unsuspected relationships and to summarise the data in novel ways that are both understandable and useful to the data owner.*

More generally, data mining is the analysis step of the Knowledge Discovery in Databases process, or KDD (see Han et al. [2006]). The goal is the extraction of patterns and knowledge from large amounts of data, not the extraction (mining) of data itself. According to the Cross Industry Standard Process for Data Mining (CRISP-DM), the six phases of the knowledge discovery in databases (KDD) process are

- Business understanding
- Data understanding
- Data preparation
- Modelling

- Evaluation
- Deployment

or a simplified process such as (1) Pre-processing, (2) Data Mining, and (3) Results Validation.
Before data mining algorithms can be used, a target data set must be assembled. A common source for data is a data mart or data warehouse. Pre-processing is essential to analyse the multivariate data sets before data mining. The target set is then cleaned. Data cleaning removes the observations containing noise and those with missing data.
There exists several functionalities to data mining, such as:

1. Data characterisation summarising the general characteristics or features of a target class of data.
2. Data discrimination comparing the general features of target class data objects with the general features of objects from a set of contrasting classes.
3. Association analysis is the discovery of association rules showing attribute value conditions occurring frequently together in a given set of data.
4. Classification is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown.

In fact, data mining involves six common classes of tasks (see Sondwale [2015]):

1. Anomaly detection (outlier/change/deviation detection): The identification of unusual data records, that might be interesting or data errors that require further investigation.
2. Association rule learning (dependency modelling): Searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
3. Clustering: is the task of discovering groups and structures in the data that are in some way or another similar, without using known structures in the data.
4. Classification: is the task of generalising known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as legitimate or as spam.
5. Regression: attempts to find a function which models the data with the least error that is, for estimating the relationships among data or datasets.
6. Summarisation: providing a more compact representation of the data set, including visualisation and report generation.

9.1.3 Modelling

In general, a model is a predictive model that we want to construct or discover from the data set. One major task of pattern recognition and data mining, within machine learning, is to construct good models from data sets (see Otero et al. [2013]). The process of generating models from data is called learning or training. There are several types of learning, such as

- supervised learning: it consists of a specified set of classes, and example objects labelled with the appropriate class. The goal being to learn from the training objects, enabling novel objects to be identified as belonging to one of the classes.

- unsupervised learning: it is the task of inferring a function that describes the structure of unlabelled data (data that has not been classified or categorised). Among neural network models, the self-organizing map (SOM) and adaptive resonance theory (ART) are commonly used in unsupervised learning algorithms.

The learned model is called a predictor or a hypothesis (or learner). There are several types of label such as categorical or numerical.

1. If the label is categorical, such as shape, the task is also called classification and the learner is also called classifier.
2. However, if the label is numerical, such as x-coordinate, the task is also called regression and the learner is also called fitted regression model.

If the label is categorical, such as shape, the task is called classification and the learner is called a classifier. If the label is numerical, such as x-coordinate, the task is called regression and the learner is called fitted regression model.
In this appendix, we will focus on supervised learning, especially classification.

9.2 Classification

9.2.1 Terminology

In the more general problem of pattern recognition, we distinguish

- Classification and clustering, which is the assignment of some sort of output value to a given input value
- Regression, which assigns a real-valued output to each input
- Sequence labelling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence)
- Parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence

Often, the individual observations are analysed into a set of quantifiable properties, known variously as explanatory variables or features. These properties may variously be

- categorical (such as A, B, AB or O, for blood type),
- ordinal (such as large, medium or small),
- integer-valued (such as the number of occurrences of a particular word in an email), or
- real-valued (such as a measurement of blood pressure).

Terminology across fields is quite varied. In statistics, where classification is often done with logistic regression or a similar procedure, the properties of observations are termed explanatory variables (or independent variables, regressors, etc.), and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, the observations are often known as instances, the explanatory variables are termed features (grouped into a feature vector), and the possible categories to be predicted are classes.
Class label is the discrete attribute having finite values (dependent variable) whose value you want to predict based on the values of other attributes (features). The class label always takes on a finite number of different values. Classification is a type of problem whereas labelling is a function trying to label an object and classify using the information. For instance, given a set of examples of the form (attribute values , class label), we want to learn a rule that computes the label from the attribute values. For example, in binary classification, we use positive $(+1)$ and negative (-1) to denote the two class labels.

9.2.2 Definitions

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known (see Alpaydin [2010]). Classification is considered an instance of supervised learning, that is, learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.
Machine learning have been traditionally used for classification. A classification process usually consists of three main stages:

1. In the first stage, data from objects have to be collected for the design of classifiers.
2. In the second stage, feature extraction is performed to extract characteristics from the collected data to be classified such that redundant information is removed and representative information is extracted resulting in reduction of input dimensions and improved classification accuracy.
3. In the third stage, a classifier is designed using the feature data.

Classification can be thought of as two separate problems:

1. binary classification: only two classes are involved (it is a better understood task).
2. multiclass classification: involves assigning an object to one of several classes. Since many classification methods have been developed specifically for binary classification, multiclass classification often requires the combined use of multiple binary classifiers.

9.2.3 The algorithms

Most algorithms describe an individual instance whose category is to be predicted using a feature vector of individual, measurable properties of the instance. Some algorithms work only in terms of discrete data and require that real-valued or integer-valued data be discretised into groups (e.g. less than 5, between 5 and 10, or greater than 10). Other classifiers work by comparing observations to previous observations by means of a similarity or distance function.
In the literature, classification techniques and methods from traditional methods to machine learning methods can be found. Examples of the former are Linear discriminant analysis (LDA), logic based method, statistical approach, or instance-based methods. In the latter, models such as decision trees, SVM, neural networks, Bayesian belief networks, genetic algorithm etc have been considered. The ability to perform classification and be able to learn to classify objects is paramount to the process of decision making.
The process of seeking relationships within a data set involves a number of steps:

1. Model or pattern structure: determining the nature and structure of the representation to be used.

   (a) a model structure is a global summary of a data set making statements about any point in the full measurement space,

   (b) pattern structures only make statements about restricted regions of the space spanned by the variables.

2. Score function: deciding how to quantify and compare how well different representations fit the data (choosing a score function).
   Score functions judge the quality of a fitted model, and should precisely reflect the utility of a particular predictive model.
3. Optimisation and search method: choosing an algorithmic process optimising the score function.

(a) optimisation problem: the task of finding the best values of parameters in models
(b) combinatorial problem: the task of finding interesting patterns (such as rules) from a large family of potential patterns, and is often accomplished using heuristic search techniques.

4. Data management strategy: deciding what principles of data management are required for implementing the algorithms efficiently.
   Data management strategy is about the ways in which the data are stored, indexed, and accessed.

Some examples of classification algorithms include:

- Linear classifiers: Fisher's linear discriminant (see Fisher [1936]), Logistic regression, Naive Bayes classifier (see Binder [1978]), Perceptron
- Support vector machines: Least squares support vector machines
- Quadratic classifiers
- Kernel estimation: k-nearest neighbour
- Boosting (meta-algorithm)
- Decision trees: Random forests
- Neural networks
- Learning vector quantization

The success of classification learning is heavily dependent on the quality of the data provided for training, as the learner only has the input to learn from. On the other hand, we want to avoid overfitting the given data set, and would rather find models or patterns generalising potential future data. Note, even though data mining is an interdisciplinary exercise, it is a process relying heavily on statistical models and methodologies. The main difference being the large size of the data sets to manipulate, requiring sophisticated search and examination methods. Further difficulties arise when there are many variables (curse of dimensionality), and often the data is constantly evolving. Recently, a lot of advances have been made on machine learning strategies mimicking human learning, and we refer the reader to Battula et al. [2013] for more details.

9.3 The challenges of computational learning

While the ability to perform classification and be able to learn to classify objects is paramount to the process of decision making, there exists several types of learning (see Mitchell [1997]), such as

- Supervised learning: learning a function from example data made of pairs of input and correct output.
- Unsupervised learning: learning from patterns without corresponding output values
- Reinforcement learning: learning with no knowledge of an exact output for a given input. Nonetheless, online or delayed feedback on the desirability of the types of behaviour can be used to help adaptation of the learning process.
- Active learning: learning through queries and responses.

More formally, these types of learning are part of what is called inductive learning where conclusions are made from specific instances to more general statements. That is, examples are provided in the form of input-output pairs $[X, f(X)]$ and the learning process consists of finding a function h (called hypothesis) which approximates a set of samples generated by the function f. The search for the function h is formulated in such a way that it can be solved by using search and optimisation algorithms.
Some of the challenges of computational learning can be summarised as follow:

- Identifying a suitable hypothesis can be computationally difficult.
- Since the function f is unknown, it is not easy to tell if the hypothesis h generated by the learning algorithm is a good approximation.
- The choice of a hypothesis space describing the set of hypotheses under consideration is not trivial.

As a result, a simple hypothesis consistent with all observations is more likely to be correct than a complex one. In the case where multiple hypotheses (an Ensemble) are generated, it is possible to combine their predictions with the aim of reducing generalisation error. For instance, boosting works as follow

- Examples in the training set are associated with different weights.
- The weights of incorrectly classified examples are increased, and the learning algorithm generates a new hypothesis from this new weighted training set. The process is repeated with an associated stopping criterion.
- The final hypothesis is a weighted-majority of all the generated hypotheses which can be based on different mixture of expert rules.

The difficult part being to know when to stop the iterative process and how to define a proper measure of error. One way forward is to consider the Probably Approximately Correct (PAC) learning which can be described as follow:

- A hypothesis is called approximately correct if its error insample lies within a small constant of the true error.
- By learning from a sufficient number of examples, one can calculate if a hypothesis has a high probability of being approximately correct.
- There is a connection between the past (seen) and the future (unseen) via an assumption stating that the training and test datasets come from the same probability distribution. It follows from the common sense that non-representative samples do not help learning.

More formally, a concept class C is said to be PAC learnable using a hypothesis class H if there exists a learning algorithm L such that for all concepts in C, for all instance distributions D on an instance space X,

$$\forall \epsilon \, , \, \delta \text{ such that } 0 < \epsilon \, , \, \delta < 1$$

when given access to the example set, produces with probability at least $(1 - \delta)$, a hypothesis h from H with error no-more than ϵ. To specify the problem we get a set of instances X, a set of hypotheses H, a set of possible target concepts C, training instances generated by a fixed, unknown probability distribution $\mathcal{D}$ over X, a target value $c(x)$, some training examples $< x, c(x) >$, and a hypothesis h estimating c. Then, the error of a hypothesis h satisfies

$$error_D = P_{x \in \mathcal{D}}(c(x) \neq h(x))$$

and the deviation of the true error from the training error satisfies

- Training error: $h(x) \neq c(x)$ over training instances.
- True error: $h(x) \neq c(x)$ over future random instances.

We must now measure the difference between the true error and the training error. Any hypothesis h is consistent when for all training samples,

$$h(x) = c(x)$$

If the hypothesis space H is finite, and $\mathcal{D}$ is a sequence of $m \geqslant 1$ independent random examples of some target concept c, then for any $0 \leqslant \epsilon \leqslant 1$, the probability that $VS_{H,D}$ contains a hypothesis with error greater than ϵ is less than

$$|H|e^{-\epsilon m}$$

and $P(1 \text{ of } |H| \text{ hyps. consistent with m exs.}) < |H|e^{-\epsilon m}$. Considering a bounded sample size, for the probability to be at most δ, that is, $|H|e^{-\epsilon m} \leqslant \delta$, then

$$m \geqslant \frac{1}{\epsilon}\big(\ln|H| + \ln\frac{1}{\delta}\big)$$

More can be found on agnostic learning and infinite hypothesis space in books by Mitchell [1997] and Bishop [2006].

9.4 Evaluation and validation

Data mining can unintentionally be misused, and can then produce results which appear to be significant, but which do not actually predict future behaviour and cannot be reproduced on a new sample of data and bear little use. Often this results from investigating too many hypotheses and not performing proper statistical hypothesis testing. Not all patterns found by the data mining algorithms are necessarily valid. It is common for the data mining algorithms to find patterns in the training set which are not present in the general data set. This is called overfitting. To overcome this, the evaluation uses a test set of data on which the data mining algorithm was not trained. The learned patterns are applied to this test set, and the resulting output is compared to the desired output. A number of statistical methods may be used to evaluate the algorithm, such as Precision and Recall which are evaluated from True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). More recently Sensitivity, Specificity, Accuracy and Receiver Operating Characteristic (ROC) curves have been used to evaluate the tradeoff between true- and false-positive rates of classification algorithms.
Sensitivity and Specificity are statistical measures of the performance of a binary classification test, also known in statistics as a classification function:

- Sensitivity (also called the true positive rate, the recall, or probability of detection in some fields) measures the proportion of actual positives that are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition).
- Specificity (also called the true negative rate) measures the proportion of actual negatives that are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition).

In the terminology true/false positive/negative, true or false refers to the assigned classification being correct or incorrect, while positive or negative refers to assignment to the positive or the negative category.
In medicine sensitivity, specificity and accuracy are widely used statistics to describe a diagnostic test since they quantify how good and reliable a test is.

- Sensitivity evaluates how good the test is at detecting a positive disease
- Specificity estimates how likely patients without disease can be correctly ruled out
- The ROC curve is a graphic presentation of the relationship between both sensitivity and specificity helping to decide the optimal model through determining the best threshold for the diagnostic test
- Accuracy measures how correct a diagnostic test identifies and excludes a given condition. Accuracy of a diagnostic test can be determined from sensitivity and specificity with the presence of prevalence.

The terms used are true positive (TP), true negative (TN), false negative (FN), and false positive (FP). If a disease is proven present in a patient, the given diagnostic test also indicates the presence of disease, the result of the diagnostic test is considered true positive. Similarly, if a disease is proven absent in a patient, the diagnostic test suggests the disease is absent as well, the test result is true negative (TN). See details in Table (6). Both true positive and true negative suggest a consistent result between the diagnostic test and the proven condition (also called standard of truth). If the diagnostic test indicates the presence of disease in a patient who actually has no such disease, the test result is false positive (FP). Similarly, if the result of the diagnosis test suggests that the disease is absent for a patient with disease for sure, the test result is false negative (FN). Both false positive and false negative indicate that the test results are opposite to the actual condition.

Outcome of the test	**Condition as determined by the Standard of Truth**		
	Positive	Negative	Row total
Positive	TP	FP	$TP + FP$
Negative	FN	TN	$FN + TN$
Column total	$TP + FN$	$FP + TN$	$N = TP + TN + FP + FN$

Table 6: Accuracy table

The evaluation measures are described in terms of TP, TN, FN and FP. The Precision and Recall are defined as follows:

$$\begin{aligned} \text{Precision}_{+} &= \frac{TP}{TP + FP} \\ \text{Precision}_{-} &= \frac{TN}{TN + FN} \\ \text{Recall}_{+} &= \frac{TP}{TP + FN} \\ \text{Recall}_{-} &= \frac{TN}{TN + FP} \end{aligned}$$

Precision is the weighted average of precision positive and negative, while Recall is the weighted average of recall positive and negative. The F-measure is given by

$$\text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The value range of these measures is between 0 and 1, where 0 indicates the worst performance, while 1 indicates the best performance.
Sensitivity, Specificity and Accuracy are defined as follows:

$$\begin{aligned} \text{Sensitivity} &= \frac{TP}{TP + FN} \\ \text{Specificity} &= \frac{TN}{TN + FP} \\ \text{Accuracy} &= \frac{TN + TP}{TN + TP + FN + FP} \end{aligned}$$

Thus, sensitivity is the proportion of true positives that are correctly identified by a diagnostic test, showing how good the test is at detecting a disease. The numerical values of sensitivity represents the probability of a diagnostic test identifies patients who do in fact have the disease. The higher the numerical value of sensitivity, the less likely diagnostic test returns false-positive results. Specificity is the proportion of the true negatives correctly identified by a diagnostic test, suggesting how good the test is at identifying normal (negative) condition. The numerical value of specificity represents the probability of a test diagnoses a particular disease without giving false-positive results. Accuracy is the proportion of true results, either true positive or true negative, in a population. It measures the degree of veracity of a diagnostic test on a condition.
Note, accuracy can be determined from sensitivity and specificity, where prevalence is known. Prevalence is the probability of disease in the population at a given time

$$\text{Accuracy} = \text{Sensitivity} \times \text{Prevalence} + \text{Specificity} \times \big(1 - \text{Prevalence}\big)$$

Note, even if both sensitivity and specificity are high, say 99%, it does not suggest that the accuracy of the test is equally high as well.
For a given diagnostic test, the true positive rate (TPR) against false positive rate (FPR) can be measured, where $TRP = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$. Thus, TPR is equivalent to sensitivity and FPR is equivalent to specificity. All possible combinations of TPR and FPR compose a ROC space. One TPR and one FPR together determine a single point in the ROC space, and the position of a point in the ROC space shows the tradeoff between sensitivity and specificity, that is, the increase in sensitivity is accompanied by a decrease in specificity. Thus the location of the point in the ROC space depicts whether the diagnostic classification is good or not. In an ideal situation, a point determined by both TPR and FPF yields a coordinates $(0,1)$, or we can say that this point falls on the upper left corner of the ROC space. This idea point indicates the diagnostic test has a sensitivity of 100% and specificity of 100%. It is also called perfect classification. Diagnostic test with 50% sensitivity and 50% specificity can be visualised on the diagonal determined by coordinate $(0,0)$ and coordinates $(1,0)$. Theoretically, a random guess would give a point along this diagonal. A point predicted by a diagnostic test fall into the area above the diagonal represents a good diagnostic classification, otherwise a bad prediction.

10 Ensemble models

We briefly introduce ensemble models. More details can be found in textbooks by Rokach [2010], Zhou [2012], among others.
We let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output spaces, $\mathcal{D}$ the probability distribution, D the data set (sample), H the set of hypotheses, $I(\cdot)$ the indicator function. We let x be an instance or feature vector. For example, we consider the training data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where the instances x_i are independent and identically distributed (i.i.d.) and $y_i = f(x_i)$ where f is the ground-truth target function. That is, x_i is a vector and y_i takes value in $\mathbb{R}$. The learning process aim at constructing a learner h minimising the generalisation error

$$\mathcal{E}(h) = E_x[I(h(x) \neq f(x))]$$

We can also let $\mathcal{X}$ be the instance space, $\mathcal{Y}$ be the label space, define the labelled data set as $L = \{(x_1, y_1), ..., (x_l, y_l)\}$ and consider the binary classification tasks where $\mathcal{Y} = \{-1, +1\}$.

10.1 Introduction

10.1.1 Presentation

Ensemble methods train multiple learners to solve the same problem by combining them. It contains a number of learners called base learners generated from a base learning algorithm (decision tree, neural network etc). The base learners are also referred to as weak learners. A weak learner is just slightly better than random guess, while a strong learner is very close to perfect performance.
Early contributions to ensemble methods are:

- combining classifiers (for pattern recognition working on strong classifiers),
- ensembles of weak learners (machine learning boosting performance from weak to strong, such as, AdaBoost, Bagging, etc), and
- mixture of experts (neural networks considering a divide-and-conquer strategy).

Hansen et al. [1990] found that predictions made by the combination of a set of classifiers are often more accurate than the ones made by the best single classifier. Further, Schapire [1990] proved that weak learners (easy to obtain) can be boosted to strong learners.
Stacking (also called stacked generalisation) proposed by Wolpert [1992] is a popular method for improving results on

data mining competitions. It is a model ensembling technique used to combine information from multiple predictive models to generate a new model (see Figure (27)). It is assumed that the stacked model (also called 2nd-level model) will outperform each of the individual models due to its smoothing nature and ability to highlight each base model where it performs best and discredit the ones where it poorly performs. Thus, stacking is most effective when the base models are significantly different. Some of the advantages of the stacking techniques are to avoid over-fitting due to K fold cross validation and to interpret non-linearity between features due to treating output scores as features.

10.1.2 Definitions

We briefly introduce ensemble methods and then present a few facts on model stacking.

Definition 10.1 *An ensemble is a set of classifiers that learn a target function, and their individual predictions are combined to classify new examples.*

An ensemble is constructed in two steps, generating the base learners, and then combining them. There are two basic ways of aggregating classification methods:

1. Before the fact: Creates solutions to be combined (for example bagging).
   Suppose we want to create a robust method for determining if a given image contains a face:
   - Problem:It may be very difficult and computationally intensive to create a classifier for that task
   - One solution: detect eyes instead of faces
     - Advantage: A lot more efficient
     - Problem: this may produce models with low accuracy
     - Solution: Combine it with other similar classifiers
2. After the fact: Combines existing solutions (for example blending).
   Netflix challenge teams merging: Different models are already built. Find an intelligent way of combining them. Given each hypothesis $\widehat{f}_1, ..., \widehat{f}_M$ and a new instance x, we can use a linear regression to compute the prediction $g(x)$. That is,

$$\widehat{f}_1, ..., \widehat{f}_M \rightarrow g(x) = \sum_{i=1}^{M} \omega_i \widehat{f}_i(x)$$

   Choose ω_i to minimise the error on aggregation set.

Each of these methods can be more or less appropriate to particular problems. There are two paradigms of ensemble methods

1. sequential ensemble methods: the base learners are generated sequentially (AdaBoost). It exploit the dependence between the base learners, since the overall performance can be boosted in a residual-decreasing way.
2. parallel ensemble methods: the base learners are generated in parallel (Bagging). It exploit the independence between the base learners, since the error can be reduced dramatically by combining independent base learners.

Some examples of ensemble methods are bagging, boosting, random forests, extra trees. Nowadays, stacking and blending are also ensemble methods (see details in textbook by Zhou [2012]). We now briefly introduce stacking:

Definition 10.2 *Model stacking is a learning method that aims to improve predictive accuracy by combining predictions from multiple models. It is a particular case of ensemble learning.*

It is argued that model stacking leads to better predictions, at the expense of interpretability [6]. In general, it is used as a component of several winning entries in Kaggle competitions.

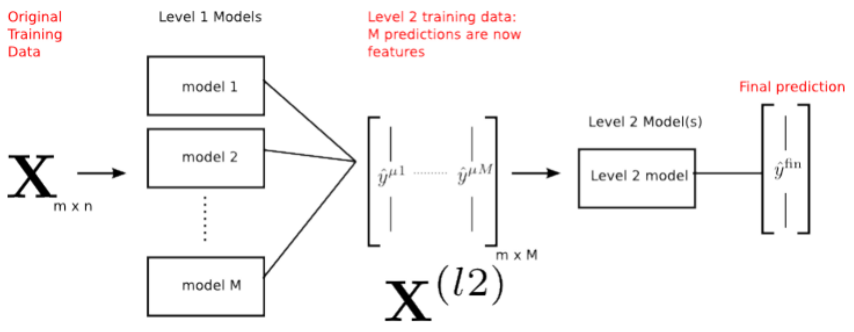


Figure 27: Two levels model stacking

10.1.3 Diversity

We call ensemble diversity the difference among the individual learners. The diversity is crucial to ensemble performance since a gain from combination requires the individual learners to be different. The major problem is that the individual learners are trained for the same task from the same training data, and thus they are usually highly correlated. However, several algorithms assume either independent or less correlated learners. Even if correlation between individual learners is considered, it is generally based on the assumption that the individual learners produce independent estimates of the posterior probabilities, which is not the case in practice. In addition, the problem of generating diverse individual learners is even more challenging if we consider that the individual learners must not be very poor, and otherwise their combination would not improve and could even worsen the performance. For example, when the performance of individual classifiers is quite poor, the added error of simple soft voted ensemble may become arbitrarily large. As a result, it is desired that the individual learners should be accurate and diverse. Nonetheless, combining only accurate learners is often worse than combining some accurate ones together with some relatively weak ones, since complementarity is more important than pure accuracy. In conclusion, the success of ensemble learning lies in achieving a good tradeoff between the individual performance and diversity. Unfortunately, there is no well-accepted formal definition of diversity, but it is crucial in ensemble learning.

10.1.4 Generalisation error

Ensemble methods are based on statistical tools and, as such, rely on error computation and error reduction. For instance, we take binary classification on classes $\{-1, +1\}$ and let f be the truth function. We suppose that each base classifier has an independent generalisation error ϵ, that is, for base classifier h_i

$$P(h_i(x) \neq f(x)) = \epsilon$$

After combining N number of such base classifiers according to

$$H(x) = \text{sgn}(\sum_{i=1}^{N} h_i(x))$$

[6] In mathematical logic, interpretability is a relation between formal theories that expresses the possibility of interpreting or translating one into the other.

the ensemble H makes an error only when at least half of its base classifiers make errors. By Hoeffding inequality, the generalisation error of the ensemble is

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor \frac{N}{2} \rfloor} \binom{N}{k} (1-\epsilon)^k \epsilon^{N-k} \leqslant e^{-\frac{1}{2}N(2\epsilon-1)^2}$$

It shows that the generalisation error reduces exponentially to the ensemble size N and ultimately approaches to zero as N approaches to infinity. Even though one can not get absolutely independent base learners generated from the same training data set, base learners with less dependence can be obtained by introducing randomness in the learning process, and a good generalisation ability can be expected by the ensemble. Further, parallel methods are favourable to parallel computing, speeding up the training process.
One can show that the generalisation error of an ensemble depends on a term related to diversity. Thus, one must estimate error decomposition to understand ensemble learning. Examples are the error-ambiguity decomposition and the bias-variance decomposition.

10.2 Some simple models

10.2.1 Decision tree

Decision tree (DT) learning is one of the most popular techniques for classification. The classification model learnt through these techniques is represented as a tree called a decision tree. It consists in a set of tree-structured decision tests working in a divide-and-conquer way as follows:

- Each non-leaf node is associated with a feature test also called a split, which splits data into different subsets according to their different values on the feature test.
- Each leaf node is associated with a label, which will be assigned to instances falling into this node.

When predicting, a series of feature tests is conducted starting from the root node, and the result is obtained when a leaf node is reached. On the other hand, the classification process starts by testing whether the value of the feature y-coordinate is larger than a threshold; if so, the instance is classified as "cross", and otherwise the tree tests whether the feature value of x-coordinate is larger than another threshold; if so, the instance is classified as "cross" and otherwise is classified as "circle". The process is recursive since in each step, a data set is given and a split is selected, then this split is used to divide the data set into subsets, and each subset is considered as the given data set for the next step. Remains to decide how to select the splits. For instance, dividing the training set D into subsets $D_1, ..., D_k$, Breiman et al. [1984] proposed the CART which uses Gini index for selecting the split maximising the Gini

$$G_{gini}(D; D_1, ..., D_k) = I(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} I(D_k)$$

where

$$I(D) = 1 - \sum_{y \in \mathcal{Y}} P^2(y|D)$$

Alternatively, Quinlan [1998] proposed to use the information gain criterion in the ID3 algorithm. Given a training set D, the entropy of D is defined as

$$\text{Ent}(D) = - \sum_{y \in \mathcal{Y}} P(y|D) \log P(y|D)$$

If the training set D is divided into subsets $D_1, ..., D_k$, the entropy may be reduced. The amount of the reduction is the information gain

$$G(D; D_1, ..., D_k) = \text{Ent}(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} \text{Ent}(D_k)$$

Thus, the feature-value pair which will cause the largest information gain is selected for the split. However, features with a lot of possible values will be favoured, disregarding their relevance to classification. Note, Quinlan [1993] had considered a variant of the information gain called the $C4.5$ based on the gain ratio

$$P(D; D_1, ..., D_k) = G(D; D_1, ..., D_k) \cdot \Big(\sum_{i=1}^{k} \frac{|D_k|}{|D|} \log \frac{|D_k|}{|D|} \Big)^{-1}$$

which takes normalisation on the number of feature values. In that setting, the feature with the highest gain ratio, among features with better-than-average information gains, is selected as the split. Note, $C4.5$ and CART can deal with numerical features.
To reduce the risk of overfitting, a general strategy is to employ pruning to cut off some tree branches caused by noise or peculiarities of the training set. Pre-pruning tries to prune branches when the tree is being grown, while post-pruning re-examines fully grown trees to decide which branches should be removed. This is achieved with the validation error: for pre-pruning, a branch will not be grown if the validation error will increase by growing the branch; for post-pruning, a branch will be removed if the removal will decrease the validation error.
Trees that are grown very deep tend to learn highly irregular patterns, but they overfit their training sets. That is, they have low bias, but very high variance.

10.2.2 Boosting

Boosting is an algorithm that converts weak learners to strong learners. We suppose the weak learner will work on any data distribution it is given, and take the binary classification task as an example. That is, we want to classify instances as positive and negative. The training instances in space $\mathcal{X}$ are drawn i.i.d. from distribution $\mathcal{D}$, and the truth function is f. Suppose the space X is made of three parts $\mathcal{X}_i$, $i = 1, 2, 3$, each takes $\frac{1}{3}$ amount of the distribution, and a learner working by random guess has 50% classification error on this problem. We want to get an accurate classifier on the problem, but can only have a weak classifier with correct classifications in spaces $\mathcal{X}_1$ and $\mathcal{X}_2$ and has wrong classifications in $\mathcal{X}_3$, thus has $\frac{1}{3}$ classification error. We denote this weak classifier as h_1. The idea of boosting is to correct the mistakes made by h_1 by deriving a new distribution $\mathcal{D}^{'}$ from $\mathcal{D}$, which makes the mistakes of h_1 more evident. For example, it focuses more on the instances in X_3. Then, we can train a classifier h_2 from $\mathcal{D}^{'}$. Again, suppose h_2 is also a weak classifier, which has correct classifications in $\mathcal{X}_1$ and $\mathcal{X}_3$ and has wrong classifications in $\mathcal{X}_2$. By combining h_1 and h_2 in an appropriate way, the combined classifier will have correct classifications in $\mathcal{X}_1$, and maybe some errors in $\mathcal{X}_2$ and $\mathcal{X}_3$. Again, we derive a new distribution $\mathcal{D}^{''}$ to make the mistakes of the combined classifier more evident, and train a classifier h_3 from the distribution, so that h_3 has correct classifications in $\mathcal{X}_2$ and $\mathcal{X}_3$. Then, by combining h_1, h_2 and h_3, we have a perfect classifier, since in each space of $\mathcal{X}_i$, $i = 1, 2, 3$, at least two classifiers make correct classifications. To conclude, boosting works by training a set of learners sequentially and combining them for prediction, where the later learners focus more on the mistakes of the earlier learners. See Algorithm (2).

Algorithm 2 Boosting procedure

```
Require: Input: sample distribution $\mathcal{D}$, base learning algorithm $L$, number of learning rounds $N$
Require: $\mathcal{D}_1 = \mathcal{D}$ initialise distribution
1: for $n = 1$ to $N$ do
2:     $h_n = L(\mathcal{D}_n)$ train a weak learner from distribution $\mathcal{D}_n$
3:     $\epsilon_n = P_{x \sim \mathcal{D}_n}(h_n(x) \neq f(x))$ evaluate the error of $h_n$
4:     $\mathcal{D}_{n+1} =$ Adjust-Distribution $(\mathcal{D}_n, \epsilon_n)$
5: end for
Require: Output: $H(x) =$ Combine-Outputs $(\{h_1(x), ..., h_n(x)\})$
```

Freund et al. [1997] proposed the AdaBoost to specify Adjust-Distribution and Combine-Outputs . See Algorithm (3).

Algorithm 3 AdaBoost algorithm

```
Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, base learning algorithm $L$, number of learning rounds $N$
Require: $\mathcal{D}_1 = \frac{1}{m}$ initialise the weight distribution
1: for $n = 1$ to $N$ do
2:     $h_n = L(D, \mathcal{D}_n)$ train a classifier $h_n$ from $D$ under distribution $\mathcal{D}_n$
3:     $\epsilon_n = P_{x \sim \mathcal{D}_n}(h_n(x) \neq f(x))$ evaluate the error of $h_n$
4:     if $\epsilon_n > \frac{1}{2}$ then
5:         break
6:     end if
7:     $\alpha_n = \frac{1}{2} \ln(\frac{1-\epsilon_n}{\epsilon_n})$ determine the weight of $h_n$
8:     $\mathcal{D}_{n+1}(x) = \frac{\mathcal{D}_n(x)}{Z_n} e^{-\alpha_n f(x) h_n(x)}$ update the distribution, where $Z_n$ is a normalisation factor which enables
       $\mathcal{D}_{n+1}$ to be a distribution
9: end for
Require: Output: $H(x) = \text{sgn}\big(\sum_{n=1}^N \alpha_n h_n(x)\big)$
```

Considering a binary classification on classes $\{-1, +1\}$, Friedman et al. [2000] modified the AdaBoost algorithm by minimising the exponential loss function

$$L_{exp}(h|\mathcal{D}) = E_{x \sim \mathcal{D}}[e^{-f(x)h(x)}]$$

using additive weighted combination of weak learners as

$$H(x) = \sum_{n=1}^N \alpha_n h_n(x)$$

Minimising the exponential loss by H, the partial derivative for every x is zero, that is, $\partial_{H(x)} e^{-f(x)H(x)} = 0$, so that

$$H(x) = \frac{1}{2} \ln \frac{P(f(x) = 1|x)}{P(f(x) = -1|x)}$$

As a result,

$$\text{sgn}(H(x)) = \arg \max_{y \in \{-1,1\}} P(f(x) = y|x)$$

which implies that $\text{sgn}(H(x))$ achieves the Bayes error rate. The H is obtained by iteratively generating h_n and α_n.

10.2.3 Bagging

Bootstrap AGGregatING (called Bagging) is composed of bootstrap and aggregation (see Breiman [1996c]). Since the combination of independent base learners will lead to a dramatic decrease of errors, we want to get the base learners as independent as possible. Ideally, given a data set, we want to sample a number of non-overlapped data subsets and then train a base learner from each of the subsets. However, this is in general not possible due to limited data. One solution is to consider data sample manipulation, such as data perturbation. For instance, bagging adopts the bootstrap distribution (see Efron et al. [1993]) for generating different base learners. It applies bootstrap sampling to obtain the data subsets for training the base learners. Given a training data set containing m number of training examples, a sample of m training examples will be generated by sampling with replacement. By applying the process N times, N samples of m training examples are obtained. Then, from each sample a base learner can be trained by applying the base learning algorithm. The outputs of the base learners are aggregated by voting for classification and averaging for regression. If the base learners are able to output confidence values, weighted voting or weighted averaging are often used. For instance, when forecasting instance (feature vector), the algorithm feeds the instance to its base classifiers and collects all of their outputs, and then votes the labels and takes the winner label as the prediction, where ties are broken arbitrarily. See Algorithm (4).

Algorithm 4 Bagging procedure

```
Require: Input: data set $\{(x_1, y_1), ..., (x_m, y_m)\}$, base learning algorithm $L$, number of base learner $N$
1: for $n = 1$ to $N$ do
2:     $h_n = L(D, \mathcal{D}_{bs})$ $\mathcal{D}_{bs}$ is the bootstrap distribution
3: end for
Require: Output: $H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{n=1}^{N} I(h_n(x) = y)$
```

Note bagging has a large variance reduction effect which is particularly effective with unstable base learners.

10.3 Random forest

Random forests (RF) or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (see Ho [1995] [1998]). The algorithm was extended by Breiman [2001]. Another extension combines the bagging idea from Breiman and random selection of features, introduced first by Ho and later independently by Amit et al. [1997], in order to construct a collection of decision trees with controlled variance.
The idea of ensemble learning is that a single classifier is not sufficient for determining class of test data since the latter is not able to distinguish between noise and pattern. Thus, random decision forests correct for decision trees' habit of overfitting to their training set. Ho established that forests of trees splitting with oblique hyperplanes can gain accuracy as they grow without suffering from overtraining, as long as the forests are randomly restricted to be sensitive to only selected feature dimensions. He eventually concluded that other splitting methods behave similarly, as long as they are randomly forced to be insensitive to some feature dimensions. Note that this observation of a more complex classifier (a larger forest) getting more accurate nearly monotonically is in sharp contrast to the common belief that the complexity of a classifier can only grow to a certain level of accuracy before being hurt by overfitting.

10.3.1 Tree bagging

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_m$ with responses $Y = y_1, ..., y_m$, bagging repeatedly (N times) selects a random sample with replacement of the training set and fits trees to these samples.

Algorithm 5 Tree Bagging

```
1: for $n = 1$ to $N$ do
2:     sample, with replacement, $n$ training examples from $X, Y$: called $X_n, Y_n$
3:     train a classification or regression tree $f_n$ on $X_n, Y_n$
4: end for
```

After training, predictions for unseen samples $x^{'}$ can be made by averaging the predictions from all the individual regression trees on $x^{'}$

$$\hat{f} = \frac{1}{N} \sum_{n=1}^{N} f_n(x^{'})$$

or by taking the majority vote in the case of classification trees.
This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Bootstrap sampling is a way of de-correlating the trees by showing them different training sets. An estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on $x^{'}$

$$\sigma^2 = \frac{1}{N-1} \sum_{n=1}^{N} \big(f_n(x^{'}) - \hat{f} \big)^2$$

An optimal number of trees N can be found using cross-validation, or by observing the out-of-bag error.

10.3.2 The procedure

Random forests (RF) is an extension of bagging incorporating randomised feature selection. During the construction of a component decision tree, at each step of split selection (deterministic), RF first randomly selects a subset of features, and then carries out the conventional split selection procedure within the selected feature subset. That is, RF use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. It is called feature bagging. Typically, for a classification problem with p features, $\sqrt{p}$ features are used in each split.
The parameter K controls the incorporation of randomness. When K equals the total number of features, the constructed decision tree is identical to the traditional deterministic decision tree, but when $K = 1$, a feature will be selected randomly. Breiman [2001] suggested $K = \log p$, the logarithm of the number of features.

Algorithm 6 Random tree algorithm

```
Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, feature subset size $K$
 1: $N$ ← create a tree node based on $D$
 2: if all instances in the same class then
 3:     return $N$
 4: end if
 5: $\mathcal{F}$ ← the set of features that can be split further
 6: if $\mathcal{F}$ is empty then
 7:     return $N$
 8: end if
 9: $\widetilde{\mathcal{F}}$ ← select $K$ features from $\mathcal{F}$ randomly
10: $N \cdot f$ ← the feature that has the best split point in $\widetilde{\mathcal{F}}$
11: $N \cdot p$ ← the best split point on $N \cdot f$
12: $D_l$ ← subset of $D$ with values of $N \cdot f$ smaller than $N \cdot p$
13: $D_r$ ← subset of $D$ with values of $N \cdot f$ no smaller than $N \cdot p$
14: $N_l$ ← call the process with parameters $(D_l, K)$
15: $N_r$ ← call the process with parameters $(D_r, K)$
16: return $N$
Require: Output: a random decision tree
```

Adding one further step of randomisation yields extremely randomized trees, or ExtraTrees. These are trained using bagging and the random subspace method, like in an ordinary random forest, but additionally the top-down splitting in the tree learner is randomised. Instead of computing the locally optimal feature/split combination (based on, information gain or the Gini impurity), for each feature under consideration, a random value is selected for the split. This value is selected from the feature's empirical range (in the tree's training set, that is, the bootstrap sample).
The technique is as follows: The first step in measuring the variable importance in a data set $D_n = \{(X_i, Y_i)\}_{i=1}^n$ is to fit a random forest to the data. During the fitting process the out-of-bag error for each data point is recorded and averaged over the forest (errors on an independent test set can be substituted if bagging is not used during training). To measure the importance of the jth feature after training, the values of the jth feature are permuted among the training data and the out-of-bag error is again computed on this perturbed data set. The importance score for the jth feature is computed by averaging the difference in out-of-bag error before and after the permutation over all trees. The score is normalised by the standard deviation of these differences.
As an example, we consider a random forest that performs sampling with replacement, such that, given N trees that are to be learnt, the results are based on these data set samples. Each tree is learnt using 3 features selected randomly. After the creation of n trees, when testing data is used, the decision which majority of trees come up with is considered as the final output. This approach also avoids problem of overfitting. The implementation is summarised in the Algorithm (7).

Algorithm 7 Random Forest

```
Require: Input: training set $D$, number of trees in the ensemble $N$
Require: Output: a composite model $M_*$
 1: for $n = 1$ to $N$ do
 2:     create bootstrap sample $D_n$ by sampling $D$ with replacement
 3:     select three features randomly
 4:     use $D_n$ and randomly select three features to derive tree $M_n$
 5: end for
 6: return $M_*$
```

10.4 Towards combination methods

Ensemble methods resort to combination to achieve a strong generalisation ability, where the combination method plays a crucial role. Some of the reasons for improvement are

- Statistical issue: high variance
- Computational issue: high computational variance
- Representational issue: high bias

Through combination, the variance as well as the bias of learning algorithms may be reduced.

10.4.1 Averaging

Averaging is the most fundamental combination method for numeric outputs. To illustrate the method, we take regression as an example. Thus, we consider the regression setting and suppose that we estimate N models (learners) $\{\mathcal{M}_1, ..., \mathcal{M}_N\}$, with $h_i = \mathcal{M}_i$, $i = 1, ..., N$, on a single data set D leading to corresponding predictions $h_1(x), ..., h_N(x)$ with instance x (or feature vector) where $h_i(x) = \hat{f}_i(x) \in \mathbb{R}$. In model averaging, we compute the prediction by directly averaging the outputs of individual learners

$$H(x) = \hat{f}_{ave}(x) = \sum_{i=1}^N \omega_i h_i(x) \tag{10.28}$$

where ω_i, $i = 1, .., N$, are fixed model weights. There are several ways for choosing the weights ω_i, $i = 1, .., N$.

10.4.1.1 Simple averaging The simplest one being to set $\omega_i = \frac{1}{N}$ to get a simple model average. Suppose the underlying true function we try to learn is $f(x)$, and x is sampled according to a distribution $p(x)$. The output of each learner can be written as the true value plus an error item as follows:

$$h_i(x) = f(x) + \epsilon_i(x) \; , \; i = 1, ..., N$$

Then, the mean squared error of h_i can be written as

$$\mathcal{E}(h) = \int \big(h_i(x) - f(x) \big)^2 p(x) dx = \int \epsilon_i^2(x) p(x) dx$$

and the averaged error made by the individual learners is

$$\overline{\mathcal{E}}(h) = \frac{1}{N} \sum_{i=1}^N \int \epsilon_i^2(x) p(x) dx$$

Similarly, the expected error of the combined learner is

$$\mathcal{E}(H) = \int \Big(\frac{1}{N} \sum_{i=1}^N h_i(x) - f(x) \Big)^2 p(x) dx = \int \big(\frac{1}{N} \sum_{i=1}^N \epsilon_i(x) \big)^2 p(x) dx$$

Thus, one can see that

$$\mathcal{E}(H) \leqslant \overline{\mathcal{E}}(h)$$

which states that the expected ensemble error will be no larger than the averaged error of the individual learners. In addition, if we assume that the errors ϵ_i, $i = 1, ..., N$, have zero mean and are uncorrelated

$$\int \epsilon_i p(x)dx = 0 \text{ and } \int \epsilon_i \epsilon_j p(x)dx = 0 \text{ for } i \neq j$$

we get

$$\mathcal{E}(H) = \frac{1}{N}\overline{\mathcal{E}}(h)$$

so that the ensemble error is smaller by a factor of N than the averaged error of the individual learners.

***Remark* 10.1** *In general, the errors are typically highly correlated since the individual learners are trained on the same problem.*

10.4.1.2 Weighted averaging More generally, assume $\omega_i \geqslant 0$ and $\sum_{i=1}^N \omega_i = 1$, then

$$\begin{aligned}
\mathcal{E}(H) &= \int \Big(\sum_{i=1}^N \omega_i h_i(x) - f(x) \Big)^2 p(x)dx \\
&= \int \Big(\sum_{i=1}^N \omega_i h_i(x) - f(x) \Big) \Big(\sum_{j=1}^N \omega_j h_j(x) - f(x) \Big) p(x)dx \\
&= \sum_{i=1}^N \sum_{j=1}^N \omega_i \omega_j C_{ij}
\end{aligned}$$

where

$$C_{ij} = \int \big(h_i(x) - f(x) \big) \big(h_j(x) - f(x) \big) p(x)dx$$

and C is the correlation matrix. Thus, the optimal weights can be solved by

$$\omega = \arg\min_{\omega} \sum_{i=1}^N \sum_{j=1}^N \omega_i \omega_j C_{ij}$$

Applying the Lagrange multiplier method, and assuming the correlation matrix C to be invertible, we get

$$\omega_i = \frac{\sum_{j=1}^N C_{ij}^{-1}}{\sum_{k=1}^N \sum_{j=1}^N C_{kj}^{-1}}$$

which provides a closed-form solution to the optimal weights.

***Remark* 10.2** *The correlation matrix is usually singular or ill-conditioned, since the errors of the individual learners are typically highly correlated and many individual learners may be similar since they are trained on the same problem.*

Therefore ω_i is generally infeasible, and moreover, it does not guarantee non-negative solutions.
Note, voting, is special cases or variants of weighted averaging. More generally any ensemble method can be regarded as trying a specific way to decide the weights for combining the individual learners, and different ensemble methods can be regarded as different implementations of weighted averaging.
For example, given a training set with m number of training examples, we can choose the coefficients by least squares

$$\min_{\omega_i : i \in [1,N]} \sum_{k=1}^{m} \big(y_k - \widehat{f}_{ave}(x_k) \big)^2$$

where we may want to impose restrictions on the weights such as non-negativity and a total sum of one. However, the least squares estimation might not work well since the regression models $\{h_1(\cdot), ..., h_N(\cdot)\}$ were previously estimated based on the same training data.
The minimisation does not take into account the complexity of the individual models, and will tend to overfit by putting too much weight on the most complex models (which will have low training errors).

***Remark* 10.3** *When the data is noisy and insufficient, the estimated weights are often unreliable.*

In general, simple averaging is appropriate for combining learners with similar performances, whereas if the individual learners exhibit non-identical strength, weighted averaging with unequal weights may achieve a better performance.

10.4.2 Voting

Voting is the most popular and fundamental combination method for nominal outputs. Thus, to illustrate the method, we consider classification and suppose given a set of N individual classifiers $\{h_1, ..., h_N\}$. We want to combine the h_i to predict the class label from a set of l possible class labels $\{c_1, ..., c_l\}$. We assume that for any instance x, the outputs of the classifier h_i are given as an l-dimensional label vector $(h_{i,1}(x), ..., h_{i,l}(x))^\top$, where $h_{i,j}(x)$ is the output of h_i for the class label c_j. The output can take different types of values according to the information provided by the individual classifiers. For instance:

- Crisp label: $h_{i,j}(x) \in \{0, 1\}$ which takes value one if h_i predicts c_j as the class label and zero otherwise.
- Class probability: $h_{i,j}(x) \in [0, 1]$ which is an estimate of the posterior probability $P(c_j|x)$.

For classifiers that produce un-normalised margins, such as SVMs, calibration methods such as Platt scaling or Isotonic Regression can be used to convert such an output to a probability.

10.4.2.1 Majority voting Majority voting (MV) is the most popular voting method. Every classifier votes for one class label, and the final output class label is the one that receives more than half of the votes; if none of the class labels receives more than half of the votes, a rejection option will be given and the combined classifier makes no prediction. The output class label of the ensemble is

$$H(x) = \begin{cases} c_j \text{ if } \sum_{i=1}^{N} h_{i,j}(x) > \frac{1}{2} \sum_{k=1}^{l} \sum_{i=1}^{N} h_{i,k}(x) \\ \text{ rejection } \text{ if a tax credit} \end{cases}$$

For example, given a total of N classifiers for a binary classification problem, the ensemble decision will be correct if at least $\lfloor \frac{N}{2} + 1 \rfloor$ classifiers choose the correct class label.
Assume that the outputs of the classifiers are independent and each classifier has an accuracy p [7]. Then, the probability of the ensemble for making a correct decision can be calculated using a binomial distribution (see Hansen et al. [1990]). The probability of obtaining at least $\lfloor \frac{N}{2} + 1 \rfloor$ correct classifiers out of N is

$$P_{mv} = \sum_{k=\lfloor \frac{N}{2}+1 \rfloor}^{N} \binom{N}{k} p^k (1-p)^{N-k}$$

Lam et al. [1997] showed that

[7] each classifier makes a correct classification at probability p

- if $p > \frac{1}{2}$, then P_{mv} is monotonically increasing in N, and

$$\lim_{N \to \infty} P_{mv} = 1$$

- if $p < \frac{1}{2}$, then P_{mv} is monotonically decreasing in N, and

$$\lim_{N \to \infty} P_{mv} = 0$$

- if $p = \frac{1}{2}$ then $P_{mv} = \frac{1}{2}$ for any N

***Remark* 10.4** *This result is obtained based on the assumption that the individual classifiers are statistically independent, but in practice the classifiers are generally highly correlated since they are trained on the same problem.*

10.4.2.2 Plurality voting Plurality voting (PV) takes the class label which receives the largest number of votes as the final winner. That is, the output class label of the ensemble is

$$H(x) = c_{\arg\max_j \sum_{i=1}^N h_{i,j}(x)}$$

and ties are broken arbitrarily. Note, plurality voting does not have a reject option. Further, in the case of binary classification, plurality voting coincides with majority voting.

10.4.2.3 Weighted voting If the individual classifiers have unequal performance, we want to give more power to the stronger classifiers in voting; this is realised by weighted voting. The output class label of the ensemble is

$$H(x) = c_{\arg\max_j \sum_{i=1}^N \omega_i h_{i,j}(x)}$$

where ω_i is the weight assigned to the classifier h_i. Again, we want $\omega_i > 0$ and $\sum_{i=1}^N \omega_i = 1$. Further, we need to decide how to obtain the weights. We let $\ell = (\ell_1, ..., \ell_N)^\top$ be the outputs of the individual classifiers, where ℓ_i is the class label predicted for the instance x by the classifier h_i and p_i denotes the accuracy of h_i. There is a Bayesian optimal discriminant function for the combined output on class label c_j

$$H_j(x) = \log\left(P(c_j)P(\ell|c_j)\right)$$

Assuming that the outputs of the individual classifiers are conditionally independent

$$P(\ell|c_j) = \prod_{i=1}^N P(\ell_i|c_j)$$

then one can show

$$H_j(x) = \log\left(P(c_j) + \sum_{i=1,\ell_i=c_j}^N \log\frac{p_i}{1-p_i} + \sum_{i=1}^N \log\left(1-p_i\right)\right.$$

Since $\sum_{i=1}^N \log(1-p_i)$ does not depend on the class label c_j, and $\ell_i = c_j$ can be expressed by the result of $h_{i,j}(x)$, the discriminant function can be simplified to

$$H_j(x) = \log\left(P(c_j) + \sum_{i=1}^N h_{i,j}(x)\log\frac{p_i}{1-p_i}\right.$$

The first term on the RHS does not rely on the individual learners, while the second term discloses that the optimal weights for weighted voting satisfy

$$\omega_i \propto \log \frac{p_i}{1-p_i}$$

which shows that the weights should be in proportion to the performance of the individual learners.

***Remark* 10.5** *The above formula obtained by assuming independence among the outputs of the individual classifiers, yet this does not hold since all the individual classifiers are trained on the same problem and they are usually highly correlated.*

10.4.2.4 Soft voting Soft voting (SV) is applied to classifiers which produce class probability outputs. The individual classifier h_i outputs a l-dimensional vector $(h_{i,1}(x), ..., h_{i,l}(x))^\top$ for the instance x, where $h_{i,j}(x) \in [0,1]$ can be regarded as an estimate of the posterior probability $P(c_j|x)$. In the case of uniform weighting, the simple soft voting average all the individual outputs, getting the final output for class c_j as

$$H_j(x) = \frac{1}{N}\sum_{i=1}^{N} h_{i,j}(x)$$

On the other hand, the weighted soft voting method can be any of the following three forms:

1. A classifier-specific weight is assigned to each classifier, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N} \omega_i h_{i,j}(x)$$

where ω_i is the weight of the classifier h_i.

2. A class-specific weight is assigned to each classifier per class, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N} \omega_{i,j} h_{i,j}(x)$$

where $\omega_{i,j}$ is the weight of the classifier h_i for the class c_j.

3. A weight is assigned to each example of each class for each classifier, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N}\sum_{k=1}^{m} \omega_{i,j,k} h_{i,j}(x)$$

where $\omega_{i,j,k}$ is the weight of the instance x_k of the class c_j for the classifier h_i.

Focussing on case (2), we have

$$h_{i,j}(x) = P(c_j|x) + \epsilon_{i,j}(x)$$

where $\epsilon_{i,j}(x)$ is the approximation error. In classification, the target output is given as a class label. In the case of an unbiased estimation the combined output $H_j(x)$ is also unbiased, and we can obtain a variance-minimised unbiased estimation of $H_j(x)$ for $P(c_j|x)$ by estimating the weights. Minimizing the variance of the combined approximation error $\sum_{i=1}^{N} \omega_{i,j}\epsilon_{i,j}(x)$ under the constraints $\omega_{i,j} \geqslant 0$ and $\sum_{i=1}^{N} \omega_{i,j} = 1$, we get the optimisation problem

$$\omega_j = \arg\min_{\omega_j} \sum_{k=1}^{m}\Big(\sum_{i=1}^{N} \omega_{i,j} h_{i,j}(x) - I(f(x_k) = c_j)\Big)^2, \; j = 1, ..., l$$

and solve for the weights. In general, soft voting is used for homogeneous ensembles. In the case of heterogeneous ensembles, one must perform a calibration where the class probability outputs are converted to class label outputs by setting $h_{i,j}(x) = 1$ if $h_{i,j}(x) = \max\{h_{i,j}(x)\}$ and 0 otherwise. The voting methods for crisp labels can then be applied.

10.5 Combining by learning

10.5.1 Stacking

The main idea of stacking is that we want to learn whether training data have been properly learned. Thus, a learner is trained to combine the individual learners. The individual learners are called the first-level learners, while the combiner is called the second-level learner, or meta-learner. Given the first-level learners we generate a new data set for training the second-level learner, where the outputs of the first-level learners are regarded as input features while the original labels are still regarded as labels of the new training data.
There are two approaches for combining models: voting and stacking. The differences between the stacking and voting framework are as follow:

- In contrast to stacking, no learning takes place at the meta-level when combining classifiers by a voting scheme.
- Label that is most often assigned to a particular instance is chosen as the correct prediction when using voting.

Practically, stacking is concerned with combining multiple classifiers generated by different learning algorithms $L_1, ..., L_N$ on a single dataset D, which is composed by a feature vector $d_i = (x_i, y_i)$. The stacking process can be broken into two phases:

1. Generate a set of base-level classifiers $h_1, ..., h_N$ where $h_n = L_n(D)$.
2. Train a meta-level classifier to combine the outputs of the base-level classifiers.

The implementation is summarised in the Algorithm (8).

Algorithm 8 Stacking procedure

```
Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, first level learning algorithms $L_1, ..., L_N$, second level learning algorithm $L$
Require: Output: a composite model $M_*$
 1: for $n = 1$ to $N$ do
 2:     $h_n = L_n(D)$ train a first-level learner by applying the first-level learning algorithm $L_n$
 3: end for
 4: $D^{'} = \emptyset$ generate a new data set
 5: for $i = 1$ to $m$ do
 6:     for $n = 1$ to $N$ do
 7:         $z_{i,n} = h_n(x_i)$ train a first-level learner by applying the first-level learning algorithm $L_n$
 8:     end for
 9:     $D^{'} = D^{'} \cup \big( (z_{i,1}, ..., z_{i,N}), y_i \big)$
10: end for
11: $h^{'} = L(D^{'})$ train the second-level learner $h^{'}$ by applying the second-level learning algorithm $L$ to the new data set $D^{'}$
Require: Output: $H(x) = h^{'}\big( h_1(x), ..., h_N(x) \big)$
```

To avoid overfitting, it is suggested that the instances used for generating the new data set are excluded from the training examples for the first-level learners, and a cross validation or leave-one-out procedure is often recommended. The stacking framework is as follows:

- The training set for the meta-level classifier is generated through a leave-one-out cross validation process

$$\forall i = 1, ..., m \text{ and } \forall n = 1, ..., N \text{ then } h_n^i = L_n(D - d_i)$$

- The learned classifiers are then used to generate predictions for $d_i : z_i^n = h_n^i(x)$
- The meta-level dataset consists of examples of the form $\Big(\big(z_i^1, ..., z_i^N \big), y_i \Big)$, where the features are the predictions of the base-level classifiers and the class is the correct class of the example in hand.

In the case of k-fold cross-validation, the original training data set D is randomly split into k almost equal parts $D_1, ..., D_k$. Let D_j and $D_{(-j)} = D \backslash D_j$ to be the test and training sets for the jth fold. Given N learning algorithms, a first-level learner $h_n^{(-j)}$ is obtained by using the nth learning algorithm on $D_{(-j)}$. For x_i in D_j, we let $z_{i,n}$ denotes the output of the learner $h_n^{(-j)}$ on x_i. Then, at the end of the entire cross-validation process, the new data set is generated from the N individual learners as

$$D^{'} = \{ \big(z_{i,1}, ..., z_{i,N}, y_i \big) \}_{i=1}^m$$

It is then applied to the second-level learning algorithm and the resulting learner $h^{'}$ is a function of $(z_1, ..., z_N)$ for y. Similarly to averaging described above, stacking is equivalent to solving the minimisation problem

$$\min_{\omega_m : m \in [1,M]} \sum_{i=1}^N \big(y_i - \widehat{f}_{avecv}(x_i) \big)^2 \tag{10.29}$$

where

$$\widehat{f}_{avecv}(x) = \sum_{m=1}^M \omega_m \widehat{f}_m^{(cv)}(x) \tag{10.30}$$

and $\widehat{f}_m^{(cv)}(\cdot)$ are cross validation predictions. That is, model stacking fits a linear regression model based on constructed predictors derived from different models. We say that the linear regression model is therefore the meta model for the stack.

***Remark* 10.6** *More generally, we have*

$$\widehat{f}_{avecv}(x) = g\big(\widehat{f}_1^{(cv)}(x), ..., \widehat{f}_M^{(cv)}(x) \big)$$

There is no reason for $g(\cdot)$ to be linear, except for simplicity.

This idea can be generalised and used with any algorithm as a meta model. For instance, it can be applied to classification. Some options are:

- Hard voting classifier: (weighted) majority voting across several models.
- Soft voting classifier: (weighted) probability averages.
- Model stacking: fit a meta model such as a logistic regression using cross validation predictions from multiple models as inputs.

10.5.2 Infinite ensemble

Since most ensemble methods exploit only a small finite subset of hypotheses, Lin et al. [2008] developed an infinite ensemble framework that constructs ensembles with infinite hypotheses. It is based on support vector machines (SVM) and corresponds to learning the combination weights for all possible hypotheses. By embedding infinitely many hypotheses into a kernel, it can be found that the learning problem reduces to an SVM training problem with specific kernels.

Let $\mathcal{H} = \{h_\alpha : \alpha \in \mathcal{C}\}$ denote the hypothesis space, where $\mathcal{C}$ is a measure space. The kernel that embeds $\mathcal{H}$ is defined as

$$K_{\mathcal{H},r}(x_i, x_j) = \int_{\mathcal{C}} \Phi_{x_i}(\alpha)\Phi_{x_j}(\alpha)d\alpha$$

where $\Phi_x(\alpha) = r(\alpha)h_\alpha(x)$ and $r : \mathcal{C} \to \mathbb{R}_+$ is chosen such that the integral exists for all x_i, x_j. α denotes the parameter of the hypothesis h_α, and $Z(\alpha)$ means that Z depends on α. It has been proved that the kernel is valid. Following SVM, the framework formulates the following (primal) problem:

$$\begin{aligned}
&\min_{\omega \in L_2(\mathcal{C}), b \in \mathbb{R}, \xi \in \mathbb{R}^m} \frac{1}{2}\int_{\mathcal{C}} \omega^2(\alpha)d\alpha + C\sum_{i=1}^m \xi_i \\
&\text{s.t. } y_i\Big(\int_{\mathcal{C}} \omega(\alpha)r(\alpha)h_\alpha(x)d\alpha + b\Big) \geqslant 1 - \xi_i \\
&\xi_i \geqslant 0 \, , \forall i = 1, ..., m
\end{aligned}$$

The final classifier obtained from this optimization problem is

$$g(x) = \text{sgn}\big(\int_{\mathcal{C}} \omega(\alpha)r(\alpha)h_\alpha(x)d\alpha + b\big)$$

By using the Lagrangian multiplier method and the kernel trick, the dual problem of the primal problem can be obtained, and the final classifier can be written in terms of the kernel $K_{\mathcal{H},r}$ as

$$g(x) = \text{sgn}\big(\sum_{i=1}^m y_i\lambda_i K_{\mathcal{H},r}(x_i, x) + b\big)$$

where the λ_i are the Lagrange multipliers. By equivalence, the above equation infinite ensemble over $\mathcal{H}$. The learning problem can be reduced to solving an SVM with the kernel $K_{\mathcal{H},r}$ and the final ensemble can be obtained by applying typical SVM solvers.

10.5.3 Other methods

There are many other combination methods available and we are going to briefly present a few of them.

10.5.3.1 The algebraic methods The algebraic methods which is based on a probabilistic framework (see Kittler et al. [1998]). Again, we let $h_{i,j}(x)$ be the class probability of c_j output from h_i. Bayesian decision theory states that given N classifiers, the instance x should be assigned to the class c_j which maximises the posteriori probability $P(c_j|h_{1,j}, ..., h_{N,j})$. From Bayes theorem, it follows that

$$P(c_j|h_{1,j}, ..., h_{N,j}) = \frac{P(c_j)P(h_{1,j}, ..., h_{N,j}|c_j)}{\sum_{i=1}^l P(c_i)P(h_{1,j}, ..., h_{N,j}|c_i)}$$

where $P(h_{1,j}, ..., h_{N,j}|c_j)$ is the joint probability distribution of the outputs from the classifiers. Assuming that the outputs are conditionally independent, that is, $P(h_{1,j}, ..., h_{N,j}|c_j) = \prod_{i=1}^N P(h_{i,j}|c_j)$, it follows that

$$\begin{aligned}
P(c_j|h_{1,j}, ..., h_{N,j}) &= \frac{P(c_j)\prod_{i=1}^N P(h_{i,j}|c_j)}{\sum_{i=1}^l P(c_i)\prod_{k=1}^N P(h_{k,j}|c_i)} \\
&= P^{N-1}(c_j)\prod_{i=1}^N P(c_j|h_{i,j})
\end{aligned}$$

Since $h_{i,j}$ is the probability output, we have $P(c_j|h_{i,j}) = h_{i,j}$. Thus, if all classes are with equal prior, we get the product rule for combination

$$H_j(x) = \prod_{i=1}^{N} h_{i,j}(x)$$

Following the same approach, Kittler et al. [1998] derived soft voting method, as well as the maximum/minimum/median rules which choose the maximum/minimum/median of the individual outputs as the combined output. For instance, the median rule generates the combined output according to

$$H_j(x) = \underset{i}{\text{med}}\big(h_{i,j}(x)\big)$$

where $\text{med}(\cdot)$ is the median statistic.

10.5.3.2 Mixture of experts Mixture of experts (ME) works in a divide-and-conquer strategy where a complex task is broken up into several simpler and smaller subtasks, and individual learners (called experts) are trained for different subtasks (see Jacobs et al. [1991], Xu et al. [1995]). Gating is usually employed to combine the experts. The individual learners are generated for different subtasks and there is no need to devote to diversity (see Figure (28)). Thus, the key problem becomes to find the natural division of the task and then derive the overall solution from sub-solutions. To do so, we make the experts local by targeting each expert to a distribution specified by the gating function, rather than the whole original training data distribution.

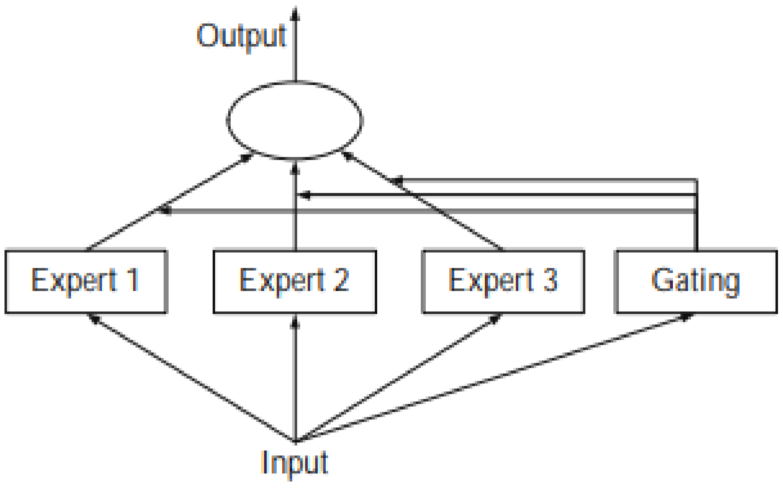


Figure 28: Mixture of experts

As an example, we assume N experts and let the output y be a discrete variable with possible values 0 and 1 for binary classification. Given an input x, each local expert h_i tries to approximate the distribution of y and obtains a local output $h_i(y|x;\theta_i)$ where θ_i is the parameter of the ith expert h_i. The gating function provides a set of coefficients $\pi_i(x;\alpha)$ weighting the contributions of experts where α is the parameter of the gating function. Thus, the final output of the ME is a weighted sum of all the local outputs produced by the experts given by

$$H(y|x;\Psi) = \sum_{i=1}^{N} \pi_i(x;\alpha) \cdot h_i(y|x;\theta_i)$$

where Ψ includes all unknown parameters. The output of the gating function is often modelled by the softmax function as

$$\pi_i(x;\alpha) = \frac{e^{v_i^\top x}}{\sum_{i=1}^N e^{v_j^\top x}}$$

where v_i is the weight vector of the ith expert in the gating function, and α contains all the elements in the vector v_i. When training, $\pi_i(x;\alpha)$ states the probability of the instance x appearing in the training set of the ith expert h_i. However, when testing, it defines the contribution of h_i to the final prediction.
The training procedure tries to achieve two goals: for given experts, to find the optimal gating function; for given gating function, to train the experts on the distribution specified by the gating function. The unknown parameters are usually estimated by the Expectation Maximisation (EM) algorithm (see Jordan et al. [1995], Xu et al. [1996]). Hierarchical mixture of experts (HME) extends mixture of experts (ME) into a tree structure where the experts are built from multiple levels of experts and gating functions (see Jordan et al. [1992]).

10.6 Diversity measures

A symmetric measure will keep the same when the values of 0 (incorrect) and 1 (correct) in binary classification are swapped.

10.6.1 Pairwise measures

When measuring diversity, the simplest approach is to measure the pairwise similarity/dissimilarity between two learners, and then average all the pairwise measurements to compute the overall diversity. One advantage of pairwise measures is that they can be visualized in $2d$-plots.
Given the data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, we consider binary classification and define the contingency table (see Table (7)) for two classifiers h_i and h_j, where $a+b+c+d = m$ are non-negative variables showing the numbers of examples satisfying the conditions specified by the corresponding rows and columns. We are going to present a few representative pairwise measures based on these variables.

Classifiers	$h_i = +1$	$h_i = -1$
$h_j = +1$	a	c
$h_j = -1$	b	d

Table 7: Contingency table

The disagreement measure (see Skalak [1996]) between h_i and h_j is defined as the proportion of examples on which two classifiers make different predictions

$$\operatorname*{dis}_{ij} = \frac{b+c}{m}$$

where dis_{ij} takes values in $[0, 1]$. The larger the value, the larger the diversity.
The Q-statistic (see Yule [1900]) of h_i and h_j is defined as

$$Q_{ij} = \frac{ad - bc}{ad + bc}$$

where Q_{ij} takes value in the range $[-1, 1]$. It is zero if h_i and h_j are independent, it is positive if h_i and h_j make similar predictions, otherwise it is negative.
The correlation coefficient (see Sneath et al. [1973]) of h_i and h_i is defined as

$$\rho_{ij} = \frac{ad - bc}{\sqrt{(a+b)(a+c)(c+d)(b+d)}}$$

which is classical when measuring the correlation between two binary vectors. Note, ρ_{ij} and Q_{ij} have the same sign, and $|\rho_{ij}| \geqslant |Q_{ij}|$.
The kappa-statistic (see Cohen [1960]) is defined as

$$\kappa_p = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}$$

where Θ_1 and Θ_2 are the probabilities that the two classifiers agree and agree by chance, respectively. The probabilities for h_i and h_j, on the data set D, are estimated as follows:

$$\begin{aligned} \Theta_1 &= \frac{a+d}{m} \\ \Theta_2 &= \frac{(a+b)(a+c)+(c+d)(b+d)}{m^2} \end{aligned}$$

Then, $\kappa_p = 1$ if the two classifiers totally agree on D, $\kappa_p = 0$ if they agree by chance, and $\kappa_p < 0$ occurs when the agreement is even less than what is expected by chance. All these measures do not require knowing the classification correctness.

10.6.2 Non-pairwise measures

We now focus on measuring the ensemble diversity directly rather than averaging pairwise measurements. We consider given a set of individual classifiers $\{h_1, ..., h_N\}$ and a data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where x_i is an instance and $y_i \in \{-1, +1\}$ is class label.
The Kohavi-Wolpert (KW) variance is based on the bias-variance decomposition of the error of a classifier (see Kohavi et al. [1996]). On an instance x, the variability of the predicted class label y is defined as

$$var_x = \frac{1}{2}\Big(1 - \sum_{y \in \{-1,+1\}} P^2(y|x)\Big)$$

Kuncheva et al. [2003] modified the variability to measure diversity by considering two classifier outputs: correct (denoted by $\tilde{y} = +1$) and incorrect (denoted by $\tilde{y} = -1$). They estimated $P(\tilde{y} = +1|x)$ and $P(\tilde{y} = -1|x)$ over individual classifiers as follows:

$$P(\tilde{y} = +1|x) = \frac{\rho(x)}{N} \text{ and } P(\tilde{y} = -1|x) = 1 - \frac{\rho(x)}{N}$$

where $\rho(x)$ is the number of individual classifiers that classify x correctly. By substituting these formulas in the KW-variance and averaging over the data set D the $\kappa\omega$ measure is obtained

$$\kappa\omega = \frac{1}{mN^2} \sum_{k=1}^{m} \rho(x_k)\big(N - \rho(x_k)\big)$$

where the larger the $\kappa\omega$ measurement, the larger the diversity.
Interrater agreement is a measure of interrater (interclassifier) reliability (see Fleiss [1981]). It can be used to measure the level of agreement within a set of classifiers (see Kuncheva et al. [2003]). It is measured as follows

$$\kappa = 1 - \frac{\frac{1}{N}\sum_{k=1}^{m} \rho(x_k)\big(N - \rho(x_k)\big)}{m(N-1)\overline{p}(1-\overline{p})}$$

where $\rho(x_k)$ is the number of classifiers that classify x_k correctly, and

$$\overline{p} = \frac{1}{mN} \sum_{i=1}^{N} \sum_{k=1}^{m} I(h_i(x_k) = y_k)$$

is the average accuracy of individual classifiers. Again, $\kappa = 1$ if the classifiers totally agree on D, and $\kappa \leqslant 0$ if the agreement is even less than what is expected by chance.
Kuncheva et al. [2003] showed that the KW-variance, the averaged disagreement dis_{avg} and the kappa-statistic κ were closely related. The relations are as follows

$$\kappa\omega = \frac{(N-1)}{2N} dis_{avg} \text{ and } \kappa = 1 - \frac{N}{(N-1)\overline{p}(1-\overline{p})} \kappa\omega$$

where $\overline{p}$ is as above.
In entropy, for an instance x_k, the disagreement will be maximised if a tie occurs in the votes of individual classifiers. Cunningham et al. [2000] directly calculated the Shannon's entropy on every instance and averaged them over D for measuring diversity, getting

$$Ent_{cc} = \frac{1}{m} \sum_{k=1}^{m} \sum_{y \in \{-1,+1\}} -P(y|x_k) \log P(y|x_k)$$

where

$$P(y|x_k) = \frac{1}{N} \sum_{i=1}^{N} I(h_i(x_k) = y)$$

can be estimated by the proportion of individual classifiers that predict y as the label of x_k. The computation of Ent_{cc} does not require to know the correctness of individual classifiers.
Assuming the correctness of the classifiers known, Shipp et al. [2002] defined their entropy measure as

$$Ent_{sk} = \frac{1}{m} \sum_{k=1}^{m} \frac{\min\big(\rho(x_k), N - \rho(x_k)\big)}{N - \lceil \frac{N}{2} \rceil}$$

where $\rho(x_k)$ is the number of individual classifiers that classify x correctly. The value of Ent_{sk} is in the range of $[0, 1]$, where 0 indicates no diversity and 1 indicates the largest diversity. Note, even though this formula does not use the logarithm function, it is easier to handle and faster to calculate.
Assuming that the diversity is maximised when the failure of one classifier is accompanied by the correct prediction of the other, Partridge et al. [1997] defined the generalised diversity as follows

$$gd = 1 - \frac{p(2)}{p(1)}$$

where

$$p(1) = \sum_{i=1}^{N} \frac{1}{N} p_i \text{ and } p(2) = \sum_{i=1}^{N} \frac{1}{N} \frac{(i-1)}{(N-1)} p_i$$

where p_i denotes the probability of i randomly chosen classifiers failing on a randomly drawn instance x. The gd value is in the range of $[0, 1]$, and the diversity is minimised when $gd = 0$. The authors also proposed a modified version of the generalised diversity, called coincident failure (CF), given by

$$cfd = \begin{cases} 0 \text{ if } p_0 = 1 \\ \frac{1}{1-p_0} \sum_{i=1}^{N} \frac{(N-i)}{(N-1)} p_i \text{ if } p_0 < 1 \end{cases}$$

In that setting, $cfd = 0$ if all classifiers give the same predictions simultaneously, and $cfd = 1$ if each classifier makes mistakes on unique instances.

11 Introduction to artificial neural networks

11.1 Neural networks

While wavelet analysis enable to decompose complex systems into simpler elements, in order to understand them, we can also gather simple elements to produce a complex system. Networks is one approach among others achieving that goal. They are characterised by a set of interconnected nodes seen as computational units receiving inputs and processing them to obtain an output. The connections between nodes, which can be unidirectional or bidirectional, determine the information flow between them. We obtain a global behaviour of the network, called emergent, since the abilities of the network supercede the one of its elements, making networks a very powerful tool. Since Neural Nets have been widely studied by computer scientists, electronic engineers, biologists, psychologists etc, they have been given many different names such as Artificial Neural Networks (ANNs), Connectionism or Connectionist Models, Multi-layer Percepetrons (MLPs), or Parallel Distributed Processing (PDP) to name a few. However, a small group of classic networks emerged as dominant such as Hopfield Networks (see Hopfield [1982]), Back Propagation (see Rumelhart et al. [1986] [1986b]), Competitive Networks and networks using Spiky Neurons.

Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. They are inspired from natural neurons receiving signals through synapses located on the dendrites, or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal through the axon, which might be sent to another synapse, and might activate other neurons. ANN is a high level abstraction of real neurons consisting of inputs (like synapses) multiplied by weights (strength of the respective signals), and computed by a mathematical function determining the activation of the neuron. Another function computes the output of the artificial neuron (sometimes in dependence of a certain threshold). Depending on the weights (which can be negative), the computation of the neuron will be different, such that by adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. In presence of a large number of neurons we must rely on an algorithm to adjust the weights of the ANN to get the desired output from the network. This process of adjusting the weights is called learning or training. The aim of training the network is to obtain the optimal network minimising the difference between the output and a target, given some input data. Then, the output of the optimal network becomes the optimal predictor. The optimisation process is usually done by gradient descent method, where the weights are updated by an amount being equal to the opposite of the gradient. This way of updating weights is called the standard back propagation.

11.1.1 The mathematical formalism

In machine learning, a task consist in learning a functional relation between a given input $U(p) \in \mathbb{R}^{N_i}$ and a desired output $\hat{Y}(p) \in \mathbb{R}^{N_o}$, for the set $p = 1, ..., P$, where P is the number of data points in the training data set $\{(U(p), \hat{Y}(p))\}$.
In a non-temporal task, the data points are independent of each other and the goal is to learn a function

$$Y(p) = f(U(p))$$

minimising an error measure $E = E(Y, \hat{Y})$.
In general, the relation between the input $U(p)$ and the desired output $\hat{Y}(p)$ can either be solved by a linear model as

$$Y(p) = W^{\top} U(p)$$

where $W \in \mathbb{R}^{N_i \times N_o}$ is a weight matrix, or by a nonlinear model

$$Y(p) = f(W^\top U(p))$$

where $f(\cdot)$ is the activation function.

***Remark* 11.1** *In general, the weight matrix is defined as $W \in \mathbb{R}^{N_o \times N_i}$ where the weight $[w_{ij}]$ goes from the jth input node to the ith output node. Instead, we define $W \in \mathbb{R}^{N_i \times N_o}$ and transpose it, to let the weight $[w_{ij}]$ goes from the ith input node to the jth output node.*

The training of neural networks (NNs) depends on the adaptation of free network parameters, the weight values. Thus, it is an optimisation task where the result is to find optimal weight set of the network that reduce an error function E. It can be formulated as the minimisation of an error function in the space of connection weights (see Appendix (13))

$$\min_W E(Y, \hat{Y})$$

In a temporal task, the input signal and target signal are set in a discrete time domain $t = 1, ..., T$ and the function we are trying to learn has memory. We are going to discuss both non-temporal and temporal tasks, but we first choose to describe the mathematical formalism in the case of temporal tasks.

11.1.1.1 The temporal tasks We let the time-varying input signal be an N_ith order column vector $U(t) = [u_i(t)]$, the generated output is an N_oth order column vector $Y(t) = [y_o(t)]$ and the target output $\hat{Y}(t)$ is an N_oth order column vector, where $t = 1, .., T$ and T is the number of data points in the training set $\{(U(t), \hat{Y}_t)\}$. The goal is to learn a function

$$Y(t) = f(\cdots, U(t-1), U(t))$$

such that $E(Y, \hat{Y})$ is minimised, where E is an error measure.
The relation between the input $U(t)$ and the desired output $\hat{Y}(t)$ can either be solved by a linear model as

$$Y(t) = W^\top U(t)$$

where $W \in \mathbb{R}^{N_i \times N_o}$, or by a nonlinear model.
In nonlinear models, we generally expand nonlinearly the input $U(t)$ into a high dimensional feature vector $X(t) \in \mathbb{R}^{N_x}$, and use linear models to get a reasonable output vector $Y(t)$. The output vector can be written as

$$Y(t) = f_{out}(W_{out}^\top X(t)) = f_{out}(W_{out}^\top \phi(\cdots, U(t-1), U(t)))$$

where $f_{out}(\cdot)$ is the output function (identity, sigmoid, or other), and $W_{out} \in \mathbb{R}^{N_x \times N_o}$ are the trained output weights. The functions

$$X(t) = \phi(\cdots, U(t-1), U(t))$$

transforming the current input $U(t)$ and its history $U(t-1), ...$ into a higher dimensional vector $X(t)$ are called kernels, and we refer to them in machine learning as expansion methods. Since these kernels have an unbounded number of parameters, we define them recursively as

$$X(t) = \phi(X(t-1), U(t)) \tag{11.31}$$

Expansion methods includes, among others, Support Vector Machines, Feedforward Neural Networks, Radial Basis Function approximators, Slow Feature Analysis, various Probability Mixture models.
As an example of such kernels, the recurrent neural networks (RNNs) can be written as

$$X(t+1) = f\big(W_{res}^\top \cdot X(t) + W_{in}^\top \cdot U(t+1) + W_{fb}^\top \cdot Y(t)\big) \tag{11.32}$$

where $f(\cdot)$ is an activation function, $W_{in} \in \mathbb{R}^{N_i \times N_x}$ is an input weight matrix, $W_{res} \in \mathbb{R}^{N_x \times N_x}$ is a connection weight matrix, and $W_{fb} \in \mathbb{R}^{N_o \times N_x}$ is a feedback weight matrix.
In addition to the function $f(\cdot)$, leaky integrator neurons performs a leaky integration of its activation from previous time steps, which can be applied before or after the activation function $f(\cdot)$. In the latter case, assuming no feedback, we get

$$X(t+1) = (1 - a\Delta t)X(t) + \Delta t f\big(W_{in}^\top \cdot U(t+1) + W_{res}^\top \cdot X(t)\big)$$

where Δt is a compound time gap between two consecutive time steps divided by the time constant of the system, and a is the delay (or leakage) rate. Setting $a = 1$ and redefining Δt as a constant α controlling the speed of the dynamics, we get

$$X(t+1) = (1-\alpha)X(t) + \alpha f\big(W_{in}^\top \cdot U(t+1) + W_{res}^\top \cdot X(t)\big) \tag{11.33}$$

which is an exponential moving average. In general, we set $a\Delta t \in [0,1]$ in the first equation and $\alpha \in [0,1]$ in the second such that a neuron neither retain, nor leak, more activation than it had. Thus, with one extra parameter we make sure that neuron activations $X(t)$ never go outside the boundary defined by $f(\cdot)$. In fact, the neuron activation performs a low-pass filtering of its activations with the cutoff frequency

$$f_c = \frac{a}{2\pi(1-a)\Delta t}$$

where Δt is the discretisation time step, allowing to tune the reservoirs for particular frequencies.
Non-temporal tasks using feedforward networks are functions, while RNNs may develop self-sustained temporal activation dynamics making them dynamical systems. Generally, supervised training of RNNs, such as gradient descent, adapt iteratively all weights according to their estimated gradients $\frac{\partial E}{\partial W}$ to minimise the output error $E = E(Y, \hat{Y})$. One can adapt backpropagation (BP) methods from feedforward neural networks in the case of RNNs, by propagating the gradient through network connections and time. For example, the Backpropagation Through Time (BPTT) has a runtime complexity of $O(N_x^2)$ per weight update per time step for a single output ($N_o = 1$). Alternatively, the Real-Time Recurrent Learning (RTRL) estimate the gradients recurrently, forward in time, but it has a runtime complexity of $O(N_x^4)$. Improvements in standard RNNs design were proposed independently by Maass et al. [2002] under the name of Liquid State Machines and Jaeger [2001] under the name of Echo State Networks. Over time, these types of models became known as Reservoir Computing.

11.1.1.2 Handling memory Maass et al. [2005a] [2005b] presented a computational theory characterising the gain in computational power that a fading memory system can acquire through feedback from trained readouts, in presence of noise. We briefly introduce this theory.
A map (or filter) F from input to output streams is defined to have fading memory if its current output at time t depends only on values of the input u during some finite time interval $[t-T, t]$.

Definition 11.1 *F has fading memory if there exists for every $\epsilon > 0$ some $\delta > 0$ and $T > 0$ so that*

$$|(Fu)(t) - (F\tilde{u})(t)| < \epsilon$$

for any $t \in \mathbb{R}$ and any input functions u, $\tilde{u}$ with $\|u(\tau) - \tilde{u}(\tau)\| < \delta$ for all $\tau \in [t-T, t]$.

This is a characteristic property of all filters that can be approximated by an integral over the input stream u, or more generally by Volterra or Wiener series.
In general, real-time computations involve memory or persistent internal states that can not be modelled with memory systems. It was shown that ANNs could be enlarged through feedback from trained readouts. As discussed above,

ANNs are special cases of dynamical systems, which themselves can be seen as having universal capabilities for analog computing.

Theorem 11.1 *A large class $\mathcal{S}_n$ of systems of differential equations of the form*

$$x_i^{'}(t) = f_i\big(x_1(t), ..., x_n(t)\big) + g_i\big(x_1(t), ..., x_n(t)\big) \cdot v(t) \; , \; i = 1, ..., n \tag{11.34}$$

where $v(t)$ is the input vector, are universal for analog computing in the following sense:
It can respond to an external input $u(t)$ with the dynamics of any n-th order differential equation of the form

$$z^{(n)}(t) = G\big(z(t), z^{'}(t), z^{''}(t), ..., z^{(n-1)}(t)\big) + u(t) \tag{11.35}$$

for arbitrary smooth functions $G : \mathbb{R}^n \to \mathbb{R}$ if the input term $v(t)$ is replaced by a suitable memoryless feedback function $K\big(x_1(t), ..., x_n(t), u(t)\big)$ and if a suitable memoryless readout function $h\big(x_1(t), ..., x_n(t)\big)$ is applied to its internal state $\big(x_1(t), ..., x_n(t)\big)$.
Also, the dynamic responses of all systems consisting of several higher order differential equations of the form Equation (11.35) can be simulated by fixed systems of the form Equation (11.34) with a corresponding number of feedbacks.

The authors characterised Equation (11.34) as follows:

$$x_i^{'}(t) = -\lambda_i x_i(t) + \sigma\big(\sum_{j=1}^{n} a_{ij} \cdot x_j(t)\big) + b_i \cdot v(t) \; , \; i = 1, ..., n \tag{11.36}$$

where $\sigma(\cdot)$ is a standard activation function. Note, it is similar to Equation (11.33). Further, if the activation function σ is also applied to the term $v(t)$, the system in Equation (11.36) can still simulate arbitrary differential equations of the form Equation (11.35) with bounded inputs $u(t)$ and bounded responses $z(t), ..., z^{(n-1)}(t)$.
As explained by Branicky [1995], all Turing machines can be simulated by systems of differential equations of the form Equation (11.35). Thus, systems of the form Equation (11.34), enlarged through feedback, are also universal for digital computing.
Casey [1996], Maass et al. [1998], among others, showed that additive noise, even with arbitrarily small bounded amplitude, reduces the non-fading memory capacity of any recurrent neural network to some finite number. As a result, arbitrary Turing machines can no-longer be simulated by such networks. Nonetheless, given this a priori limitation, feedback still provides noisy fading memory systems with maximum possible computational power. Thus, any finite state machine can be emulated by a fading memory system with feedback, despite the noise in the system.

Theorem 11.2 *Feedback allows linear and nonlinear fading memory systems, even in the presence of additive noise with bounded amplitude, to employ the computational capability and non-fading states of any given finite state machine for real-time processing of time varying inputs.*

Thus, the learning induced generation of high-dimensional attractors through feedback provides a new model for capturing the characteristics of persistent signal response.

11.1.2 Presenting ANNs

Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples. The basic architecture of an ANN is a network comprised of nodes (called *neurons* in biology) interacting with each other via incoming and outgoing weighted connections indicating the strength degree of dendrite between neurons. While McCulloch et al. [1943] introduced the first ANN model, most modelling of neural learning networks has been based on synapses of a general type described by Hebb [1949] and Eccles [1953]. Given some input, the network will be activated and this activation signal will be passed throughout the rest of the network through the connections. Generally, an ANN model is specified by its topology, node characteristics and the training algorithms. There are a variety of ANNs with various topologies, node properties and training algorithms. Rosenblatt [1962] introduced the principles of perceptrons. Perceptrons were mainly modelled with neural connections in a forward direction $A \rightarrow B \rightarrow C \rightarrow D$. Consequently, the analysis of networks with strong backward coupling $A \leftrightarrows B \leftrightarrows C$ proved intractable. Further, perceptron modelling required synchronous neurons like a conventional digital computer. Hopfield [1982] overcame these difficulties and triggered the research interest in the area by demonstrating the computational capabilities of networks with symmetric connections. The main contribution of the paper was the introduction of the energy function of the network which turns to be their most useful property in the context of optimisation. Two of the most important distinct types of ANNs are feedforward and feedback network. ANNs with cycles are *feedback* networks, which are also referred to as *recurrent* neural networks (RNNs). Those networks with acyclic connections are called *feedforward* neural networks (FNNs). Algorithms such as back-propagation use gradient descent to tune network parameters to best fit a training set of input-output pairs, making the learning process robust to errors in the training data (see Rumelhart et al. [1986] [1986b]). Various successful applications to practical problems developed, such as learning to recognise handwritten characters (see LeCun et al. [1989]) and spoken words (see Lang [1990]), learning to detect fraudulent use of credit cards, drive autonomous vehicles on public highways (see Pomerleau [1993]). Rumelhart et al. [1994] provided a survey of practical applications. Since McCulloch et al. [1943] introduced the first ANN model, various models where developed with different functions, accepted values, topology, learning algorithms, hybrid models where each neuron has a larger set of properties etc. While there is various types of classifiers, neural network based classifiers dominate the literature. Yet, compared to traditional NNs, higher order neural networks (HONNs) have several unique characteristics, including

- stronger approximation with faster convergence property
- greater storage capacity
- higher fault tolerance capability

However, its major drawback is the exponential growth in the number of weights with the increasing order of the network. As a special case of the feedforward HONN, the Pi-Sigma networks (PSNs) introduced by Shin et al. [1991] have the capability of higher order neural networks, but using a smaller number of weights. In order to enhance the learning capability of neural network, many researchers have improved the system by combining other techniques such as fuzzy logic (see Zadeh [1994]), genetic algorithm (see Harrald et al. [1997]).

For simplicity of exposition we are first going to present the Hopfield networks (see Hopfield [1982]) and then briefly describe the back-propagation algorithm proposed by Rumelhart et al. [1986].

11.1.3 The Hopfield networks

11.1.3.1 Description The Hopfield neural network model (see Hopfield [1982]) consists of a fully connected network of units (or neurons). The connections between the units are weighted, such that for any two units i and j, w_{ij} is the weight of the connection between them. The model assumes symmetrical weights ($w_{ij} = w_{ji}$) and in most of the cases zero self-coupling terms ($w_{ii} = 0$). A connection with positive weight is an excitatory connection, while a connection with negative weight is an inhibitory connection. A unit i is characterised by its output (or state) v_i, the activation (or network input) u_i that receives from the other units, and a threshold θ_i. The network state is given by the output (or state) vector $v = \big(v_1, v_2, ..., v_n\big)$.
Each unit receives input from all the other units and forwards its output to all the other units. The output of a unit is updated by the dynamics of the network. In general, the output behaviour is described as a function of the activation, where the activation depends on the weighted summation of the inputs and the threshold. In general, we follow the dynamics rule from McCulloch et al. [1943], given by

$$v_i(t+1) = \Phi(u_i(t)) = \Phi\big(\sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i\big) \tag{11.37}$$

where the activation is

$$u_i(t) = w_i \cdot v(t) + \theta_i = \sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i \tag{11.38}$$

and θ_i is the threshold. The activation function $\Phi(x)$ can take several forms:

- it can be the unit step function (Heaviside function)

  $$\Theta(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x \leqslant 0 \end{cases}$$

  in which case we have a discrete Hopfield network with binary states $\{0, 1\}$.

- the sign function

  $$\text{sgn}(x) = \begin{cases} 1 \text{ if } x \geqslant 0 \\ -1 \text{ if } x < 0 \end{cases}$$

  in which case the network is discrete but with values $\{1, -1\}$.

- a network with continuous-valued units (analog Hopfield network), where the values fall in the range $[0, 1]$ or $[-1, 1]$ can be obtained with the sigmoid (or logistic) function

  $$g_\beta(x) = \frac{1}{1 + e^{-2\beta x}}$$

  The parameter β is usually taken as $\beta = \frac{1}{T}$ where T is a virtual temperature. The latter adjusts the sharpness of the sigmoid (or hyperbolic tangent) function and at the limit $T \to 0$ (absolute temperature) the output becomes discrete. We can also use the hyperbolic tangent function

  $$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Note that

$$\text{sgn}(x) = 2\Theta(x) - 1 \text{ and } tanh(\beta x) = 2g_\beta(x) - 1$$

so that the $\{0, 1\}$ (or $[0, 1]$) model can be easily transformed to $\{-1, 1\}$ (or $[-1, 1]$) and vice-versa. In the limit $\beta \to \infty$ the values become discrete.

All the networks above are deterministic in the sense that the next state is an explicit function of the previous state and the characteristics of the network. In the case of the stochastic Hopfield networks the probability of a unit to be in a particular state is drawn from a probability distribution such as Boltzmann or Cauchy. Alternatively, the stochastic network can be viewed as a deterministic network, where the threshold of a unit is variable and is drawn from a probability density.

Note, the updating policy can be

- synchronous, in which case all the units are updated simultaneously at each time step, or
- asynchronous, where either the units are updated in sequence one at each time step (sequential asynchronous update) or at each time step one randomly chosen unit is updated (random asynchronous update).

In summary, the properties that characterize different Hopfield networks are

- Discrete vs Continuous-Valued Units
- Deterministic vs Stochastic Networks
- Synchronous vs Asynchronous Update (Sequential or Random)

11.1.3.2 The energy function The essential ingredient of the learning process is the Hebbian property which is the modification of w_{ij} by correlations like

$$\Delta w_{ij} = \big[v_i(t)v_j(t)\big]_{average}$$

where the average is some appropriate calculation over past history. In addition, the Hopfield model has stable limit points. That is, the energy function of a Hopfield network is a function defined over the state space of the network given by

$$\begin{aligned} E(v) &= -\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n v_i v_j w_{ij} - \sum_{i=1}^n v_i\theta_i \\ &= -\sum_{i=1}^n v_i\big(\frac{1}{2}\sum_{j=1}^n v_j w_{ij} + \theta_i\big) \\ &= -\sum_{i=1}^n v_i u_i = -u\cdot v \end{aligned} \quad (11.39)$$

Note, this function always decreases (not necessarily monotonically because of the threshold) during the evolution of the network due to the dynamics of the systems given in Equation (11.37). State changes (altering v_i) will continue until a least (local) E is reached. This case is isomorphic with an Ising model.
Thus, the energy function can be viewed as defining an energy landscape and the dynamics can be thought of as the motion of a small sphere on the energy surface under the influence of gravity and friction. Consequently, the network performs as a minimiser of the energy function and will reach a local or global minimum of the function during its evolution.
One can show that whenever a unit i changes state, the energy difference ΔE_i is given by

$$\Delta E_i = -u_i\Delta v_i = -\big(\sum_{j=1}^n w_{ij}v_j + \theta_i\big)\Delta v_i \quad (11.40)$$

where u_i is the activation in Equation (11.38). As an example, we consider the $\{0,1\}$ network with dynamics given in Equation (11.37). Then

- If the activation is positive, then either $\Delta u_i = 0$ $(1 \to 1)$ or $\Delta u_i = 1$ $(0 \to 1)$ and thus $\Delta E_i \leqslant 0$.
- If the activation is negative, then either $\Delta u_i = 0$ $(0 \to 0)$ or $\Delta u_i = -1$ $(1 \to 0)$ and thus $\Delta E_i \leqslant 0$.

The argument is similar for the other models. Note, the main property of the energy function given in Equation (11.39) is held only if the weights of the connections are symmetric and the self-coupling terms are zero or positive. This is the case for the Hopfield network so that its energy function is called Lyapunov function.

11.1.4 The multilayer perceptron

The *multilayer perceptron* (MLP) is formed in layers, the idea being that multilayer ANN can approximate any continuous function. This algorithm is a layer feed-forward ANN, since the artificial neurons are organised in layers with signals sent forward and with errors propagated backwards. The network receives inputs via neurons in the input layer, and the output of the network is given via neurons on the output layers. There may be one or more intermediate hidden layers. For clarity, we consider a network with three layers, that is, an input layer, a hidden layer, and an output layer. The hidden layer is used for capturing the non-linear relationships among variables. It is illustrated in Figure 29, where the bottom layer is the *input layer*. Then the information is propagated to the middle layer, called the *hidden layer*. Finally, the output layer receives the incoming value. This process is termed the *forward pass*. Note that the S-shaped curve is the squashing function which forces the output of the unit to fall between certain bounded interval. Except for the input layer, the output value of each unit in the other layers should be passed through the squashing function before transferring to the next layer. The back-propagation algorithm uses supervised learning where we provide the algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. Defining the network function f_{out} as a particular implementation of a composite function from input to output space, the learning problem consists in finding the optimal combination of weights so that f_{out} approximates a given function f as closely as possible (see Appendix (11.1.1)). However, in practice the function f is not given explicitly but only implicitly through some examples. Cybenko [1989] hinted at the universal approximation theorem for sigmoid activation functions. Funahashi [1989] and Hornik et al. [1989] showed that a MLP with one hidden layer and sufficient non-linear units could approximate any continuous function on a compact input domain to arbitrary precision. Therefore, MLP are recognised as universal approximators. Hornik [1991] showed that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators. The universal approximation theorem can be stated as follows:

Theorem 11.3 *The universal approximation theorem*
Let $\Phi(\cdot)$ be a nonconstant, bounded, and continuous function. Let I_m denotes the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\epsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, ..., N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \Phi(w_i^\top x + b_i)$$

as an approximate realisation of the function f where f is independent of Φ. That is,

$$|F(x) - f(x)| < \epsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

Note, the theorem still holds when replacing I_m with any compact subset of $\mathbb{R}^m$. Kurkova [1992] took advantage of techniques developed by Kolmogorov to give a direct proof of the universal approximation capabilities of perceptron type networks with two hidden layers.

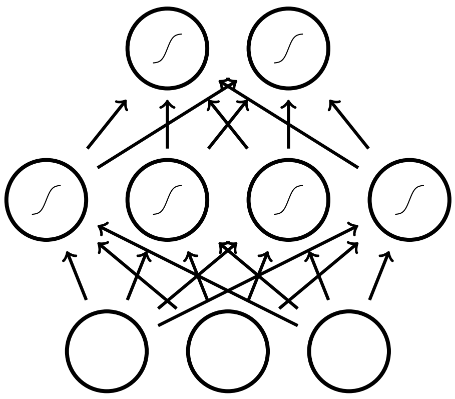


Figure 29: **A multilayer perceptron.** The three layers are input, hidden, and output layers, respectively, from bottom to top. The S-shaped curves in the hidden and output layers are the sigmoid squashing functions applying to the net values.

11.1.5 Gradient descent and the delta rule

As explained by Rumelhart et al. [1986] [1986b], one way forward to finding the optimal combination of weights in the multilayer perceptron is to consider the delta rule, which uses gradient descent to search the hypothesis space of possible weight vectors to find the weights best fitting the training examples. The gradient descent provides the basis for the backpropagation algorithm, which can learn networks with many interconnected units.
Given a vector of input $x \in \mathbb{R}^n$ together with a weight vector $w \in \mathbb{R}^n$, we consider the task of training an unthresholded perceptron, that is, a linear unit for which the output O is given by

$$O(x) = w \cdot x$$

A linear unit is the first stage of a perceptron without the threshold. We must specify a measure for the training error of a hypothesis (weight vector), relative to the set $\mathcal{D}$ of training examples (see Appendix (11.1.1)). A common measure is to use

$$E(w) = \frac{1}{2} \sum_{p=1}^{P} (O_p - d_p)^2$$

where d_p is the target output for training example p and O_p is the output of the linear unit for training example p. One can show that under certain conditions, the hypothesis minimising E is also the most probable hypothesis in H given the training data. The entire hypothesis space of possible weight vectors and their associated E values produce an error surface. Given the way in which we defined E, for linear units, this error surface must always be parabolic with a single global minimum.
The gradient descent search (see Appendix (13.2.1)) determines a weight vector minimising E by starting with an arbitrary initial weight vector, and then repeatedly modifying it in small steps. At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface, until a global minimum error is reached. This direction is found by computing the derivative of E with respect to each component of the vector w, called the gradient of E with respect to w, given by

$$\nabla E(w) = [\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}]$$

The gradient specifies the direction that produces the steepest increase in E, and its negative produces the steepest decrease. Hence, the training rule for gradient descent is

$$w \leftarrow w + \Delta w$$

where

$$\Delta w = -\eta \nabla E(w)$$

where $\eta > 0$ is the learning rate determining the step size in the gradient descent search. The negative sign implies that we move the weight vector in the direction that decreases E. Given the definition of $E(w)$ we can easily differentiate the vector of $\frac{\partial E}{\partial w_i}$ derivatives as

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{p=1}^{P} (O_p - d_p)^2 \text{ , } i = 1, ..., n$$

which gives after some calculation

$$\frac{\partial E}{\partial w_i} = \sum_{p=1}^{P} (O_p - d_p) x_{i,p}$$

where $x_{i,p}$ is the single input component x_i for training example p. The weight update rule for gradient descent is

$$\Delta w_i = -\eta \sum_{p=1}^{P} (O_p - d_p) x_{i,p} = \eta \sum_{p=1}^{P} (d_p - O_p) x_{i,p}$$

In the case where η is too large, the gradient descent search may overstep the minimum in the error surface rather than settling into it. One solution is to gradually reduce the value of η as the number of gradient descent steps grows. Gradient descent is a strategy for searching through a large or infinite hypothesis space which can be applied whenever

1. the hypothesis space contains continuously parameterised hypotheses.
2. the error can be differentiated with respect to these hypothesis parameters.

The key practical difficulties in applying gradient descent are

1. converging to a local minimum can sometimes be quite slow.
2. if there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

While the gradient descent training rule computes weight updates after summing over all the training examples in $\mathcal{D}$, the stochastic gradient descent approximate this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example. Hence, as we iterate through each training example, we update the weight according to

$$\Delta w_i = \eta (d - O) x_i$$

where the subscript i represents the ith element for the training example in question. One way of viewing this stochastic gradient descent is to consider a distinct error function $E_p(w)$ defined for each individual training example p as follow

$$E_p(w) = \frac{1}{2} (d_p - O_p)^2$$

where d_p and O_p are the target value and the unit output value for training example p. We therefore iterates over the training examples p in $\mathcal{D}$, at each iteration altering the weights according to the gradient with respect to $E_p(w)$. The sequence of these weight updates provides a reasonable approximation to descending the gradient with respect to the original error function $E(w)$. This training rule is known as the delta rule, or sometimes the least-mean-square (LMS) rule. Note, the delta rule converges only asymptotically toward the minimum error hypothesis, but it converges regardless of whether the training data are linearly separable.

11.1.6 Introducing multilayer networks

While a single perceptron can only express linear decision surfaces, multilayer networks are capable of expressing a rich variety of nonlinear decision surfaces. Since multiple layers of cascaded linear units can only produce linear functions, we need to introduce another unit to represent highly nonlinear functions. That is, we need a unit whose output is a nonlinear function of its inputs, but which is also a differentiable function of its input. We are now going to discuss such a unit and then describe how to apply the gradient descent in the case of multilayer networks.

11.1.6.1 Describing the problem We consider a feedforward network with N_i input units, N_o output units and N_k hidden layers which can exhibit any desired feedforward connection pattern. We are also given a training set $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_P, \overline{d}_P)\}$ consisting of P ordered pairs of n- and m-dimensional vectors called the input and output patterns, respectively. Note, the training set may consits of sequential data, or time-varying input, which we then represent as $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_T, \overline{d}_T)\}$ with T ordered pairs. We assume that the primitive functions at each node of the network is continuous and differentiable, and that the weights of the edges are real numbers selected randomly so that the output $\overline{O}_p$ of the network is initially different from the target $\overline{d}_p$. The idea being to minimise the distance between $\overline{O}_p$ and $\overline{d}_p$ for $p = 1, ..., P$ by using a learning algorithm searching in a large hypothesis space defined by all possible weight values for all the units in the network. There are different measures of error, and for simplicity we let the error function of the network be given by

$$E = \frac{1}{2} \sum_{p=1}^{P} ||\overline{O}_p - \overline{d}_p||_2^2$$

which is a sum of L_2 norms. After minimising the function with a training set, we can consider new unknown patterns and use the network to interpolate it. We use the backpropagation algorithm to find a local minimum to the error function by computing recursively the gradient of the error function and correcting the initial weights. Every one of the j output units of the network is connected to a node evaluating the function $\frac{1}{2}(O_{p,j} - d_{p,j})^2$ where $O_{p,j}$ and $d_{p,j}$ denote the jth component of the output vector $\overline{O}_p$ and the target vector $\overline{d}_p$, respectively. The outputs of the additional m nodes are collected at a node which adds them up and gives the sum E_p as its output. The same network extension is built for each pattern $\overline{d}_p$. We can then collect all the quadratic errors and output their sum, obtaining the total error for a given training set $E = \sum_{p=1}^{P} E_p$. Since E is computed exclusively through composition of the node functions, it is a continuous and differentiable function of the q weights $w_1, .., w_q$ in the network. Therefore, we can minimise E by using an iterative process of gradient descent where we need to calculate the gradient

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_q}\right)$$

and each weight is updated with the increment

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \text{ for } i = 1, .., q$$

where η is a learning constant, that is, a proportionality parameter defining the step length of each iteration in the negative gradient direction. Hence, once we have a method for computing the gradient,we can adjust the weights iteratively until we find a minimum of the error function where $\nabla E \approx 0$.

***Remark* 11.2** *In the case of multilayer networks, the error surface can have multiple local minima. Hence, the minimisation by steepest descent will only produce a local minimum of the error function, and not necessarily the global minimum error.*

11.1.6.2 Describing the algorithm For simplicity of exposition we first describe the algorithm in the case where there is no hidden layer and the training set consists of a single input-output pair ($P = 1$). We assume a set of artificial neurons $\{f_{i,j}\}$, each receiving $\{x_i\}_{i=1}^{N_i}$ input and computing $\{O_j\}_{j=1}^{N_o}$ output, and we focus our attention on one of the weights, $w_{i,j}$, going from input i to neuron j in the network. We let W denote the $N_i \times N_o$ weight matrix with element $w_{i,j}$ at the ith row and the jth column and we let $w(j)$ be the $N_i \times 1$ row vector for the jth column. In the back-propagation algorithm, given the input vector $x = (x_1, .., x_{N_i})$, the activation function (also called net_j) for the jth neuron satisfies the weighted sum

$$A_j(x, w) = x \cdot w(j) = \sum_{i=1}^{N_i} x_i w_{i,j} \, , j = 1, .., N_o$$

which only depends on the inputs and the weights $w_{i,j}$ from input i to neuron j. We let the output function (or threshold box, or activation function) be a function g of the activation function, getting

$$O_j(x,w) = g(\tilde{A}_j(x,w)) \ , \ j = 1, .., N_o$$

where $\tilde{A}_j(x,w) = A_j(x,w) + b_j$ and b_j is a bias value. In compact form, the output vector of all units is

$$O(x,w) = g(xW)$$

using the convention that we apply the function $g(\cdot)$ to each component of the argument vector. The simplest output function is the identity function. When using a threshold activation function, if the previous output of the neuron is greater than the threshold of the neuron, the output of the neuron will be one, and zero otherwise. Further, to simplify computation, the threshold can be equated to an extra weight. The error being the difference between the actual and the desired output, it only depends on the weights. Hence, to minimise the error by adjusting the weights, we define the error function for the output of each neuron. The error function for the jth neuron satisfies

$$E_j(x,w,d) = (O_j(x,w) - d_j)^2 \ , \ j = 1, .., N_o$$

where d_j is the jth element the desired target vector $\overline{d}$. In that setting, the total error of the network is simply the sum of the errors of all the neurons in the output layer

$$E(x,w,d) = \frac{1}{2} ||\overline{O}(x,w) - \overline{d}(x)||_2^2 = \frac{1}{2} \sum_{j=1}^{N_o} E_j(x,w,d)$$

To minimise the error, we will update the weights of the network so that the expected error in the next iteration is lower using the method of gradient descent. Each weight is updated by using the increment

$$\Delta w_{i,j} = w_{i,j}^{l+1} - w_{i,j}^l = -\eta \frac{\partial E}{\partial w_{i,j}^l}$$

where l is the iteration counter, and $\eta \in (0,1]$ is a learning rate. The size of the adjustment depends on η and on the contribution of the weight to the error of the function. The adjustment will be largest for the weight contributing the most to the error. We repeat the process until the error is minimal.

11.1.6.3 Describing the nonlinear transformation We use the chain rule to compute the gradient of the error function. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function. One way forward is to use the sigmoid, or logistic function, a real function $f_c : \mathbb{R} \to (0,1)$ defined as

$$f_c(x) = \frac{1}{1 + e^{-cx}}$$

where the constant c can be chosen arbitrarily, and its reciprocal $\frac{1}{c}$ is called the temperature parameter. For $c \to \infty$ the sigmoid converges to a step function at the origin. Further, $\lim_{x\to\infty} f_c(x) = 1$, $\lim_{x\to-\infty} f_c(x) = 0$, and $\lim_{x\to 0} f_c(x) = \frac{1}{2}$. The first derivative of the sigmoid with respect to x is

$$\frac{d}{dx} f_c(x) = \frac{ce^{-cx}}{(1+e^{-cx})^2} = cf_c(x)(1 - f_c(x)) \tag{11.41}$$

and the second derivative of the sigmoid with respect to x is

$$\frac{d^2}{dx^2} f_c(x) = cf_c^{'}(x) - 2cf_c(x)f_c^{'}(x)$$

where $f_c^{'}(x) = \frac{d}{dx} f_c(x)$. To get a symmetric output function, we can consider the symmetrical sigmoid $f_s(x)$ defined as

$$f_s(x) = 2f_1(x) - 1 = \frac{1 - e^{-x}}{1 + e^{-x}}$$

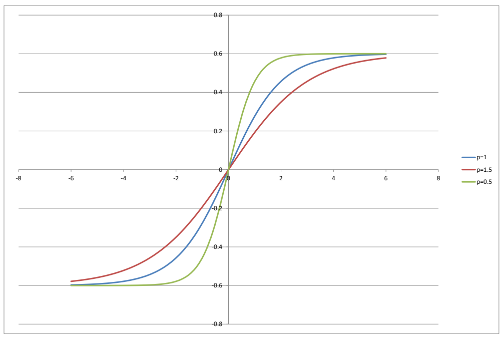
which is the *hyperbolic tangent* for the argument $\frac{x}{2}$, written $tanh(\frac{x}{2})$. An alternative solution is simply to linearly translate the domain of definition of the logistic function in the range $[b_L, b_H]$ where b_L is the lower bound and b_H the upper bound. Hence, the real function $f_{b,c} : \mathbb{R} \to (b_L, b_H)$ is defined as

$$f_{b,c}(x) = b_L + (b_H - b_L) f_c(x)$$

with $\lim_{x \to \infty} f_c(x) = b_H$, $\lim_{x \to -\infty} f_c(x) = b_L$, and $\lim_{x \to 0} f_c(x) = \frac{1}{2}(b_L + b_H)$. The derivative of this function with respect to x is

$$\frac{d}{dx} f_{b,c}(x) = (b_H - b_L) \frac{d}{dx} f_c(x) = (b_H - b_L) c f_c(x)(1 - f_c(x))$$

Hence, in order to get a symmetric output function we can set $b_L = -1.5$ and $b_H = 1.5$. Example of translated logistic function and its derivative are given in Figure (30) for $b_L = -0.6$ and $b_H = 0.6$ with $c = \frac{1}{p}$ for $p = 0.5, 1, 1.5$. Several other output functions can be used and have been proposed in the back-propagation algorithm. However, smoothed output function can lead to local minima in the error function which would not be there if the Heavised function had been used. Further, the slope of the sigmoid function given in Equation (11.41) takes values in the range $[0, \frac{1}{4}c]$ where the maximum is reached at $x = 0$. Thus, the steepest slope of the sigmoid function occurs at the origin, and depending on the size of the constant c, the slope can become greater than 1, leading to gradient explosion. In addition, the derivative of the sigmoid function is asymptotically zero for large values of x, such that the gradient will decay much more when the net value is taking large of small values. The decaying or explosion of the gradient are both called the vanishing gradient problem.



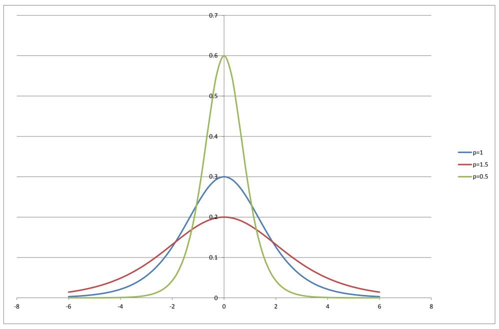


Figure 30: Translated logistic function and its derivative for $b_L = -0.6$ and $b_H = 0.6$.

11.1.6.4 A simple example To explain the method, we first consider a two-layers ANN model, where the most common output function used with a threshold is the sigmoidal function

$$O_j(x, w) = \frac{1}{1 + e^{-A_j(x,w)}}$$

with the limits $\lim_{A_j \to \infty} = 1$, $\lim_{A_j \to -\infty} = 0$, and $\lim_{A_j \to 0} = \frac{1}{2}$ allowing for a smooth transition between the low and high output of the neuron. We compute the gradient of the total error with respect to the weight $w_{i,j}$, noted

$\nabla E(w_{i,j})$, where it is understood that the subscripts i and j are fixed integers within $i = 1, ..., N_i$ and $j = 1, .., N_o$. From the linearity of the total error function, the gradient is given by

$$\frac{\partial E}{\partial w_{i,j}} = \frac{1}{2} \sum_{j=1}^{N_o} \frac{\partial E_j}{\partial w_{i,j}} \, , i = 1, ..., N_i$$

where the subscripts i and j are fixed integers in the partial derivatives. We first differentiate the total error with respect to the output function

$$\frac{\partial E}{\partial O_j} = (O_j - d_j)$$

and then differentiate the output function with respect to the weights

$$\frac{\partial O_j}{\partial w_{i,j}} = \frac{\partial O_j}{\partial A_j} \frac{\partial A_j}{\partial w_{i,j}} = O_j(1 - O_j)x_i$$

since $\frac{\partial A_j}{\partial w_{i,j}} = x_i$. Putting terms together, the adjustment to each weight becomes

$$\Delta w_{i,j} = -\eta(O_j - d_j)O_j(1 - O_j)x_i = \eta(d_j - O_j)O_j(1 - O_j)x_i$$

We can use the above equation to train an ANN with two layers. Given a training set with p input-output pairs, the error function can be computed by creating p similar networks and adding the outputs of all of them to obtain the total error of the set.

11.1.7 Multi-layer back propagation

In the case of a multilayer network, again we consider a single input-output pair and assume a set of artificial neurons $\{f_{i,j,k}\}_{k=0}^K$ where the multilayer subscript $k = 0$ corresponds to the set of inputs $\{x_i\}_{i=1}^{N_i}$, and the remaining subscript k corresponds to the set of outputs $\{y_i\}_{i=1}^{N_{o,k}}$, where $N_{o,k}$ is the number of output in the k-th layer. In that setting, we define the output function of the jth node for the k-th layer as

$$O_{j,k}(x, w) = g(\tilde{A}_{j,k-1}(x, w)) \, , j = 1, .., N_{o,k}$$

where $\tilde{A}_{j,k-1}(x, w) = A_{j,k-1}(x, w) + b_{j,k}$ with the bias $b_{j,k}$, and we let the corresponding activation function satisfies

$$A_{j,k-1}(x, w) = \sum_{i=1}^{N_{o,k-1}} O_{i,k-1}(x, w)w_{i,j}^{k-1} \text{ and } O_{j,0}(x, w) = A_{j,-1}(x, w) = \sum_{i=1}^{N_i} x_i$$

where $w_{i,j}^{k-1}$ is the weight going from the ith node in layer $k-1$ to the jth node in layer k. Note, since $\frac{d\tilde{A}_{j,k-1}(x,w)}{dA_{j,k-1}(x,w)} = 1$, we get $\frac{\partial O_{j,k}(x,w)}{\partial A_{j,k-1}(x,w)} = \frac{\partial O_{j,k}(x,w)}{\partial \tilde{A}_{j,k-1}(x,w)}$ where $\frac{\partial O_{j,k}(x,w)}{\partial \tilde{A}_{j,k-1}(x,w)} = \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x, w))$.

When running backpropagation, the error signal travels from the output layer to the input layer. Given a hidden layer with subscript k, and focusing on neuron with index h, we define the pre-error signal and error signal as

$$\begin{aligned} e_{h,k} &= \frac{\partial}{\partial O_{h,k}} E \\ \delta_{h,k} &= \frac{\partial}{\partial \tilde{A}_{h,k-1}(x, w)} E = \frac{\partial}{\partial net_{h,k}} E \end{aligned}$$

Using these results, we are going to summarise the feedforward neural network by considering the multilayer perceptron (MLP).

11.1.7.1 Forward pass We consider an MLP consisting of K hidden layers with N_i input units, $N_{o,k}$ output units in the k-th hidden layer and $N_{o,K}$ output units in the output layer. In each unit of the hidden layer, or in the output layer, we first calculate the weighted sum of the incoming values, which is referred to as the *net value* of the input unit. The *activation function*, denoted by f, is then applied to the net value. The value of the activation function, y, is the output of the unit. The activation function is required to be bounded, differentiable and monotonous. The two most common functions used in machine learning are the *hyperbolic tangent* and the *logistic sigmoid* functions, which are both nonlinear. This important feature makes it possible for the network to model nonlinear equations. As the number of hidden layers increases, the network can approximate more complex nonlinear functions. Methods using a network with a large number of hidden layers are referred to as *deep learning* network. Since the input domain of the activation function is infinite, while the output domain is finite, the activation functions are also termed as *squashing functions*. Letting h be the index of the first hidden unit, the net value and output function satisfy

$$\begin{aligned} net_{h,1} &= \sum_{i=1}^{N_i} w_{ih} x_i \\ y_h &= f(net_{h,1}) \end{aligned} \tag{11.42}$$

Considering two adjacent layers, denoted by $k-1$ and k, with index h and h', respectively, the summation and the activation process are similar to that of Equation (11.42), given by

$$\begin{aligned} net_{h',k} &= \sum_{h=1}^{N_{o,k-1}} w_{hh'} y_h \\ y_{h'} &= f(net_{h',k}) \end{aligned} \tag{11.43}$$

The net value and activation function of the output layer are calculated in the same way as those in the hidden layer. We let s be the index ranging over the output layer, and $N_{o,K-1}$ is the number of neurons in the $(K-1)$-th hidden layer closest to the output layer. We get

$$\begin{aligned} net_{s,K} &= \sum_{h=1}^{N_{o,K-1}} w_{hs} y_h \\ y_s &= f(net_{s,K}) \end{aligned} \tag{11.44}$$

11.1.7.2 Backward pass In general, after a forward pass the network output is not the target input, and we need to measure the distance between the actual output and the target output to serve as a measurement of the network performance. This distance is called the *error function*. Given the input-target pair (x, z), and the network output y, then the error function is given by

$$E(x,z) = \frac{1}{2} \sum_{s:output} (y_s - z_s)^2, \tag{11.45}$$

where $y = (y_1, \cdots, y_s, \cdots, y_K)$, $z = (z_1, \cdots, z_s, \cdots, z_K)$, and $N_{o,K}$ is the number of units in the output layer. As long as the error function is larger than a given tolerence level, we need to modify the weights of the network to further decrease the measure E. The algorithm designed for stepwise updating the weights that minimise the error function is called the *training algorithm*, or *learning algorithm* of the network. It is natural to relate the weights updates to the *gradient*, which is a vector of derivatives of the error function with respect to all the weights. It costs the network certain number of steps to reach a tolerably small error. Putting all the weights in a vector, the weight vector at step n is denoted by w^n, which is then updated by an amount Δw^n. We denote the gradient as $\nabla E = \frac{\partial E}{\partial w^n}$ and apply the *gradient descent* repeatedly using the chain rule. By working backward, from computing the derivative with respect to the output layer to computing the derivatives with respect to all the internal weights, the error is back propagated in the network. This procedure is called the standard *back propagation*, discovered independently by different researchers (see Werbos [1974], Parker [1985], Rumelhart et al. [1986b]). In standard back propagation, the weight update of the n-th step is

$$\Delta w^n = -\eta \frac{\partial E}{\partial w^n}, \tag{11.46}$$

where η is the *learning rate*, a real number between 0 and 1. Following the previous notations, the detailed procedure to calculate the gradient $\frac{\partial E}{\partial w_{ij}}$ is as follows:

1. Calculate the derivatives of the error function with respect to the output units:

$$\frac{\partial E}{\partial y_s} = y_s - z_s \tag{11.47}$$

$$\frac{\partial E}{\partial net_{s,K}} = \frac{\partial E}{\partial y_s}\frac{\partial y_s}{\partial net_{s,K}} \tag{11.48}$$

2. Calculate the derivatives of the error function with respect to the hidden units. Work backward through hidden layers, using the chain rule. To do so, we introduce the *error signal*:

$$\delta_j := \frac{\partial E}{\partial net_j}, \tag{11.49}$$

where j is the index of an arbitrary unit in the network.

(a) Calculate the error signals of the last hidden layer:
Let j be the index of a unit in the last hidden layer which is the closest to the output layer. Since the error E depends on hidden unit j only through its connection to the output units, we have

$$\begin{aligned}\delta_j &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_{j,K-1}} \\ &= \frac{\partial y_j}{\partial net_{j,K-1}} \sum_{s=1}^{N_{o,K}} \frac{\partial E}{\partial net_{s,K}}\frac{\partial net_{s,K}}{\partial y_j}\end{aligned} \tag{11.50}$$

From Equation (11.43) and (11.44), and given the definition of the error signal in Equation (11.49), we have,

$$\delta_j = f^{'}(net_{j,K-1}) \sum_{s=1}^{N_{o,K}} \delta_s w_{js} \tag{11.51}$$

(b) Calculate the error signals for hidden layers before the last hidden layer:
The error signals for the hidden units in hidden layer k ($k < K$) are based on the calculation of the error signal of the $(k+1)$-th layer due to the chain rule. That is to say, except for the units of the last hidden layer, the error signal of each unit in other hidden unit can be computed recursively as

$$\delta_i = f^{'}(net_{i,k}) \sum_{j=1}^{N_{o,k+1}} \delta_j w_{ij} \tag{11.52}$$

where i and j are indicators of unit in the hidden layers k and $k+1$, respectively, before the last hidden layer.

Having computed all the error signals for all the hidden units as well as the one for the output units, the derivatives of error function with respect to the weights w_{ij} can be written as:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_{j,k+1}} \frac{\partial net_{j,k+1}}{\partial w_{ij}} = \delta_j y_i \tag{11.53}$$

Then, the change of weight from unit i to unit j is given by

$$\Delta w_{ij} = -\alpha \delta_j y_i \tag{11.54}$$

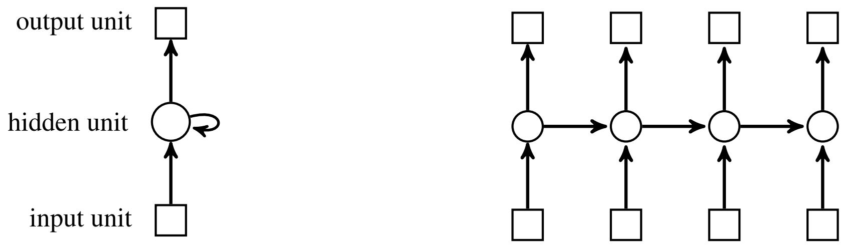
Thus, introducing the concept of error signal makes the back propagation process more comprehensible, as only the error signal propagates backward through the network. The changing amount of the weight is proportional to the product of the error signal of its destination unit and the output of its source unit. To summarise, the forward pass calculates the network output, while the backward pass updates the weights to minimise the error. The method used in backward pass is called back propagation, and a forward pass together with a backward pass is regarded as one *loop*. The training process continue until some *stopping criteria* is met, such as the error function is small enough, or, it stops to decrease after a certain number of loops.

11.2 Presenting recurrent neural networks

While the feedforward neural networks discussed in Section (11.1.7) have no cycles for connections, a *Recurrent Neural Network* (RNN) has feedback connection, meanning that the nodes of the network have cyclical connections between them (see Figure 31a). The existence of cycles allows RNNs to develop a self-sustained temporal activation dynamics along its recurrent connection pathways, even in the absence of input, making them a dynamical system. There are two main classes of RNNs, the first one being characterised by an energy-minimising stochastic dynamics and symmetric connections (see Taylor et al. [2007]), and the second featuring a deterministic update dynamics and directed connections. We are going to concentrate on the latter. As shown in Figure 31b, RNNs can be visualised by unfolding the RNNs along the whole input sequence. That is, in the case of time series, the RNN are unfolded through time. The unfolded graph has no cycles, which is the same as FNNs, so that the forward pass and backward pass of MLP can be applied. Recall that the MLP is an approximator for nonlinear functions, and since RNN can be unfolded to a deep feedforward network, it has the advantage of the MLP, making it a better approximator. In fact, RNNs have proved to be an attractive form for modelling non-linearity due to their ability to approximate any dynamical system with arbitrary precision (see Siegelmann et al. [1991]). Further, Funahashi et al. [1993] showed that under mild and general assumptions, RNNs are universal approximatiors of dynamical systems. Indeed, as an equivalent theory of universal approximation for MLPs, it is said that a single hidden layer RNN with sufficient hidden units can approximate any sequence to arbitrary accuracy (see Hammer [2000]). For simplicity of exposition, we focus on a simple RNN with one self-connected hidden layer, as shown in Figure 31a. As trivial the difference in topology between FNNs and RNNs may seems, the advantage of the RNNs over the FNNs is profound. The latter can only map a limited number of inputs to a limited number of outputs, while a simple RNN can map the entire time series to some outputs. This is significant for time series prediction, as the long history can be fed to the network, and the existence of the recursive connection allows the network to memorise information of previous time steps, which is useful considering that financial time series are mostly serially correlated.

11.2.1 The algorithm

11.2.1.1 Forward pass We now consider a sequence of length T with N_i inputs, one hidden layer ($K = 2$) with $N_{o,K-1}$ hidden units and N_K output units. The i-th external input at time t is denoted by $x_i(t)$, and we let $net_{h,K-1}(t)$ and $y_h(t)$ be the *net value* (summation of the value received) and the output of unit h, respectively, in the hidden layer at time t. After unfolding the network, the forward pass of the RNN is similar to that of a MLP. The only difference being that the unit in the hidden layer receives values from both the current external inputs and the previous output of all hidden units. Mathematically, the net value of a hidden unit is given by



(a) A simple RNN (b) RNN unfolded

Figure 31: Recurrent Neural Network

$$net_{h,K-1}(t) = \sum_{i=1}^{N_i} w_{ih} x_i(t) + \sum_{h'=1}^{N_{o,K-1}} w_{h'h} y_{h'}(t-1) \tag{11.55}$$

where $h^{'}$ is a unit on the previous hidden layer. Then the output value of the hidden unit is calculated by applying the sigmoid activation f to the net value

$$y_h(t) = f(net_{h,K-1}(t)) \tag{11.56}$$

such that the output of hidden units at each time step can be computed recursively, starting from the first time step $t = 1$. Initial value of $y_h(0), \forall h$ is commonly set to be zero, meaning that the network has not received information before the beginning of forward pass. Still, some researchers tend to have nonzero intial value by which they found that the network is more stable (see Zimmermann [2006]). The net value of the output unit, $net_{s,K}$, depends only on the output value of the hidden layer so that the output units are synchronised with the hidden units

$$net_{s,K}(t) = \sum_{h=1}^{N_{o,K-1}} w_{hs} y_h(t)$$

The error function can be defined as

$$E = \sum_{t=1}^{T} \sum_{s=1}^{N_K} \frac{1}{2}(y_s(t) - z_s(t))^2 \tag{11.57}$$

where $z_s(t)$ is the target value of the output unit s at time t. However, this is not the unique form of error function, as it should be based on the training algorithms of the RNN, which we will discuss in the next section.

11.2.1.2 Backward pass We explained in Section (11.1.7) how to calculate the derivatives of the error function with respect to the weights in an MLP. In the case of RNNs, there are mainly two algorithms to calculate the weight derivatives: the Real Time Recurrent Learning (RTRL) (see Robinson et al. [1987], Williams et al. [1989]) and the Backpropagation Through Time (BPTT) (see Werbos [1990], Williams et al [1992]). In ordrer to describe the learning algorithm of the RNN and detail its associated problems, we focus on the BPTT, as it is simple conceptually and more efficient in computation time (see Graves [2005]). Figure 32 illustrates the scheme for updating the weights. Note that the same weights are used in every time step. The error function is defined in Equation (11.57), which include all the time steps.

For MLPs, the error signals are computed in Equations (11.48), (11.50) and (11.52) according to the type of unit. However, what is common for the error signals of the hidden units is that they depend on the sum of error signals that flow back to them. Since unfolded RNNs are equivalent to MLPs, such rule can also be applied. As illustrated in

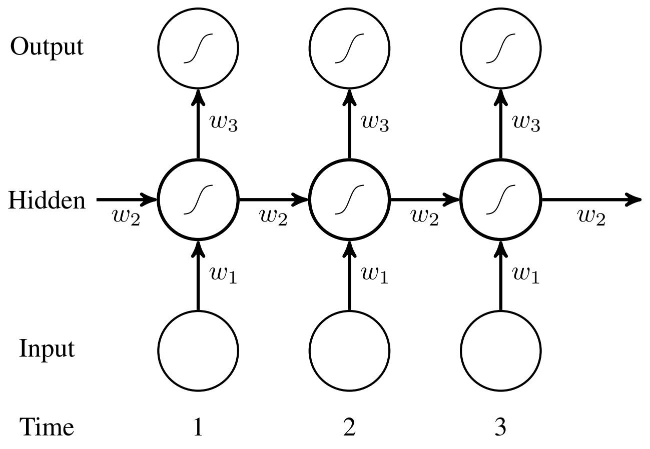


Figure 32: Weights in BPTT

Figure 32, we consider the hidden unit h_2 at time $t = 2$, which has two outgoing connections. We can see that the error signal of hidden unit h_3 at time $t = 3$, and that of the output unit o_2 at time $t = 2$ would flow back to unit h_2 during back propagation through time. The red dashed arrows in Figure 33 illustrates the flow of error signals backward to a particular hidden unit. Hence, the error signal at unit h_2 can be computed as

$$\delta_{h_2} = \delta_{h_3} + \delta_{o_2}$$

Generally, the error function depends on the output of hidden layer not only through the current output layer, but also through the hidden layer next time. Thus, we have

$$\delta_h(t) = f^{'}(net_{h,K-1}(t))(\sum_{s=1}^{N_K} \delta_s(t) w_{hs} + \sum_{h'=1}^{N_o} \delta_{h'}(t+1) w_{hh'}) \tag{11.58}$$

where $\delta_j(T+1) = 0$, $\forall j$. Thus, starting the backward pass at the last time T, we apply Equation (11.58) recursively. Having computed the error signals, we can now calculate the derivative of the error function with respect to the weight via chain rule, getting

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial net_j(t)} \frac{\partial net_j(t)}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j(t) y_i(t) \tag{11.59}$$

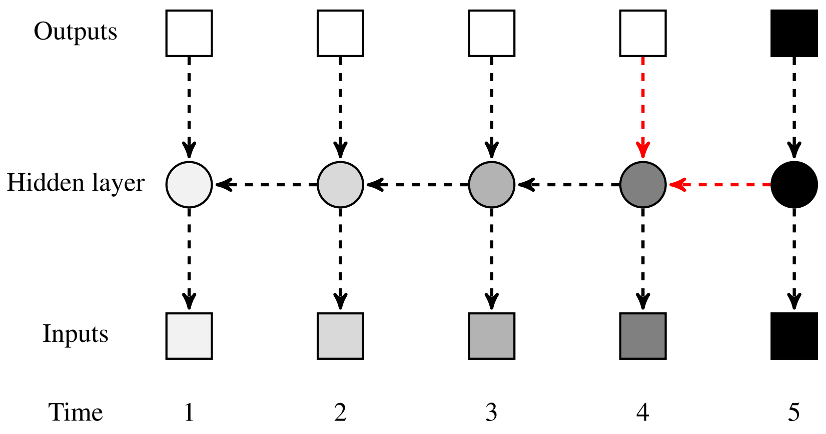


Figure 33: The flow of error signals and vanishing gradient problem in back propagation. The darkness of the shading of the nodes in the unfolded recurrent network indicates the degree of influence (sensitivity) of error signal of output unit at time $t = 5$. The darker the shade represents higher sensitivity. The sensitivity decays as the error signal is back propagated through time.

11.2.2 The long short-term memory

The impact of RNNs in nonlinear modelling has been limited because they are difficult to train by gradient-descent methods. The gradual change of network parameters during the learning process leads the network to bifurcations, where the gradient information degenerates and may become ill-defined (see Doya [1992]). Further, many update cycles may be necessary to obtain convergence of a few parameters, leading to long training times. In addition, when dealing with long-range memory, the gradient information may dissolves exponentially over time. One remedy is to use the Long Short-Term Memory networks (LSTM), which we are now going to describe.

11.2.2.1 The vanishing gradient problem Even though recurrent neural networks (RNNs) are capable of processing serially correlated sequences, the length of sequence that a standard recurrent neural network can access is actually very limited. It results from the fact that the gradients would either decay or blow up when cycling around the recursive connections for too many times, which is usually referred to as *vanishing gradient problem* (see Bengio et al. [1994]). We can see in Equation (11.59) that the derivative of the error with respect to a weight depends on the error signal of the weight's destination unit, j, which itself depends on the derivative of the activation function. Recall, the derivative of the sigmoid function can explode or vanish depending on the slope of the function. As the number of time steps get larger and larger, so does the number of layers in the unfolded RNN. Assuming $c = 1$ in the sigmoid function, the error signal of a hidden unit at time $t = T$ would be propagated back through $(T - 1)$ layers, until a hidden layer at time $t = 1$. That is, it would be multiplied by $(T - 1)$ sigmoid activations' derivatives, all in the range $[0, \frac{1}{4}]$. For T not too large, the vanishing problem does not have much impacts on the network, but for larger T the gradient would decay exponentially. Similarly, for large enough c, the derivative of the sigmoid function near the origin ($x = 0$) would be larger than 1, leading to gradient explosion. This property of sigmoid function is illustrated in Figure 34. For these reasons, training RNNs with standard gradient descent algorithm is only feasible for small time steps. For longer time dependencies, the gradient vanishes as the error signal is propagated back through time, so that the network weights are never adjusted correctly when taking the events far back in the past into account (see Hochreiter et al. [2001]).

11.2.2.2 The constant error carousel One consequence of the vanishing, or exploding, gradient problem described above is that the traditional RNN will fail when involved with large time series. Since the vanishing gradient problem stems from the fact that the tangent line of the sigmoid function is either flat or steep, it is reasonable for some

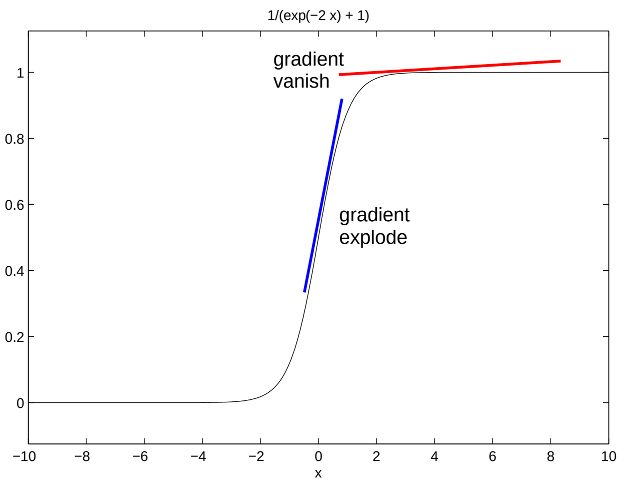


Figure 34: Plot of $\frac{1}{1+e^{-2x}}$ and its tangent lines. The black curve is the sigmoid function ranged $[0,1]$ with $c=2$. The red line is the flat tangent line.

units to have constant gradient equating to 1. This kind of unit has been presented as the *Constant Error Carousel* (CEC) (see Hochreiter et al. [1997]). To describe the advantageous properties of CEC, we consider a single unit, j, with a single connection to itself. According to the rule of calculation of error signals, the jth error signal satisfies

$$\delta_j(t) = f_j^{'}(net_j(t))\delta_j(t+1)w_{jj}$$

where $f_j(\cdot)$ is the activation function of unit j. A constant error flow implies that $\delta_j(t) = \delta_j(t+1)$, leading to

$$f_j^{'}(net_j(t))w_{jj} = 1$$

This is an ordinary differential equation (ODE)

$$\frac{\partial f_j(net_j(t))}{\partial net_j(t)} = \frac{1}{w_{jj}}$$

which we can integrate, getting

$$f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$$

for arbitrary $net_j(t)$. Thus, the activation function f_j for the jth unit should be **linear**

$$y_j(t+1) = f_j(net_j(t+1))$$

Since unit j has only one connection to itself, we get

$$net_j(t+1) = w_{jj}f_j(t)$$

and from the equation of $f_j(net_j(t))$, we get

$$f_j(w_{jj}y_j(t)) = \frac{w_{jj}y_j(t)}{w_{jj}} = y_j(t)$$

We can therefore set the activation function to be the identity function $f_j(x) = x, \forall x$, obtaining $w_{jj} = 1$. The unit j above can be extended to CEC by adding some extra features. A multiplicative *input gate unit* and a multiplicative *output gate unit* are introduced to control the input and output of the CEC, respectively. The input gate unit can prevent the memory of unit j from being overwritten by irrelevant inputs, while the output gate unit avoid perturbation of other units by controlling the output of unit j. The CEC ensures that the error signal arriving at the memory cell would not be scaled up or down during back propagation, and thus can avoid exponential gradient.

11.2.2.3 Network architecture A unit which include the input gate unit, the output gate unit and the CEC is called a *memory cell* (see Figure 35). We are now going to describe the *Long Short-Term Memory* (see Hochreiter et al. [1997]) which is a recurrent neural network consisting of set of *memory blocks* where each block contains one or more memory cells. In fact, the LSTM network is the same as that of a standard recurrent neural network, except that the conventional hidden units are replaced by a memory block with cells and gates encapsulated in. Surely, the hidden layer of the LSTM network can be a mixture of conventional hidden units and memory blocks, but the former is not necessary. The v-th memory cell of the j-th memory block is denoted by c_j^v, the net value of the cell, the input gate and the output gate are $net_{c_j^v}$, net_{in_j} and net_{out_j}, respectively, the output value of the cell, the input gate and the output gate are $y^{c_j^v}$, y^{in_j} and y^{out_j}, respectively. Note that memory cells in the same memory block are controlled by the same input gate and output gate (see Figure 35).

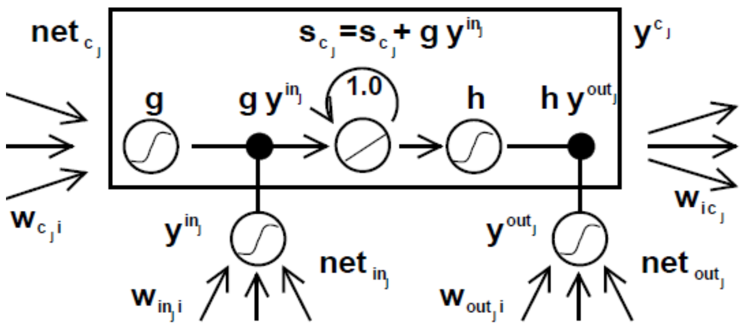


Figure 35: Architecture of a memory block with one memory cell. The CEC is a central linear unit with a self-connected weight 1.0. gate units use inputs from other units to decide whether to access or discard certain information.

As with a standard recurrent network, we have

$$y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

$$y^{out_j} = f_{out_j}(net_{out_j}(t))$$

where f_{in_j} and f_{out_j} are the activation function of the input gate and output gate of memory block i, and

$$\begin{aligned}
net_{in_j}(t) &= \sum_u w_{in_j u} y^u(t-1) + \sum_{n=1}^{N_i} w_{in_j n} x_n(t) \\
net_{out_j}(t) &= \sum_u w_{out_j u} y^u(t-1) + \sum_{n=1}^{N_i} w_{out_j n} x_n(t) \\
net_{c_j^v}(t) &= \sum_u w_{c_j^v u} y^u(t-1) + \sum_{n=1}^{N_i} w_{c_j^v n} x_n(t)
\end{aligned}$$

where the superscript u stands for memory blocks and other traditional hidden unit. The output of the memory cell is different from that of a conventional hidden layer. An additional variable $s_{c_j^v}$, called *internal state*, should be considered

$$\begin{aligned} s_{c_j^v}(t) &= s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t)),\ t > 0 \\ s_{c_j^v}(0) &= 0, \end{aligned} \tag{11.60}$$

Actually, the first line of Equation (11.60) can be written as

$$s_{c_j^v}(t) = 1.0 \times s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t))$$

where 1.0 corresponds to the self-recursive connection with weight 1.0 in Figure 35, which is essentially w_{jj} obtained by the introduction of a simple self-connected unit j without the vanishing gradient problem. So the output of the cell can be computed as

$$y^{c_j^v}(t) = y^{out_j}(t) h(s_{c_j^v}(t))$$

where g and h are the squashing function of the net value of memory cell and the internal sate $s_{c_j^v}$.

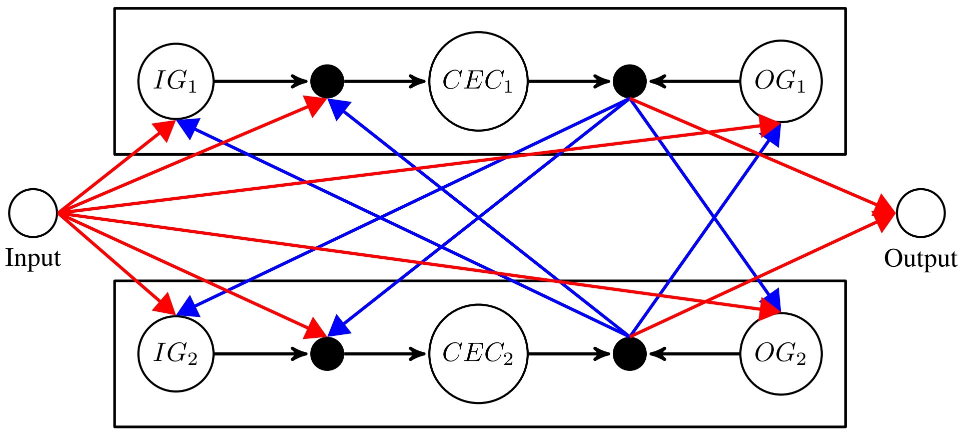


Figure 36: Connections in LSTM network. The example network consists of one input unit, a hidden layer of two single-cell LSTM memory blocks and one output units. Suppose the input is at time t. The the sources of the blue connections are from the previous one time step $t-1$. while the red connections convey information of current time step t. IG:input gate, OG: output gate, CEC:constant error carrousel

It is important that the gate units and the internal states are only visible within the cell, and that only the output of the cell can connect to other blocks (including gates and cell) in the hidden layer. Therefore, the net value is the summation of current input values and the output of memory cells $y^{c_i^j}$ from previous time step. Further, Graves [2005] made some modifications to the original model of Schmidhuber and Hochreiter by modifying the rule connecting the units. Only the outputs memory cells are allowed to connect to other blocks, and the CECs and outputs of gate units are only visible within the memory blocks they are located in. As an example, Figure 36 shows the types of connections between two blocks. The gates units enable the memory cells in LSTM to store as well as to access information for a long period of time, and thus alleviate the vanishing gradient problem. For instance, if the activation function of the input gate is zero (the input gate is closed), then the memory of the cell will not be overwritten by the current input, thereby making it accessible to the network at a later time, as long as the output gate is open. However, the original LSTM has a weakness. When the sequence of time is long and have not been segmented to reset the network at certain time, Equation (11.60) implies that the internal state will grow infinitely and the network will collapse. To remedy this problem, Gers et al. [2000] proposed to add a *forget gate* to the self-connection with weight 1.0 in the original LSTM cell (see Figure 35). Then the revised equation for the internal state is extended from Equation (11.60) as follow

$$\begin{aligned} s_{c_j^v}(t) &= y^{\phi_j}(t)s_{c_j^v}(t-1) + y^{in_j}(t)g(net_{c_j^v}(t)),\ t > 0 \\ s_{c_j^v}(0) &= 0. \end{aligned} \tag{11.61}$$

where $y^{\phi_i}(t)$ is the value of the forget gate of the i-th memory block, which is squashed between zero and one. The forget gate is analogous to the reset operation of the memory cell. So far, the gate units are not entitled to control the internal state, as the gates have merely two sources of input: from the current input units and the previous output of all memory cells. In other words, the gates unit can only observe the cells' output. Once the output gate is closed, there is no way for gates to access the CEC they are supposed to control, which results in insufficient information that do harm to the performance of the network. In order to avoid the lack of information of internal states, another augmentation of the LSTM with forget gate was also introduced by Gers et al. [2002]. Weighted *peephole weights* are added in order to connect from the CEC to all gates of the same memory block. This effective remedy ensures that the all gates can "inspect" the internal state currently, even if the output gate is closed (see the dashed arrow in Figure 37).

We can summarise the types of connections of the modern LSTM as follow

- Outgoing connections
  - outside memory block
    Cells' output can feed to any types of units in any blocks in the hidden layer and the output units and they are the only output that can be connected to other blocks and output units.
  - inside memory block
    The gate unit can only pass value forward to cells of the block where the gate belong to. The internal state can connect to all types of gate within the same block.
- Incoming connections
  The memory cell's input can receive outputs from the hidden layer and the input layer;

  - The gate units can receive from the outputs from the hidden layer, input layer and the value of internal states.
  - The output unit can only receive value from the outputs of conventional hidden units and memory cells.

By including the forget gate and peephole weights, the traditional LSTM evolved to the modern LSTM and the update scheme in Hochreiter et al. [1997] needs refinement.

11.2.2.4 The learning algorithm Details and proofs of the learning algorithm of the LSTM can be found in Hochreiter et al. [1997]. We display the learning algorithm in Figure 38. We emphasise here that the update scheme involves truncated derivative which are used to enhance the efficiency of the network. Note that the *training error* is the average of the $|z_t - y_t|,\ t = 1, \cdots, T$, in Figure 38. The weight update scheme is illustrated in Figure 39. The updated algorithm for the backward pass is a combination of truncated BPTT and customised RTRL. The former refers to BPTT using truncated derivatives. To be specific, standard BPTT is applied to output units, truncated BPTT is implemented by output gates, while weights related to input gates, forget gates and memory cells use a truncated RTRL. By comparing Figure 32 with Figure 39 we can get an idea of the differences between the full BPTT and the truncated one used by LSTM. In the BPTT algorithm for standard RNN, Equation (11.58) shows that the information of all time steps has to be saved for the update of weight and the weights remain unchanged during the entire feedforward process. Whereas the LSTM network updates its weights at each time step during the forward pass, so that there is no need to store all values. To summarise, the BPTT with full gradient change the weight for one time by passing

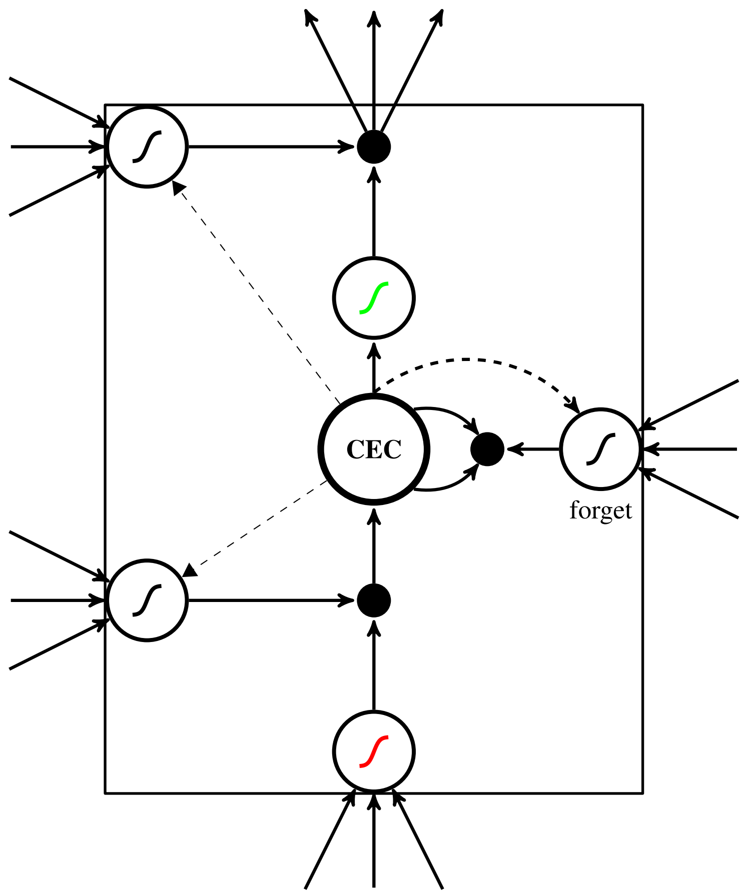


Figure 37: A modern LSTM memory block with one cell. The input, output and forget gates are the analogous of write read and reset operations for the cells. The three gates are nonlinear summation units that collect outputs from outside and inside the block. The small black dots are multiplicative units by which the activation of the cell is controlled. The gate units have sigmoid function f with range $[0, 1]$, so that 0 corresponds to the gate being closed and 1 to the gate being open. The unit with red S-shaped curve is the activation function that squashes the net value of cells' input, and the unit with green S-shaped curve is the activation function of cells' output. Usually, the functions g and h are task-specific and are encouraged to have ranges different from $[0, 1]$. The input and output gates multiply the input and output of the cell, and the forget gate multiply the previous internal state. Peephole weights from internal states to gates are shown by dashed arrow. Note, except for the self-recursive connection of the CEC, connections within the blocks have fixed weights of 1.0. The only outputs from the memory block to the rest of the network is from the upper multiplicative unit.

forward from time $t = 1$ to $t = T$ and passing backward from $t = T$ to $t = 1$ while the truncated BPTT passes forward from $t = 1$ to $t = T$ and the weights are updated for T times.

Computational Complexity of LSTM. The LSTM algorithm is efficient and its computational complexity is of order $O(W)$ per time step, where W is the number of weights (see Hochreiter et al. [1997]). Compared with BPTT, LSTM is more economical in terms of space. In fact, calculating the full gradient of LSTM also has advantage since it is easy to debug and can be checked by numerical approximations (see Graves et al. [2005]). With the CEC, a LSTM block may be considered as a smart network unit, compared to the conventional hidden unit, to store information for

```
for $i = 1$ to $n$ do
    Initialize.  Initialize all weights $\{w_{ij}(1)\}$ randomly.
    while stopping criteria not met do
        for $t = 1$ to $T - 1$ do
            Feedforward from $x_t$ to output $y_t$ using weights $\{w_{ij}(t)\}$.
            Back propagation.  calculate error signals based on $y_t - z_t$.  Take down $|y_t - z_t|$.
            Update weights.  $\{w_{ij}(t)\} \rightarrow \{w_{ij}(t+1)\}$
        end for
        Reset network except for its weights.
    end while
end for
```

Figure 38: Pseudo-code of LSTM. The subscripts i and j standards for any units that have connections. y_t and z_t denote the actual output and target output of the network at time t. $z_t = x_{t+1}$. n is the number of simulations.

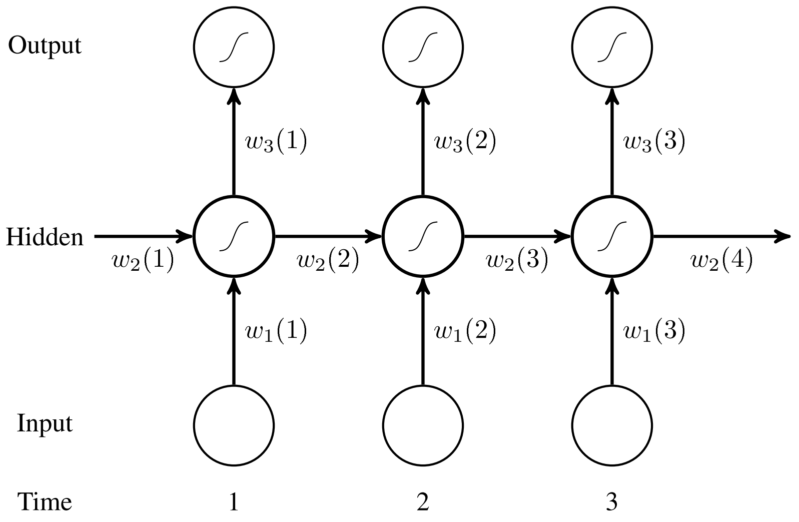


Figure 39: Algorithm of weight update of LSTM. Note that the memory cell is regarded as a normal hidden unit for simplicity. Only one hidden unit is shown.The number in bracket represents the time step.Weights are updated at each time step.

arbitrary length of time. Therefore, the LSTM is well-suited to process and predict time series when there are unknown size of long time lags between important events. Since financial time series has long term memory, predicting volatility by LSTM network may be rather promising. While the traditional RNN with weight update algorithm BPTT (Back Propagation Through Time) (see Williams et al. [1990]) and RTRL (Real-time Recurrent Learning) (see Robinson et al. [1987] and Williams et al. [1992]) and the combinations of the former two (see Schmidhuber [1992]) have been proved to occur learning failure when processing sequences with only 10 time steps (see Bengio et al. [1994], Hochreiter et al. [1997], Gers et al. [2000], Hochreiter et al. HochreiterEtAl01), the LSTM can deal with 1000 time steps and even more, outperforming those traditional RNN algorithms. In fact, except for prediction, LSTM outperforms other RNNs in numerous aspects such as the best performance in speech recognition (see Graves et al. [2011]) and the ICDAR handwritting competition in 2009.

11.2.3 Reservoir computing

Reservoir Computing (RC) is a special RNN originating from Echo State Networks (ESNs) (see Jaeger [2001]) and Liquid State Machines (LSMs) (see Maass et al. [2002]), which assumes that supervised adaptation of all interconnection weights are not necessary, and only training a memoryless supervised readout from it is sufficient to obtain good results. Thus, it avoids the shortcomings of the gradient-descent training of RNNs. RC is based on the computational seperation between a dynamic reservoir and a recurence-free readout. The former is an RNN as a nonlinear temporal expansion function, randomly created and unchanged during training. The latter produces the desired output from the expansion. This is a very simple approach avoiding the problem of bifurcations encountered during the training of RNNs, and the use of complex memory cells when learning long-term dependencies, as in the LSTM described in Section (11.2.2). Even though several studies aimed at understanding RC and the factors affecting its performances, none have been completely satisfactory due to the complexity of the reservoir, often resulting in contradictory conclusions. Introduction to the concepts and methodologies can be found in the literature (see Lukosevicius et al. [2009], [2012]). Goudarzi et al. [2014] compared the performance of three methods, the delay line, the NARX network, and the ESN and concluded that the first two have higher memorisation capability, but fall short of the generalisation power of the latter. Thus, we are going to briefly describe the ESN.

11.2.3.1 Describing the Reservoir methods We let the time-varying input signal be an N_i-th order column vector $U(t) = [u_i(t)]$, the reservoir state is an N_x-th order column vector $X(t) = [x_j(t)]$, and the generated output is an N_o-th order column vector $Y(t) = [y_o(t)]$. A Reservoir Computer is a collection of internal nodes, whose state vector $X(t)$ evolve in discrete time t according to the nonlinear map in Equation (11.31) and given in the form

$$X(t+1) = (1-\alpha)X(t) + \alpha f\big(W_{res}^{\top} \cdot X(t) + W_{in}^{\top} \cdot U(t+1) + W_{fb}^{\top} \cdot Y(t)\big) \tag{11.62}$$

for some activation function $f(\cdot)$, where the leaky integrator α is a constant controlling the speed of the dynamics. The input weight matrix is an $N_i \times N_x$ matrix $W^{in} = [w_{ij}^{in}]$ where w_{ij}^{in} is the weight of the connection from input node i to reservoir node j. The connection weights inside the reservoir are represented by an $N_x \times N_x$ matrix $W_{res} = [w_{jk}^{res}]$ where w_{jk}^{res} is weight from node j to node k in the reservoir. In presence of a bias, the output matrix is an $(N_x+1) \times N_o$ matrix $W_{out} = [w_{ko}^{out}]$, where w_{ko}^{out} is the weight of the connection from the reservoir node k to the output node o. In the case where the output nodes are also connected to the input nodes and to themselves the size of the matrix becomes $(N_x + N_i + N_o) \times N_o$. The feedback matrix is an $N_x \times N_o$ matrix $W_{fb} = [w_{ko}^{fb}]$, where w_{ko}^{fb} is the weight of the connection from the reservoir node k to the output node o. If no output feedback is needed, then W_{fb} is null. The generated output is given by

$$Y(t) = f_{out}(W_{out}^{\top} \cdot X(t)) \tag{11.63}$$

where $f_{out}(\cdot)$ is an output activation function, typically the identity or a sigmoid. The output weights are trained to minimise the squared output error

$$E = ||Y(t) - \hat{Y}(t)||^2$$

given the target output $\hat{Y}(t)$. Once we know the optimum weights of the network, we can use the model to perform a forecast. To do so, we let T be a network state update, where

$$X(t+h) = T(X(t), \overline{U}^h)$$

denote the network state resulting from an iterated application of Equation (11.62) when the input sequence $\overline{U}^h = U(t+1), ..., U(t+h)$ is fed to the network being in state $X(t)$ at time t.

We present some of the major characteristics one has to consider when defining a relevant model (see Lukosevicius et al. [2009] and Lukosevicius [2012]). There are several features one has to consider when designing RC:

1. Connectivity: $\{W_{in}, W_{res}, W_{fb}\}$ is generated randomly. The input, bias, reservoir and feedback scaling (respectively named s_{in}, s_{bias}, s_{res} and s_{fb}) will be considered as micro parameters. For the choice of topology, the authors suggested a permutation matrix with a medium number and different lengths of connected cycles, or a general orthogonal matrix.
   (a) N_x: The principle is that the bigger the reservoir, the better the obtainable performance, provided appropriate regularisation measures are taken against over-fitting. As ESNs are computationally cheap, we can use as big a reservoir as we can afford computationally. N_x will be considered as a macro parameter.
   (b) Sparsity: It enables fast reservoir updates. One rule of thumb is to connect at most each node to 10 other nodes in the reservoir.
   (c) Distribution: Uniform or Normal. W_{in}, W_{res} and W_{fb} have usually the same distribution. In general, the seed of the pseudo-random generator is fixed.
2. Leaking Rate α: The leaking rate can be seen as the time interval between two discrete realisations. In general, we expect its value to be close to 1.
3. Echo State Property: The state of the reservoir $X(t)$ should be uniquely defined by the fading history of the input $U(t)$ [...] The spectral radius should be greater in tasks requiring longer memory of the input. Finding clear characteristics remain a large debate among practitioners and researchers.

From these features, one can see that many meta parameters are involved, so that it is necessary to perform an additional optimisation aiming at minimising the RMSE on the validation sample. It is known as meta-optimisation (see Pedersen et al. [2008a]). The idea is to have an optimisation method act as an overlaying meta-optimiser, trying to find the best performing behavioural parameters for another optimisation method (see Pedersen et al. [2008a]). Note, parameter tuning via meta-optimiser is done in an offline manner, while adaptation of DE parameters is done online during optimisation.
According to Lukosevicius et al. [2009], the investigation shows that the error surfaces in the combined global parameter and W_{out} spaces may have very high curvature and multiple local minima. Thus, gradient descent methods are not always practical. In general, optimising reservoir is quite a hard task as we have many parameters involved. Nevertheless, the computationally cheap property of Reservoir Computing leads to the following idea: *"generate k reservoirs and pick the best."*.

Echo State Networks (ENSs) assume that if a random RNN possesses certain algebriac properties, it suffises to train a linear readout from it to obtain good performances. Even though there are several readout methods, the most popular one is the linear regression where W_{out} solve a system of linear equations. From Remark (11.1), we get

$$W_{out}^\top X_m = \hat{Y}_m$$

where both the matrix $X_m \in \mathbb{R}^{N_x \times T}$ and the matrix $\hat{Y}_m \in \mathbb{R}^{N_o \times T}$ [8], over the training period $t = 1, ..., T$, have a column for every training time step t. The output weights can be estimated with direct pseudo-inverse calculations, or they can be estimated with the ordinary linear regression (Wiener-Hopf solution). Thus, we get

$$W_{out}^\top X_m \cdot X_m^\top = \hat{Y}_m \cdot X_m^\top$$

and the weight matrix becomes

$$W_{out}^\top = \hat{Y}_m \cdot X_m^\top (X_m \cdot X_m^\top)^{-1} = PR^{-1}$$

where $R = X_m \cdot X_m^\top$ is the correlation matrix of the reservoir states, and $P = \hat{Y}_m \cdot X_m^\top$ is the cross-correlation matrix between the states and the desired outputs. Note, $P \in \mathbb{R}^{N_o \times N_x}$ and $R \in \mathbb{R}^{N_x \times N_x}$ do not depend on the training length

[8] These matrices are transposed compared to the conventional notation.

T and can be calculated incrementally. When R is ill-conditioned the method is numerically unstable, but computing the Moore-Penrose pseudo-inverse R^+ instead of R can improve the solution. Alternatively, we can decompose the matrix R into two triangular matrices with Cholesky or the LU decomposition (see Press et al. [1992]) and solve

$$W_{out}^\top R = P$$

by two steps of substitution to get $W_{out}^\top = PR^{-1}$.
Evolutionary search can also be used for training the linear readouts. There exists several bias to reduce the error in the validation set. One can smooth the model with regularisation functions with the Ridge regression

$$W_{out}^\top = P(R + \beta^2 I)^{-1} \tag{11.64}$$

where β controls the smoothing effect, and $I \in \mathbb{R}^{N_x \times N_x}$ is the identity matrix.

The following algorithm sums up the steps involved in the training of the ESN framework.

Algorithm 9 RC offline Training

```
Require: get a linear system solver (we use Cholesky Decomposition).
Require: set $t_0$ as a wash-out size.
Require: set $\beta$ as a regularisation parameter.
1: $Y \leftarrow [y(t_0)\| \dots \|y(T)]$
2: $Y \leftarrow f_{out}^{-1}(Y)$
3: $X \leftarrow [1\|U(t_0)\| \dots \|U(T)\|X(t_0)\| \dots \|X(T)]$
4: $W_{out} \leftarrow Y \cdot X^T \cdot (X \cdot X^T + \beta^2 I)^{-1}$
5: return $W_{out}$
```

Note, we can also add noise $\epsilon(t)$ (sampled from uniform or Gaussian distribution) to the reservoir states

$$X(t+1) = (1-\alpha)X(t) + \alpha f\big(W_{res}^\top \cdot X(t) + W_{in}^\top \cdot U(t+1) + W_{fb}^\top \cdot Y(t)\big) + \epsilon(t)$$

to stabilise solutions in the model (see Jaeger [2007]). We can further adjust global control parameters to make the echo state network dynamically similar to the system we model. For instance, we can use fully connected reservoirs or sparsely connected ones.

In order to produce a reservoir with a rich enough set of dynamics the number of internal connections N_x should be large, the weight matrix W should be sparse, and the weights of the connections should be generated randomly from a uniform distribution symmetric around the zero value. Further, the network should have the echo state property (ESP), which relates asymptotic properties of the excited reservoir dynamics to the driving signal (see Jaeger [2001]). It states that the effect of a previous state $X(t)$ and a previous input $U(t)$ on a future state $X(t+h)$ should vanish gradually as $h \to \infty$, and not persist or be amplified. In reservoirs using the $tanh$ squashing function, and for zero input, the reservoir weight matrix W^{res} must be scaled so that its spectral radius [9] $\rho(W^{res})$ satisfies $\rho(W^{res}) < 1$. For any kind of inputs (including zero) and state vectors, we require $\sigma_{max}(W^{res}) < 1$ where $\sigma_{max}(W^{res})$ is the largest singular value of W^{res}. If the input comes from a stationary source, the ESP holds with probability 1 or 0. Due to the auto-feedback nature of RNNs, the reservoir states $X(t)$ reflect traces of the past input history, which can be seen as a dynamical short-term memory. Assuming a single input ESN, the short-term capacity is given by

$$C = \sum_i r^2(U(t-i), Y_i(t))$$

[9] the largest absolute eigenvalue

where $r^2(\cdot, \cdot)$ is the squared correlation coefficient between the input signal delayed by i and an output signal $Y_i(t)$ trained to memorise $U(t-i)$ on the input signal $U(t)$. For an i.i.d. input, the memory capacity C of an echo state network of size N is bounded by N. Thus, we can not train ESN on tasks requiring unbounded-time memory.

11.2.3.2 Some improvements Separating the reservoir from the readout training allows for two research directions to be pursued independently,

1. the generation of the reservoir, and
2. the output training.

There is no reasons why the reservoir should be randomly generated and alternative methods could be used to obtain optimal reservoir design. However, no single type of reservoir can be optimal for all types of problems (no free lunch principle). Several methods have been proposed for generating the reservoir (see Lukosevicius et al. [2009]), which can be classified as

1. generic methods for generating $RNNs$ with different neuron models, connectivity patterns and dynamics.
2. unsupervised adaptation of the reservoir based on the input data $U(t)$ but not the target value $\widehat{Y}(t)$.
3. supervised learning, adaptation of the reservoir using task-specific information from both $U(t)$ and $\widehat{Y}(t)$.

In order to deal with different time scales simultaneously, one can divide the reservoir into decoupled sub-reservoirs and introduce inhibitory connections among all the sub-reservoirs. However, the inhibitory connections should be heuristically computed from the rest of W and W_{fb} such that they predict the activations of the sub-reservoirs one time step ahead (see Xu et al. [2007]). Alternatively, the Evolvino transfers the idea of ESNs to a LSTM type of RNNs where the LSTM RNN used for its reservoir consists of specific small memory-holding modules. In that model, the weights of the reservoir are trained using evolutionary methods (see Schmidhuber et al. [2007]). Further, ESNs like any other RNNs has only a single layer of neurons (see Figure 31a), making it unsuitable for some types of problems requiering multilayers. A solution is to use Layered ESNs (see Lukosevicius [2007]), where part of the reservoir connections are instantaneous, and the rest takes one time step for the signals to propagate as in normal ESNs. One can also add leaky integrator neurons to ESNs, getting leaky integrator ESNs (Li-ESNs) performing at least as well as the simple ESN (see Lukosevicius et al. [2006]). Since the parameters a and Δt control the speed of the reservoir dynamics, small values result in reservoirs reacting slowly to the input. Note, depending on the speed at which the input $U(t)$ changes, we can vary Δt on-the-fly, getting a warping invariant ESNs (TWIESNs). Since checking the performance of a resulting ESN is relatively inexpensive, evolutionary methods for pre-training the reservoir developed (see Ishii et al. [2004]). Generally, one separate the topology and weight sizes of W_{res} to reduce the search space (see Bush et al. [2005]). Jiang et al. [2008] showed that by only adapting the slopes of the reservoir unit activation functions $f(\cdot)$ with an evolutionary algorithm, and having W_{out} random and fixed, a very good prediction performance of an ESN could be achieved. Note, evolutionary algorithms can also be used to train the readouts. One can increase the expressiveness of the ESN by having k linear readouts trained and an online switching mechanism among them, or by averaging outputs over several instances of ESNs (see Bush et al. [2006]).

11.3 Other

11.3.1 SVM model

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. Introduced by Vapnik and Chervonenkis in 1963, Boser et al. [1992] suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. SVM is a learning system using a high dimensional feature space where points are classified by means of assigning them to one of the two disjoint half spaces, either in the pattern space or in a higher dimensional

feature space. The main objective is to identify maximum margin hyper plane, that is, the margin of separation between positive and negative examples is maximised. SVM finds the maximum margin hyper plane as the final decision boundary.
Assume that $x_i \in \mathbb{R}^d$, $i = 1, ..., N$ forms a set of input vectors with corresponding class labels $y \in \{+1, -1\}$, $i = 1, ..., N$. SVM can map the input vectors $x_i \in \mathbb{R}^d$ into a high dimensional feature space $\Phi(x_i) \in H$.
A kernel function $K(x_i, x_j)$ performs the mapping $\phi(\cdot)$, that is, $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. The value w is also in the transformed space with $w = \sum_{i=1}^N \alpha_i y_i \phi(x_i)$. Dot products with w for classification can again be computed by the kernel trick $w \cdot \phi(x) = \sum_{i=1}^N \alpha_i y_i K(x, x_i)$.
Computing the (soft-margin) SVM classifier amounts to minimising an expression of the form

$$f(w,b) = \Big[\frac{1}{N}\sum_{i=1}^N \max\big(0, 1 - y_i(w \cdot x_i - b)\big)\Big] + \lambda \|w\|^2$$

The primal is a constrained optimisation problem with a differentiable objective function. By solving for the Lagrangian dual of the above problem, one obtains the simplified problem called the dual problem.
We can also apply the kernel trick. Suppose now that we would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the transformed data points $\phi(x_i)$. Moreover, we are given a kernel function defined as above. Then, the classification vector w in the transformed space satisfies $w = \sum_{i=1}^N \alpha_i y_i \phi(x_i)$. The resulting decision boundary is

$$f(x) = \text{sgn}\big(w \cdot \phi(x) + b\big) = \text{sgn}\big(\sum_{i=1}^N y_i \alpha_i \cdot K(x, x_i) + b\big)$$

In order to estimate the values α_i, $i = 1, ..., N$ we solve the following quadratic programming problem

$$\begin{aligned}
&\max_{\alpha_i}\Big(\sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{i=j}^N \alpha_i \alpha_j \cdot y_i y_j \cdot K(x_i, x_i)\Big) \\
&\text{subject to } 0 \leqslant \alpha_i \leqslant c \\
&\sum_{i=1}^N y_i \alpha_i = 0 \text{ , } i = 1, ..., N
\end{aligned}$$

where the parameter c controls the tradeoff between the margin and misclassification error.
The polynomial and radial basis kernel functions are given as follows:

$$\begin{aligned}
&\text{Polynomial (inhomogeneous) function } : K(x_i, x_j) = (x_i \cdot x_j + 1)^d \\
&\text{Gaussian radial basis function } : K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}
\end{aligned}$$

where d is the degree of the polynomial function and $\gamma > 0$ is the constant of the radial basis function. The latter is sometime given by $\gamma = \frac{1}{2}\sigma^2$. Note, if we want the polynomial function to be homogeneous we set $K(x_i, x_j) = (x_i \cdot x_j)^d$.
Thus, the parameters of the SVM model are (d, γ, c). In general, several levels are considered to determine these parameters efficiently.
Recent algorithms for finding the SVM classifier include sub-gradient descent and coordinate descent. Both techniques have proven to offer significant advantages over the traditional approach when dealing with large, sparse datasets?sub-gradient methods are especially efficient when there are many training examples, and coordinate descent when the dimension of the feature space is high.
Sub-gradient descent algorithms for the SVM work directly with the expression $f(w, b)$. Since f is a convex function, traditional gradient descent (or SGD) methods can be adapted, where instead of taking a step in the direction of the functions gradient, a step is taken in the direction of a vector selected from the function's sub-gradient.

11.3.2 Variable weights neural networks

In conventional neural networks, connection weights are fixed. That is, the characteristics of the neural network is invariant (the invariant type neural network). The adaptive ability of neural networks to an unknown environment depends on its generalisation capability. However, there exists a limit of generalisation ability in invariant type neural networks. Yasuda et al. [2006] proposed the neural network with variable connection weights, which changes its connection weights and its characteristic according to the changing environment. Lam et al. [2014] presented the variable weight neural network (VWNN), allowing its weights to be changed in operation according to the characteristic of the network inputs.

11.3.2.1 Description We are now going to briefly describe the VWNN. It consists of two traditional neural networks, namely tuned and tuning neural networks. The former is the one which actually classifies the input data, while the latter provides the weights to the tuned neural network according to the characteristic of the input data. The VWNN works on the principle that the connection weights are function to the external input signal.
As an example, we consider a three-layer feed-forward fully-connected neural network (denoted fixed model) with n_{in} inputs and n_{out} outputs, weights $\omega_{ji}^{(k)}$ and bias terms $b_j^{(k)}$, $k = 1, 2$. The input vector is $x(t) = [x_1(t), ..., x_{n_{in}}(t)]$ and the output vector is $y(t) = [y_1(t), ..., y_{n_{out}}(t)]$ (see Bryson et al. [1969]). We are going to modify that network to obtain its associated VWNN. To do so, we need the tuning neural networks NN_k, $k = 1, 2$, which provide connection weights $\omega_{ji}^{(k)}$, and bias terms $b_j^{(k)}$ to the fixed model. The tuning networks rely on the modified input vector $x^{'}(t)$, which consists of some selected features from $x(t)$. Given a predetermined constant selection matrix $S \in \mathbb{R}^{n^{'}_{n_{in}} \times n_{n_{in}}}$, then

$$x^{'}_{n_{in}}(t) = S \cdot x_{n_{in}}(t)$$

where $n^{'}_{n_{in}} \leqslant n_{n_{in}}$. For example

$$x^{'}_{n_{in}}(t) = \begin{bmatrix} x^{'}_1(t) \\ x^{'}_2(t) \end{bmatrix}, x_{n_{in}}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} \text{ and } S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

such that $x^{'}_{n_{in}}(t)$ selects $x_1(t)$ and $x_3(t)$ as the input of the tuning neural networks NN_k. The latter will produce output weight vector $W(t)$ consisting of all connection weights of the tuned neural network (fixed model). Eventually, the tuned neural network will then use $W(t)$ to process the input $x(t)$ and produce the output $y(t)$. See Figure (40) and Figure (41).

11.3.2.2 Application to finance Clearly, the role of the matrix S is to reduce the dimensionality of the input signal x in view of generating dynamical weights as a function of the latter.
The method was applied to large size neural network. In the case of financial time series, we can let the matrix S be a lowpass filter or other denoising technique, so that the modified input $x^{'}$ of the tuning network becomes the filtered time series. This way, the fixed weights of the tuning network are associated with a smooth signal and the variable weights of the tuned network become a function of the filtered market returns.

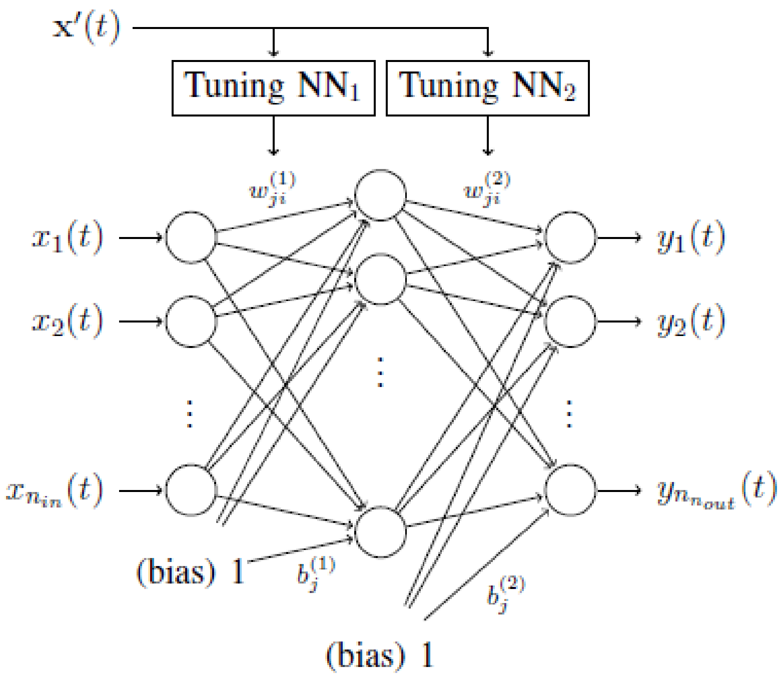


Figure 40: A three-layer variable-weight neural network

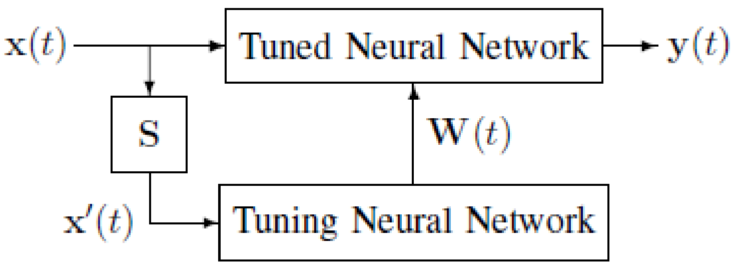


Figure 41: A block diagram of variable-weight neural network

12 Towards dynamically stable reservoirs

12.1 A specific implementation

12.1.1 Overview of the topology

We let $\mathbb{T} := [1, N_T]$ be a discrete time set and define:

- $u \in \mathbb{R}^{N_T \times N_U}$ the input signal at time t and we define $U := [1, N_U]$.

- $X := (x^k)_{k \in N_K} \in \mathbb{R}^{N_K \times N_T \times N_{X_K}}$ a sequence of reservoir neuron activations (often called kernels). $\forall k \in \mathbb{K} := [1, N_K]$, x_k is a factor which can be seen as a fraction of the reservoir. Factors are usually defined with the same matrix design and can be correlated or not (in terms of dimension and jump). By allowing this third dimension, we can represent several high dimensional signal in a same reservoir and in a dependent or independent manner. They all have their own set of Equations (11.62) (11.63) and parameters coming with it. We also define $x = [x^1 \| \dots \| x^K]$.

- $y, \; \hat{y} \in \mathbb{R}^{N_T \times N_Y}$ are respectively the output and forecast signals. And we also define $\mathbb{Y} \equiv \hat{\mathbb{Y}} := [1, N_Y]$.

- $\forall k \in N_K$, $W_{in}^k \in \mathbb{R}^{N_{X_k} \times N_U}$, $W_{res}^k \in \mathbb{R}^{N_{X_k} \times N_{X_k}}$ and $W_{fb}^k \in \mathbb{R}^{N_{X_k} \times N_Y}$ are matrices mapping respectively u to x^k, x^k to x^k and y (or $\hat{y}$) to x^k. These matrices are randomly constructed but supported by a well chosen topology. We present our choice of topology in Appendix (12.1.3). They stand for the part of the topology learned in an unsupervised manner.

- The output matrix is slightly different as we gather all the reservoir factors in a unique vector $s = [1\|u\|x^1\| \dots \|x^{N_K}] \in \mathbb{R}^{N_T \times N_S}$ where $N_S = 1 + N_u + \sum_{k=1}^{K} N_{X_k}$. Then, the output matrix is $W_{out} \in \mathbb{R}^{N_Y \times N_S}$. This matrix is the only part of the topology learned in a supervised manner.

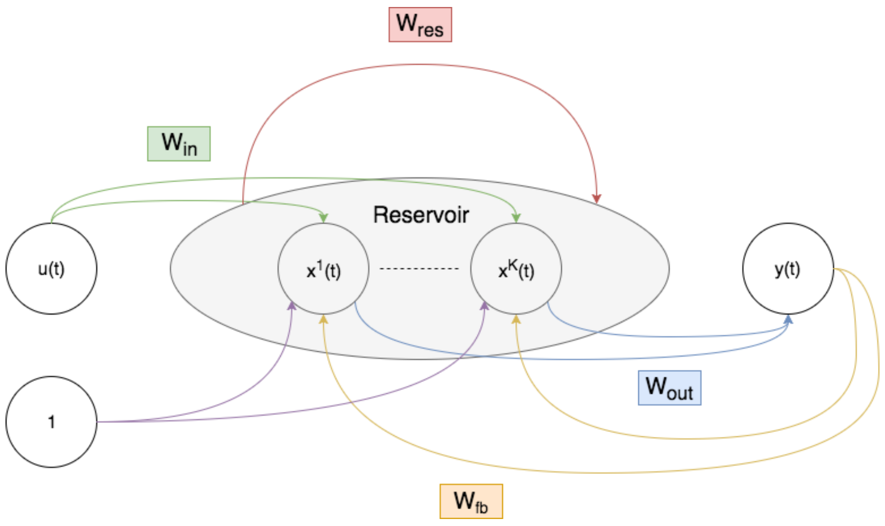


Figure 42: An Echo State Network with $N_x = K$ factors

12.1.2 Cost function

We are interested in the following temporal task: to find a model mapping u to y such that it minimises a given error measure. Mathematically, we can define a function ϕ corresponding to the network in Figure (42) as follows:

$$\begin{cases} \phi : T \times U \to Y \\ \phi : t, u \to \hat{y} = g(W_{out}[1\|u(t)\|x(t)]) \\ x : T \times U \times X \times Y \to Y \\ x : t, u, x, y \to x(t) = f(W_{in}u(t) + W_{res}x(t-1) + W_{fb}y(t-1)) \end{cases} \tag{12.65}$$

Note, x behaves as a simple network but we will introduce a more complex function later.
We recall the definition of the Root-Mean-Square Error (RMSE) between the train and target vectors:

$$E(y_{train}, y_{target}) = \sqrt{\frac{1}{T \times N_y} \sum_{t=1}^{T} \sum_{i=1}^{N_y} (y_{target,i}(t) - y_{train,i}(t))^2} \tag{12.66}$$

12.1.3 Random generation of the input and feedback matrices

These matrices are chosen randomly as suggested by the ESN framework. They can be initialised by using a particular random variable distribution, such as, the uniform or Gaussian distribution. In our experiments we choose to use a Gaussian distribution. We also consider a scaling parameter for each matrix in order to optimise them, denoted by $s_{in}, s_{bias}, s_{res}$ and s_{fb}. As discussed in Section (2.2), the feedback scaling parameter is of key importance in our network as the reservoir needs to recall its last forecasting value in the best manner regarding our cost function defined in Equation (12.66).

12.1.4 Topology of the reservoir matrix

Using a completely random network leads the reservoir structure to be poorly understood. Rodan et al. [2012] demonstrated that simple cycle reservoir (SCR) shows comparable performance to the randomised one. Further, they have introduced a new model adding regular jumps to the SCR, called cycle reservoir with jumps (CRJ). The latter has shown superior performance compared to the traditional ones in non-linear system identification, more particularly in time series prediction. We consider this last model in our system. For example, if each of our reservoir factor was made of six neurons, the reservoir would be as in Figure (43).

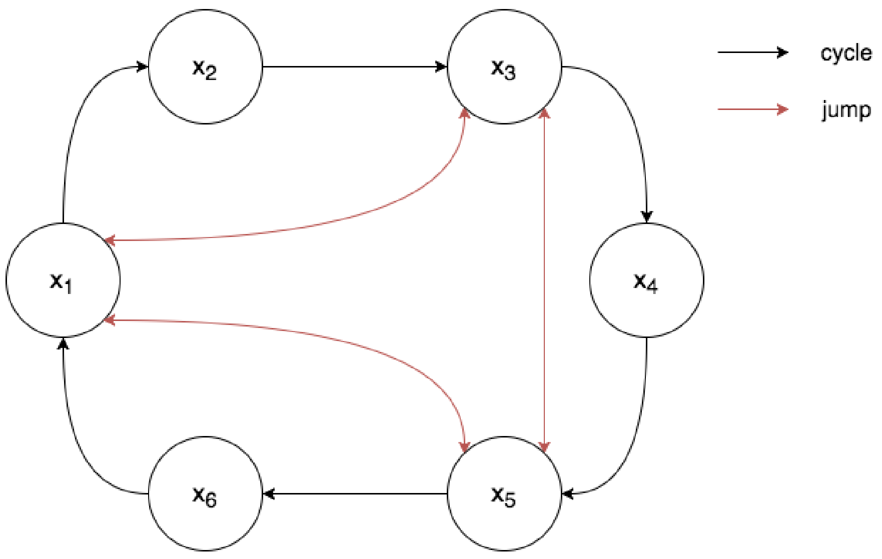


Figure 43: CRJ model for a 6 neurons reservoir factor

In the CRJ model, four parameters are considered for each factor: the cycle weight r_c, the jump weight r_j, the jump level telling at which neuron the jumps start and the jump sizing how many neurons are jumped. Given these parameters, the reservoir is entirely deterministic. Considering Figure (43) above, the reservoir matrix is constructed as follows:

$$W_{res} = \begin{bmatrix} 0 & 0 & r_j & 0 & r_j & r_c \\ r_c & 0 & 0 & 0 & 0 & 0 \\ r_j & r_c & 0 & 0 & r_j & 0 \\ 0 & 0 & r_c & 0 & 0 & 0 \\ r_j & 0 & r_j & r_c & 0 & 0 \\ 0 & 0 & 0 & 0 & r_c & 0 \end{bmatrix}$$

12.1.5 Reservoir dynamics

In a temporal task, it is important to constantly update the kernels with a function taking both new input and the previous kernels. This leads to the notion of dynamical short-term memory. The state vector x is updated at every time $t \in \mathbb{T}$ using the following recursion equations:

$$\forall k \in \mathbb{K} \quad \begin{cases} \tilde{x}^k(t) = \delta^k S_{a^k}(W^k_{in} \cdot [1; u(t)] + W^k_{res} \cdot x^k(t-1) + W^k_{fb} \cdot y(t-1) + \epsilon^k) \\ \qquad\quad + (1-\delta^k)(W^k_{in} \cdot [1; u(t)] + W^k_{res} \cdot x^k(t-1) + W^k_{fb} \cdot y(t-1) + \epsilon^k) \\ x^k(t) = (1-\gamma^k) x^k(t-1) + \gamma^k \tilde{x}^k(t) \end{cases} \tag{12.67}$$

where γ^k is the learning rate of the k-th kernel telling how much it learns from what has happened in the previous state as new data come in. ϵ^k is a matrix noising the kth kernel and thus preventing overfitting. It is monitored by a parameter determining the standard deviation of the Gaussian distribution. a^k is the active rate which is linked to the parametrised sigmoid function S_{a^k}. This function is applied element wise to our multidimensional factors. A plot is given in Figure (44), illustrating its behaviour as a varies.

$$S_a : x \in \mathbb{R} \to \frac{2}{1 + e^{-\frac{x}{a}}} - 1 \tag{12.68}$$

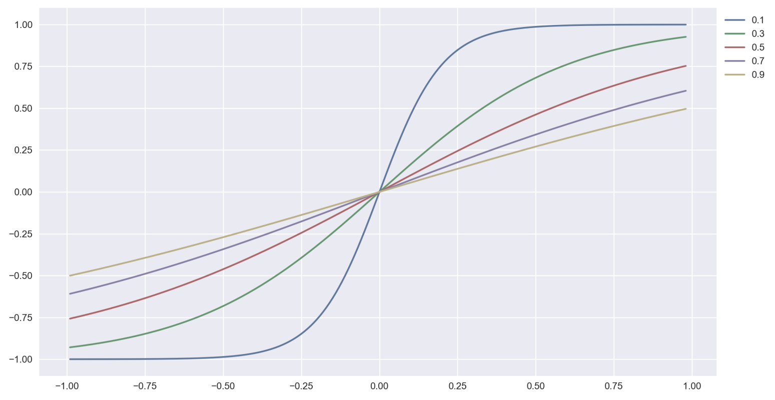


Figure 44: parametric sigmoid function

Sigmoidal functions are often used in machine learning to squash values to their limits. They are widely used in classification tasks. The reason being that this function is a way of assigning the negative value -1 or positive value

$+1$ to a neuron in a smooth manner. The active rate a defines how important this squashing is. Further, δ^k monitors the importance of this squashing given the neuron's value.

For a given time t, the resulting $x(t)$ is called echoes of its input history $\{u(1) \dots u(t)\}$. We can set $\gamma = \alpha \times \Delta t$, where Δt is the time interval at which the data is coming into the system and α is a parameter. The smaller α and Δt are, the more the reservoir reacts slowly to new input. Note that from a signal processing point of view, the exponential moving average in Equations (12.67) acts as a low-pass filtering of its activations with cutoff frequency:

$$f_c = \frac{\alpha}{2\pi(1-\alpha)\Delta t}$$

12.2 Optimisation methods

We sum up the parameter set in the following picture and table. For each parameter, we provide its location, its characteristics such as its type (micro or macro) and its RMSE variability impact.

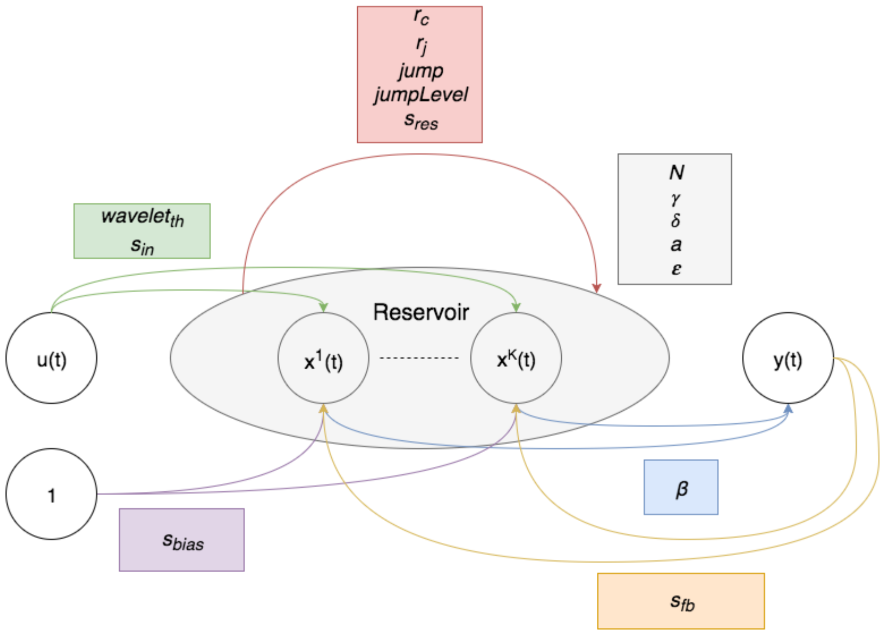


Figure 45: Parameters to be optimized.

	Type		RMSE Influence		
Parameter	Micro	Macro	Low	Med	High
r_c	$\times$		$\times$		
r_j	$\times$		$\times$		
$jump$	$\times$		$\times$		
$jumpLevel$	$\times$		$\times$		
$wavelet_{th}$		$\times$			$\times$
β	$\times$		$\times$		
N		$\times$	$\times$		
s_{in}	$\times$			$\times$	
s_{bias}	$\times$		$\times$		
s_{res}	$\times$			$\times$	
s_{fb}	$\times$			$\times$	
γ	$\times$				$\times$
δ	$\times$		$\times$		
a	$\times$				$\times$
ϵ	$\times$		$\times$		

Table 8: Parameter characteristics. Micro/Macro corresponds to a matrix ladder (a parameter is micro if it defines only its elements, macro if it defines its shape). Thanks to a graph of the RMSE through the optimization process, we can classify the influence of each parameter.

Note that some parameters depend on the factor's number. The number of parameters can grow very fast as we increase this number. Only β and $wavelet_{th}$ are independent of the reservoir shape. If we define m_f as the factors' number, then we have the following number of parameters:

$$n_{param} = 13 * m_f + 2 \tag{12.69}$$

We have focused on a 2-factor reservoir in our study as it provides two high dimensional signals and at the same time keeps the optimisation time relatively low. Therefore, the number of parameters is 28. This is relatively high and we cannot rely on an exhaustive line search, as it would take too much time to run, and also the system would be over-optimised, leading to overfitting the data.

12.2.1 Random Generation: GEN

One way of solving the above problem is to consider the following functions:

- global optimisation: it generates uniformly random values inside a space $globalMax$ times in a same loop and update the best parameters upon the RMSE obtained on the validation sample by running training and validation.
- local optimisation: exactly the same as the global one with $localMax$ times, except that it is now restricted to a smaller space (20% of the global one).
- cross optimisation: it consists in optimizing the parameters one by one $crossMax$ times and see if there is any improvement on the validation sample. It also performs two local optimisations at the end (one at 20% followed by one at 10%).
- While loop: it combines a more exhaustive global optimization ($whileMax$ times) with local and cross optimisations if there is improvement in the global.

***Remark* 12.1** *Note that at each call of any of these functions, new training and validation running are performed.*

Algorithm 10 Optimisation Strategy 1: GEN

```
Require: get $whileMax$.
Require: set $count \leftarrow 0$, $best_{RMSE} \leftarrow \infty$.
 1: $RMSE \leftarrow$ global()
 2: $RMSE \leftarrow$ local()
 3: $RMSE \leftarrow$ cross()
 4: while $count < whileMax$ do
 5:     $\mathbf{p} \leftarrow$ random()
 6:     $node \leftarrow$ Reservoir($\mathbf{p}$)
 7:     $prenode \leftarrow$ LeakyIntegrator($node$)
 8:     $prenode$.train()
 9:     $RMSE \leftarrow prenode$.valid()
10:     if $RMSE < best_{RMSE}$ then
11:         $RMSE \leftarrow$ cross()
12:         $RMSE \leftarrow$ local()
13:     end if
14:     $count \leftarrow count + 1$
15: end while
16: return $RMSE$
```

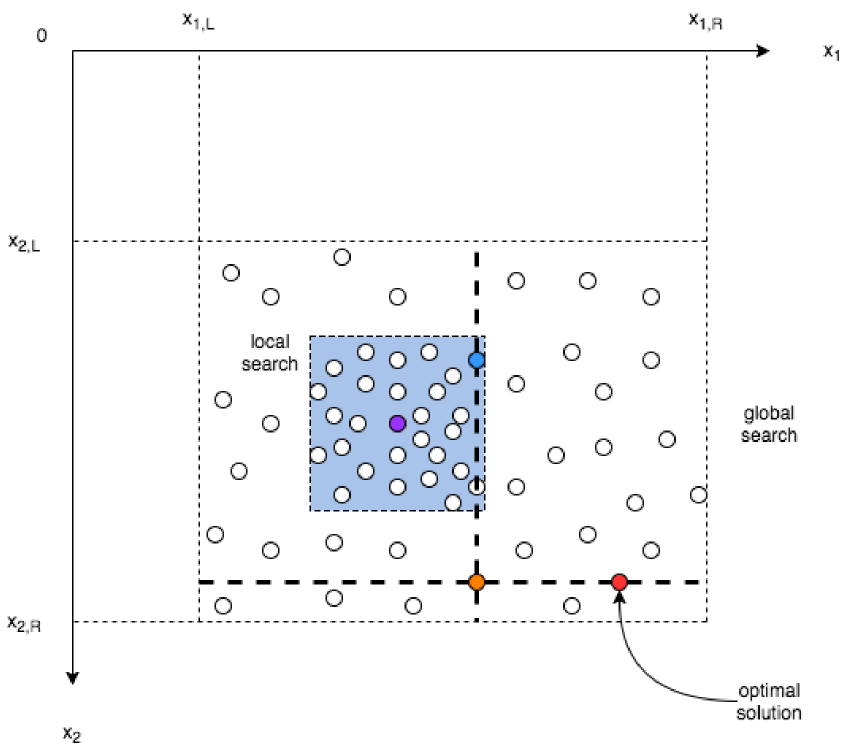


Figure 46: Random optimisation strategy with 2 parameters. The global optimisation finds an optimal point (purple). Then a local optimisation is done in the blue area where another optimum is found (blue). Then comes the cross optimisation where an optimum is firstly found along x_2 (orange) and secondly along x_1 (red). The $while$ loop part is not represented here in order to lighten the figure but the reader can easily see what it consists in.

***Remark* 12.2** *In our experiments, we note that the while loop is often useless when the cross optimisation is long enough.*

12.2.2 Stochastic Gradient Descent: SGD

The idea of this optimisation strategy is to still perform a global search and also try to find as quickly as possible the local optimum by performing a stochastic gradient descent.

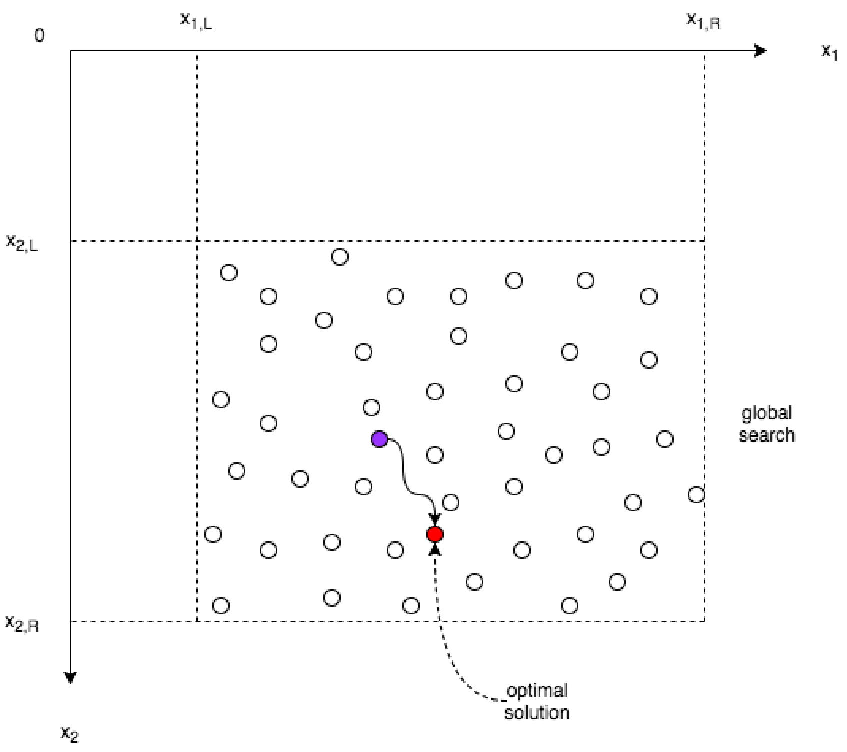


Figure 47: SGD optimisation strategy with 2 parameters. For every global generation (purple) we refine it by changing only a few parameters in order to get to a local optimum (red).

Tang et al. [2014] implemented this strategy by changing only the parameters r_c and r_j. They claimed that this method converges faster to an optimum and does not affect the accuracy. In this section, we lay out this method for the following set of parameters: r_c, r_j, δ, γ and a. The authors also define the reservoir matrix W_{res} with a cycle reservoir with jumps (CRJ). The idea of their method is to train the system with a hybrid optimisation strategy:

- Readouts: the output weights of the matrix W_{out} is trained using Equation (11.64).
- Reservoir: each parameter θ is optimised on a grid search. E is defined as in Equation (12.66).

$$\frac{\partial E}{\partial \theta} = 2\sum_{t=t_0+1}^{t_1}(\hat{y}(t)-y(t))W_{out}\frac{\partial x(t)}{\partial \theta} + \underbrace{2\sum_{t=t_0+1}^{t_1}(\hat{y}(t)-y(t))x(t)\frac{\partial W_{out}}{\partial \theta}}_{=0} \tag{12.70}$$

where θ is the parameter we optimise. Indeed, the second term is null as we use W_{out} from Equation (11.64), which implies that $\frac{\partial E}{\partial \theta} = \sum_{t=t_0+1}^{t_1}(\hat{y}(t)-y(t))x(t) = 0$.

Remark 12.3 *Note that this fact is only true when W_{out} is constant, which is not the case in online algorithms. Thus, we will mainly use the GEN optimiser in our study in order to fairly compare the different methods.*

Recall the two equations updating a factor of the reservoir at time t:

$$\begin{cases} \tilde{x}(t) = \delta S_a(W_{in}\cdot[1;u(t)] + W_{res}\cdot x(t-1) + W_{fb}\cdot y(t-1)+\epsilon) \\ \qquad + (1-\delta)(W_{in}\cdot[1;u(t)] + W_{res}\cdot x(t-1) + W_{fb}\cdot y(t-1)+\epsilon) \\ x(t) = (1-\gamma)x(t-1)+\gamma\tilde{x}(t) \end{cases} \tag{12.71}$$

From these equations we can derive $\frac{\partial x(t)}{\partial \theta}$:

$$\begin{cases} \frac{\partial x(t)}{\partial r_c} = (1-\gamma)\frac{\partial x(t-1)}{\partial r_c} + \gamma(1-\delta+\frac{\delta}{a}S_a(\omega)(1-S_a(\omega))) \odot (\frac{\partial W}{\partial r_c}x(t-1) + W\frac{\partial x(t-1)}{\partial r_c}) \\ \frac{\partial x(t)}{\partial r_j} = (1-\gamma)\frac{\partial x(t-1)}{\partial r_j} + \gamma(1-\delta+\frac{\delta}{a}S_a(\omega)(1-S_a(\omega))) \odot (\frac{\partial W}{\partial r_j}x(t-1) + W\frac{\partial x(t-1)}{\partial r_j}) \\ \frac{\partial x(t)}{\partial \delta} = (1-\gamma)\frac{\partial x(t-1)}{\partial \delta} + \gamma(S_a(\omega)-\omega) \\ \frac{\partial x(t)}{\partial \gamma} = -x(t-1) + \tilde{x}(t) \\ \frac{\partial x(t)}{\partial a} = (1-\gamma)\frac{\partial x(t-1)}{\partial a} - \gamma\delta\frac{u}{a^2}S_a(\omega)(1-S_a(\omega)) \end{cases} \quad (12.72)$$

where $\omega = W_{in} \cdot [1; u(t)] + W_{res} \cdot x(t-1) + W_{fb} \cdot y(t-1) + \epsilon$. Then we update parameter θ using the following equation:

$$\boxed{\theta_{n+1} = \theta_n - \alpha_\theta * \frac{\partial E}{\partial \theta_n}} \quad (12.73)$$

Algorithm 11 Optimisation Strategy 2: SGD

```
Require: get $whileMax$, $SGD_{Max}$, $V$ (parameters' set).
Require: set $RMSE$, count ← 0, $whileCount$ ← 0, $SGD_{count}$ ← 0 p.
1:  while $whileCount < whileMax$ do
2:      p ← random()
3:      while $SGD_{count} < SGD_{Max}$ do
4:          for $\theta$ in $V$ do
5:              $\theta = \theta - \alpha_\theta * \frac{\partial E}{\partial \theta}$
6:          end for
7:          $node$ ← Reservoir(p)
8:          $prenode$ ← LeakyIntegrator($node$)
9:          $prenode$.train()
10:         $RMSE$ ← $prenode$.valid()
11:         $SGD_{count}$ ← $SGD_{count} + 1$
12:     end while
13: end while
14: return $RMSE$
```

12.3 Adaptive filters for online learning

There exists several ways of defining an online training algorithm, such as gradient descent and recursive least squares (see Haykin [2000], Bishop [2006]).

12.3.1 A review of adaptive filtering theory

We consider the adaptive filtering theory and justify its ability to improve the offline model presented in the previous part. Douglas [1999] gave the following definition:

Definition 12.1 *Adaptive Filter*
An adaptive filter is a computational device that attempts to model the relationship between signals in real time in an iterative manner.

Later, Steil [2004] argued that an adaptive filter is able to adapt extremely fast to the main characteristics of the task in a sort of one-shot online-learning.
In general, an adaptive filter can be identified by four different features:

1. The signal being processed by the filter. For instance, it can correspond to the daily closed prices.
2. The structure/scheme/process that is used to compute the output signal from the input signal it processes. This is represented by Figure (42).
3. The parameters defined in its structure/scheme/process that can be modified in an iterative manner. The aim being to modify the input-output relationship to reduce a cost function.
4. The algorithm describing how these parameters are modified through time.

Given $\mathbb{T} = [1, N_T]$ a discrete time set, for each $t \in \mathbb{T}$, the input signal $u(t)$ is fed into the reservoir, now called adaptive filter, which computes an output signal $\hat{y}(t)$. The reservoir defines parameters that can adjust the values of $\hat{y}(t)$. At time $t+1$ we know the real world value of the output, therefore it is natural to define the difference signal as follows:

$$\forall t \in [1, T] \, , \; e(t) = y(t) - \hat{y}(t)$$

This signal, described in the diagram in Figure (48), is called the error signal. As time t is incremented, we hope that our adaptive filter learns from the environment and gets better and better at matching the response signal $y(t)$ through an adaptation process.

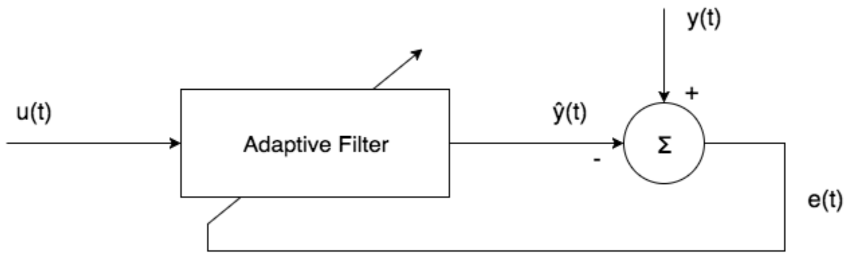


Figure 48: Scheme of the adaptive filtering purpose

There exists several types of application such as system identification, inverse modelling and linear prediction. For our task, we will use the latter and describe it in the diagram in Figure (49).

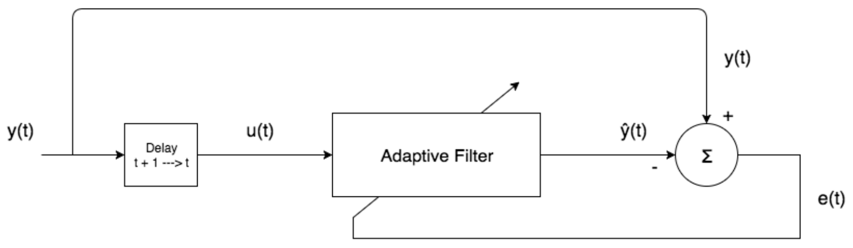


Figure 49: Linear Predictor problem

In our task, we have the following:

$$\forall t \in [1, T] \,, \; u(t) = y(t - \Delta)$$

with $\Delta = 1$ when the non-filtered data are fed in the adaptive filter.
The diagram above indicates that if a certain output signal $y(t)$ is desired, we can consider it as input to compute the output of our model $\hat{y}(t)$. If the system we wish to predict is stable enough, then the next forecast should be even closer.
There are several measures to quantify the task performance such as the root mean square error (RMSE). An adaptive algorithm aims at adjusting a set of parameters of the corresponding adaptive filter in order to reduce as much as possible the RMSE. Given Definition (12.1), an adaptive signal must define some parameters which can be adapted at every point in time to achieve this goal. From RC theory, since the significant matrix in the learning process is W_{out}, we should only adapt the latter. A general equation to model an adaptive filtering algorithm in our task is as follow:

$$W_{out}(t+1) = W_{out}(t) + \mu(t) G\left(e(t), u(t), s(t)\right)$$

where $G(\cdot)$ is a vector-valued non-linear function, $\mu(t)$ a parameter controlling the step size at time t. $e(t)$ and $u(t)$ are respectively the error and input signals evaluated at time t. Douglas [1999] also suggested *"$s(t)$ [as] a vector of states that store pertinent information about the characteristics of the input and error signals and/or the coefficients at previous time instants."* s stands as the reservoir states in our task. The author pointed out that the success or failure of an adaptive filtering algorithm heavily relies on the choice of the step size μ. Therefore, μ requires significant attention when optimising in the validation set.

12.3.1.1 Simple gradient descent We recall the gradient descent update equation below:

$$\boxed{\omega_{i,j}(t+1) = \omega_{i,j}(t) - \mu_t \cdot \frac{\partial SE(t)}{\partial \omega_{i,j}(t)}} \tag{12.74}$$

where $\omega_{i,j}(t)$ is the (i,j)th element of the output matrix W_{out} at time t. We also recall the equation computing the forecast at time t:

$$\hat{y}(t) = g\left(W_{out} s(t)\right) \tag{12.75}$$

where $g \in C^1(\mathbb{R})$ and applies to a vector in an element-wise manner and $\forall t \in \mathbb{T} \;\; s(t) = [1 \| u(t) \| x(t)]$.
We derive $\frac{\partial E(t)}{\partial \omega_{i,j}(t)}$ below using the framework defined in Appendix (11.1.5).

$$SE(t) = \sum_{i=1}^{N} \left(y_i(t) - \hat{y}_i(t)\right)^2 \tag{12.76}$$

$$\Rightarrow \quad \frac{\partial SE(t)}{\partial \omega_{i,j}(t)} = -2 \sum_{i=1}^{N} (y_i(t) - \hat{y}_i(t)) \frac{\partial \hat{y}_i(t)}{\partial \omega_{i,j}(t)} \tag{12.77}$$

$$\Rightarrow \quad \frac{\partial SE(t)}{\partial \omega_{i,j}(t)} = -2 \sum_{i=1}^{N} (y_i(t) - \hat{y}_i(t)) g'(W_{out} s(t)) s_j(t) \tag{12.78}$$

12.3.1.2 Recursive least squares The Recursive Least Squares (RLS) algorithm can be used for iterative computation of the inverse of the correlation matrix R_{ij}^{-1} and the readout weights ω (see Appendix (11.2.3.1)). It can be summarised as follows:

$$
\begin{aligned}
\kappa(t) &= \frac{\nu^{-1}\Gamma(t-1)x(t)}{1+\nu^{-1}x^{\top}(t)\Gamma(t-1)x(t)} \\
\omega(t) &= \omega(t-1)+\kappa(t)\left(y(t)-\hat{y}(t)\right) \\
\Gamma(t) &= \nu^{-1}\Gamma(t-1)-\nu^{-1}\kappa(t)x^{\top}(t)\Gamma(t-1)
\end{aligned}
$$

where $\kappa(t)$ is the gain vector and $\Gamma(t)$ is the estimate of the inverse of the correlation matrix. The forgetting factor ν is set to 1. This method ensures faster convergence of the readout weights but is much more computationally intensive than the simple gradient descent algorithm.

12.3.2 Online algorithm

We are now going to describe the online algorithm for real time learning of output matrix W_{out}. While we have considered the actual output of the financial time series y, it is more relevant to take the filtered signal as the target signal. Indeed, this signal is by definition smoother and therefore more continuous in time.

Algorithm 12 RC online Training

```
Require: choose a gradient descent algorithm $\mathbb{A}$
1: $W_{out} \leftarrow$ Algorithm ( 9)
2: set $SE \leftarrow 0$
3: for $t \leftarrow 0$ to $T_{valid}$ - 1 do
4:     update each reservoir factor using Equation ( 12.67).
5:     compute forecast using Equation ( 12.75).
6:     $SE \leftarrow SE + (y(t+1) - \hat{y}(t+1))^2$
7:     update $W_{out}$ with Equation ( 12.74) using $\mathbb{A}$.
8: end for
```

In Algorithm (12) we use $\mathbb{A}$ as the scaled conjugate gradient implemented in GNU Scientific Library [2017]. This method is very fast for a well-chosen initial guess and it makes sense to suppose that the output matrix should be close to its previous state as the signal we try to calibrate is smooth.

***Remark* 12.4** *We can also replace y in Equation (12.67) with its last feedback $\hat{y}$. Consequently, the reservoir has more information about how accurate its last forecast was.*

12.4 Associative Reservoir Computing

12.4.1 Paradigm

12.4.1.1 Definitions So far, we have taught the reservoir the input to output mapping. Equation (12.79) introduce the function ϕ defining this mapping and carrying the forward relation between the input sample U and the output sample Y without studying its properties. We are now going to understand the function ϕ in view of making better prediction.

Definition 12.2 *Ambiguous relation*
Let $\phi : U \to Y$. We say that ϕ is ambiguous if it is not injective i.e. $\exists (u_1, u_2) \in U^2 | u_1 \neq u_2 \Rightarrow \phi(u_1) = \phi(u_2)$

Definition 12.3 *State space*
A state space is the set of all possible states [the reservoir states x in our case] of a dynamical system. Each state of the system corresponds to a unique point in the state space (see Terman et al. [2008]).

Definition 12.4 *Attracting set*
Let S be a dynamical system and X its associated state space. We say of a set A to be an attracting set of S if it is a closed subset of X such that $\forall a \in A, \exists x(0) = x_0 | x(t) \to a$ as $t \to +\infty$.
We call its elements attractors.

Definition 12.5 *Basin of attraction*
Let a be an attractor of A. We define the basin of attraction of A $B(A)$ as the set of initial conditions such that $x(t) \to a$ as $t \to +\infty$.

In associative reservoir computing, the learning process aims at shaping the reservoir dynamics in a particular attractor. A major problem when learning ϕ is that it may be an ambiguous relation. Following Reinhart [2011], we develop a method for resolving the ambiguity of ϕ using a dynamical approach.

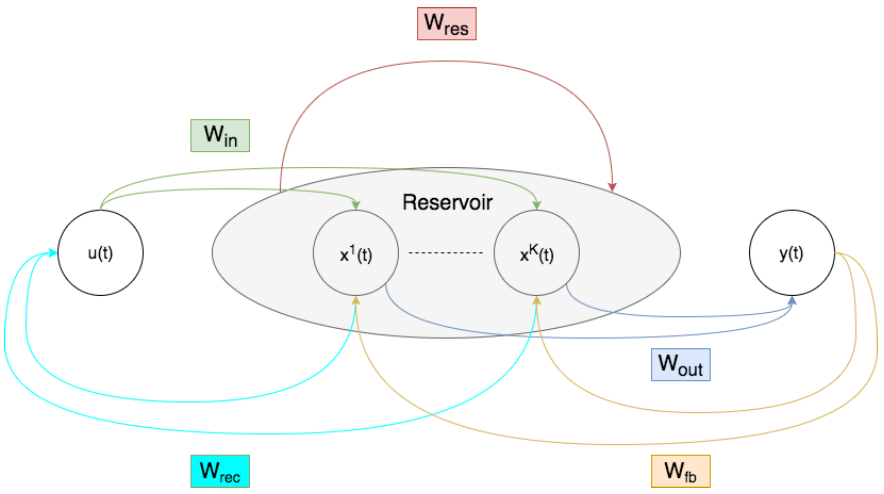
12.4.1.2 The model In the ARC model, we overwrite the reservoir forward representation by a joint representation involving input u and output y. To access the latter, we need a matrix $W_{fb} \neq 0$. Furthermore, we define a matrix mapping the reservoir states to the input layer and called it $W_{rec}(\in \mathbb{R}^{N_U \times N_S})$. We want to learn the mapping ψ from output to input. Figure (50) defines the different connections involved in this model. We have the following system:

$$\begin{cases} \phi : T \times U \times X \to Y \\ \phi : t, u, x_{fwd} \to \hat{y} = g(W_{out}[1 \| u(t) \| x_{fwd}(t)]) \\ \psi : T \times Y \times X \to U \\ \psi : t, y, x_{bwd} \to \hat{u} = g(W_{rec}[1 \| y(t) \| x_{bwd}(t)]) \\ x_{fwd} : T \times U \times X \times Y \to X \\ x_{fwd} : t, u, x, y \to x_{fwd}(t) = f(W_{in} u(t) + W_{res} x(t-1) + W_{fb} \hat{y}(t-1)) \\ x_{bwd} : T \times U \times X \times Y \to X \\ x_{bwd} : t, u, x, y \to x_{bwd}(t) = f(W_{in} \hat{u}(t-1) + W_{res} x(t-1) + W_{fb} y(t)) \end{cases} \quad (12.79)$$

Equations (12.79) above define the associative system. The system is now composed of two functions ϕ and ψ mapping respectively input to output and output to input. We have also defined two states x_{fwd} and x_{bwd} corresponding respectively to the states obtained with forward model ϕ and backward model ψ.

***Remark* 12.5** *In practice the backward model cannot be use in our task (at least on the testing sample) as we obviously do not know the asset price we want to predict. However this framework demonstrates that we can compute its corresponding input $\hat{u}$ by simply using the last prediction $\hat{y}(t-1)$.*

Figure 50: ARC model.



Typically, it is often recommended to use the output at time t when learning on the training sample, we call this process teacher forcing. On the validation set, we can keep training the W_{out} and W_{rec} matrices in order to teach the reservoir how to stay close to the target signal. That is to say, we use equation 12.74 to update W_{out} and W_{rec}, the idea being to stay close to an attractor as time goes on. In the next subsection we lay out our algorithm for converging to an attractor at every time.

12.4.1.3 Attractor-based computation In the Figure (51) below, the system sits at time t and iterates the reservoir dynamics for a certain number of time, k_{max}, using always the same input $u(t)$. This artificial dynamics causes the reservoir $x(t)$ to evolve toward a stable state. A good forecast of the output attractor can be performed using Equations (12.79) afterwards.

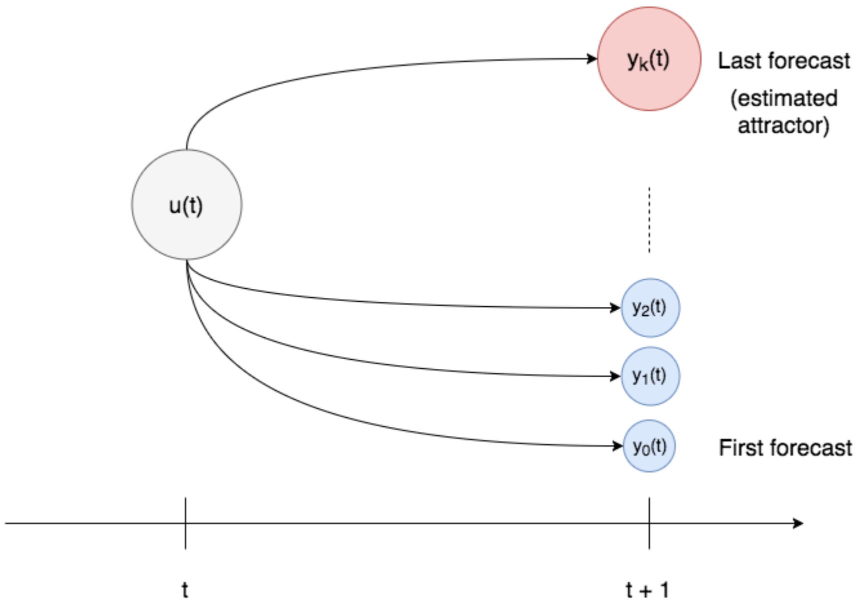


Figure 51: Dynamical system model to fit an attractor.

As explained by Reinhart [2011], the dynamical approach to model ambiguous inverse problems crucially depends on the ability to shape multi-stable attractor dynamics that reflect the multiplicity of solutions.

***Remark* 12.6** *The attractors we try to fit are the different filters we may use to denoise the initial signal. For instance, it can correspond to several wavelets, decomposition and thresholding levels.*

Algorithm 13 Attractor Convergence Algorithm

```
Require: get an input data at time $u(t)$.
Require: set $k_{max}$ as a maximum number of iteration.
Require: set $\epsilon$ as a tolerance level.
1: while $k < k_{max}$ and $norm < \epsilon$ do
2:     compute $x_{fwd}(t)$ using Equation ( 12.79).
3:     compute $\hat{y}(t)$ and $\hat{u}(t)$ using Equation ( 12.79).
4:     $norm \leftarrow ||x_{fwd}(k+1) - x_{fwd}(k)||^2$
5:     $k \leftarrow k + 1$
6: end while
```

In Algorithm (13) we eliminate the transient effect by iterating the states using the same input. It is as if we would have frozen the series at time t and went on feeding the reservoir, leading to convergence to an attractor. Basically, we want to fit the reservoir dynamics to this attractor in order to teach the reservoir how to reproduce its behaviour. If it is the case, then the network should not need this trick any more and it would predict well this attractor. Ambiguity can be resolved by applying Algorithm (13), causing the reservoir to settle in a particular attractor.

12.4.2 States stabilisation techniques

A major issue of our task is that only noised data are observable in the market. However, the reservoir states need to be smooth (in the sense of its norm) in order to hope learning. Indeed, if the states are stables as time goes, then the predicted values should be stables as well. From Table (2) we have seen that there is a good learning when dealing with filtered input. However we cannot use the wavelet filter in the testing sample. To offset this problem, we could imagine teaching the reservoir how to remain consistently stable while feeding it with noisy input.

12.4.2.1 Reservoir dynamics calibration to observable stable states Suppose that we can observe stable states x^* on the training sample. We wish to find a dynamics $W_{opt} = [W_{in} \| W_{res} \| W_{fb}]$ such that it minimises the following cost function:

$$E = \sum_{t=1}^{T_{training}} ||x^*(t) - x(t)||^2 \tag{12.80}$$

Given u, x^* and y on the training sample, the problem is stated as follows:

$$\min_{W_{opt}} \sum_{t=1}^{T_{training}-1} ||x^*(t+1) - f(W_{opt}[u(t) \| x^*(t) \| y(t)])||^2 \tag{12.81}$$

Let us define

$$S^* = [[u(0) \| x^*(0) \| y(0)] \| \dots \| [u(T_{training}-1) \| x^*(T_{training}-1) \| y(T_{training}-1)]]$$

and

$$A^* = [f^{-1}(x^*(1))\| \dots \| f^{-1}(x^*(T_{training}))]$$

Then we can solve this problem using Ridge regression:

$$\hat{W}_{opt} = A^* \cdot {S^*}^T \cdot (S^* \cdot {S^*}^T + \lambda I)^{-1} \tag{12.82}$$

where λ is a well chosen regularization parameter. Given these calibrated matrices, the reservoir can now learn from noisy input while staying relatively stable.

***Remark* 12.7** *The filter is of key importance as it will define how the system can map noisy input to smooth states.*

In practice we want the reservoir to remain stable when fed with unseen noisy input. Thus, we must consider a large enough training sample to ensure the states remains stable for exploitation.

12.4.2.2 Learning with filtered inputs Another idea would be to work with filtered inputs only and to directly teach the reservoir to predict the filtered data (coloured in green). Indeed, this filter can be seen as an indicator of the initial dataset (see Technical Indicators in Appendix (8)). One could then argue that getting a good prediction of the filtered data could lead to good real time series forecasting if one can control the forecasting error.

Suppose that we observe stable states x^* on the training and validation samples. In these samples we can use all the information at any time, so that we can fit a filtering technique such as wavelets and feed the reservoir using the filtered inputs only. The states should then be relatively stable on the training and validation, as previously discussed. Then, using Equation (12.74) (for both W_{out} and W_{res}), we teach the network to fit this filtered dynamics. At the end of the validation process, we hope that our network will be well enough trained and that we no-longer need Equation (12.74) to obtain good forecasts. That is, we should have the functions ϕ and ψ relatively constant with respect to the parameters $(W_{in}, W_{res}, W_{fb}, W_{out}, W_{rec})$.

We present in Figure (52) our risk management scheme when making a forecast. The idea being to use, under particular assumptions, the information at time t with the backward mapping function ψ, in order to have a control over the forecast error.

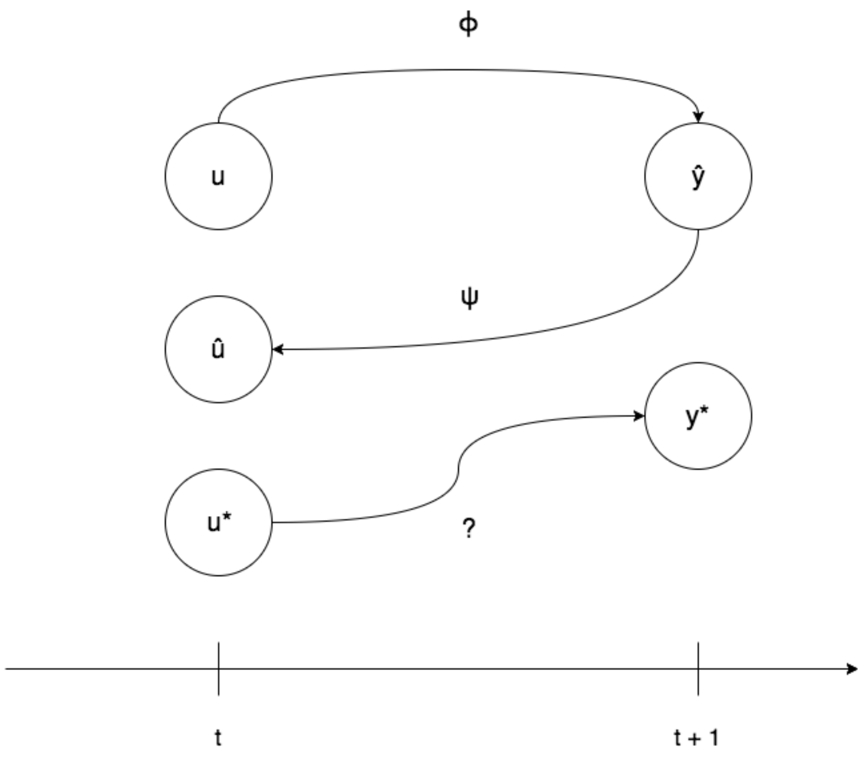


Figure 52: Risk management scheme.

Suppose that $\exists t_0 \in [0, T_{valid}]$ such that $\forall t \in \mathbb{T}_0 := [t_0, T_{valid}], \phi(u(t)) = \hat{y}(t)$ and $\psi(\hat{y}(t)) = u(t)$. Then $\psi^{-1}_{|\mathbb{T}_0} = \phi_{|\mathbb{T}_0}$ (injective property is verified at time t if Algorithm (13) converges). Suppose also that $\phi_{|\mathbb{T}_0}$ is time-independent and that $\forall t \in \mathbb{T}_0, |u^*(t) - \hat{u}(t)| \leqslant K$. Then, we could set up the following strategy:

Algorithm 14 Out-of-sample forecasting risk management

```
Require: set $l_V = \sup\{|u^*(t) - \hat{u}(t)|, \forall t \in \mathbb{T}_0\}$
Require: set $l_T \leftarrow 0$
1: while $l_T < l_V$ do
2:     update the reservoir $x$ up to time t using Equations ( 11.62) and ( 11.63).
3:     compute forecast $\hat{y}(t)$ and associative input $\hat{u}(t)$ using Equation ( 12.75).
4:     $l_T \leftarrow |u^*(t) - \hat{u}(t)|$
5: end while
```

The ARC framework is very powerful as it allows us to evaluate the input error at time t to have an idea of the forecast accuracy at time $t + 1$.

13 Introduction to constrained optimisation

See textbooks by Bertsekas [1996] [2015], Bazaraa et al. [1993], Nocedal et al. [1999], Boyd et al. [2004], Sun et al. [1999], among other.

13.1 Introduction

13.1.1 Defining the problem

The optimisation problems constitute a large class of problems which gave rise to a research area known as Combinatorial Optimisation. In such problems, we try to optimise (maximise or minimise) some quantity, while satisfying some constraints. Optimisation problems are of particular interest in the area of Operations Research and they appear in many real situations.

13.1.1.1 The constrained problem We consider a system with the real-valued properties

$$g_m \text{ for } m = 0, .., P-1$$

making the objectives of the system to be optimised. Given a N-dimensional vector of real-valued parameter $X \subset \mathbb{R}^N$ the optimisation problem can always be written as

$$\min f_m(X)$$

where $f_m(\cdot)$ is a function by which g_m is calculated and where each element $X(i)$ of the vector is bounded by lower and upper limits $L_i \leqslant X(i) \leqslant U_i$ which define the search space $\mathcal{S}$. Depending on the linearity or non-linearity of the objective function we can distinguish between linear and non-linear optimisation.
We follow Lueder [1990] who showed that all functions $f_m(\cdot)$ can be combined in a single objective function $H : X \subset \mathbb{R}^N \to \mathbb{R}$ expressed as the weighted sum

$$H(X) = \sum_{m=1}^{P} w_m f_m(X)$$

where the weighting factors w_m define the importance of each objective of the system. Hence, the optimisation problem becomes

$$\min H(X)$$

so that all the local and global minima (when the region of eligibility in X is convex) can be found. However, most problems involves a single objective function, so that the optimisation function simplifies. Most complex search problems such as optimisation problems are constrained numerical problem (CNOP) more commonly called general nonlinear programming problems with constraints given by

$$g_i(X) \leqslant 0 \, , \, i = 1, .., p$$
$$h_j(X) = 0 \, , \, j = 1, .., q$$

Equality constraints are usually transformed into inequality constraints by

$$|h_j(X)| - \epsilon \leqslant 0$$

where ϵ is the tolerance allowed. Given the search space $\mathcal{S} \subset \mathbb{R}^N$, we let $\mathcal{F}$ be the set of all solutions satisfying the constraints of the problems called the feasible region. It is defined by the intersection of $\mathcal{S}$ and the set of $p+q$ additional constraints. At any point $X \in \mathcal{F}$, the constraints $g_i(\cdot)$ that satisfy $g_i(X) = 0$ are active constraints at X while equality constraints $h_j(\cdot)$ are active at all points of $\mathcal{F}$. Many practical problems have objective functions that are non-differentiable, non-continuous, non-linear, noisy, multi-dimensional and have many local minima.

13.1.1.2 Some examples We let $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^l$ and consider the optimisation problem

$$\min_x f(x) \text{ such that } g(x) \leqslant 0 \, , \, h(x) = 0 \tag{13.83}$$

As discussed above, any equality constraint $h_i(x)$ can be transformed into two inequality constraints

$$h_i(x) \geqslant 0 \text{ and } -h_i(x) \geqslant 0$$

such that

$$\min_x f(x) \text{ such that } g(x) \leqslant 0 \tag{13.84}$$

is sufficiently general.
Some examples of optimisation problems are

- Linear regression:
$$\min_w \|xw - y\|^2$$
- Classification (logistic regression or SVM)
$$\min_w \sum_{i=1}^n \log\big(1 + e^{-y_i x_i^\top w}\big)$$
or
$$\|w\|^2 + C \sum_{i=1}^n \xi_i \text{ such that } \xi_i \geqslant 1 - y_i x_i^\top w \, , \, \xi_i \geqslant 0$$
- Maximum likelihood estimation:
$$\max_\theta \sum_{i=1}^n \log p_\theta(x_i)$$
- k-,means
$$\min_{\mu_1,\dots,\mu_k} J(\mu) = \sum_{j=1}^k \sum_{i \in C_j} \|x_i - \mu_j\|^2$$

Some important examples in machine learning are

- SVM loss:
$$f(w) = \big(1 - y_i x_i^\top w\big)^+$$
- Binary logistic loss:
$$f(w) = \log\big(1 + e^{-y_i x_i^\top w}\big)$$

13.1.2 Conditions for a local optimum

13.1.2.1 Local minimum Necessary and sufficient conditions for a local optimum: x^* is a local minimum of $f(x)$ if and only if

1. f has a zero gradient at x^*:
$$\nabla_x f(x^*) = 0$$
2. and the Hessian of f at x^* is positive semi-definite:
$$v^\top \big(\nabla^2 f(x^*)\big) v \geqslant 0 \, , \forall v \in \mathbb{R}^n$$
where
$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \dots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \dots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix}$$

13.1.2.2 Local maximum Similarly, x^* is a local maximum of $f(x)$ if and only if

1. f has a zero gradient at x^*:
$$\nabla_x f(x^*) = 0$$
2. and the Hessian of f at x^* is negative semi-definite:
$$v^\top \big(\nabla^2 f(x^*)\big) v \leqslant 0 \, , \forall v \in \mathbb{R}^n$$

13.1.3 Conditions for constrained local optimum

We explore the condition to decrease the cost function in the case of both an equality constraint and an inequality constraint. We use the notion of steepest descent described in Appendix (13.2.1).

13.1.3.1 The equality constraint At any point x the direction of the steepest descent of the cost function $f(x)$ is given by $\Delta x = -\nabla_x f(x)$. Thus, to move δx from x such that $f(x + \delta x) < f(x)$ we must have

$$\delta x \cdot \big(-\nabla_x f(x)\big) > 0$$

We then consider the conditions to remain on the constraint surface: Normals to the constraint surface are given by $\nabla_x h(x)$. Note, the direction of the normal is arbitrary as the constraint can be imposed as either $h(x) = 0$ or $-h(x) = 0$. To move a small δx from x and remain on the constraint surface we need to move in a direction which is orthogonal to $\nabla_x h(x)$.
More generally, for x_F a feasible point, we consider the case

$$\nabla_x f(x_F) = \nu \nabla_x h(x_F)$$

where ν is a scalar. It occurs if δx is orthogonal to $\nabla_x h(x_F)$, so that

$$\delta x \cdot \big(-\nabla_x f(x_F)\big) = -\delta x \cdot \nu \nabla_x h(x_F) = 0$$

So a constrained local optimum occurs at x^* when $\nabla_x f(x^*)$ and $\nabla_x h(x^*)$ are parallel, that is,

$$\nabla_x f(x^*) = \nu \nabla_x h(x^*)$$

13.1.3.2 The inequality constraint Again, x_F denotes a feasible point. The necessary and sufficient conditions for a constrained local minimum are the same as for an unconstrained local minimum. However, we observe two cases

1. the constraint is not active at the local minimum ($g(x^*) < 0$): therefore the local minimum is identified by the same conditions as in the unconstrained case.

2. the constrained local minimum occurs on the surface of the constraint surface: thus, we have an optimisation problem with an equality constraint $(g(x) = 0)$. A local optimum occurs when $\nabla_x f(x)$ and $\nabla_x g(x)$ are parallel

$$-\nabla_x f(x) = \lambda \nabla_x g(x)$$

However, for some points, we do not have a local minimum since $-\nabla_x f(x_F)$ points in towards the feasible region. Thus, the constrained local minimum occurs when $-\nabla_x f(x)$ and $\Delta_x g(x)$ point in the same direction

$$-\nabla_x f(x) = \lambda \nabla_x g(x) \text{ and } \lambda > 0$$

To summarise, if x^* corresponds to a constrained local minimum, then

1. unconstrained local minimum occurs in the feasible region
   (a) $g(x^*) < 0$
   (b) $\nabla_x f(x^*) = 0$
   (c) $\nabla_{xx} f(x^*)$ is a positive semi-definite matrix

2. unconstrained local minimum lies outside the feasible region
   (a) $g(x^*) = 0$
   (b) $-\nabla_x f(x^*) = \lambda \nabla_x g(x^*)$ with $\lambda > 0$
   (c) $v^\top \nabla_{xx} L(x^*) v \geqslant 0$ for all v orthogonal to $\nabla_x g(x^*)$

13.1.4 Conditions for a global optimum

Convexity plays a key role in mathematical programming. Convex programs minimise convex optimisation functions subject to convex constraints ensuring that every local minimum is always a global minimum. The second order convexity conditions are defined as follows:

Theorem 13.1 *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable. Then f is convex if and only if for all $x \in \text{dom} f$,*

$$\nabla^2 f(x) \geqslant 0$$

In general, we want the optimisation problem to be convex. We define a convex optimisation problem as follow:

Definition 13.1 *An optimisation problem is convex if its objective is a convex function, the inequality constraint g is convex and the equality constraint is affine.*

$$\min_x f(x) \textit{ (convex function) such that } g(x) \leqslant 0 \textit{ (convex set) }, h(x) = 0 \textit{ (affine)}$$

Theorem 13.2 *If $\hat{x}$ is a local minimiser of a convex optimisation problem, it is a global minimiser.*

The following theorem emphasises the reasons why we want a convex problem. It applies to smooth functions.

Theorem 13.3 $\nabla f(x) = 0$ *if and only if x is a global minimiser of $f(x)$.*

Proof: For $\nabla f(x) = 0$ we have

$$f(y) \geqslant f(x) + \nabla f^\top(x)(y - x) = f(x)$$

For $\nabla f(x) \neq 0$ there is a direction of descent.

13.1.5 Some solutions

When dealing with constrained optimisation problem we want to transform the constraint problem to one of the following:

- a series of unconstrained problems
- a single but larger unconstrained problem
- another simpler constraint problem (dual, convex)

Some of the existing solutions are

1. Penalty and barriers
   - Associate a (adaptive) penalty cost with violation of the constraint
   - Associate an additional force compensating the gradient into the constraint (augmented Lagrangian)
   - Associate a log barrier with a constraint, becoming ∞ for violation (interior point method)
2. Gradient projection methods (mostly for linear constraints)
   - For active constraints, project the step direction to become tangantial
   - When checking a step, always pull it back to the feasible region
3. Lagrangian and dual methods
   - Rewrite the constrained problem into an unconstrained one
   - Or rewrite it as a (convex) dual problem
4. Simplex methods (linear constraints)
   - Walk along the constraint boundaries

13.2 Unconstrained optimisation

13.2.1 The gradient descent

One of the simplest approach for solving convex unconstrained optimisation problem is the Gradient Descent. The gradient descent search determines a vector x minimising $f(x)$ by starting with an arbitrary initial vector, and then repeatedly modifying it in small steps. At each step, the vector is altered in the direction that produces the steepest descent along the error surface, until a global minimum error is reached. This direction is found by computing the derivative of f with respect to each component of the vector x, called the gradient of f with respect to x, given by

$$\nabla f(x) = [\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}]$$

***Remark* 13.1** *The gradient specifies the direction that produces the steepest increase in f, and its negative produces the steepest decrease.*

The Gradient Descent is described in Algorithm (15).

Algorithm 15 Gradient Descent

```
1: for $t = 1$ to $T$ do
2:     $x_{t+1} \leftarrow x_t + \Delta x_t$ where $\Delta x_t = -\eta_t \nabla f(x_t)$
3: end for
```

In that setting t is the time step and η_t is the step size or learning rate. There are several ways for choosing the step size:

- Exact line search
$$\eta_t = \arg\min_{\eta} f\big(x - \eta \nabla f(x)\big)$$
- Backtracking line search. Let $\alpha \in (0, \frac{1}{2})$, $\beta \in (0,1)$. Multiply $\eta = \beta \eta$ until
$$f\big(x - \eta \nabla f(x)\big) \leqslant f(x) - \alpha \eta \|\nabla f(x)\|^2$$

13.2.2 The subgradient descent

The concepts of subderivative and subdifferential can be generalised to functions of several variables.

Definition 13.2 *Subgradient*
If $f : U \to \mathbb{R}$ is a real-valued convex function defined on a convex open set in the Euclidean space $\mathbb{R}^n$, a vector v in that space is called a subgradient at a point x_0 in U if for any x in U one has

$$f(x) - f(x_0) \geqslant v \cdot (x - x_0)$$

The set of all subgradients at x_0 is called the subdifferential at x_0 and is denoted $\partial f(x_0)$. The subdifferential is always a nonempty convex compact set.
We can simplify the Gradient Descent. Subgradient methods are iterative methods for solving convex minimisation problems (see Shor [1985]). They are convergent even when applied to a non-differentiable objective function. When the objective function is differentiable, sub-gradient methods for unconstrained problems use the same search direction as the method of steepest descent. See Algorithm (16).

Algorithm 16 Subgradient Descent

```
1: for $t = 1$ to $T$ do
2:     $x_{t+1} \leftarrow x_t - \eta_t \psi_t$
3: end for
```

where η_t is the step size and $\psi_t \in \partial f(x_t)$ denotes a subgradient of f at x_t and x_t is the t-th iterate of x. If f is differentiable, then its only subgradient is the gradient vector ∇f itself.
Many different types of step-size rules are used by subgradient methods. We consider methods with proof of convergence, where the step-sizes are determined off-line, before the method is iterated such as

- Constant step size: $\eta_t = \eta$
- Constant step length: $\eta_t = \frac{\gamma}{\|\psi_t\|_2}$ which gives $\|x_{t+1} - x_t\|_2 = \gamma$.
- Square summable but not summable step size, that is, any step sizes satisfying
$$\eta_t \geqslant 0 \; , \; \sum_{t=1}^{\infty} \eta_t^2 \leqslant \infty \; , \; \sum_{t=1}^{\infty} \eta_t = \infty$$
- Nonsummable diminishing, that is, any step sizes satisfying
$$\eta_t \geqslant 0 \; , \; \lim_{t \to \infty} \eta_t = 0 \; , \; \sum_{t=1}^{\infty} \eta_t = \infty$$

Some examples of non-differentiable convex functions used in machine learning are

$$f(x) = \big(1 - a^\top x\big)^+ \ , \ f(x) = \|x\|_1 \ , \ f(x) = \sum_{r=1}^k \sigma_r(x)$$

where σ_r is the rth singular value of x.

13.2.3 The Newton's method

The idea behind the Newton's method is to use a second-order approximation to the function

$$f(x + \Delta x) \approx f(x) + \nabla f^\top(x)\Delta x + \frac{1}{2}\Delta x^\top \nabla^2 f(x) \Delta x$$

Choose Δx to minimise the above equation, we get

$$\Delta x = -\big(\nabla^2 f(x)\big)^{-1} \Delta f(x)$$

We get the descent direction

$$\nabla f^\top(x)\Delta x = -\nabla f^\top(x)\big(\nabla^2 f(x)\big)^{-1}\Delta f(x) < 0$$

13.2.4 The convergence

The convergence rate is as follows:

- Strongly convex case: $\nabla^2 f(x) \geqslant mI$, then we have Linear convergence. For some $\gamma \in (0,1)$, $f(x_t) - f(x^*) \leqslant \gamma^t$, $\gamma < 1$. We get
$$f(x_t) - f(x^*) \leqslant \gamma^t \text{ or } t \geqslant \frac{1}{\gamma}\log\frac{1}{\epsilon} \Rightarrow f(x_t) - f(x^*) \leqslant \epsilon$$
- Smooth case: $\|\nabla f(x) - \nabla f(y)\| \leqslant C\|x - y\|$. We get
$$f(x_t) - f(x^*) \leqslant \frac{K}{t^2}$$
- Newton's method is often faster, especially when f has long valleys.

Note, inverting a Hessian is very expensive $O(d^3)$. One can approximate the inverse Hessian with BFGS or Limited-memory BFGS. We can also use the Conjugate Gradient Descent. For unconstrained non-convex problems, theses methods will find local optima.

13.2.5 Adding constraints

13.2.5.1 The projected subgradient descent One extension is the projected subgradient descent (PSD) which solves a constrained optimisation problem (see Lemarechal [2001]). Subgradient-projection methods are suitable for convex minimisation problems with very large number of dimensions, since they require little storage. Given a convex constraint set X we want to minimise $\min_{x \in X} f(x)$. We do subgradient steps and project x_t back into X at every iteration

$$x_{t+1} = \prod_X \big(x_t - \eta \psi_t\big)$$

where $\prod_X$ is a projection on X and ψ_t is any subgradient of f at x_t. Sketch of proof:

$$\| \prod_X (x_t) - x^* \| \leqslant \| x_t - x^* \|$$

if $x^* \in X$.
Any decreasing, non-summable step size $\eta_t \to 0$, $\sum_{t=1}^{\infty} \eta_t = \infty$ gives

$$f(x_{avg(t)}) - f(x^*) \to 0$$

Further, with $\eta_t \propto \frac{1}{\sqrt{t}}$, we get

$$f(x_{avg(t)}) - f(x^*) \leqslant \frac{C}{\sqrt{t}}$$

13.2.5.2 The subgradient descent The subgradient method can be extended to solve the inequality constrained problem

$$\min f(x) \text{ such that } g_i(x) \leqslant 0 \, , \, i = 1, ..., m$$

where $g_i(\cdot)$, $i = 1, ..., m$, are convex functions. The algorithm takes the same form as the unconstrained case in Algorithm (16). However, ψ_t is a subgradient of the objective or one of the constraint functions at x. That is,

$$\psi_t = \begin{cases} \partial f(x) \text{ if } g_i(x) \leqslant 0 \forall i = 1, ..., m \\ \partial g_j(x) \text{ for some } j \text{ such that } g_j(x) > 0 \end{cases}$$

where ∂f denotes the subdifferential of f. If the current point is feasible, the algorithm uses an objective subgradient, while if the current point is infeasible, the algorithm chooses a subgradient of any violated constraint.

13.2.5.3 The Newton method The simpler case is Linear constraints $Ax = b$. For example, in the case of Newton method, we consider $f(x + \Delta x) - f(x)$ and minimise

$$\min_{\Delta x} \nabla f^\top(x) \Delta x + \frac{1}{2} \Delta x^\top \nabla^2 f(x) \Delta x \text{ such that } A \Delta x = 0$$

The solution Δx satisfies

$$A(x + \Delta x) = Ax + A\Delta x = b$$

In the case of inequality constraints we get

$$\min_{\Delta x} \nabla f^\top(x) \Delta x + \frac{1}{2} \Delta x^\top \nabla^2 f(x) \Delta x \text{ such that } g_i(x + \Delta x) \leqslant 0$$

13.2.6 Optimisation for machine learning

As discussed in Appendix (11.1.1), given the samples (x_i, y_i), $i = 1, 2, ..., m$, of m pairs of input vector x_i and output vectors y_i, the goal of machine learning is to minimise the expected loss (also called risk function)

$$\hat{L}(h) = E[L(h(x), y)]$$

The learning can be understood as finding a mapping $h(\cdot, w)$ such that $h(x_i, w) \approx y_i$ with w an N_w dimensional vector of parameters to be learned. However, we do not know $P(X, Y)$ and can not estimate it. One solution is to consider Empirical Risk minimisation:

- we substitute the sample mean for the expectation
- minimise empirical loss:
$$\overline{L}(h)=\frac{1}{m}\sum_{i=1}^{m}L(h(x_i),y_i)$$
- A.K.A. sample average approximation

Mathematically, we want to find

$$w^*=\arg\min_w \overline{L}(w)$$

where $\overline{L}(w)=\frac{1}{m}\sum_{i=1}^{m}L(w;x_i,y_i))$ approximates the expected value of the loss. We can also write the loss function as $L(h(x_i,w),y_i)$.

13.2.6.1 The batch gradient descent We can use batch gradient descent: we minimise the empirical loss, assuming it is convex and unconstrained. The gradient descent on the empirical loss is described in Algorithm (17).

Algorithm 17 Gradient Descent on the Empirical Loss

```
1: repeat
2:     $w_{k+1} \leftarrow w_k - \eta_t\big(\frac{1}{m}\sum_{i=1}^{m}\frac{\partial L(w;x_i,y_i)}{\partial w}\big)$
3: until convergence or $k=K$
```

At each step, the gradient is the average of the gradient for all samples $i=1,...,m$. Thus, it is very slow when m is large.

13.2.6.2 The stochastic gradient descent Alternatively, we can compute the stochastic gradient descent (SGD) from just one sample (or a few samples). The algorithm is described in Algorithm (18).

Algorithm 18 Stochastic Gradient Descent on the Empirical Loss

```
1: repeat
2:     $w_{k+1} \leftarrow w_k - \eta_t\frac{\partial L(w;x_i,y_i)}{\partial w}$
3: until convergence or $k=K$
```

Choose one sample i and compute the gradient for that sample only. Note, the gradient of one random sample is not the gradient of the objective function. However, the SGD converges to the empirical loss minimum as well as to the expected loss minimum. Nonetheless, convergence to the expected loss is slow

$$f(w_t)-E[f(w^*)]\leqslant O(\frac{1}{t})\text{ or }O(\frac{1}{\sqrt{t}})$$

In principle, if the training set is small we consider the batch learning using quasi-Newton or conjugate gradient descent. On the other hand, if the training set is large we consider the stochastic gradient descent. However, the convergence is very sensitive to the learning rate. It needs to be determined by trial and error (model selection or cross-validation).

13.3 A first approach to constrained optimisation

Since equality constraints can be expressed in terms of inequality constraints, we will focus on the latter.

13.3.1 Penalty and barriers

We use the following convention

- A barrier is ∞ for $g(x) > 0$
- A penalty is zero for $g(x) \leqslant 0$ and increases with $g(x) > 0$

We let $I(a) = 1$ if $a > 0$, 0 otherwise. We let $I_{-}(a) = \infty$ if $a > 0$, 0 otherwise. That is, $I_{-}(a) = \infty I(a)$. Further, we let $I_0(a) = \infty$ unless $a = 0$.
We present in Appendix (13.6) a generalisation of the penalty function approach. The penalty function has the general form

$$P(g(x)) = \Psi(g(x))$$

where the function $\Psi(x)$ needs to be defined. We are now going to discuss some of these functions.

13.3.1.1 Log barrier method In the logarithmic barrier method, rather than considering the problem in Equation (13.83), we rewrite the problem as

$$\min_x f(x) + \sum_{i=1}^{m} I_{-}(g_i(x))$$

where $I_{-}(a) = \infty$ if $a > 0$, 0 otherwise. That is, $I_{-}(a) = \infty I(a)$. Note, we get the approximation $I_{-}(a) \approx -\mu \log{(-a)}$ for small μ. Thus, we can rewrite the minimisation problem as

$$\min_x f(x) - \mu \sum_{i=1}^{m} \log(-g_i(x))$$

Some of the consequences of this re-formulation are:

- For $\mu \to 0$ then $-\mu \log(-g)$ converges to $\infty I(g > 0) = I_{-}(g > 0)$
- The barrier gradient $\nabla - \log(-g) = \frac{\nabla g}{g}$ pushes away from the constraint
- Eventually we want to have a very small μ, but doing so makes the barrier very non-smooth, which is bad for gradient and 2nd order methods

13.3.1.2 Central path In this approach, every μ defines a different optimal $x^*(\mu)$ given by

$$x^*(\mu) = \arg\min_x f(x) - \mu \sum_{i=1}^{m} \log(-g_i(x))$$

As a result, each point on the path can be understood as the optimal compromise of minimising $f(x)$ and a repelling force of the constraints (which corresponds to the dual variables $\lambda^*(\mu)$). Detail is given in the Algorithm (19).

Algorithm 19 Log Barrier method

```
Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances $\theta$, $\epsilon$
Require: Output: $x$
Require: Initialise $\mu = 1$
1: repeat
2:     find $x \leftarrow \arg\min_x f(x) - \mu \sum_{i=1}^{m} \log(-g_i(x))$ with tolerance $10\theta$
3:     decrease $\mu \leftarrow \frac{\mu}{10}$
4: until $|\Delta x| < \theta$ and $\forall i : g_i(x) < \epsilon$
```

13.3.1.3 Squared penalty method In the Squared penalty method, rather than considering the problem in Equation (13.83), we rewrite the problem as

$$\min_x f(x) + \mu \sum_{i=1}^m I(g_i(x) > 0) g_i^2(x)$$

Note, this method will always lead to some violation of constraints. A better idea would be to add an out-pushing gradient/force $-\nabla g_i(x)$ for every constraint $g_i(x) > 0$ that is violated. Ideally, the out-pushing gradient mixes with $-\nabla f(x)$ exactly such that the result becomes tangential to the constraint. This idea leads to the augmented Lagrangian approach. Detail is given in the Algorithm (20).

Algorithm 20 Squared Penalty method

```
Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances $\theta$, $\epsilon$
Require: Output: $x$
Require: Initialise $\mu = 1$
1: repeat
2:     find $x \leftarrow \arg\min_x f(x) + \mu \sum_{i=1}^m I(g_i(x) > 0) g_i^2(x)$ with tolerance $10\theta$
3:     decrease $\mu \leftarrow \frac{\mu}{10}$
4: until $|\Delta x| < \theta$ and $\forall i : g_i(x) < \epsilon$
```

13.3.2 The augmented Lagrangian approach

13.3.2.1 The equality constraint We first consider an equality constraint in Equation (13.83) before addressing inequalities which we rewrite as

$$\min_x f(x) + \mu \sum_{i=1}^m h_i^2(x) + \sum_{i=1}^m \lambda_i h_i(x)$$

Some of the consequences of this re-formulation are:

- The gradient $\nabla h_i(x)$ is always orthogonal to the constraint
- By tuning λ_i we can induce a virtual gradient $\lambda_i \nabla h_i(x)$
- The term $\mu \sum_{i=1}^m h_i^2(x)$ penalises the function as before

We first minimise the above equation for some μ and λ_i, which will lead to a (slight) penalty $\mu \sum_{i=1}^m h_i^2(x)$. For the next iteration, we choose λ_i to generate exactly the gradient that was previously generated by the penalty. The optimality condition after an iteration is

$$x^{'} = \arg\min_x f(x) + \mu \sum_{i=1}^m h_i^2(x) + \sum_{i=1}^m \lambda_i h_i(x)$$

which gives

$$0 = \nabla f(x^{'}) + \mu \sum_{i=1}^m 2 h_i(x^{'}) \nabla h_i(x^{'}) + \sum_{i=1}^m \lambda_i \nabla h_i(x^{'})$$

We then update the λ for the next iteration

$$\begin{aligned}\sum_{i=1} \lambda_{i,new} \nabla h_i(x^{'}) &= \mu \sum_{i=1}^{m} 2h_i(x^{'}) \nabla h_i(x^{'}) + \sum_{i=1} \lambda_{i,old} \nabla h_i(x^{'}) \\ \lambda_{i,new} &= \lambda_{i,old} + 2\mu h_i(x^{'})\end{aligned}$$

Detail is given in the Algorithm (21). As a consequence

- We do not have to take the penalty limit $\mu \to \infty$ but still can have exact constraints
- If f and h were linear (∇f and ∇h_i constant), the updated λ_i would be exactly right. That is, in the next iteration we would exactly hit the constraint (by construction)
- The penalty term is like a measuring device for the necessary virtual gradient, which is generated by the augmentation term in the next iteration
- The λ_i are meaningful: they give the force/gradient that a constraint exerts on the solution

Algorithm 21 Augmented Lagrangian Equality method

```
Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances $\theta$, $\epsilon$
Require: Output: $x$
Require: Initialise $\mu = 1$, $\lambda_i = 0$
1: repeat
2:   find $x \leftarrow \arg\min_x f(x) + \mu \sum_{i=1}^m h_i^2(x) + \sum_{i=1}^m \lambda_i h_i(x)$
3:   $\forall i : \lambda_i \leftarrow \lambda_i + 2\mu h_i(x^{'})$
4: until $|\Delta x| < \theta$ and $|h_i(x)| < \epsilon$
```

13.3.2.2 The inequality constraint We then consider the inequality constraint in Equation (13.83), which we rewrite as

$$\min_x f(x) + \mu \sum_{i=1}^{m} I\big(g_i(x) \geqslant 0 \vee \lambda_1(0)\big) g_i^2(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

Note the λ_i are zero or positive but never negative. A constraint is either active or inactive:

- When active $\big(g_i(x) \geqslant 0 \vee \lambda_1(0)\big)$ we aim for equality $g_i(x) = 0$
- When inactive $\big(g_i(x) \geqslant 0 \wedge \lambda_1(0)\big)$ we do not penalise / augment equality $g_i(x) = 0$

Detail is given in the Algorithm (22).

Algorithm 22 Augmented Lagrangian Inequality method

```
Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances $\theta$, $\epsilon$
Require: Output: $x$
1: Initialise $\mu = 1$, $\lambda_i = 0$
2: repeat
3:   find $x \leftarrow \arg\min_x f(x) + \mu \sum_{i=1}^m I\big(g_i(x) \geqslant 0 \vee \lambda_1(0)\big) g_i^2(x) + \sum_{i=1}^m \lambda_i g_i(x)$
4:   $\forall i : \lambda_i \leftarrow \max\big(\lambda_i + 2\mu h_i(x^{'}), 0\big)$
5: until $|\Delta x| < \theta$ and $g_i(x) < \epsilon$
```

13.4 The Lagrangian approach

13.4.1 The KKT conditions

Given a constraint problem in Equation (13.83), we define the Lagrangian as

$$L(x,\lambda,\nu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{l} \nu_j h_j(x) \tag{13.85}$$

where $\lambda_i \geqslant 0$, $\nu_j \in \mathbb{R}$ are called dual variables or Lagrange multipliers. We will focus on the case

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

The Lagrangian is useful when computing optima analytically. It implies the Karush-Kuhn-Tucker (KKT) conditions of optimality (see Kuhn et al. [1951]). That is, the KKT conditions encode the conditions for local minima described in Appendix (13.1.3). Note, the optima are necessarily at saddle points of the Lagrangian. The Lagrangian implies a dual problem, which is sometimes easier to solve than the primal one.
As discussed in Appendix (13.1.3), at the optimum there must be a balance between the cost gradient $-\nabla f(x)$ and the gradient of the active constraints $-\nabla g_i(x)$. Formally, for optimal $x : \nabla f(x) \in \text{Span}\{\nabla g_i(x)\}$ or for optimal x, there must exist λ_i such that

$$-\nabla f(x) = \sum_{i=1}^{m} (-\lambda_i \nabla g_i(x))$$

For optimal x, the Karush-Kuhn-Tucker conditions must hold (necessary condition): $\exists \lambda$ such that

$$\begin{aligned}
\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) &= 0 \text{ force balance (or stationarity)} \\
\forall i : g_i(x) &\leqslant 0 \text{ primal feasibility} \\
\forall i : \lambda_i &\geqslant 0 \text{ dual feasibility} \\
\forall i : \lambda_i g_i(x) &= 0 \text{ complementary} \\
\text{plus positive definite constraints on } \nabla_{xx} L(x,\lambda) &
\end{aligned}$$

The first condition can be equivalently expressed as $\exists \lambda$ such that

$$\nabla_x L(x,\lambda) = 0$$

The KKT conditions imply

1. Case 1: inactive constraint
   - when $\lambda^* = 0$ then we have $L(x^*,\lambda^*) = f(x^*)$
   - condition KKT [1] $\Rightarrow \nabla_x f(x^*) = 0$
   - condition KKT [2] $\Rightarrow x^*$ is a feasible point
2. case 2: active constraint
   - when $\lambda^* > 0$ then we have $L(x^*,\lambda^*) = f(x^*) + \lambda^* g(x^*)$
   - condition KKT [1] $\Rightarrow \nabla_x f(x^*) = -\lambda^* \nabla_x g(x^*)$

- condition KKT [4] $\Rightarrow g(x^*) = 0$
- condition KKT [4] also implies $\Rightarrow L(x^*, \lambda^*) = f(x^*)$

Thus, the Lagrangian can be viewed as the energy function that generates (for good choice of λ) the right balance between cost and constraint gradients. Note, this is like in the augmented Lagrangian approach, where however we have an additional augmented squared penalty that is used to tune the λ_i.
In the case of multiple equality and inequality constraints, the Karush-Kuhn-Tucker conditions must hold (necessary condition): $\exists \lambda$ such that

$$
\begin{aligned}
\nabla_x L(x, \lambda, \nu) &= 0 \text{ force balance (or stationarity)} \\
\forall i : g_i(x) &\leqslant 0 \text{ primal feasibility} \\
\forall i : \lambda_i &\geqslant 0 \text{ dual feasibility} \\
\forall i : \lambda_i g_i(x) &= 0 \text{ complementary slackness} \\
h(x) &= 0 \text{ primal feasibility} \\
\text{plus positive definite constraints on } \nabla_{xx} L(x, \lambda)
\end{aligned}
$$

***Remark* 13.2** *Concerning the stationarity condition: for a differentiable function f, we can not use $\partial f(x) = \{\nabla f(x)\}$ unless f is convex.*

13.4.2 Implication from the KKT conditions

13.4.2.1 Equality constraints In the case of the equality, the Lagrangian is given by

$$
L(x, \nu) = f(x) + \sum_{j=1}^{l} \nu_j h_j(x) = f(x) + \nu^\top h(x)
$$

Given the solution in Appendix (13.1.3), we get the following relations

- $\min_x L(x, \nu) \Rightarrow 0 = \nabla_x L(x, \nu) \longleftrightarrow$ force balance
- $\max_\nu L(x, \nu) \Rightarrow 0 = \nabla_\nu L(x, \nu) = h_i(x) \longleftrightarrow$ constraint

As a result, the optima (x^*, ν^*) are saddle points where

- $\nabla_x L = 0$ ensures force balance, and
- $\nabla_\nu L = 0$ ensures the constraint

13.4.2.2 Inequality constraints In the case of the inequality, the Lagrangian is given by

$$
L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) = f(x) + \lambda^\top g(x)
$$

and we get

$$
\max_{\lambda \geqslant 0} L(x, \lambda) = \begin{cases} f(x) \text{ if } g(x) \leqslant 0 \\ \infty \text{ otherwise} \end{cases}
$$

and

$$\max_{\lambda_i \geqslant 0} L(x,\lambda) = \begin{cases} \lambda_i = 0 \text{ if } g(x) < 0 \\ 0 = \nabla_{\lambda_i} L(x,\lambda) = g_i(0) \text{ otherwise} \end{cases}$$

It implies either $\big(\lambda_i = 0 \wedge g_i(x) < 0\big)$ or $g_i(0) = 0$, which is exactly equivalent to the KKT conditions. Again, the optima (x^*, λ^*) are saddle points where

- $\min_x L$ enforces force balance, and
- $\max_\lambda L$ enforces the KKT conditions

13.4.3 The Lagrangian dual problem

13.4.3.1 Definition We define the Lagrange dual function as

$$l(\lambda,\nu) = \inf_x L(x,\lambda,\nu)$$

which implies two problems:

1. primal problem: $\min_x f(x)$ such that $g(x) \leqslant 0$ and $h(x) = 0$
2. dual problem: $\max_{\lambda,\nu} l(\lambda,\nu)$ such that $\lambda \geqslant 0$

Note, the dual problem is convex, even if the primal problem is non-convex. That is, $l(\cdot,\cdot)$ is concave. Thus, we can rewrite the two problems as

1. primal problem: $\min_x\big[\sup_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu)\big]$
2. dual problem: $\max_{\lambda \geqslant 0,\nu}\big[\inf_x L(x,\lambda,\nu)\big]$

The primal problem is equivalent to a min-max optimisation because $\max_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu)$ ensures the constraints. For example, consider a two-player game. If player 1 chooses x that violates a constraint $g_1(x) > 0$, player 2 choose $\lambda_1 \to \infty$ so that $L(x,\lambda,\nu) = \cdots + \lambda_1 f_1(x) + \cdots \to \infty$. Therefore, player 1 is forced to satisfy the constraints.

13.4.3.2 Relation between primal and dual solutions We now discuss the relation between primal and dual solutions in the case of inequality constraints. For any $\lambda_i \geqslant 0$, the dual function is always a lower bound

$$l(\lambda) = \min_x L(x,\lambda) \leqslant \min_x f(x) \text{ such that } g(x) \leqslant 0$$

and consequently

$$\max_{\lambda \geqslant 0}\big[\min_x L(x,\lambda)\big] \leqslant \min_x\big[\max_{\lambda \geqslant 0} L(x,\lambda)\big]$$

Including equality and inequality, we get:

Lemma 13.1 *Weak duality*
If $\lambda \geqslant 0$, then

$$l(\lambda,\nu) \leqslant f(x^*)$$

Proof: We have

$$\begin{aligned} l(\lambda,\nu) &= \inf_x L(x,\lambda,\nu) \leqslant L(x^*,\lambda,\nu) \\ &= f(x^*) + \sum_{i=1}^m \lambda_i g_i(x^*) + \sum_{j=1}^l \nu_j h_j(x^*) \leqslant f(x^*) \end{aligned}$$

Strong duality holds if and only if

$$\max_{\lambda \geqslant 0,\nu} \big[\min_x L(x,\lambda,\nu) \big] = \min_x \big[\max_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu) \big]$$

If the primal is convex, and there exists an interior point, we get the Slater condition

$$\exists x : \forall i : g_i(x) < 0 \text{ and } \forall j : h_j(x) = 0$$

then we have strong duality. That is, primal and dual solutions are equivalent.

Theorem 13.4 *Strong duality*
For reasonable convex problems,

$$\sup_{\lambda \geqslant 0,\nu} l(\lambda,\nu) = f(x^*)$$

13.4.3.3 Interpretation We can interpret duality as a linear approximation. We let $I_-(a) = \infty$ if $a > 0$ and 0 otherwise. We let $I_0(a) = \infty$ unless $a = 0$. We can rewrite the problem as

$$\min_x f(x) + \sum_{i=1}^m I_-(g_i(x)) + \sum_{j=1}^l I_0(h_j(x))$$

We replace $I_-(g_i(x))$ with $\lambda_i g_i(x)$, a measure of displeasure when $\lambda_i \geqslant 0$, $g_i(x) > 0$. Further, $\nu_j h_j(x)$ is a lower bound for $I_0(h_j(x))$. Thus, we get

$$\min_x f(x) + \sum_{i=1}^m \lambda_i g_i(x) + \sum_{j=1}^l \nu_j h_j(x)$$

As an example, we consider the linearly constrained least squares. The problem is as follows:

$$\min_x \frac{1}{2} \|Ax - b\|^2 \text{ such that } Bx = d$$

From the Lagrangian, we get

$$L(x,\nu) = \frac{1}{2} \|Ax - b\|^2 + \nu^\top (Bx - d)$$

Take the infimum, we get

$$\nabla_x L(x,\nu) = A^\top A x - A^\top b + B^\top \nu \Rightarrow x = (A^\top A)^{-1} \big(A^\top b - B^\top \nu \big)$$

Replace the value of x and we get a simple unconstrained quadratic problem

$$\inf_x L(x,\nu) = \frac{1}{2} \| A \big[(A^\top A)^{-1} \big(A^\top b - B^\top \nu \big) \big] - b \|^2 + \nu^\top \Big(B \big[(A^\top A)^{-1} \big(A^\top b - B^\top \nu \big) \big] - d \Big)$$

13.4.3.4 Summary When we have a constrained optimisation problem which is hard to solve, we might consider the dual problem since it may have simpler constraints. Further the solution of the latter is also the solution of the former.
Given the primal feasible x and the dual feasible λ and ν, the quantity

$$f(x) - l(\lambda, \nu)$$

is called the duality gap between x and λ, ν. We get a certificate of optimality: if we have a feasible x and know the dual $l(\lambda, \nu)$, then

$$\begin{aligned} l(\lambda, \nu) \leqslant f(x^*) \leqslant f(x) \quad & \Rightarrow \quad f(x^*) - f(x) \geqslant l(\lambda, \nu) - f(x) \\ & \Rightarrow \quad f(x) - f(x^*) \leqslant f(x) - l(\lambda, \nu) \end{aligned}$$

Thus, if the duality gap is zero, then x is primal optimal (and λ, ν are dual optimal). That is, all these inequalities are actually equalities. It provides a stopping criterion for a numerical implementation: if $f(x) - l(\lambda, \nu) \leqslant \epsilon$, then we are guaranteed that $f(x) - f(x^*) \leqslant \epsilon$.
We now introduce necessity and sufficiency for an optimum solution:

- Necessity states that if x^* and λ^*, ν^* are primal and dual solutions, with zero duality gap, then x^*, λ^*, ν^* satisfy the KKT conditions.
- Sufficiency states that if there exists x^*, λ^*, ν^* that satisfy the KKT conditions, then

$$\begin{aligned} l(\lambda^*, \nu^*) &= f(x^*) + \sum_{i=1}^{m} \lambda_i^* g_i(*) + \sum_{j=1}^{l} \nu_j^* h_j(*) \\ &= f(x^*) \end{aligned}$$

  where the first equality holds from stationarity, and the second holds from complementary slackness. Therefore, duality gap is zero so that x^* and λ^*, ν^* are primal and dual optimal.

In summary, KKT conditions are

- always sufficient
- necessary under strong duality

One of the most important uses of duality is that, under strong duality, we can characterise primal solutions from dual solutions. That is, for a problem with strong duality (assume Slater's condition) x^* and λ^*, ν^* are primal and dual solutions $\Longleftrightarrow$ x^* and λ^*, ν^* satisfy the KKT conditions.

If $\min_x L(x, \lambda, \nu)$ can be solved analytically, so does the convex dual problem. More generally, we have

$$\text{Optimisation problem} \rightarrow \text{Solve KKT conditions}$$

Thus, we can apply standard algorithms for solving an equation system $r(x, \lambda) = 0$ such as the Newton method

$$\nabla r \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -r$$

It leads to primal-dual algorithms that adapt x and λ concurrently. They use the curvature $\nabla^2 f$ to estimate the right λ to push out of the constraint.

13.4.3.5 Equivalence In general, one switch back and forth between the constrained form

$$\min_x f(x) \text{ such that } g(x) \leqslant d \text{ (C)}$$

where $d \in \mathbb{R}$ is a tuning parameter, to the Lagrange form

$$\min_x f(x) + \lambda \cdot g(x) \text{ (L)}$$

where $\lambda \geqslant 0$ is a tuning parameter, and claim the two methods are equivalent. We get two cases

1. (C) to (L): if the problem (C) is strictly feasible, then strong duality holds, and there exists some $\lambda \geqslant 0$ (dual solution) such that any solution x^* in (C) minimises

$$f(x) + \lambda \cdot \big(f(x) - d\big)$$

so that x^* is also a solution in (L).

2. (L) to (C): if x^* is a solution in (L), then the KKT conditions for (C) are satisfied by taking $d = g(x^*)$ so that x^* is a solution in (C).

Thus, we get the equivalence

$$\begin{aligned} \bigcup_{\lambda \geqslant 0} \{ \text{ solutions in (L) } \} &\subseteq \bigcup_{d} \{ \text{ solutions in (C) } \} \\ \bigcup_{\lambda \geqslant 0} \{ \text{ solutions in (L) } \} &\supseteq \bigcup_{d \text{ such that (C) is strictly feasible}} \{ \text{ solutions in (C) } \} \end{aligned}$$

Note, this is not a perfect equivalence (albeit minor non-equivalence).
Using the KKT conditions and simple probability arguments, we get the following result:

Theorem 13.5 *Let f be differentiable and strictly convex, $A \in \mathbb{R}^{n \times p}$, $\lambda > 0$. Consider*

$$\min_x f(Ax) + \lambda \|x\|_1$$

If the entries of A are drawn from a continuous probability distribution (on $\mathbb{R}^{n \times p}$), then with probability 1, the solution $x^* \in \mathbb{R}^p$ is unique and has at most $\min\{n,p\}$ nonzero components.

Note, while f must be strictly convex, there is no restrictions on the dimensions of A and one could choose $p >> n$.

13.4.4 Additional

13.4.4.1 The log barrier method revisited Going back to the log barrier method in Appendix (13.3.1.1), for a given μ, the optimality condition is

$$\nabla f(x) - \sum_{i=1}^m \frac{\mu}{g_i(x)} \nabla g_i(x) = 0$$

or equivalently, we get the modified (approximated) KKT conditions

$$\nabla f(x) - \sum_{i=1}^m \lambda_i \nabla g_i(x) = 0 \, , \, \lambda_i g_i(x) = -\mu$$

We see that centring in the log barrier method is equivalent to solving the modified KKT conditions. In addition, on the central path, the duality gap is μ:

$$l(\lambda^*(\mu)) = f(x^*(\mu)) + \sum_i \lambda_i g_i(x^*(\mu)) = f(x^*(\mu)) - \mu$$

13.4.4.2 Non-convex optimisation We consider the optimisation problem with equality constraint

$$\min_x f(x) \text{ such that } h(x) = 0$$

where both $f(\cdot)$ and $h(\cdot)$ are neither convex nor linear. Given the Lagrangian

$$L(x,\nu) = f(x) + \nu^\top h(x)$$

we get the min-max problem

$$\min_x \max_\nu L(x,\nu)$$

The idea is to let x_0 be an initial value of x and compute the increment dx and the value of ν by solving a linearised version of the above problem, replacing x by $x + dx$, and iterates. If the function $f(\cdot)$ is differentiable, the increment dx is normally computed at each iteration by solving the linear system

$$\begin{pmatrix} \eta I & \nabla_x h^\top \\ \nabla_x h & 0 \end{pmatrix} \begin{pmatrix} dx \\ \nu \end{pmatrix} = \begin{pmatrix} -\nabla_x f \\ -h(x) \end{pmatrix}$$

which amounts to projected gradient descent. The projection is performed onto the hyperplanes of linearised constraints. This system is derived from the KKT conditions that dx and ν must satisfy.
Note, when the function $f(\cdot)$ is a sum of squared residuals (typical in regression problems), one can replace the projected gradient by Gauss-Newton (GN) or Levenberg-Marquardt (LM) (see Fua et al. [2010]). The iteration scheme remains the same but the KTT conditions become

$$\begin{pmatrix} J^\top J + \eta I & \nabla_x h^\top \\ \nabla_x h & 0 \end{pmatrix} \begin{pmatrix} dx \\ \nu \end{pmatrix} = \begin{pmatrix} -J^\top r(x_t) \\ -h(x_t) \end{pmatrix}$$

where r is the vector of residuals and $J = \nabla_x r$ is its Jacobian matrix evaluated at x_t.

13.5 Standard convex programs

There exist a large variety of mathematical programs, each being a different research area in itself with extensive theory and algorithms. Given the constraint problem in Equation (13.83), we review a few basic convex optimisation programming models and present some examples. See Nocedal et al. [1999] for details. See Press et al [1992] for numerical implementation.

13.5.1 Simple programming models

13.5.1.1 Quadratic programming Quadratic programming is used in problems minimising least squares loss function. A quadratic program (QP) has a quadratic objective with linear constraints. The problem is stated as

$$\min_x \frac{1}{2} x^\top Q x + C^\top x \text{ such that } a_i x \leqslant b_i \ , \ i \in I \ , \ a_j x = b_j \ , \ j \in \varepsilon$$

where the Hessian matrix Q is $n \times n$ symmetric, I and ε are finite sets of indices and a_i, $i \in I \bigcup \varepsilon$ are $n \times 1$ vectors. As discussed above, if the matrix Q is positive, $x^\top Q x \geqslant 0$ for any x, then the problem is convex. For convex QP, any local solution is also a global solution. A QP can always be solved or shown to be infeasible in a finite number of iterations. AS discussed in Appendix (13.4), the KKT optimal conditions are

1. $Qx + \sum_{i \in I} \lambda_i a_i + \sum_{j \in \varepsilon} \nu_j a_j = 0$ Dual Feasibility
2. $a_i^\top x \leqslant b_i$ for $i \in I$ Primal Feasibility
3. $a_j^\top x = b_j$ for $j \in \varepsilon$ Primal Feasibility
4. $\lambda_i \big(a_i^\top x - b_i \big) = 0$ for $i \in I$ Complementary

In the special case where there are no constraints ($I = 0$), then the KKT point can be found by simply solving a system of linear equations. It is more difficult in presence of inequality constraints. Two methods exists

1. interior point methods
2. active-set methods: the optimal active set is the set of constraints satisfied as equalities at the optimal solutions. Active set methods work by making educated guesses as to the active set and solving the resulting equality constrained QP. In the case of wrong guess, one can use gradient and Lagrangian multiplier information to determine constraints to add to or subtract from the active set.

13.5.1.2 Linear programming Linear programming optimises a linear function subject to linear constraints. Since linear functions and constraints are convex, a linear program (LP) is always a convex program. Linear programming is a special case of the QP above, with the Hessian Q equal to zero. The problem is stated as

$$\min_x C^\top x \text{ such that } a_i x \leqslant b_i \; , \; i \in I \; , \; a_j x = b_j \; , \; j \in \varepsilon$$

In general, one uses interior point methods and simplex methods (active set methods) to solve LP problems.

13.5.1.3 Second-order cone programming The second-order cone programming (SOCP) problems have a linear objective, second-order cone constraints, and possibly additional linear constraints. The problem is stated as

$$\min_x C^\top x \text{ such that } \|R_i x + d_i\|_2 \leqslant a_i x + b_i \; , \; i \in I \; , \; a_j x = b_j \; , \; j \in \varepsilon$$

where $R_i \in \mathbb{R}^{n_i \times n}$ and $d_i \in \mathbb{R}^{n_i}$. These problems are often solved by using interior point algorithms.

13.5.1.4 Semidefinite programming Semidefinite programs (SDP) are the generalisation of linear programs (LP) to matrices. In standard form, a SDP minimises a linear function of a matrix subject to linear equality constraints and a matrix non-negativity constraint. The problem is stated as

$$\min_X < C, X > \text{ such that } < A_i, X >= b_i \; , \; i \in I \; , \; X \succeq 0$$

where X, C and A_i take values in $\mathbb{R}^{n \times n}$ and $b_i \in \mathbb{R}$. Further, $X \succeq 0$ means X must be positive semidefinite and $< C, X >= \text{trace}(CX)$. In general, SDP are solved via interior programming methods (see Mittelmann [2003]).

13.5.2 Some examples

13.5.2.1 Quadratic programming As an example we consider a quadratic function with equality constraints. Given $Q \geqslant 0$, the optimisation problem is

$$\min_x \frac{1}{2} x^\top Q x + c^\top x \text{ such that } Ax = 0$$

It is a convex problem with no inequality constraints so that by the KKT conditions x is a solution if and only if

$$\begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -c \\ 0 \end{pmatrix}$$

for some λ. Linear system combines stationarity and primal feasibility (complementary slackness and dual feasibility are vacuous).

13.5.2.2 The Lasso problem Another example is the Lasso problem: given the response $y \in \mathbb{R}^n$, the predictors $A \in \mathbb{R}^{n \times p}$ (columns $A_1, ..., A_p$), the optimisation problem is

$$\min_{x \in \mathbb{R}^p} \frac{1}{2} \|y - Ax\|^2 + \nu \|x\|_1$$

The KKT conditions are

$$A^\top (y - Ax) = \lambda s$$

where $s \in \partial \|x\|_1$, that is,

$$s_i \in \begin{cases} \{1\} \text{ if } x_i > 0 \\ \{-1\} \text{ if } x_i < 0 \\ (-1, 1] \text{ if } x_i = 0 \end{cases}$$

Note, if $|A_i^\top (y - Ax)| < \nu$ then $x_i = 0$.

13.6 Generalising the penalty function approach

13.6.1 The energy function

As already seen above, the penalty function method consists of transforming the constrained optimisation problem in Equation (13.83) (but with $g_i(x) \geqslant 0$) into an unconstrained one based on a penalty function (see Luenberger [1984]). That is, the penalty function $E(\cdot, k)$ satisfies

$$E(x, k) = \alpha f(x) + \sum_{i=1}^m k_i P(g_i(x))$$

where $\alpha = +1$ for minimisation problems and $\alpha = -1$ for maximisation problems. The penalty term $P(g_i(x))$ should satisfy

$$\begin{aligned} P(g_i(x)) &= 0 \text{ if } g_i(x) \geqslant 0 \text{ the corresponding constraint is satisfied} \\ P(g_i(x)) &> 0 \text{ if } g_i(x) < 0 \end{aligned}$$

The constants k_i, $i = 1, ..., m$, define the relative weight concerning the satisfaction of some constraints against some other and/or the relative weight of satisfying all the constraints or minimising $sf(x)$. The task of tuning these parameters provides a means of customising the problem according to the current needs.
As discussed in Appendix (11.1.3), the function $E(\cdot, k)$ is also called the energy function for the corresponding network (see Cichocki et al. [1993]).
Assuming differentiability, a local minimum of the penalty function is obtained by using the dynamic gradient scheme

$$\frac{dx}{dt} = -\mu \nabla_x E(x, k) \; , \; x(0) = x^{(0)}$$

where t is the time step and

$$\mu = \text{diag}\big(\mu_1, \mu_2, ..., \mu_n\big)$$

or in extended form

$$\begin{aligned}
\frac{dx_1}{dt} &= -\mu_1\Big(\frac{\partial f(x)}{\partial x_1} + \sum_{i=1}^m k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_1}\Big) \, , x_1(0) = x_1^{(0)} \\
\frac{dx_2}{dt} &= -\mu_2\Big(\frac{\partial f(x)}{\partial x_2} + \sum_{i=1}^m k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_2}\Big) \, , x_2(0) = x_2^{(0)} \\
&\vdots \\
\frac{dx_n}{dt} &= -\mu_n\Big(\frac{\partial f(x)}{\partial x_n} + \sum_{i=1}^m k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_n}\Big) \, , x_n(0) = x_n^{(0)}
\end{aligned}$$

where $\mu_j > 0$ and $k_i > 0$. Usually one takes $\mu_j = \mu = \frac{1}{\tau}$, $j = 1, ..., m$, where τ is a time constant, and $k_i = k$, $i = 1, ..., n$.

13.6.2 The penalty functions

The penalty function has the general form

$$P(g(x)) = \Psi(g(x))$$

where the function $\Psi(x)$ should penalise only configurations where $x > 0$. One possibility is to use a sigmoid function or

$$\Psi(x) = x\Theta(x)$$

where $\Theta(x)$ is a Heaviside function (see Ohlsson [1993]). The penalty term is zero if and only if the constraint is satisfied, otherwise the penalty is proportional (linear) to the degree of violation. The slope of $\Psi(x)$ is implicitly given by the strength of the constraint k in the energy function.
Some examples of penalty terms are

$$\begin{aligned}
P(g_i(x)) &= \big[\min(0, g_i(x))\big]^2 \\
P(g_i(x)) &= -\min(0, g_i(x))
\end{aligned}$$

13.6.2.1 Type I In the first case, the energy function becomes

$$E(x,k) = f(x) + \frac{1}{2}\sum_{i=1}^m k_i\big[\min(0, g_i(x))\big]^2$$

Assuming that $g_i(x)$, $i = 1, ..., m$, have continuous first derivatives, one can show that the same is true for $P(g_i(x))$, so that the energy function is continuously differentiable. The gradient of $P(g_i(x))$ is

$$\nabla_x\big[\min(0, g_i(x))\big]^2 = \min(0, g_i(x))\nabla_x g_i(x)$$

Using the gradient strategy for minimising the energy function $E(\cdot, k)$, we get the system of ordinary differential equations

$$\frac{dx_j}{dt} = -\mu_j \Big(\frac{\partial f(x)}{\partial x_j} + \sum_{i=1}^m k_i S_i g_i(x) \frac{\partial g_i(x)}{\partial x_j} \Big) \ , j = 1, ..., n$$

where the control signals are

$$S_i = \begin{cases} 1 \text{ if } g_i(x) \leqslant 0 \\ 0 \text{ if } g_i(x) > 0 \end{cases}$$

and

$$S_i g_i(x) = \min(0, g_i(x))$$

Kennedy et al. [1988] and Cichocki et al. [1993] showed that the functional scheme for simulating these equations could be considered as a neural network (NN), where the integrators represent the neurons (basic units) and functional nonlinear generators build up the connections between them. This approach replaces the constrained problem in Equation (13.83) by an unconstrained minimisation of the differentiable penalty function $E(\cdot, k)$. Luenberger [1984] explained that theoretical results on the penalty function method show that equivalence of the problems in Equation (13.83) and $\min_x E(x, k)$ is only obtained in the limit, as

$$\min_{i=1,...,m} \{k_i\} \to \infty$$

Since the values of the parameters k_i, $i = 1, ..., m$, used in a NN are finite, it follows that the unconstrained minima of the energy function obtained by the NN will only be approximations to the true solutions of the constrained problem.

13.6.2.2 Type II We now consider the case of the second penalty term, the energy function becomes

$$E(x, k) = f(x) - \sum_{i=1}^m k_i \min(0, g_i(x))$$

Due to the non-differentiability of the penalty term, this penalty function is non-differentiable, even though the functions $g_i(x)$ are assumed differentiable. Luenberger [1984] proved that any unconstrained minimum of the above penalty function is also a solution of the constrained problem in Equation (13.83) provided that $\min_{i=1,...,m}\{k_i\}$ is sufficiently large. Thus, an unconstrained minimisation will yield the true solution if k_i, $i = 1, ..., m$ are sufficiently large but finite. This solution is termed exact. In that case, the corresponding system of ordinary differential equations is

$$\frac{dx_j}{dt} = -\mu_j \Big(\frac{\partial f(x)}{\partial x_j} - \sum_{i=1}^m k_i S_i \frac{\partial g_i(x)}{\partial x_j} \Big) \ , j = 1, ..., n$$

where $\mu_j > 0$ and the control signals are

$$S_i = \begin{cases} 1 \text{ if } g_i(x) \leqslant 0 \\ 0 \text{ if } g_i(x) > 0 \end{cases}$$

The functional scheme is a simpler NN than with the previous penalty function using simple switches instead of the expensive analog multipliers. However due the non-smooth nature of the penalty function, the first derivative discontinuities can lead to parasitic effects slowing up the solution speed.

13.6.2.3 Type III Rather than adding together the penalty terms $-k_i \min(0, g_i(x))$, Mladenov et al. [1999] proposed to take the maximum of these terms, getting the penalty (energy) function

$$\begin{aligned} E(x,k) &= f(x) + \max_{i=1,...,m} \big(k_i P(g_i(x)) \big) \\ &= f(x) + \max_{i=1,...,m} \big(\max\big(0, -k_i g_i(x)\big) \big) \\ &= f(x) + \max\big(0, -k_1 g_1(x), -k_2 g_2(x), ..., -k_m g_1(m)\big) \end{aligned}$$

In that setting, only the most violated inequality constraint is taken into account. Note, this system is non-differentiable. Again, it has been shown that it is an exact penalty function (see Polak [1997]).
To apply the gradient strategy, we need to select

$$\frac{\partial x}{\partial t} \in -\mu \partial E(x,k)$$

where $\partial E(x,k)$ is the generalised gradient of the penalty function $E(\cdot,k)$. We use the same system of ordinary differential equations as in Type II except that the control signals S_i are defined as follows:

$$S_i = \begin{cases} 1 \text{ if } \max\big(0, -g_i(x)\big) = \max_{j=1,...,m} \big(\max\big(0, -g_i(x)\big)\big) \\ 0 \text{ if } \max\big(0, -g_i(x)\big) < \max_{j=1,...,m} \big(\max\big(0, -g_i(x)\big)\big) \end{cases}$$

The proposed NN architecture consists in a winner-take-all block included into the Control Network (see Cichocki et al. [1993]). This block is used to determine dynamically the index $i*$ of the most violated inequality and to generate the control signal $S_{i*} = 1$ corresponding to this inequality, while all other control signals (corresponding to other inequalities) are equal to 0. The same non-smooth character of the exact penalty function as in Type II applies.

13.7 Solving optimisation problems with NNs

In general, optimisation problems that can not be solved with an exact algorithm are solved by applying a global search optimisation such as genetic algorithms, particle swarm optimisation, simulated annealing, ant colony optimisation etc. Some authors proposed the use of neural networks (NN) to resolve optimisation problems in those cases where the use of linear programming or Lagrange multipliers is not feasible.

13.7.1 A short review

Artificial neural networks (ANN) have been used to obtain solution of constrained optimisation problems (see Appendix (13)). For instance, Chua et al. [1984] developed the canonical non-linear programming circuit, using the Kuhn-Tucker conditions from mathematical programming theory (see Appendix (13.4)). Hopfield et al. [1985] [1987] published some results on how to go about using neural networks to solve optimisation problems. The Travelling Salesman Problem was the first problem formulated in terms of neural network. The approach was then used to demonstrate how circuits of simple unit can solve hard problems. Tank et al. [1986] developed their optimisation network for solving linear programming problems. Smith et al. [1989] discussed some practical design problems of the Tank and Hopfield (TH) network along with its stability properties. Kennedy et al. [1988] extended the results of TH to more general non-linear programming problems. Further, they showed that the network introduced by TH was a special case of the canonical non-linear programming network proposed Chua et al. [1984] with capacitors added to account for the dynamic behaviour of the circuit. Lillo et al. [1993a] analysed the dynamics of the canonical nonlinear programming circuit and showed that it was a gradient system minimising an unconstrained energy function that can be viewed as a penalty method approximation of the original problem. Thus, all of these methods (Hopfield-type networks) use the penalty function method (see Appendix (13.6.2)) whereby a constrained optimisation problem is approximated by an unconstrained optimisation problem. This is due to the network property of reducing their energy function during evolution, leading to a local or global minimum. Some authors proposed an exact penalty function approach and presented a neural optimisation network for solving constrained optimisation problems (see Lillo et al. [1993b], Mladenov et al. [1999]). The parallel nature of some neural network models seems to be very promising in reducing computation time (see Cichocki et al. [1993], Takefuji et al. [1996]). However, the problem of local minima is the main limitation of the approach and several techniques have been proposed to overcome it, such as stochastic networks, simulated annealing, and other global optimisation methods. Villarrubia et al. [2017] proposed to apply a multilayer perceptron to approximate the objective functions. The same process could be followed in the restrictions. The objective function is approximated with a non-linear regression with the objective of obtaining a new function that facilitates the solution of the optimisation problem. The activation function of the neural network must be selected so that the derivative of the transformed objective functions should be polynomial.

13.7.2 Mapping optimisation problems to networks

Since a Hopfield network (see Appendix (11.1.3)) performs as a minimiser of its energy function, if the optimisation problem can be coded as an energy function, then a network that corresponds to this energy function can be used to minimise (locally or globally) the function and thus provide an optimal or near-optimal solution. Thus, the procedure starts with the construction of the energy function and then the parameters of the network (number of units, weights of connection, thresholds, update policy or even the dynamics) are adjusted to reflect the problem. Then the network is initialised to some initial state and is let to run until it comes to equilibrium from where a solution can be drawn. However, in order to construct the appropriate energy function the latter must be quadratic to satisfy Equation (11.39). The most common approach is the penalty (or cost) function approach (see Appendix (13.6)). The energy function is initialised to the objective function of the problem and for each constraint a penalty term is added. Thus, the problem from a constrained optimisation form is reduced to an unconstrained minimisation problem.
As soon as the energy function has been constructed, the elements of the network (number of units, weights, thresholds) can be derived. For each unit i the energy difference ΔE_i is calculated (derivatives can be used for this purpose) and it is transformed into a form similar to Equation (11.40). By analogy, the coefficient of v_j will give the weight w_{ij}, whereas the constant terms will give the threshold θ_i. Note, as the problem size grows, the network size (units and connections) can be intractable large and inapplicable to practical domains. The limitations are due to storage and computing power requirements, but several techniques for parallel implementations have been proposed. The initialisation of the network state is a problem since if it is initialised to a state corresponding to a possible solution, it will be stuck as all the constraints are satisfied, although this solution may not be optimal. Such states corresponds to local minima and are not desirable as initial states. Moreover, different initialisation may lead to different solutions, with different quality and/or computation time. One solution is to consider a random state for initialisation.

13.7.3 Some optimisation models

We briefly present some simple networks used for optimisation problems.

13.7.3.1 Discrete synchronous Hopfield network In the discrete synchronous Hopfield network, the dynamics of the system are given by Equation (11.37) and all the units are updated in parallel. The network is not guaranteed to decrease at each step and since the system may fail to come into equilibrium a solution is not always derived. However, it has the advantage of being fully parallelisable.

13.7.3.2 Discrete asynchronous Hopfield network This model is similar to the previous one with the difference that the units are updated sequentially. The network will eventually come into equilibrium, from where a solution can be drawn. However, the system can be easily trapped into a local minimum and is highly dependent on the initial state. Further, it can not be parallelised since it is strictly sequential.

13.7.3.3 Analog Hopfield network In the analog Hopfield network, the dynamics of the system are given by Equation (11.37). Note, synchronous, asynchronous or continuous updating can be applied. Although the output of such networks is continuous they can be used to solve discrete optimisation problems. In the equilibrium state the outputs closer to 1 are taken as 1 and the output closer to 0 (-1) are taken as 0 (-1). When using the sigmoid function, if we start the network at the temperature T where we want to measure the outputs, it may take a long time to come to equilibrium. In general, the simulated annealing technique is applied. We start the network at a relatively high temperature and gradually we cool it down. The potential disadvantage of this network is that it may be trapped in local minima inside the hypercube without reaching any of the corners.

13.7.3.4 Boltzmann machine The operation of the Boltzmann machine integrates the dynamics of the discrete asynchronous Hopfield model with the simulated annealing technique. At each step t a unit i of the network is selected randomly and the energy difference ΔE_i that will be caused by a change of its state is calculated using Equation (11.40). If ΔE_i is negative (the energy is decreased) the change is definitely accepted, otherwise it is accepted with probability $P_{B,i}(t)$ that depends on the quantity $e^{\frac{\Delta E_i}{T_B}}$ where T_B is a temperature that decreases according to some annealing schedule. In general, the Metropolis criterion is used as the acceptance criterion, which is given by

$$P_{B,i}(t) = \begin{cases} 1 \text{ if } \Delta E_i(t) < 0 \\ \frac{1}{1+e^{\frac{\Delta E_i}{T_B}}} \text{ if } \Delta E_i(t) \geqslant 0 \end{cases}$$

The logarithmic schedule can be used to decrease the temperature:

$$T_B(t) = \frac{T_B(t-1)}{1 + t\log(1+r)}$$

where r is a parameter that adjusts the speed of the schedule.
The Boltzmann machine can be very effective when used with the appropriate annealling schedule and is able to perform a wide exploration of the problem state space. However, it is strictly sequential and can not be parallelised. Several attempts at parallelising its operation (group updates) were proposed, all having to cope with some kind of trade-off between the solution quality and the actual speed-up.

13.7.3.5 Cauchy machine The Cauchy machine extends the discrete synchronous Hopfield model. The behaviour of a unit i at each time step t is given by

$$v_i(t) = \Theta(u_i(t))$$

where u_i is the activation in Equation (11.38). During the operation, the activation is updated using the following motion equation (gradient descent dynamics)

$$\frac{du_i(t)}{dt} = -\frac{\partial E(v)}{\partial v_i}$$

For simulation purposes, we use the first-order approximation for the above dynamics, getting

$$\frac{\Delta u_i}{\Delta t} = -\frac{\Delta E_i}{\Delta v_i} \text{ and } u_i(t+\Delta t) = u_i(t) + \Delta u_i$$

Note, the activation plays the role of an accumulator, a sort of a memory that stores the cumulative activation of the unit during the network's operation time. Hence, the probability of two units to change state simultaneously is reduced significantly, so that oscillation phenomena are avoided. Consequently, the system will eventually come to equilibrium. This is strengthened also by the fact that each unit follows the above gradient descent dynamics. When the energy function is given by Equation (11.39), then using Equation (11.40) and the above gradient descent dynamics we can derive the motion equation for each unit i as

$$\frac{\Delta u_i}{\Delta t} = \sum_{j=1}^n w_{ij}v_j + \theta_i \text{ and } u_i(t+\Delta t) = u_i(t) + \big(\sum_{j=1}^n w_{ij}v_j + \theta_i\big)\Delta t$$

A state vector v constitutes an equilibrium state for the network if for all i the following condition is satisfied

$$\big(v_i = 1 \text{ and } \Delta u_i \geqslant 0\big) \text{ or } \big(v_i = 0 \text{ and } \Delta u_i \leqslant 0\big)$$

In order to provide the system with stochastic hill-climbing capabilities, the Distributed Cauchy Machine has been developed, where Cauchy color noise is added in the updating procedure. The output of unit i at each time step t is stochastically updated with probability $P_{C,i}(t)$ which depends on the Cauchy distribution:

$$P_{C,i}(t) = \begin{cases} s_i(t) \text{ if } v_i(t) = 0 \\ 1 - s_i(t) \text{ if } v_i(t) = 1 \end{cases}$$

where

$$s_i(t) = P(v_i(t) = 1) = \frac{1}{2} + \frac{1}{\pi} arctan\big(\frac{u_i(t)}{T_C(t)}\big)$$

The virtual temperature $T_C(t)$ is usually given by a fast annealing schedule:

$$T_C(t) = \frac{T_{C,0}}{1 + \beta t}$$

where $T_{C,0}$ is the initial temperature and β is real parameter in the range $[0, 1]$ controlling the speed of the schedule. In the extreme case where $T_C = 0$ the network becomes deterministic.
In general, the Cauchy machine provides solutions of lower quality than the Boltzmann machine, but it has the advantage of being fully parallelisable. Further, since the activation is cumulative, changes at the state of a unit can be done only after many time steps, for the activation must change sign. The lack of this flexibility becomes more obvious when the system has operated for a long time and the activations have large absolute values.

13.7.3.6 Hybrid Scheme Papageorgiou et al. [1998] presented the hybrid update scheme as an attempt at combining both the advantages of the Boltzmann and the Cauchy machine. It extends the synchronous discrete Hopfield model and the stochastic update rule is based on a convex combination of the Boltzmann and Cauchy machine update rules. At each time step t, the probability of accepting a state change concerning unit i is given by

$$P_{H,i}(t) = \alpha P_{C,i}(t) + (1 - \alpha) P_{B,i}(t)$$

where α is a control parameter in the range $[0, 1]$. A large value of α will result to a typical Cauchy machine, whereas a small value will result to a synchronous Boltzmann machine destroying the convergence property. The two temperatures need not be equal and a good choice would be to update $T_C(t)$ according to some annealing schedule and then take $T_B(t) = \lambda T_C(t)$ where λ is an adjustable parameter. Even though the above stochastic update rule accepts changes suggested by the energy difference (through $P_{B,i}(t)$), such changes should be reflected on the activation $u_i(t)$, otherwise at the next time step the unit would return to the previous value, since it is still suggested by the activation. The following rule ensures that this will not happen:

$$\begin{array}{rl} \text{if} & \big(u_i(t + \Delta t) <> u_i(t)\big) \text{ and } \big(P_{C,i}(t) < 0.25\big) \text{ and } \big(P_{B,i}(t) > 0.75\big) \\ \text{then} & \big(u_i(t + \Delta t) = -u_i(t)\big) \end{array}$$

Lagoudakis [1997] showed that $\big(P_{B,i}(t) > 0.75\big)$ was necessary for the network to converge. Experimental results showed that the solutions produced by the hybrid scheme were similar to the solutions produced by the Boltzmann machine but with larger convergence time, similar to that of the Cauchy machine. However, the hybrid scheme is fully parallelisable, so that the parallel implementation provides solutions similar to that of the Boltzmann machine but in substantially less time.

13.7.4 Approximation with non-linear regression

13.7.4.1 The method The main idea is to approximate the objective function f with some heuristics methods. Approximation functions are usually defined around a point, which would make it possible to use polynomials to approximate functions by applying the Taylor theorem. Based on this idea, one can solve non-linear optimisation problems by applying Taylor non-linear functions. For instance, Nanculef et al. [2014] applied this method in Frank-Wolfe algorithms, allowing for the linearisation of the objective functions by applying derivatives in a point to calculate the straight line, plane or hyperplane crosses through that point. The solutions are calculated iteratively with a new hyperplane for each iteration. Note, the MAP (Method of Approximation Programming) is a generalisation of the Frank-Wolfe algorithm, which permits to linearise the restrictions.
Villarrubia et al. [2017] proposed to generalise this approach by using the universal approximation theorem (see Theorem (11.3) and approximate the objective function with neural networks. That is, rather than calculating a new approximation for each tentative solution, they chose to infer the approximation by applying neural networks. Given a known objective function f, the system generates a dataset in the domain of the variables to train a neural network. The objective function of the optimisation problem is redefined with the multilayer perceptron that transforms the function, making it possible to generate a polynomial equation to resolve the optimisation problem. Finally, when the new objective function is calculated another solution (such as the Lagrangian method) can be applied to resolve the problem.
To define a neural network, it is necessary to establish parameters, such as the connections, number of layers, activation functions, propagation rules etc. In the case of the multilayer perceptron, the authors considered two different stages: the learning stage, and the prediction process. In both stages, the number of layers and activation functions have to be the same.

13.7.4.2 Defining the network Following the Hopfield networks (see Appendix (11.1.3)) and multilayer networks (see Appendix (11.1.6)), the propagation rule is the weighted sum (see activation in Equation (11.38)) defined by

$$u_j(t) = \sum_{i=1}^{n} w_{ij} x_i(t) + \theta_j$$

where w_{ij} is the weight connecting neuron i in the input layer with neuron j in the hidden layer , x_i is the output from neuron i in the input layer, n is the number of neurons in the input layers, t is the pattern, and θ_j is the bias.
We now need to select the activation function Φ of the network such that that the derivatives of the transformed objective functions are polynomial. This is because it will simplify calculations when using the Lagrangian method to solve the optimisation problem. If the function Φ is linear, the output of neuron j is a linear combination of the neurons in the input layer and, consequently, y_j is a linear function. Therefore, if the activation function is the identity, the net output would correspond to the output of the neuron. Thus, if neuron j in the hidden layer has the activation function Φ , the output is given by

$$y_j(t) = \Phi(u_j(t))$$

which is equivalent to a non-linear regression.
Since the multilayer perceptron has three layers, we apply the propagation rule twice in order to transmit the value from the input layer to the neurons in the output layer. If neuron k in the output layer has the activation function Φ_{out}, the output is given by

$$y_k(t) = \Phi_{out}\big(\sum_{j=1}^{m} w_{jk} y_j(t) + \theta_k\big)$$

where m is the number of neurons in the hidden layer. In the special case where Φ_{out} is the identity function, the output simplifies to

$$y_k(t) = \sum_{j=1}^{m} w_{jk} y_j(t) + \theta_k$$

Replacing $y_j(t)$ with its value defined above, the output in neuron k is defined as

$$y_k(t) = \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k \tag{13.86}$$

Using the universal approximation theorem (see Theorem (11.3)), we can deduce that there exists $\epsilon > 0$ such that

$$|y_k(t) - f| < \epsilon$$

In the special case where the function Φ is the identity, the output in neuron k from the output layer is calculated as a linear combination of the inputs, so that the function is linear. Thus, if we train the multilayer perceptron with an identity activation function, it would be possible to make an approximation of the trained function. In the case where an optimisation problem has a non-linear objective function, it would then be possible to redefine the function according to Equation (13.86) so that it would be linear. However, with this activation function we can not prevent the approximation functions from becoming hyperplane, so that the activation function Φ can not be linear. Another choice would be to consider the arctan activation function because the equation of the derivative [10] is simpler than the expression of the sigmoidal function.

13.7.4.3 The Lagrangian method In order to solve the optimisation problem defined in Equation (13.84), we can approximate the objective function f with the neural network defined in Equation (13.86), getting

$$f(x) \approx \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k \text{ such that } g_i(x) \leqslant 0 \, , \, i = 1, ..., m$$

We can then use the Lagrangian approach (see Appendix (13.4)) to solve this optimisation problem. In that case the Lagrangian in Equation (13.85) becomes

$$L(x, \lambda) = \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k + \sum_{i=1}^{m} \lambda_i \cdot g_i(x)$$

with the Karush-Kuhn-Tucker (KKT) conditions

$$\begin{aligned} \frac{\partial L}{\partial x_i} &= 0 \, , \, i = 1, ..., n \\ \frac{\partial L}{\partial \lambda_j} &= 0 \, , \, j = 1, ..., m \end{aligned}$$

The stationarity condition can be written as

$$\forall j \;\; \frac{\partial f(x)}{\partial x_j} \pm \sum_{i=1}^{m} \lambda_i \frac{\partial g_i(x)}{\partial x_j} \leqslant 0$$

and the complementary condition is

$$\forall i : \lambda_i g_i(x) = 0$$

[10] $arctan(f) = \frac{f'}{1+f^2}$

We clearly see that we need to use an activation function Φ which is analytically differentiable to simplify the derivatives in order to calculate the solution more easily. For restrictions with inequality, it is possible to introduce a threshold based on the training carried out in the neural network. For the threshold to be lower, it would be best to have previously trained the neural network with values in the variables around their definition. That is, in a problem with restrictions given by

$$\begin{aligned} a_i &\leqslant x_i \leqslant b_i \\ x_i &\geqslant a_i \\ x_i &\leqslant b_i \end{aligned}$$

we should generate a dataset for the training phase that matches the restrictions for the variable x_i. The lower the difference among consecutive values, the lower the error to define the threshold. The authors got the restriction

$$g_s(x) \leqslant 0$$

and given Equation (13.86), they got

$$\sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k + \mu_s = 0$$

Finally, the optimisation problem becomes

$$f(x) = \sum_{j=1}^{m^1} w^1_{jk} \Phi\big(\sum_{i=1}^{n^1} w^1_{ij} x^1_i(t) + \theta^1_j\big) + \theta^1_k$$

such that

$$\begin{aligned} \sum_{j=1}^{m^1} w^1_{jk} \Phi\big(\sum_{i=1}^{n^1} w^1_{ij} x^1_i(t) + \theta^1_j\big) + \theta^1_k + \mu^1_s &= 0 \\ &\vdots \\ \sum_{j=1}^{m^n} w^n_{jk} \Phi\big(\sum_{i=1}^{n^n} w^n_{ij} x^n_i(t) + \theta^n_j\big) + \theta^n_k + \mu^n_s &= 0 \end{aligned}$$

14 Global search optimisation

14.1 Evolutionary algorithms

14.1.1 A brief history of evolutionary algorithms

Evolutionary algorithms (EAs) introduced by Holland [1962] [1975] and Fogel [1966] are robust and efficient optimisation algorithms based on the theory of evolution proposed by Darwin [1882], where a biological population evolves over generations to adapt to an environment by mutation, recombination and selection. They are stochastic search algorithms, searching from multiple points in space instead of moving from a single point like gradient-based methods do. These algorithms are typically initiated with a population of potential solutions, that may be drawn randomly or specified prior to the beginning of the search. Iteration on the population is based on the principles of natural selection, with each iteration, or generation, improving in fitness as defined by some pre-determined measure. The methods by which each generation is determined are specific to the particular algorithm in question, however, all evolutionary algorithms rely on classes of stochastic operator known as selection and reproduction operators. The selection operator acts to ensure that individuals with greater fitness in each generation are selected as parents for the next generation, whilst the reproduction operator determines how the next generation is derived from the parents selected. Moreover, they work on function evaluation alone (fitness) and do not require derivatives or gradients of the objective functions. McKay [2008] claimed that despite their simplicity, given modest resources and a relatively tough optimisation problem, Evolutionary Algorithms reliably converge to good solutions, and what's more, they are suited to parallel implementation due to their speed scaling almost linearly with the number of processors.

There is a large literature describing different evolutionary algorithms (EAs) commonly used to solve constrained nonlinear programming problems (CNOPs) such as evolutionary programming, evolution strategies, genetic algorithms (GAs), differential evolution (DE) and many more. For instance, GAs are general purpose search algorithms based on an evolutionary paradigm where the population members are represented by strings, corresponding to chromosomes. Search starts with a population of randomly selected strings, and, from these, the next generation is created by using genetic operators (mutation). At each iteration individual strings are evaluated with respect to a performance criteria and assigned a fitness value. Strings are randomly selected using these fitness values to either survive or to mate to produce children for the next generation. However, among the different EA's commonly used DE became very popular. DE is a population-based approach to function optimisation generating a new position for an individual by calculating vector differences between other randomly selected members of the population. The DE algorithm is found to be a powerful evolutionary algorithm for global optimisation in many real problems. As a result, since the original article of Storn and Price [1995] many authors improved the DE model to increase the exploration and exploitation capabilities of the DE algorithm when solving optimisation problems.

Since EAs are search engines working in unconstrained search spaces they lacked until recently of a mechanism to deal with the constraints of the problems. The first attempts to handle the constraints were either to incorporate methods from mathematical programming algorithms within EAs such as penalty functions, or, to exploit the mathematical structure of the constraints. Then, a considerable amount of research proposed alternative methods to improve the search of the feasible global optimum solution. Most of the research on DE focused on solving CNOPs by using a sole DE variant, a combination of variants or combining DE with another search method. One of the most popular constraint handling mechanisms is the use of the three feasibility rules proposed by Deb [2000] on genetic algorithms. Using some of the improvements to the DE algorithm combined with simple and robust constraint handling mechanisms we propose a modified algorithm for solving our optimisation problem under constraints which greatly improves its performances.

14.1.2 Introduction to genetic algorithms

GA is an optimization technique that produces optimisation of the problem by using natural evolution based on survival of the fittest. The search space is initialised with a set of solution called chromosome, which is a set of genes. Quality and fitness of a chromosome is measured by fitness function. The behaviour of Genetic algorithm is determined by exploration and exploitation with the help of operators like reproduction (selection), crossover (recombination) and mutation.
The general procedure for genetic algorithm is (see Witten et al. [2005]):

1. START: Generate random population.
2. FITNESS: Evaluate the fitness f(x) of each chromosome x in the population.
3. NEW POPULATION: Create a new population by repeating following steps until the new population is complete
    - REPRODUCTION OR SELECTION: Parents chromosomes are selected from population according to their fitness to crossover and produce new offspring.

- CROSSOVER: crossover operator produce new two offspring from selected two parents based crossover probability.
- MUTATION: Mutation operator produce new offspring by mutate single bit position in chromosome. Mutation used to maintain genetic diversity.

4. ACCEPTING: Place new offspring in the new population.
5. REPLACE: Use new generated population for a further run of the algorithm
6. TEST: If the end condition is satisfied, stop, and return the best solution in current population.
7. LOOP: Go to stop.

Performance of GA depends on various parameters like population size, crossover and mutation rate and computation time. Performance of genetic algorithm can be optimized by parallel genetic algorithm. Parallel Genetic Algorithm (PGA) is an algorithm that works by dividing a large problem into smaller tasks. Genetic algorithm with parallel processing reduces the computation time.
Some of the advantages and disadvantages of GA are:

1. Advantages of GA
   - GA has faster and more efficient as compared to the traditional methods.
   - It has very good parallel capabilities.
   - It optimises both continuous and discrete functions and also multi-objective problem.
   - It is Useful when the search space is very large and there are a large number of parameters involved.
2. Disadvantage of GA
   - Fitness values are calculated repeatedly which might be computationally expensive for some problems.
   - Being stochastic, there is no guarantee of the optimality or the quality of the solution.

When using GAs with classification algorithms, some of the issues and challenges are

- Local Convergence: GA sometimes converge on local optima, incorrect peak populate due to sampling errors. One needs to modify selection pressure to prevent premature convergence.
- Parameter Setting: GA parameters such as selection rate, chromosome length, population size, crossover probability, mutation probability and total number of generations has a large impact on performance of genetic algorithm. Setting these parameters is a complex task.

14.2 Introduction to differential evolution

14.2.1 The DE algorithm

According to Feoktistov [2006], Differential Evolution (DE) is first and foremost an optimisation algorithm, and in particular, one of the most powerful tools for global optimisation, regardless of it simplicity. It is so named because it identifies differences in individuals through the use of a simple and fast linear operator (differentiation), and in doing so, realises the evolution of a population of individuals in some intelligent manner. That is, the main characteristic of DE is an adaptive scaling of step sizes resulting in fast convergence behaviour. The Differential Evolution (DE) proposed by Storn and Price [1995] is an algorithm that can find approximate solutions to nonlinear programming problems. It is a parallel direct search method using NP parameter vectors, where each vector (or individual) is of dimension D equals the number of objective function parameters. The vectors

$$X_{i,G} \text{ for } i = 0, .., NP - 1$$

forms a population for each generation G, like any evolutionary algorithms. The initial vector population is chosen randomly, covering the entire parameter space. The number of vectors NP is a function of the dimension D, such as $NP \in [5D, 10D]$ but NP must be at least 4 to ensure that DE will have enough mutually different vectors to play with (see Storn et al. [1997]). Each of the NP parameter vectors undergoes mutation, recombination and selection.

14.2.1.1 The mutation The role of mutation is to explore the parameter space by expanding the search space giving its name to the DE algorithm. For a given parameter vector $X_{i,G}$ called the Target vector, the DE generates a Donor vector V made of three or more independent parent vectors $X_{r_l,G}$ for $l = 1, 2, ..$ where r_l is an integer chosen randomly from the interval $[0, NP - 1]$ and different from the running index i. In the spirit of Wright [1991], the main idea is to perturbate a Base vector $\hat{V}$ with a weighted difference vector (called differential vectors)

$$V = \hat{V} + F \sum_{l=1} \big(X_{r_{2l-1},G} - X_{r_{2l},G} \big) \tag{14.87}$$

where the mutation factor F is a constant taking values in $[0, 2]$ and scaling the influence of the set of pairs of solutions selected to calculate the mutation value. Most of the time, the Base vector is defined as the arithmetical crossover operator

$$\hat{V} = \lambda X_{best,G} + (1 - \lambda) X_{r_1,G}$$

where $\lambda \in [0, 1]$ allows for a linear combination between the best element $X_{best,G}$ of the parent population vectors and a randomly selected vector $X_{r_1,G}$. It is called a global selection when $\lambda = 1$ while when $\lambda = 0$ the base vector is the same as the target vector, $X_{r_1,G} = X_{i,G}$ and we get a local selection. In the special case where the mutation factor is set to zero, the mutation operator becomes a crossover operator. Figure (53) displays graphically the solution space and how the mutation vector V is realised. Note, the parameter vector X_{r_3} that currently resides outside of the solution space demonstrates how the mutation procedure allows for exploration of alternative solution spaces and therefore, how the algorithm is able to reliably converge to global minima.

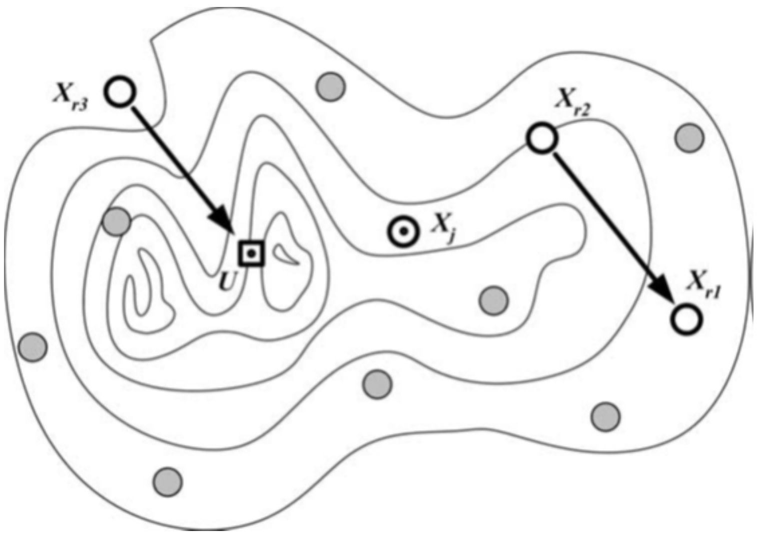


Figure 53: Creation of differential mutation vector. (Feoktistov [2006])

14.2.1.2 The recombination Recombination incorporates successful solutions from the previous generation. That is, according to a rule, we combine elements of the Target vector $X_{i,G}$ with elements of the Donor vector $V_{i,G}$ to create an offspring called the Trial vector $U_{i,G}$. In order to increase the diversity of the parameter vectors, elements of the Donor vector enter the Trial vector with probability CR. In the DE algorithm, each element of the Trial vector satisfies

$$U_{i,G}(j) = \begin{cases} V_{i,G}(j) \text{ for } j = n_r \mod dim, (n_r + 1) \mod dim, ..., (n_r + L - 1) \mod dim \\ X_{i,G}(j) \text{ for all other } j \in [0, .., NP - 1] \end{cases}$$

where $< n_r >_{dim} = n_r \mod dim$ is the modulo of n_r with modulus dim, dim is the dimension of the vector V (here $dim = N$), and the starting index n_r is a randomly chosen integer from the interval $[0, dim - 1]$. Hence, a certain sequence of the element of U is equal to the element of V, while the other elements get the original element of $X_{i,G}$. We only choose a subgroup of parameters for recombination, enhancing the search in parameter space. The integer L denotes the number of parameters that are going to be exchanged and is drawn from the interval $[1, dim]$ with probability

$$P(L > \nu) = (CR)^\nu \, , \nu > 0$$

The random decisions for both n_r and L are made anew at each new generation G. The term $CR \in [0, 1]$ is the crossover factor controlling the influence of the parent in the generation of the offspring. A higher value means less influence from the parent. Generally, values of CR in the range $[0.8, 1]$ lead to good results (see Lampinen et al. [2004]). Most of the time, the mutation operator in Section (14.2.1.1) is sufficient and one can directly set the Trial vector equal to the Donor vector.

14.2.1.3 The selection Unlike the previous two procedures, the selection procedure is typically deterministic under Differential Evolution. The tournament selection only needs part of the whole population to calculate an individual selection probability where subgroups may contain two or more individuals. In the DE algorithm, the selection is deterministic between the parent and the child. The best of them remain in the next population. We compute the objective function with the original vector $X_{i,G}$ and the newly created vector $U_{i,G}$. If the value of the latter is smaller than that of the former, the new Target vector $X_{i,G+1}$ is set to $U_{i,G}$ otherwise $X_{i,G}$ is retained

$$X_{i,G+1} = \begin{cases} U_{i,G} \text{ if } H(U_{i,G}) \leq H(X_{i,G}) \, , i = 0, .., NP - 1 \\ X_{i,G} \text{ otherwise} \end{cases}$$

This method, as applied in many Evolutionary Algorithms, ensures that the population fitness will always increase or remain constant through each generation. Mutation, recombination and selection continue until some stopping criterion is reached. The mutation-selection cycle is similar to the prediction-correction step in the EM algorithm or in the filtering problems. Figure (54) gives a graphical account of the tournament selection process.

14.2.1.4 Simple convergence criteria We allow for different convergence criterion in such a way that if one of them is reached, the algorithm terminates. We let $f^* = f_{min}(G)$ be the fittest design in the population so far, and we define

$$f_{a,G} = \frac{1}{NP} \sum_{i=0}^{NP-1} f(X_{i,G})$$

as the average objective value at generation G. The variance of the objective value at generation G is given by

$$f_{v,G} = \frac{1}{NP} \sum_{i=0}^{NP-1} \big(f(X_{i,G}) - f_{a,G} \big)^2 = \frac{1}{NP} \sum_{i=0}^{NP-1} f^2(X_{i,G}) - f_{a,G}^2$$

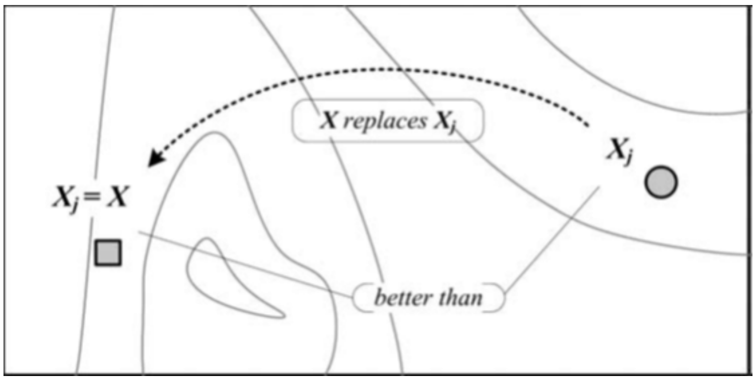


Figure 54: Tournament selection. (Feoktistov, [[2006]])

Then, when the percentage difference between the average value and the best design reaches a specified small value ϵ_1

$$\frac{|f_{a,G} - f_{min}(G)|}{|f_{a,G}|} \times 100 \leqslant \epsilon_1$$

we terminate the algorithm. Also, we let $f_{min}(G-1)$ be the fittest design in the previous generation $(G-1)$, and consider as a criterion the difference

$$|f_{min}(G-1) - f_{min}(G)| < \epsilon_2$$

where ϵ_2 is user defined. In that case, the DE algorithm will continue until there is no appreciable improvement in the minimum fitness value or some predefined maximum number of iterations is reached.

14.2.2 Pseudocode

We now present the pseudo code of a standard DE algorithm.

```
Initialise vectors of the population NP
Evaluate the cost of each vector
    for i=0 to Gmax do
        repeat
            Select some distinct vectors randomly
            Perform mutation
            Perform recombination
            Perform selection
                if offspring is better than main parent then
                replace main parent in the population
                end if
        until population is completed
        Apply convergence criterions
    next i
```

14.2.3 The strategies

Over the years, Storn and Price as well as a large number of other authors made improvement to the DE model so that there are now many different DE models (see Price et al. [2005], Storn [2008]). These models vary in the type of recombination operator used as well as in the number and type of solutions used to calculate the mutation values. We are now going to list the main schemes perturbating the base vector for mutation.

14.2.3.1 Scheme DE1 To improve convergence on a set of optimisation problems using the Scheme DE4 introduced in Section (14.2.3.4), Storn and Price considered for each vector $X_{i,G}$ a trial vector V generated according to the rule

$$V = X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)$$

where the integer r_l for $l = 1, 2, 3$ are chosen randomly in the interval $[0, NP - 1]$ and are different from the running index i. Also, F is a scalar controlling the amplification of the differential variation $X_{r_2,G} - X_{r_3,G}$. We can slightly modify the scheme by writing

$$V = X_{r_1,G} + F\big(X_{r_2,G} + X_{r_3,G} - X_{r_4,G} - X_{r_5,G}\big)$$

More generally, the base vector can be expressed as a linear combination of other distinct vectors, as in Equation (14.87). As a simple rule, the differential weight F is usually in the range $[0.5, 1]$ (see Lampinen et al. [2004]). The population size should be between $3N$ and $10N$ and generally we increase NP if misconvergence happens. In the case where NP is increased we should decrease F.

14.2.3.2 Scheme DE2 Similarly to the Scheme DE1 in Section (14.2.3.1), the trial vector V is generated according to the rule

$$V = X_{i,G} + \lambda\big(X_{best,G} - X_{i,G}\big) + F\big(X_{r_2,G} - X_{r_3,G}\big)$$

where λ is a scalar controlling the amplification of the differential variation $X_{best,G} - X_{i,G}$. It enhances the greediness of the scheme by introducing the current best vector $X_{best,G}$. It is useful for non-critical objective functions, that is, when the global minimum is relatively easy to find. It gives a balance between robustness and fast convergence.

14.2.3.3 Scheme DE3 In the same spirit, Mezura-Montes et al. [2006] also modified the Scheme DE1 in Section (14.2.3.1) by incorporating information of the best solution as well as information of the current parent in the current population to define the new search direction.

$$V = X_{r_3,G} + \lambda\big(X_{best,G} - X_{r_2,G}\big) + F\big(X_{i,G} - X_{r_1,G}\big)$$

where λ is a scalar controlling the amplification of the differential variation $X_{best,G} - X_{r_2,G}$. This scheme has the same properties as the Scheme DE2 in Section (14.2.3.2).

14.2.3.4 Scheme DE4 The oldest strategy developed by Storn and Price [1997] is for the trial vector V to be generated according to the rule

$$V = X_{best,G} + F\big(X_{r_1,G} - X_{r_2,G}\big)$$

where the weight F is a scalar. However, in that setting they found several optimisation problems where misconvergence occurred. To improve the convergence, Price et al. [2005] proposed to perturb the differential weight F by introducing the Dither and Jitter schemes. In the former, Karaboga et al. [2004] randomised the weight as follow

$$\lambda_G = F_l + U_G(0,1)\big(F_u - F_l\big)$$

where F_l is a lower weight, F_u is an upper weight, and $U_G(0,1)$ is uniformly distributed in the range $[0,1]$ for each generation G. In the latter, the random weight is picked for each element $j = 0, ..., D-1$ of the vector being updated

$$\lambda_j = F \times \left[1 + \delta \times (U_j(0,1) - \frac{1}{2})\right]$$

where δ determines the scale of perturbation. It must be small, and is usually set to $\delta = 0.0001$. For example, we can consider the strategy

$$V = X_{best,G} + \lambda_j \left(X_{r_1,G} - X_{r_2,G}\right)$$

It is a jitter which add fluctuation to the random target. The jitter is the time variation of a periodic signal in electronics and telecommunications (swing dancer). It is tailored for small population sizes and fast convergence. We can also modify the scheme by doing

$$V = X_{best,G} + \lambda_j \left(X_{r_1,G} + X_{r_2,G} - X_{r_3,G} - X_{r_4,G}\right)$$

Going one step further, Storn [2000] combined the jitter scheme with the dither one, getting

$$\lambda_{j,G} = \Big(F_l + U_G(0,1)\left(F_u - F_l\right)\Big)\left[1 + \delta \times (U_j(0,1) - \frac{1}{2})\right]$$

14.2.3.5 Scheme DE5 Das et al. [2005] improved the DE's convergence by applying the dither scheme to every difference vector. Similarly to the Scheme DE1 in Section (14.2.3.1), the trial vector V is generated according to the rule

$$V = X_{r_1,G} + \lambda_{d,i}\left(X_{r_2,G} - X_{r_3,G}\right)$$

where $\lambda_{d,i}$ is a computer dithering factor

$$\lambda_{d,i} = F + U_i(0,1) \times (1 - F) \, , i = 0, ..., NP - 1$$

also written

$$\lambda_{d,i} = F_l + U_i(0,1)\left(F_u - F_l\right) \, , i = 0, ..., NP - 1$$

where $U_i(0,1)$ is uniformly distributed in the range $[0,1]$. It is a per-vector dither, making the Scheme DE1 more robust. Note, as discussed in Section (14.2.3.4) we can also have

$$\lambda_{d,G} = F + U_G(0,1) \times (1 - F)$$

which is a per-generation dither factor. In that algorithm, choosing $F = 0.3$ is a good start. As explained by Pedersen [2010], the dither and jitter schemes are similar, except that the dither draws a random weight once for each agent-vector to be updated and the jitter draws a random weight for each element of that vector. To see this, we simply set $F_l = F \times (1 - \frac{\delta}{2})$ and $F_u = F \times (1 + \frac{\delta}{2})$. The dither can simply be rewritten as

$$\lambda_{d,i} \sim U_i(F_l, F_u) \, , i = 0, ..., NP - 1$$

where $U_i(F_l, F_u)$ is uniformly distributed in the range $[F_l, F_u]$, and the jitter can simply be rewritten as

$$\lambda_j \sim U_j(F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2})) \, , j = 0, ..., D - 1$$

where $U_i(F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2}))$ is uniformly distributed in the range $[F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2})]$. Defining a midpoint F_{mid} and a range F_{range}, the dither becomes

$$\lambda_{d,i} \sim U_i(F_{mid} - F_{range}, F_{mid} + F_{range}) \, , i = 0, ..., NP - 1$$

and the jitter becomes

$$\lambda_j \sim U_j(F_{mid} - F_{range}, F_{mid} + F_{range}) \, , j = 0, ..., D-1$$

We therefore need to select two parameters, F_{mid} and F_{range}, to determine the limits of perturbation for the weight F. Pedersen chose the midpoint in the interval $F_{mid} \in [0,2]$ and the range in the interval $F_{range} \in [0,3]$ allowing for negative differential weights to occur. Thus, perturbing behavioural parameters introduce new parameters in the form of the boundaries for the stochastic sampling ranges. Further, when the behavioural parameters are completely random, it takes statistically a lot of samples before finding an optimum.

14.2.3.6 Scheme DE6 A more complex algorithm than the Scheme DE1 in Section (14.2.3.1) is to introduce the choice between two strategies. In the either-or algorithm, the trial vector V is generated according to the rule

$$V = \begin{cases} \Big(X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big) I_{(\delta < \frac{1}{2})} \\ \Big(X_{r_1,G} + \frac{1}{2}(\lambda + 1)\big(X_{r_2,G} + X_{r_3,G} - 2X_{r_1,G}\big)\Big) I_{(\delta \geq \frac{1}{2})} \end{cases}$$

where δ is as above. It alternates between differential mutation and three-point recombination.

14.2.3.7 Scheme DE7 Alternatively, we have the two possible strategies using an either-or algorithm. If we favor the r_1 event we have

$$V = \Big(X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big) I_{(\delta < P_e)}$$

while if we favor the best event we have

$$V = \Big(X_{best,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big) I_{(\delta < P_e)}$$

It alternates between differential mutation and doing nothing.

14.2.3.8 Scheme DE8 Still another more complex algorithm than the Scheme DE1 in Section (14.2.3.1) is to introduce two strategies V_l for $l = 1,2$. We first build the Base vector $\hat{V}_1$ as the linear combination of the two original base vector $X_{r_1,G}$ and $X_{best,G}$, getting

$$\hat{V}_1 = \lambda_{M,G} X_{best,G} + \overline{\lambda}_{M,G} X_{r_1,G}$$

where $\overline{\lambda}_{M,G} = 1 - \lambda_{M,G}$. This time $\lambda_{M,G}$ is a Gaussian variable per-generation with mean μ and variance ξ to be defined. The second Base vector $\hat{V}_2$ is generated according to the rule

$$\hat{V}_2 = 2X_{best,G} - \hat{V}_1 = (2 - \lambda_{M,G}) X_{best,G} - \overline{\lambda}_{M,G} X_{r_1,G}$$

The two Trial vectors V_i for $i = 1,2$ becomes

$$\begin{aligned} V_1 &= \hat{V}_1 + F_b\big(X_{r_3,G} - X_{r_2,G}\big) \\ V_2 &= \hat{V}_2 + F_b\big(X_{r_2,G} - X_{r_3,G}\big) = 2X_{best,G} - V_1 \end{aligned}$$

where F_b is a Gaussian variable with mean $\mu = 0$ and variance $\xi = F$. In the special case where $\mu = 2$ and $\xi = 0$ we get

$$\begin{aligned} \hat{V}_1 &= 2X_{best,G} - X_{r_1,G} \\ \hat{V}_2 &= X_{r_1,G} \end{aligned}$$

and the system simplifies to

$$\begin{aligned} V_1 &= 2X_{best,G} - X_{r_1,G} + F_b\big(X_{r_3,G} - X_{r_2,G}\big) \\ V_2 &= X_{r_1,G} + F_b\big(X_{r_2,G} - X_{r_3,G}\big) \end{aligned}$$

and V_2 recover a pseudo Secheme DE1 where $F_b \in [-F, F]$.

14.2.4 Improvements

The DE algorithm is found to be a powerful evolutionary algorithm for global optimisation in many real problems. As the DE algorithm performs mutation based on the distribution of the solutions in a given population, search directions and possible step sizes depend on the location of the individuals selected to calculate the mutation values. As a result, since the original article of Storn and Price [1995] many authors improved the DE model to increase the exploration and exploitation capabilities of the DE algorithm when solving optimisation problems. We are going to review a few changes to the DE algorithm which greatly improved the performances of our problem.

14.2.4.1 The tuning parameters The tuning of the DE is mainly contolled by three variables: the number of vector (or individual) NP, the differential weight F, and the crossover factor CR. Finding bounds for their values has been a topic of intensive research. Zaharie [2002] studied theoretically the convergence of the DE by analysing the behavioural parameters of the DE models. She proved that the mutation scale factor F should never be smaller than F_{crit} defined as

$$F_{crit} = \sqrt{\frac{(1 - \frac{CR}{2})}{NP}}$$

Price et al. [2005] showed that only high values of CR guarantee the contour matching properties of DE. Further, only when $CR = 1$ is the mean number of function evaluation for an objective function and its rotated counterpart the same. In that setting, the DE is called rotationally invariant. As a rule of thumb (see Storn et al. [1997]), we get

- $F \in [0.5, 1.0]$
- $CR \in [0.8, 1.0]$
- $NP = 10D$

but they lack generality. Hence, the need to compute these parameters automatically.

14.2.4.2 Ageing The DE selection is based on local competition only. The number of children that may be produced to compete against the parent $X_{i,G}$ should be chosen sufficiently high so that a sufficient number of child will enter the new population. Otherwise, it would lead to survival of too many old population vectors that may induce stagnation. To prevent the vector $X_{i,G}$ from surviving indefinitely, Storn [1996] used the concept of ageing. One can define how many generations a population vector may survive before it has to be replaced due to excessive age. If the vector $X_{i,G}$ is younger than Num generations it remains unaltered otherwise it is replaced by the vector $X_{r_3,G}$ with $r_3 \neq i$ being a randomly chosen integer in $[0, NP - 1]$.

14.2.4.3 Constraints on parameters Commonly, we are searching for a solution to an optimisation problem between certain bounds. Given the parent vector $X_{i,G}$ for $i = 0, .., NP - 1$ we define upper and lower bounds for each initial parameters as

$$L(j) \leqslant X_{i,G_0}(j) \leqslant U(j) \ , j = 0, ..., D - 1$$

where G_0 is inception, and we randomly select the initial parameter values uniformly on the interval $[L(j), U(j)]$ as

$$X_{i,G_0}(j) = L(j) + U_j(0,1)\big(U(j) - L(j)\big) \text{ , } j = 0, ..., D-1$$

where $U_j(0,1)$ generates a random number in the range $[0,1]$ with a uniform distribution for each element j of the vector. Obviously, as the number of generation G increases, the DE algorithm will generate elements of the vector outside of the limits established (lower and upper) by an amount. Several alternatives exist for handling boundary constraints (see Onwubolu [2004]). Following Mezura-Montes et al. [2004a], this amount is substracted or added to the limit violated (reflecting barrier), in order to shift the value inside the limits. Should this cause the new value to violate the other bound, a random value will be generated, as when creating the initial parameter set. We can also set the element of the vector half way between the old position and the limit as follow

$$U_{i,G+1}(j) = \begin{cases} \frac{1}{2}(X_{i,G}(j) + U(j)) \text{ if } U_{i,G+1}(j) > U(j) \\ \frac{1}{2}(X_{i,G}(j) + L(j)) \text{ if } U_{i,G+1}(j) < L(j) \\ U_{i,G+1}(j) \text{ otherwise} \end{cases}$$

where $U_{i,G+1}(j)$ is the new Trial vector.

14.2.4.4 Convergence In order to accelerate the convergence process, when a child replaces its parent, Mezura-Montes et al. [2004a] copied its value both into the new generation and into the current generation. It allows the new child, which is a new and better solution, to be selected among the r_l solutions and create better solutions. Therefore, a promising solution does not need to wait for the next generation to share its genetic code. Similarly, to improve performance and to accelerate the convergence process, Storn [1996] explored the idea of allowing a solution to generate more than one offspring. Once a child is better than its parent, the multiple offspring generation ends. Following the same idea, Coello Coello and Mezura-Montes [2003] and then Mezura-Montes et al. [2006] allowed for each parent at each generation to generate $k > 0$ offspring. Among these newly generated solutions, the best of them is selected to compete against its parent, increasing the chances to generate fitter offspring.

14.2.4.5 Self-adaptive parameters It was proved that key control parameters in the DE algorithm, such as the crossover CR and the weight applied to random differential F, should be altered in the evolution process itself (see Liu et al. [2002]). These parameters can be adjusted by using heuristic rules, or they can be self-adapted (see Liu et al. [2005], Brest et al. [2006], Balamurugan et al. [2007]). That is, the control parameters are not required to be pre-defined and can change during the evolution process. These control parameters are applied at the individual levels in the population, such that better values should lead to better individuals producing better offspring and hence better values. However, in general, the random change to the parameters are applied irrespective of the quality of the current parameters. Noman et al. [2011] proposed to preserve better parameter choices, while changing the non-productive ones. We first describe the algorithm proposed by Brest et al. [2006] (jDE), and then introduce the adaptative algorithm of Noman et al. [2011] (aDE). The parameter F is a scaling factor controlling the amplification of the difference between two individuals to avoid search stagnation. At generation $G = 1$ the amplification factor $F_{i,G}$ for the ith individual ($i = 0, ..., NP - 1$) is generated randomly in the range $[0.1, 1.0]$. Then, at the next generations the control parameter is given by

$$F_{i,G+1} = \begin{cases} F_L + r_1 \times F_U \text{ if } r_2 < \tau_1 \\ F_{i,G} \text{ otherwise} \end{cases}$$

where r_j, for $j = 1, 2$ are uniform random values in $[0,1]$, $F_L = 0.1$, $F_U = 0.9$ and τ_1 represent the probability to adjust the parameter F. Using the notation in Section (14.2.3.5), we can rewrite the scaling factor as

$$F_{i,G+1} = \begin{cases} U_i(F_l, F_u) \text{ if } r_2 < \tau_1 \\ F_{i,G} \text{ otherwise} \end{cases}$$

where $F_l = 0.1$ and $F_u = F_L + F_U = 1.0$. The only difference with the dither is the fact that the randomness of the differential weight depends on a probability of adjustment. According to Feoktistiv [2006], at the beginning of the evolution procedure, the mutation step length, and hence $F_{i,G}$, should be large, as individuals are far away from each other and the procedure could benefit from exploring beyond the current solution space. Since the individuals of a generation converge for subsequent generations, the step length, and hence $F_{i,G}$ should become smaller to allow a more concentrated search around the successful solution space. Thus, by applying custom parameters at individual levels, better values in the population lead to better individuals producing better Donor vectors and so on. Similarly, we can extend the crossover parameter CR as follow

$$CR_{i,G+1} = \begin{cases} r_3 \text{ if } r_4 < \tau_2 \\ CR_{i,G} \text{ otherwise} \end{cases}$$

where r_j, for $j = 3, 4$ are uniform random values in $[0, 1]$ and τ_2 represent the probability to adjust the parameter CR. The new parameter CR takes random values in the range $[0, 1]$. Since $F_L = 0.1$, $F_U = 0.9$, Brest et al. [2006] proposed to set $\tau_1 = \tau_2 = 0.1$ such that F takes random values in the range $[0.1, 1]$. Note, both $F_{k,G+1}$ and $CR_{k,G+1}$ are obtained before the mutation is performed in order to influence the mutation, crossover, and selection of the new vector $X_{i,G+1}$. Given that random adjustment can only be good when the parameter setting is not suitable, Noman et al. [2011] proposed to compare the fitness of the offspring $f(U_G)$ with the average fitness value of the current generation, $f_{a,G}$. Then, the choice of the amplification factor F in offspring U_G for the parent $X_{i,G}$ is given by

$$F_G^{child} = \begin{cases} F_{i,G} \text{ if } f(U_G) < f_{a,G} \\ r_1(0.1, 1.0) \text{ otherwise} \end{cases}$$

and that of the crossover parameter is given by

$$CR_G^{child} = \begin{cases} CR_{i,G} \text{ if } f(U_G) < f_{a,G} \\ r_2(0.1, 1.0) \text{ otherwise} \end{cases}$$

where $r_j(a, b)$ are uniform random number in the range $[a, b]$. Initially, the parameters F and CR are created randomly for each individual. Note, the objective vector part of the offspring is created by using the original mutation and crossover values of the DE.

***Remark* 14.1** *Again, perturbing behavioural parameters introduce new parameters in the form of the boundaries for the stochastic sampling ranges and the adjustment probabilities.*

14.2.4.6 Selection Santana-Quintero et al. [2005] maintained two different populations (primary and secondary) according to some criteria and considered two selection mechanisms that are activated based on the total number of generation G_{max} and the parameter $sel_2 \in [0.2, 1]$ which regulates the selection pressure. That is

$$\text{Type of selection } = \begin{cases} \text{Random if } G < (sel_2 * G_{max}) \\ \text{Elitist otherwise} \end{cases}$$

a random selection is first adopted followed by an elitist selection. In the random selection, three different parents are randomly selected from the primary population while in the elitist selection they are selected from the secondary one. In both selections, a single parent is selected as a reference so that all the parents of the main population will be reference parents only once during the generating process.

14.2.5 Convergence criteria revised

When applying a genetic algorithm to solve some NP-hard optimisation problem (the optimum is unkown), it is usually infeasible to compute the optimal solution of the problem. Still, we need to determine whether or not the algorithm has converged to some optimum. Since a genetic algorithm is said to converge when there is no significant improvement in the values of fitness of the population from one generation to the next, there is no defined difference between stopping criteria and convergence criteria. That is, the stopping criterion provides the user a guideline in stopping the algorithm with an acceptable solution close to the optimal solution. While, mathematically, that closeness may be judged in various ways, termination criteria should avoid needless computation and prevent premature termination. Thus, stopping criteria should account for

- Reliability guarantees termination within a finite time.
- Performace guarantees no premature termination and no needless computation.

Stopping criteria are generally based on time or fitness value. The simpliest termination criteria, called exhaustion-based criteria, is to select a maximum number of generations of evaluation functions, or, a maximal time budget (absolute time, CPU time). Alternatively, we can terminate the search when the best objective value $f^* = f_{min}$ reaches or surpasses a bound, $f^* \leqslant f_{lim}$. Usually, if there is no change in the best fitness value for K consecutive iterations, the algorithm is terminated. This method works well if the optimum or a lower bound is known. Some authors proposed tight bounds on the number of iterations required to achieve a level of confidence to guarantee that a Genetic Algorithm has seen all strings (see Aytug et al. [1996]). Giggs et al. [2006] empirically studied a way to determine the maximum number of generations. However, determining the optimum time, or, finding a lower bound is a challenge. Thus, stopping criteria should contain the advantage of reacting adaptively to the state of the optimisation run.

The criteria based on objective function values use the underlying fitness function values to calculate auxiliary values as a measure of the state of the convergence of the GA. For instance,

- improvement-based criteria monitor the improvement of the best objective function value (ImpBest) (or its average ImpAvg) along the optimisation process, and stops when it falls below a user-defined threshold for a given number of generations.
- movement-based criteria monitor the distances between the population members in successive iterations (see Schwefel [1995]). The movement in the population can be calculated with respect to the average objective function value (MovObj), or with respect to positions (MovPar). Termination occurs when it is below a threshold for a given number of generations.
- distribution-based criteria monitor the distances of the population members at each iteration (see Zielinski et al. [2008]). It is assumed that all individuals converge to the optimum, such that convergence is reached when they are close to each other. MaxDist is when the maximum distance from every vector to the best population vector is below a threshold.
- combined criteria use several criteria in combination.

It is understood that the algorithm often focuses on global optimisation at the beginning, leading to large movements between population members in successive iterations, and that at the final stages of the optimisation process, the population generally converges to one point. There are different ways of measuring these distances, such as the standard deviation of positions to all, or part, of the population members, or the distance between the individuals with the best and worst objective function value. In any case, when the distance falls below a user-defined threshold for a given number of iterations, the algorithm terminates. For example, the Running Mean is the difference between the current best objective value $f_{min}(G)$, at generation G, and the average of the best objective value

$$f_{a,min}(n) = \frac{1}{G_{last}} \sum_{i=0}^{n} f_{min}(G_i) \, , G_i = G - i\Delta G$$

over a period of time $G_{last} = n\Delta G$, where ΔG is one generation. It must be less or equal to a given threshold, that is,

$$|f_{min}(G) - f_{a,min}(n)| \leqslant \epsilon$$

The Best-Worst is the difference between the best objective value $f_{min}(G)$ and the worst one $f_{max}(G)$ at generation G. At least $p\%$ of the individuals must be feasible, and it must be less or equal to a given threshold, $|f_{min}(G) - f_{max}(G)| \leqslant \epsilon$. We can also consider relative termination criterions such as $\frac{f_{min}}{f_{a,G}} \leqslant \epsilon$, or letting d_{ij} be the sum of all normalised distance between all individuals of the current generation, the ratio $\frac{d_{ij}}{K_{max}} \leqslant \epsilon$. The latter values the spacial speading of individuals of the current generation in the search space (normalised Euclidian distances $\frac{d_{ij}}{d}$ where d is the length of diagonal of the search space). Jain et al. [2001] proposed the Clus Term, combining information from the objective values and the distribution of individuals in the search space. They perform a cluster analysis (single linkage method) of the fittest individuals and determine the total amount $N(t)$ of of individuals in clusters. The search is terminated when the change of the average of $N(t)$ is equal or less than ϵ. Analysis of stopping criteria reacting adaptively to the state of an optimisation were perfomed (see Zielinski et al. [2005], Zielinski et al. [2007]).

Generally, EAs are stopped or terminated when the variance of fitness values of all the strings in the current population is less than a predefined threshold ϵ. It is assumed that after significantly many iterations the fitness values of the strings present in the population are all close to each other (the population becomes homogeneous), thereby making the variance of the fitness values close to 0. Thus ϵ should be chosen close to zero. Further, one should select a significant number of iterations from which the fitness values will be considered in calculating the variance so that the algorithm gets enough opportunity to yield improved (better) solution. However, this is not correct since

- in elitist model, or other GAs, only the best string is preserved.
- any population containing an optimal string is sufficient for the convergence of the algorithm.
- there is a positive probability of obtaining a population after infinitely many iterations with exactly one optimal string and others are being not optimal.

A lot more iterations often occur after the global optimum has been reached, or nearly reached, to improve other inferior individuals. However, only the best objective function value is important, rather than the convergence of the whole population. Since at the beginning, global search dominates the optimisation algorithm, while at final stages, the algorithm focuses on local optimisation, Liu et al. [2009] proposed a combination of global and local methods. They monitor the average improvement of the whole population in the former, and they monitor the best objective function value in the latter. Bhandari et al. [2012] established theoretically that the variance of the best fitness values obtained in the iterations can be considered as a measure to decide the termination criterion of a GA with elitist model (EGA). The proposed criterion uses only the fitness function values and takes into account the inherent properties of the objective function. We let $f_{min}(i)$ be the best fitness function value obtained at the end of ith iteration, such that $f_{min}(1) \geqslant f_{min}(2) \geqslant ... \geqslant f_{min}(n) \geqslant ... \geqslant F_1$, where F_1 is the global optimal value of the fitness function (F_i denotes the ith lowest fitness function value). Then we get the statistical mean and variance of the best fitness values obtained up to the nth iteration as

$$\begin{aligned} f_{min,1}(n) &= \frac{1}{n}\sum_{i=1}^{n} f_{min}(i) \\ b(n) &= Var(f_{min}(n)) = \frac{1}{n}\sum_{i=1}^{n}\big(f_{min}(i) - f_{min,1}(n)\big)^2 = f_{min,2}(n) - f_{min,1}^2(n) \end{aligned}$$

where $f_{min,2}(n) = \frac{1}{n}\sum_{i=1}^{n} f_{min}^2(i)$. The variance can be iteratively calculated as follow

$$b_{n+1} = \frac{1}{n+1}\Big(\big(nf_{min,2}(n) + f_{min}^2(n+1)\big) - \big(f_{min,1}(n) + f_{min}(n+1)\big)^2\Big)$$

such that only $f_{min,1}(n)$ and $f_{min,2}(n)$ at step n are required to keep in memory when computing the variance at step $(n+1)$. Alternatively, following Equation (5.2), we can write recursively the sample mean as

$$f_{min,1}(n+1) = f_{min,1}(n) + \frac{f_{min}(n+1) - f_{min,1}(n)}{n+1}$$

and following Equation (5.3), the sample variance becomes

$$b(n+1) = b(n) + f^2_{min,1}(n) - f^2_{min,1}(n+1) + \frac{f^2_{min}(n+1) - b(n) - f^2_{min,1}(n)}{n+1}$$

So far, the variance based criterion is not scale invariant, meaning it is sensitive to transformations of the fitness function [11]. One can easily avoid the impact of the scaling effect by a simple transformation of the fitness function, such as

$$g(x) = \frac{f(x)}{f^1_{min}}$$

where f^1_{min} is the minimum value of the fitness function obtained in the first iteration. If we let $b_n(g)$ be the variance of the best fitness values obtained up to the nth iteration for the function $g(x)$, then we get

$$b_n(g) = \frac{1}{n(f^1_{min})^2} \sum_{i=1}^{n} \big(f_{min}(i) - f_{min,1}(n)\big)^2 = \frac{1}{(f^1_{min})^2} b_n(f)$$

such that assuming the tolerence level ϵ_f as the value ϵ for the function f is equivalent to assuming $\epsilon_f \times (f^1_{min})^2$ as the value of ϵ for the function g. It implies that the user has to adjust the value of ϵ for the applied transformation. Note, we now need to use Equation (5.4) to obtain recursively the sample mean, and Equation (5.5) to obtain recursively the sample variance. In that setting, the GA is stopped or terminated after N iterations when the variance of the best fitness values obtained so far is bounded. That is, $b_N < \epsilon$, where $\epsilon > 0$ is a user defined small quantity corresponding to the difference between the fitness value of the best solution obtained so far and the global optimal solution.

15 Hybrid intelligent modelling

Data mining (see Appendix (9)) and more recent machine-learning methodologies (see Appendix (11.1.2)) provide a range of general techniques for the classification, prediction, and optimisation of structured and unstructured data. All of these methods require the use of quantitative optimisation techniques known as stochastic optimisation algorithms, such as combinatorial optimisation, simulated annealing (SA), genetic algorithms (GA), or reinforced learning. We find GA, ANN and SVM algorithms among the most popular classifiers. The artificial neural network (ANN) is the most widely used technique for classification and prediction. It is an optimisation task where the result is to find optimal weight and bias set of the network that reduce an error function. It can be formulated as the minimisation of an error function in the space of connection weights (see Appendix (11.1.1)). However, ANN has many disadvantages and some of its limitations are follows:

- ANN has long trained time
- High computational cost
- Adjustment of weight is difficult
- Output quality of an ANN may be unpredictable

To overcome these disadvantages Neural Networks can be combined with another technique. One way forward is to combine ANN with another algorithm such as evolutionary computing tools that can take care of a specific problem. Hybridisation is a technique which combines two or more classifiers to improve the performance of the classifier.

[11] $g(x) = k \times f(x)$ where k is a constant.

Many researchers proposed hybrid technique with an evolutionary algorithm (EA) to improve the performance of the classifier. For instance, GAs has been hybridised with many other classification algorithms (see Montana et al. [1989]). Alternatively, Ilonen et al. [2003] applied Differential Evolution to train the weights of neural networks, obtaining convergence to a global minimum.

15.1 ANN plus GA in finance

15.1.1 An overview

The underlying principles of genetic algorithm (GA) are to generate an initial population of chromosomes (search solutions) and then use selection and recombination operators to generate a new, more effective population which eventually will have the fittest chromosome (optimal value) among them. Some examples of combition between ANN and GA are Whitley et al. [1990] who used GA to optimise weighted connections and find a good architecture for neural network connections. Kim [2006] proposed a hybrid model of ANN with GA that performs instance selection to reduce dimensionality of data. The genetic algorithm is used to optimise the connection weights between layers of neural network and for selection of relevant instance. Due to the selected instances, the learning time is reduced and prediction performance is enhanced. Bai et al. [2006] presented an effective hybrid system on rough set and neural network. This classifier is used for managing the large document that is growing on the internet. The system creates clusters for organising the documents. The rough set is used for feature reduction and Neural Network is used for classification. Vivekanandan et al. [2010] built a rule-based classifier by using the genetic algorithm. The proposed work tries to reduce the learning time by incremental learning. To do so the classifier reduces the cost of learning and thus making it scalable for large data sets. Large researches have been performed in applying genetic algorithms for classification purpose. For instance, Bhardwaj et al. [2015] proposed a model called GONN which hybridise genetic programming with neural network for multi class classification problem. The model simultaneously deals with structure and weight of the neural network, bringing more diversity of the GP population and reaching a solution faster with more generalised solutions. GAs has been also widely used for the discovery of classification rules.

15.1.2 Some examples

15.1.2.1 Feature selection Sangwan et al. [2015] proposed an integrated ANN and GA for predictive modelling and optimization of turning parameters to minimize surface roughness. Following the same approach, Inthachot et al. [2016] used GA to solve a feature selection problem to find effective subsets of input into ANN. Considering four input variables for each technical indicator, to speed up computation time, they used GA to devise a small number of effective subsets of input variables (see Figure (55)). Accuracy is used to determine chromosome selection as well as to measure the performance of the prediction model.

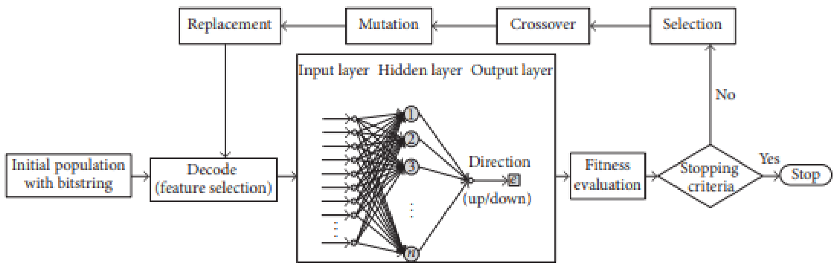


Figure 55: Steps of operation of ANN and GA hybrid intelligence

15.1.2.2 Weight selection Karimi et al. [2012] used GA to find a set of weights for connections to each node in an ANN model and determine correlation of density in nanofluids. Qiu et al. [2016] proposed to determine the optimal set of weights and biases to enhance the accuracy of an ANN model (backpropagation algorithm) by using genetic algorithms (GA). They reported improved prediction accuracy of the direction of stock price index movement with their optimised ANN model. They considered a three-layer ANN model where the output layer consists of only one neuron that represents the predicted direction of the daily stock market index. The signals are generated on a daily basis at time interval $\delta = 1$ day and the forecasting period is $d = 1$ day. GA technique is used to optimise the (initial) weights and biases of the ANN model. Then, the ANN model is trained by the BP algorithm using the determined weights and bias values. The steps are as follow:

- The data is first normalised (see Equation (6.19)), then all the weights and biases are encoded in a string and the initial population is generated. Each solution from the GA is called chromosome (or individual) describing the ANN with a certain set of weights and bias values.
- Train the ANN model using the BP algorithm and then evaluate each chromosome of the current population using a fitness function based on the MSE.
- Rank all the individuals using the fitness proportion method and select the individuals with a higher fitness value to pass on to the next generation directly.
- Apply crossover and mutation to the current population and create new chromosomes. Evaluate the fitness value of the new chromosomes and insert these new chromosomes into the population to replace the worse individuals of the current population.
- Repeat the previous steps until the stop criterion is satisfied.

15.2 ANN plus DE in finance

The training (error) of artificial neural networks (ANNs) depends on the adaptation of free network parameters, such as the weight values and the bias. Thus, it is an optimisation task where the result is to find optimal weight and bias set of the network that reduce an error function. It can be formulated as the minimisation of an error function in the space of connection weights (see Appendix (11.1.1)). The methods for training ANNs are backpropagation (BP), Levenberg-Marquadt (LM), Quasi-Newton (QN). In general, the error backpropagation method (EBP), based on gradient method, is preferred. However, the objective function describing the artificial neural networks training problem is a multi-modal function, so that the algorithms based on gradient methods can easily be stuck in local extremes. To avoid this problem one can use a global search optimisation, such as simulating annealing (SA) algorithm (see Aarst et al. [1989]), Particle Swarm Optimization (PSO) and Evolutionary Algorithm (EA) (see Michalewicz [1996]). For instance, Junyou [2007] used PSO-trained neural networks to forecast the stock price.
We have seen above that when training the ANN by using GA method (GANN), the weight coefficients of neural network are encoded in chromosome, and selection, cross-over and mutation operators are used to minimise the error. However, GANN suffers from early convergence. Even though GA has diversity in its population, it lacks of convergence speed towards global optimia. On the other hand, PSO has faster convergence speed than that of GA but it lacks of diversity in population.

We focus on differential evolution (DE), which has the following advantages: the possibility of finding the global minimum of a multi-modal function regardless of initial values of its parameters, quick convergence and a small number of parameters to set up at the start of the algorithm operation.
Ilonen et al. [2003] applied Differential Evolution to train the weights of neural networks, obtaining convergence to a global minimum but in a very long time. To do so, the weights of all neurons are stored in a real valued solution vector X. The algorithm is used to minimise the sum of squared errors (SSE) or a similar criterion function. The evaluation of this function requires iterating through all elements of the training set T and summing all the partial results (squared errors in the case of SSE) obtained for all the elements of T. Note, different choices of behavioural parameters cause it to perform worse or better than BP on particular problems and the selection of good parameters is a challenge. One

way forward is to try and make the DE parameters automatically adapt to new problems during optimisation, hence alleviating the need for the practitioner to select the parameters by hand. Since some meta parameters are kept fix throughout the entire evolution process, one must tune value of these control parameters. One approach is to use self-adaptive strategy for controlling these parameters (see Brest et al. [2006]). Slowik et al. [2008] applied DE technique to train a feed-forward flat ANN to classification of parity-p problem by introducing the adaptive selection of control parameters to the algorithm. The method proposed is called DE-ANNT (Differential Evolution ? Artificial Neural Network Training). As a result, only one parameter is set at the start of proposed algorithm. Similarly, Bui et al. [2015] use self-adaptive strategy to control the DE parameters. Rather than considering self-adaptive parameters, Pedersen et al. [2008b] proposed an overlaying optimisation method known as meta-optimisation.

Even though EA simultaneously process a population of problem solutions, it comes at the expense of very high computational complexity. To remedy this problem, Fan et al. [2003] introduced Trigonometric DE (TDE) algorithm and applied to train the ANN as a test case for their proposed algorithm. Bandurski et al. [2009] proposed to combine differential evolution algorithm with a gradient-based approach. This hybridisation comes in a number of ways, some of them are as follows:

1. One solution is for an EA to be used to locate a promising region of the weight space, and then use a gradient descent method to fine-tune the best solution (or all the solutions from the population) obtained by the EA (see Sherrod [2008]).
2. Another solution is to apply Lamarckian evolution (see Ross [1999]) to ANN and let the gradient descent procedure be incorporated into an EA as a new search operator. This operator is applied to the population members in each EA iteration, in addition to standard operators such as mutation and crossover (see Cortez et al. [2002]).

The authors applied the conjugate gradient (CG) algorithm (see Hestenes et al. [1952]) to each candidate solution $U_{i,G}$ (Trial vector in Appendix (14.2.1)) before the computation of its fitness. The number of CG iterations is set by the user and remains constant throughout the entire experiment. The results obtained were significantly faster than the original version of DE.
Similarly, to keep a reasonable balance between convergence speed and the capability of global search, Mingguange et al. [2009] combined the BP and DE algorithm to optimise the weights and threshold value adjustments of ANN.

Si et al. [2012] used DE with global and local neighbourhood based mutation (DEGL) algorithm to search the synaptic weight coefficients of neural network and to minimise the learning error in the error surface. It is a modification of DE where both global and local neighbourhood-based mutation operator is combined to create donor vector.

References

[1989] Aarst E.H.L., Korst J., Simulated annealing and Boltzmann machines. John Wiley.

[1993] Abry P., Goncalves P., Flandrin P., Wavelet-based spectral analysis of $1/f$ processes. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, **3**, pp 237–240.

[1995] Abry P., Goncalves P., Flandrin P., Wavelets, spectrum estimation and $1/f$ processes. in A. Antoniadis and G. Oppenheim, eds, *Lecture Notes in Statistics: Wavelets and Statistics*, Springer-Verlag, pp 15–30.

[1996] Abry P., Sellan F., The wavelet-based synthesis for fractional Brownian motion. proposed by F. Sellan and Y. Meyer, Remarks and Fast Implementation, *Applied and Computational Harmonic Analysis*, **3**, (4), pp 377–383.

[1998] Abry P., Veitch D., Wavelet analysis of long-range-dependent traffic. *IEEE Transaction on Information Theory*, **44**, (1), pp 2–15.

[1999] Abry P., Sellan F., A wavelet-based joint estimator of the parameters of long-range dependence. *IEEE Transaction on Information Theory*, **45**, (3), pp 878–897.

[2000] Abry P., Flandrin P., Taqqu M.S., Veitch D., Wavelets for the analysis, estimation, and synthesis of scaling data. in *Self-similar Network Traffic and Performance Evaluation*, ed. K. Park, W. Willinger, Wiley, pp 39–87.

[2002] Abry P., Baraniuk R., Flandrin P., Riedi R., Veitch D., Multiscale nature of network traffic. *IEEE Signal Processing Magazine*, **19**, (3), pp 28–46.

[2001] Abu-Mostafa Y.S., Atiya A.F., Magdon-Ismail M., White H., Neural networks in financial engineering. *IEEE Transactions on Neural Networks*, **12**, (4), pp 653–656.

[1998] Adya M., Collopy F., How effective are neural networks at forecasting and prediction? A review and evaluation. *Journal of Forecasting*, **17**, pp 481–495.

[2010] Ahmed N.K., Atiya A.F., El Gayar N., El-Shishiny H., An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, **29**, (5), pp 594–621.

[2010] Alpaydin E., Introduction to machine learning. MIT Press

[1997] Amit Y., Geman D., Shape quantization and recognition with randomized trees. *Neural Computation*, **9**, (7), pp 1545–1588.

[2015] Antonik P., Duport F., Smerieri A., Haelterman P., Massar S., FPGA implementation of reservoir computing with online learning. Université Libre de Bruxelles.

[2016] Antonik P., Hermans M., Duport F., Haelterman M., Massar S., Towards pattern generation and chaotic series prediction with photonic reservoir computers. Laboratoire d'Information Quantique, Universite Libre de Bruxelles, Belgium.

[1974] Appel G., Double your money every three years. Brightwaters, NY: Windsor Books.

[1999] Appel G., Technical analysis power tools for active investors. Financial Times Prentice Hall.

[2008] Appel G., Appel M., A quick tutorial in MACD: Basic concepts. Working Paper.

[2001] Armstrong J.S., Principles of forecasting: A handbook for researchers and practitioners. ed.: Norwell, MA: Kluwer Academic Publishers.

[1991] Arneodo A., Bacry E., Muzy J-F., Wavelets and multifractal formalism for singular signals: Application to turbulence data. *Phys. Rev. Lett.*, **67**, pp 3515–3518.

[1995] Arneodo A., Bacry E., Graves P.V., Muzy J-F., Characterizing long-range correlations in DNA sequences from wavelet analysis. *Phys. Rev. Lett.*, **74**, 3293.

[1998] Arneodo A., Muzy J-F., Dornette D., Discrete causal cascade in the stock market. *Eur. Phys. J.*, **2**, pp 277–282.

[2017] Arras L., Montavon G., Muller K-R., Samek W., Explaining recurrent neural network predictions in sentiment analysis. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pp. 159–168.

[2000] Atiya A.F., Parlos A.G., New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on neural networks*, **11**, (3), pp 697–709.

[1996] Aytug H., Koehler G.J., New stopping criterion for genetic algorithms. Working Paper, University City Blvd and University of Florida.

[2001] Bacry E., Delour J., Muzy J-F., Multifractal random walk. *Physical Review E*, **64**, 026103–026106.

[2006] Bai R., Wang X., An effective hybrid classifier based on rough set and neural network. *International Conference IEEE*, 0-7695-2749-3/06.

[1996] Baillie R.T., Long memory processes and fractional integration in econometrics. *Journal of Econometrics*, **73**, pp 5–59.

[1996] Baillie R.T., Bollerslev T., Mikkelsen H.O., Fractionally integrated generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **74**, pp 3–30.

[2007] Balamurugan R., Subramanian S., Self-adaptive differential evolution based power economic dispatch of generators with valve-point effects and multiple fuel options. *International Journal of Computer Science and Engineering*, Winter.

[2009] Bandurski K., Kwedlo W., Training neural networks with hybrid differential evolution algorithm. Zeszyty Naukowe Politechniki Bialostockiej.

[2012] Barunik J., Understanding the source of multifractality in financial markets. *Physica A*, **391**, (17), pp 4234–4251.

[2013] Battula B.P., Satya Prasad R., An overview of recent machine learning strategies in data mining. *International Journal of Advanced Computer Science and Applications*, **4**, (3), pp 50–54.

[1993] Bazaraa M.S., Sherali H.D., Shetty C.M., Nonlinear programming theory and applications. 2nd ed. John Wiley and Sons.

[2008] Bekiros S.D., Georgoutsos D.A., Direction-of-change forecasting using a volatility based recurrent neural network. *Journal of Forecasting*, **27**, (5), pp 407–417.

[1999] Bellgard C., Goldschmidt P., Forecasting foreign exchange rates: Random walk hypothesis, linearity and data frequency. Working Paper, The University of Western Australia.

[1994] Bengio Y., Simard P., Frasconi P., Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, **5**, (2), pp 157–166.

[1996] Bertsekas D.P., Constrained optimization and Lagrange multiplier methods. MIT, Athena Scientific.

[2015] Bertsekas D.P., Convex optimization algorithms. Belmont, MA., Athena Scientific.

[1998] Bessembinder H., Chan K., Market efficiency and the returns to technical analysis. *Financial Management*, **27**, (2), pp 5–27.

[2009] Beynon-Davies P., Business information systems. Basingstoke, UK

[2012] Bhandari D., Murthy C.A., Pal S.K., Variance as a stopping criterion for genetic algorithms with elitist model. *Fundamenta Informaticae*, **120**, pp 145–164.

[2015] Bhardwaj A., Tiwari A., Breast cancer using genetically optimized neural network model. *International journal Expert System with Application*.

[2017] Bianchi F.M., Kampffmeyer M., Maiorino E., Jensen R., Temporal overdrive recurrent neural network. Working Paper, University of Tromso, arXiv:1701.05159v1.

[1978] Binder D.A., Bayesian cluster analysis. *Biometrika*, **65**, pp 31–38.

[2006] Bishop C.M., Pattern recognition and machine learning. Springer Verlag.

[2014] Bloch D., A practical guide to quantitative portfolio trading. Working Paper, University of Paris 6 Pierre et Marie Curie.

[1986] Bollerslev T., Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **31**, pp 307–327.

[1999] Bollerslev T., Jubinski D., Equality trading volume and volatility: Latent information arrivals and common long-run dependencies. *Journal of Business & Economic Statistics*, **17**, pp 9–21.

[1992] Bollinger J., Using Bollinger bands. *Technical Analysis of Stocks and Commodities*, **10**, February.

[1992] Boser B.E., Guyon I.M., Vapnik V.N., A training algorithm for optimal margin classifiers. in *Proceedings of the fifth annual workshop on Computational learning theory*, pp 144.

[2017] Bouizi R., Financial time series forecasting using wavelet transform and reservoir computing paradigm. Quant Finance Ltd, MSc Thesis, Department of Mathematics, Imperial College London.

[2004] Boyd S., Vandenberghe L., Convex optimization. Cambridge University Press.

[2000] Brabazon T., A connectivist approach to index modelling in financial markets. Department of Accountancy, University College, Dublin.

[1995] Branicky M.S., Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, **138**, pp 67–100.

[1998] Breidt F.J., Crato N., de Lima P., On the detection and estimation of long memory in stochastic volatility. *Journal of Econometrics*, **83**, pp 325–348.

[1984] Breiman L., Friedman J., Stone C.J., Olshen R.A., Classification and regression trees. Chapman and Hall/CRC, Boca Raton, FL.

[1996] Breiman L., Stacked regressions. *Machine Learning*, **24**, (1), pp 49–64.

[1996b] Breiman L., Out-of-bag estimation. Technical report, Department of Statistics, University of California.

[1996c] Breiman L., Bagging predictors. *Machine Learning*, **24**, (2), pp 123–140.

[2001] Breiman L., Random forests. *Machine learning*, **45**, (1), pp 5–32

[2006] Brest J., Greiner S., Boskovic B., Mernik M., Zumer V., Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, **10**, (6), pp 646–657.

[1992] Brock W., Lakonishok J., LeBaron B., Simple technical trading rules and the stochastic properties of stock returns. *Journal of Finance*, **47**, (5), pp 1731–1764.

[1999] Brown C., Technical analysis for the trading professional. 1st Edition, McGraw-Hill.

[1969] Bryson A.E., Ho Y.C., Applied optimal control. U.K. Blaisdell.

[2015] Bush N.T., Hasegawa H., Training artificial neural network using modification of differential evolution algorithm. *International Journal of Machine Learning and Computing*, **5**, (1), pp 1–6.

[2005] Bush K., Tsendjav B., Improving the richness of echo state features using next ascent local search. in *Proceedings of the Artificial Neural Networks in Engineering Conference*, pp 227–232, St. Louis.

[2006] Bush K., Anderson C., Exploiting iso-error pathways in the N, k-plane to improve echo state network performance.

[1994] Cai J., A Markov model of switching-regime ARCH. *Journal of Business*, **12**, pp 309–316.

[2004] Cajueiro D.O., Tabak B.M., The Hurst exponent over time: Testing the assertion that emerging markets are becoming more efficient. *Physica A*, **336**, pp 521–537.

[2007] Cajueiro D.O., Tabak B.M., Long-range dependence and multifractality in the term structure of LIBOR interest rates. *Physica A*, **373**, pp 603–614.

[1995] Caldwell R.B., Performances metrics for neural network-based trading system development. *NeuroVet Journal*, **3**, (2), pp 22–26.

[2001] Calvet L., Fisher A., Forecasting multifractal volatility. *Journal of Econometrics*, **105**, pp 27–58.

[2002] Calvet L., Fisher A., Multifractality in asset returns: Theory and evidence. *The Review of Economics and Statistics*, **84**, (3), pp 381–406.

[2004] Calvet L., Fisher A., Regime-switching and the estimation of multifractal processes. *Journal of Financial Econometrics*, **2**, pp 44–83.

[2006] Calvet L., Fisher A., Thompson S., Volatility comovement: A multi-frequency approach. *Journal of Econometrics*, **31**, pp 179–215.

[2013] Calvet L., Fisher A., Wu L., Staying on top of the curve: A cascade model of term structure dynamics. Working Paper.

[1997] Campbell J.Y., Lo A.W., MacKinlay A.C., The econometrics of financial markets. Princeton University Press, New jersey.

[2008] Cao M., Qiao P., Neural network committee-based sensitivity analysis strategy for geotechnical engineering problems. *Neural Comput Appl*, **17**, pp 509–519

[2004] Carbone A., Castelli G., Stanley H., Time-dependent Hurst exponents in financial time series. *Physica A*, **344**, pp 267–271.

[2007] Carbone A., Algorithm to estimate the Hurst exponent of high-dimensional fractals. Working Paper, Physics Department, Politecnico di Torino.

[1996] Casey M., The dynamics of discrete-time computation with application to recurrent neural networks and finite state machine extraction. *Neural Computation*,**8**, pp 1135–1178.

[2001] Castiglione F., Bernaschi M., Market fluctuations: Simulation and forecasting. Working Paper

[1994] Chande T.S., Kroll S., The new technical trader. John Wiley, New York.

[2004] Chen Y., Yang B., Dong J., Nonlinear system modelling via optimal design of neural trees. *International Journal of Neural Systems*, **14**, (2), pp 125–137.

[2009] Chen C-W., Huang C-S., Lai H-W., The impact of data snooping on the testing of technical analysis: An empirical study of Asian stock markets. *Journal of Asian Economics*, **14**, (5), pp 580–591.

[2010] Chitra A., Uma S., An ensemble model of multiple classifiers for time series prediction. *International Journal of Computer Theory and Engineering*, **2**, (3), pp 1793–8201

[2014] Cho K., Van Merrienboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y., Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv: 1406.1078.

[1984] Chua L.O., Lin G.N., Non-linear programming without computation. *IEEE Trans. Circuits and Systems*, CAS-31, pp 182–186.

[2015] Chung J., Gulcehre C., Cho K., Bengio Y., Gated feedback recurrent neural networks. CoRR, abs:1502.02367.

[1993] Cichocki A., Unbehauen R., Neural networks for optimization and signal processing. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore.

[2006] Clegg R.G., A practical guide to measuring the Hurst parameter. *International Journal of Simulation: Systems, Science and Technology*, **7**, (2), pp 3–14.

[2000] Coello Coello C.A., Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, Vol. **17**, pp 319–346.

[2002] Coello Coello C.A., Mezura-Montes E., Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatic*, Vol. **16**, pp 193–203.

[2003] Coello Coello C.A., Mezura-Montes E., Increasing successful offspring and diversity in differential evolution for engineering design. *Advanced Engineering Informatic*, Vol. **16**, pp 193–203.

[1960] Cohen J., A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, **20**, (1), pp 37–46.

[1994] Connor J.T., Martin R.D., Atlas L.E., Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, **5**, (2), pp 240–254.

[2016] Contras D., Matei O., Translation of the mutation operator from genetic algorithms to evolutionary ontologies. in *International Journal of Advanced Computer Science and Applications (IJACSA)*, **7**, (1).

[2002] Cortez P., Rocha M., Neves J., A lamarckian approach for neural network training. *Neural Processing Letters*, **15**, pp 105–116.

[2013] Cortez P., Embrechts M.J., Using sensitivity analysis and visualization techniques to open black box data mining models. *Inf Sci*, **225**, pp 1–17

[2003] Costa R.L., Vasconcelos G.L., Long-range correlations and nonstationarity in the Brazilian stock market. *Physica A*, **329**, pp 231–248.

[2000] Cunningham P., Carney J., Diversity versus quality in classification ensembles based on feature selection. Technical Report TCD-CS-2000-02, Department of Computer Science, Trinity College Dublin.

[1989] Cybenko G., Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, **2**, (4), pp 303–314

[2003] Dablemont S., Van Bellegem S., Verleysen M., Modelling and forecasting financial time series of tick data by functional analysis and neural networks. Universite catholique de Louvain, Machine Learning Group, DICE, Louvain-la-Neuve, Belgium.

[1993] Dacorogna M.M., Muller U.A., Nagler R.J., Olsen R.B., Pictet O.V., A geographical model for the daily and weekly seasonal volatility in the foreign exchange market. *Journal of International Money and Finance*, **12**, (4), pp 413–438.

[1998] Dacorogna M.M., Muller U.A., Olsen R.B., Pictet O.V., Modelling short-term volatility with GARCH and HARCH models. in C. Dunis and B. Zhou, (eds.), *Nonlinear Modelling of High Frequency Financial Time Series*, John Wiley & Sons, pp 161–176.

[2001] Dacorogna M.M., Gencay R., Muller U.A., Olsen R.B., Pictet O.V., An introduction to high frequency finance. Academic Press, San Diego, CA.

[2003] Damodaran A., Investment philosophies. John Wiley & Sons, New York.

[1998] Darbellay G.A., Predictability: An information-theoretic perspective. in *Signal Analysis and Prediction*, edt, A. Prochazka, J. Uhlir, P.W.J. Rayner and N.G. Kingsbury, pp 249–262.

[2000] Darbellay G.A., Slama M., Forecasting the short-term demand for electricity? Do neural networks stand a better chance? *International Journal of Forecasting*, **16**, pp 71–83.

[1882] Darwin C.R., The variation of animals and plants under domestication. Murray, London, second edition.

[2005] Das S., Konar A., Chakraborty U.K., Two improved differential evolution schemes for faster global search. in *Proceedings of the 2005 conference on Genetic and Evolutionary Computing*, (GECCO 2005), pp 991–998.

[2002] Day T.E., Wang P., Dividends, nonsynchronous prices, and the returns from trading the Dow Jones Industrial Average. *Journal of Empirical Finance*, **9**, (4), pp 431–454.

[2000] Deb K., An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, **186**, (2/4), pp 311-338.

[2014] De Ona J., Garrido C., Extracting the contribution of independent variables in neural network models: A new approach to handle instability. *Neural Computing and Applications*, **25**, pp 859–869.

[1993] Ding Z., Granger C.W.J., Engle R.F., A long memory property of stock market returns and a new model. *Journal of the Empirical Finance*, **1**, pp 83–106.

[2017] Dingli A., Fournier K.S., Financial time series forecasting: A machine learning approach. *Machine Learning and Applications: An International Journal* (MLAIJ), **4**, No.1/2/3, September.

[2000] Dominey P.F., Ramus F., Neural network processing of natural language: I. Sensitivity to serial, temporal and abstract structure of language in the infant. *Language and Cognitive Processes*, **15**, (1), pp 87–127.

[1999] Douglas S.C. Introduction to adaptive filters. University of Utah. Published by CRC Press LLC.

[1992] Doya K., Bifurcations in the learning of recurrent neural networks. in *Proceedings of IEEE International Symposium on Circuits and Systems*, **6**, pp 2777–2780.

[1999] Drummond C., Hearne T., The lessons. A series of 30 multi-media lessons, Drummond and Hearne Publications, Chicago.

[1953] Eccles J.G., The Neurophysiological basis of mind. Clarendon, Oxford.

[1993] Efron B., Tibshirani R., An introduction to the bootstrap. Chapman & Hall, New York.

[1995] El Hihi S., Bengio Y., Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, **400**, pp 409.

[1982] Engle R.F., Autoregressive conditional heteroskedasticity with estimates of the variance of U.K. inflation. *Econometrica*, **50**, pp 987–1008.

[1986] Engle R.F., Granger C.W.j., Rice j., Weiss A., Semiparametric estimates of the relation between weather and electricity sales. *Journal of the American Statistical Association*, **81**, pp 310–320.

[1987] Engle R.F., Granger C.W.j., Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, **55**, 2, pp 251–276.

[2003] Fan H.Y., Lampinen J., A trigonometric mutation operation to differential evolution. *International Journal of Global Optimization*, **27**, pp 105–129.

[2006] Feoktistov V., Differential evolution, in search of solutions. Springer, Optimisation and its Applications, **5**.

[2008] Fernandez-Blanco P., Technical market indicators optimization using evolutionary algorithms. In proceedings of the 2008 GECCO conference companion on genetic and evolutionary computation. Atlanta, USA.

[1936] Fisher R.A., The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, **7**, pp 179–188.

[1981] Fleiss J.L., Statistical methods for rates and proportions. John Wiley & Sons, New York, NY, 2nd edition.

[1966] Fogel L.J., Artificial intelligence through simulated evolution. John Wiley, New York.

[2008] Fok W.W.T., Tam V.W.L., Computational neural network for global stock indexes prediction. *Proceedings of the World Congress on Engineering*, **2**, pp 1171–1175.

[1986] Fox R., Taqqu M.S., Large-sample properties of parameter estimates for strongly dependent stationary Gaussian time series. *The Annals of Statistics*, **14**, (2), pp 517–532.

[1997] Freund Y., Schapire R.E., A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, **55**, (1), pp 119–139.

[2000] Friedman J., Hastie T., Tibshirani R., Additive logistic regression: A statistical view of boosting (with discussions). *Annals of Statistics*, **28**, (2), pp 337–407.

[2010] Fua P., Aydin V., Urtasun R., Salzmann M., Least-squares minimization under constraints. Technical Report EPFL-REPORT-150790.

[1989] Funahashi K., On the approximate realization of continuous mappings by neural networks. *Neural Network*, **2**, (3), pp 183–192.

[1993] Funahashi K., Nakamura Y., Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, **6**, pp 801–806.

[1997] Gencay R., The predictability of security returns with simple technical trading rules. *Journal of Empirical Finance*, **5**, pp 347–359.

[2000] Gers F.A., Schmidhuber J., Cummins F., Continual prediction with LSTM. *Neural Computation*,**12**, (10), pp 2451–2471.

[2002] Gers F.A., Schraudolph N., Schmidhuber J., Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, **3**, pp 115–143.

[2006] Giggs M.S., Maier H.R., Dandy G.C., Nixon J.B., Minimum number of generations required for convergence of genetic algorithms. in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp 2580–2587.

[2014] Goudarzi A., Banda P., Lakin M.R., Teuscher C., Stefanovic D., A comparative study of reservoir computing for temporal signal processing. Working Paper, University of New Mexico and Portland State University.

[2004] Granger C., Hyng N., Occasional structural breaks and long memory with an application to the SP500 absolute stock returns. *Journal of Empirical Finance*, **11**, pp 399–421.

[1976] Graves J.E., Granville's new strategy of daily stock market timing for maximum profit. Prentice-Hall, Inc.

[2005] Graves A., Schmidhber J., Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, **18**, (5-6), pp 602–610.

[2005] Graves A., Supervised sequence labelling with recurrent neural networks. PhD thesis, Technische Universitat Munchen.

[2011] Graves A., Mohamed A-R., Hinton G., Speech recognition with deep recurrent neural networks. *International Conference on Acoustics, Speech and Signal Processing*, pp 1033–1040.

[2013] Graves A., Generating sequences with recurrent neural networks. arXiv:1308.0850.

[2004] Grech D., Mazur Z., Can one make any crash prediction in finance using the local Hurst exponent idea? *Physica A: Statistical Mechanics and its Applications*, **336**, (1), pp 133–145.

[2005] Grech D., Mazur Z., Statistical properties of old and new techniques in detrended analysis of time series. *Acta Physica Polonica B*, **36**, (8), pp 2403–2413.

[2015] Greff K., Srivastava R.K., Koutnik J., Steunebrink B.R., Schmidhuber J., LSTM: A search space odyssey. Transactions On Neural Networks And Learning Systems.

[2017] GNU Scientific Library. https://www.gnu.org/software/gsl/.

[2010] Gu G-F., Zhou W-X., Detrending moving average algorithm for multifractals. *Physical Review E*, **82**, 011136, pp 1–12.

[2015] Hallac D., Leskovec J., Boyd S., Network lasso: clustering and optimization in large graphs. Stanford University, arXiv:1507.00280.

[2000] Hammer B., On the approximation capability of recurrent neural networks. *Neurocomputing*, **31**, (1-4), pp 107–123.

[2006] Han J., Kamber M., Data mining: Concepts and techniques. Elsevier ; Morgan Kaufmann, 2nd ed. Amsterdam, Boston, San Francisco, CA.

[2013] Han Y., Yang K., Zhou G., A new anomaly: The cross-sectional profitability of technical analysis. *Journal of Financial and Quantitative Analysis*, **48**, (5), pp 1433–1461.

[2001] Hand D., Mannila H., Smyth P., Principles of data mining. MIT Press.

[1990] Hansen L.K., Salamon P., Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**, (10), pp 993–1001.

[2013] Hargreaves C., Hao Y., Prediction of stock performance using analytical techniques. *Journal of Emerging Technologies in Web Intelligence*, **5**, (2), pp 136–142.

[1997] Harrald P.G., Kamstra M., Evolving artificial neural networks to combine financial forecasts. *IEEE Transactions on Evolutionary Computation*, **1**, (1), pp 40–52.

[2005] Harrington N., The link between bollinger bands and the commodity channel index Harrington. *Technical Analysis of Stocks and Commodities*, **23**, (10), pp 32–38.

[1979] Harrison J.M., Kreps D., Martingale and arbitrage in multiperiods securities markets. *Journal of Economic Theory*, **20**, pp 381–408.

[1981] Harrison J.M., Pliska S.R., Martingales and stochastic integrals in the theory of continuous trading. *Stochastic Processes Applications*, **11**, pp 215–260.

[2002] Harvey C.R., Travers K.E., Costa M.J., Forecasting emerging market returns using neural networks. *Emerging Markets Quarterly*, , pp 1–12.

[2009] Hastie T., Tibshirani R., Friedman J., The elements of statistical learning: Data mining, inference, and prediction. Springer Series in Statistics, Second Edition.

[2000] Haykin S., Adaptive filter theory. Prentice-Hall, Upper Saddle River, New Jersey.

[1949] Hebb D.O., The organization of behavior. Wiley, New York.

[1998] Helstrom T., Holmstrom K., Predicting the stock market. Published as Opuscula ISRN HEV-BIB-OP-26-SE.

[1952] Hestenes M.R., Stiefel E., Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, **49**, pp 409–436.

[2000] Hill J.R., Pruitt G., Hill L., The ultimate trading guide. John Wiley & Sons, Wiley Trading Advantage.

[1994] Ho T.K., Hull J.J., Srihari S.N., Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**, (1), pp 66–75.

[1995] Ho T.K., Random decision forests. in *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, pp 14–16

[1998] Ho T.K., The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, (8), pp 832–844

[2004] Ho D-S., Lee C-K., Wang C-C., Chuang M., Scaling characteristics in the Taiwan stock market. *Physica A*, **332**, pp 448–460.

[1997] Hochreiter S., Schmidhuber J., Long short-term memory. *Neural Computation*, **9**, (8), pp 1735–1780.

[2001] Hochreiter S., Bengio Y., Frasconi P., Schmidhuber J., Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. in A field guide to dynamical recurrent neural networks. IEEE Press.

[1962] Holland J.H., Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, **9**, pp 297-314.

[1975] Holland J.H., Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor.

[1982] Hopfield J.J., Neural networks and physical systems with emergent collective computational abilities. *Proceedings of National Academy of Sciences USA*, **79**, pp 2554–2558.

[1985] Hopfield J.J., Tank D.W., Neural computation of decisions in optimization problems. *Biological Cybernetics*, **52**, pp 141–152.

[1987] Hopfield J.J., Tank D.W., Computing with neural circuits: A model. *Science*, **233**, pp 625–633.

[1989] Hornik K., Stinchcombe M., White H., Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, (5), pp 359–366.

[1991] Hornik K., Approximation capabilities of multilayer feedforward networks. *Neural Networks*, **4**, (2), pp 251–257.

[2017] Hudson R., Urquhart Y., Wang P., Why do moving average rules work? Comprehensive evidence from world markets.

[1951] Hurst H.E., Long-term storage capacity of reservoirs. *Trans. Amer. Soc. Civil Engineers*, **116**, pp 770–799.

[2012] Ihlen E.A.F., Introduction to multifractal detrended fluctuation analysis in Matlab. *Frontiers in Physiology*, **3** (141), pp 1–18.

[2013] Ihlen E.A.F., Multifractal analyses of response time series: A comparative study. *Behav. Res.*, **45**, pp 928–945.

[2014] Ihlen E.A.F., Vereijken B., Detection of co-regulation of local structure and magnitude of stride time variability using a new local detrended fluctuation analysis. *Gait & Posture*, **39**, pp 466–471.

[2000] Ilkka T., Data is more than knowledge. *Journal of Management Information Systems*, **6**, (3), pp 103–117.

[2003] Ilonen J., Kamarainen J.K., Lampinen J., Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, **17**, pp 93–105.

[2016] Inthachot M., Boonjing V., Intakosum1 S., Artificial neural network and genetic algorithm hybrid intelligence for predicting Thai ttock price index trend. *Computational Intelligence and Neuroscience*, **2016**, pp 1–8.

[2004] Ishii K., van der Zant T., Becanovic V., Ploger P., Identification of motion with echo state network. in *Proceedings of the OCEANS 2004 MTS/IEEE Conference*, **3**, pp 1205–1210.

[1991] Jacobs R.A., Jordan M.I., Nowlan S.J., Hinton G.E., Adaptive mixtures of local experts. *Neural Computation*, **3**, (1), pp 79–87.

[2001] Jaeger H., The echo state approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology.

[2003] Jaeger H., Adaptive nonlinear system identification with Echo State Networks. International University Bremen.

[2004] Jaeger H., Seminar slides. Seminar Spring.

[2004] Jaeger H., Haas H., Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, **304**, pp 78–80.

[2007] Jaeger H., Echo state network. *Scholarpedia*, **2**, (9), pp 2330.

[2001] Jain B.J., Pohlheim H., Wegener J., On termination criteria of evolutionary algorithms. GECCO 2001 - Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kauffmann, San Francisco.

[2016] Jamous R.A., El. Seidy E., Bayoum B.I., A novel efficient forecasting of stock market using particle swarm optimization with center of mass based technique. *International Journal of Advanced Computer Science and Applications*, **7**, (4), pp 342–347.

[1991] Jansen D.W., de Vries C.G., On the frequency of large stock market returns: Putting booms and busts into perspective. *Review of Economics and Statistics*, **73**, (1), pp 18–24.

[2011] Jhankal N.K., Adhyaru D., Bacterial foraging optimization algorithm: A derivative free technique. Engineering (NUiCONE), Nirma University International Conference on 2011.

[2008] Jiang F., Berry H., Schoenauer M., Supervised and evolutionary learning of echo state networks. in *Proceedings of 10th International Conference on Parallel Problem Solving from Nature*, **5199** of LNCS, pp 215–224, Springer.

[2012] Jizba P., Korbel J., Methods and techniques for multifractal spectrum estimation in financial time series. FNSPE, Czech Technical University, Prague.

[1992] Jordan M.I., Jacobs R.A., Hierarchies of adaptive experts. In J.E. Moody, S.J. Hanson, and R. Lippmann, editors, Advances in *Neural Information Processing Systems 4*, Morgan Kaufmann, San Francisco, pp 985–992.

[1995] Jordan M.I., Xu L., Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, **8**, (9), pp 1409–1431.

[2013] Jordehi A., Jasni J., Parameter selection in particle swarm optimisation: A survey. *Journal of Experimental & Theoretical Artificial Intelligence*, **25**, (4), pp 527–542.

[2007] Junyou B., Stock price forecasting using PSO-trained neural networks. in *IEEE Congress on Evolutionary Computation*, pp 2879–2885.

[2011] Kamber H., Jaiwei P., Jian M., Data mining: Concepts and techniques. Morgan Kaufmann, (3rd ed.).

[2002] Kantelhardt J., Zschiegner S., Koscielny-Bunde E., Bunde A., Havlin S., Stanley H.E., Multifractal detrended fluctuation analysis of nonstationary time series. *Physica A*, **316**, (1-4), pp 87–114.

[2011] Kara Y., Boyacioglu M.A., Baykan O.K., Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert Systems with Applications*, **38**, (5), pp 5311–5319.

[2004] Karaboga D., Okdem S., A simple and global optimization algorithm for engineering problems: Differential evolution algorithm. *Turkish Journal of Electrical Engineering & Computer Sciences*, **12**, (1), pp 53–60.

[2012] Karimi H., Yousefi F., Application of artificial neural network-genetic algorithm (ANN-GA) to correlation of density in nanofluids. *Fluid Phase Equilibria*, **336**, pp 79–83.

[1960] Keltner C.W., How to make money in commodities. Keltner Statistical Service.

[1988] Kennedy M.P., Chua L.O., Neural networks for non-linear programming. *IEEE Trans. Circuit and Systems*, **35**, pp 554–562.

[1995] Kennedy J., Eberhart C., Particle swarm optimization. in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Australia, pp 1942–1948.

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

[2004] Kim K., Choi J-S., Yoon S-M., Multifractal measures for the yen-dollar exchange rate. *Journal of the Korean Physical Society*, **44**, (3), pp 643–646.

[2004b] Kim K-J., Won Boo L., Stock market prediction using artificial neural networks with optimal feature transformation. *Neural computing and applications*, **13**, (3), pp 255–260.

[2006] Kim K-J., Artificial neural networks with evolutionary instance selection for financial forecasting. *Expert Systems with Applications*, **30**, (3), pp 519–526.

[2006] Kim M-J., Min S-H., Han I., An evolutionary approach to the combination of multiple classifiers to predict a stock price index. *Expert Systems with Applications*, **31**, (2), pp 241–247.

[1990] Kimoto T., Asakawa K., Yoda M., Takeoka M., Stock market prediction system with modular neural netorks. in *Proceedings of the 1990 International Joint Conference on Neural Networks*, **1**, Washington, DC, USA, pp 1–6.

[1998] Kittler J., Hatef M., Duin R., Matas J., On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, (3), pp 226–239.

[1996] Kohavi R., Wolpert D.H., Bias plus variance decomposition for zero one loss functions. In *Proceedings of the 13th International Conference on Machine Learning*, Bari, Italy, pp 275–283.

[2013] Kristoufek L., Vosvrda M., Measuring capital market efficiency: Global and local correlations structure. *Physica A*, **392**, (1), pp 184–193.

[2010] Krollner B., Vanstone B., Finnie G., Financial time series forecasting with machine learning techniques: A survey. in *ESANN 2010 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium.

[2006] Kucukemre A.U., Echo state networks for adaptive filtering. Master Thesis, University of Applied Sciences, Bohn-Rhein-Sieg, Germany.

[1951] Kuhn H.W., Tucker A.W., Nonlinear programming. in *Proceedings of 2nd Berkeley Symposium*. Berkeley, University of California Press, pp 481–492.

[2003] Kuncheva L.I., Whitaker C.J., Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, **51**, (2), pp 181–207.

[1992] Kurkova V., Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, **5**, pp 501–506.

[1997] Lagoudakis M.G., Neural networks and optimization problems A case study: The minimum cost spare allocation problem. The Center for Advanced Computer Studies, University of Southwestern Louisiana.

[1997] Lam L., Suen S.Y., Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, **27**, (5), pp 553–568.

[2014] Lam H.K., Ekong U., Xiao B., Ouyang G., Liu H., Chan K.Y., Ling S.H., Variable weight neural networks and their applications on material surface and epilepsy seizure phase classifications. Working Paper, King's College, London.

[1980] Lambert D.R., Commodity channel index: Tool for trading cyclic trends. Commodities Magazine, 219 Parkade, Cedar Falls, IA 50613.

[2004] Lampinen J., Storn R., Differential evolution. in *New Optimization Techniques in Engineering*, G.C. Onwubolu and B. Babu, Eds., Springer-Verlag, Berlin, pp 123–166.

[1984] Lane G., Lane's stochastics. *Technical Analysis of Stocks and Commodities*, pp 87–90.

[1990] Lang K.J., Waibel A.H., Hinton G.E., A time-delay neural network architecture for isolated word recognition. *Neural Networks*, **3**, pp 33–43.

[1987] Lapedes A., Farber R., Nonlinear signal processing using neural networks: Prediction and modeling. Technical Report, LA-UR87-2662, Los Alamos, New Mexico.

[1989] LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D., Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1**, (4), pp 541–551.

[1991] Lee Y., Oh S., Kim M., The effect of initial weights on premature saturation in back propagation learning. *Int Jt Conf Neural Netw*, **1**, pp 65–70.

[2001] Lemarechal C., Lagrangian relaxation. In Michael Junger and Denis Naddef *Computational combinatorial optimization: Papers from the Spring School held in Schloss Dagstuhl*, May 15-19, 2000. Lecture Notes in Computer Science. Berlin, Springer-Verlag, pp 112–156.

[2010] Liao Z., Wang J., Forecasting model of global stock index by stochastic time effective neural network. *Expert Systems with Applications*, **37**, (1), pp 834–841.

[1993a] Lillo W.E., Loh M.H., Hui S., Zak S.H., On solving constrained optimisation problems with neural networks : A penalty method approach. *IEEE Transactions on Neural Networks*, **4**, (6), pp 931–940.

[1993b] Lillo W.E., Hui S., Zak S.H., Neural network for constrained optimization problems. *International Journal of Circuit Theory and Applications*, **21**, pp 385–399.

[2008] Lin H-T., Li L., Support vector machinery for infinite ensemble learning. *Journal of Machine Learning Research*, **9**, pp 285–312.

[2009] Lin X., Yang Z., Song Y., Short-term stock price prediction based on echo state networks. *Expert Systems with Applications*, **36**, pp 7313–7317.

[2002] Liu J., Lampinen J., On setting the control parameter of the differential evolution method. in *Proc. 8th Int. Conf. Soft Computing*, pp 11–18, Brno University of Technology, Czech Republic.

[2005] Liu J., Lampinen J., A fuzzy adaptive differential evolution algorithm. *Soft Computing*, **9**, (6), pp 448–462.

[2009] Liu B., Fernandez F.V., De Jonghe D., Gielena G., Less expensive and high quality stopping criteria for MC-based analog IC yield optimization. Working Paper, ESAT-MICAS, Katholieke Universiteit Leuven and IMSE, CSIC and University of Sevilla.

[1991] Lo A.W., Long-term memory in stock market prices. *Econometrica*, **59**, (5), pp 1279–1313.

[2008] Lo A.W., Hedge funds, systemic risk, and the financial crisis of 2007-2008. Written Testimony of A.W. Lo, Prepared for the US House of Representative.

[2008] Lo A.W., Patel P.N., 130/30: The new long-only. *The Journal of Portfolio Management*, pp 12–38.

[2010] Lo A.W., Hasanhodzic J., The evolution of technical analysis: Financial prediction from Babylonian tablets to Bloomberg terminals. John Wiley and Sons, USA.

[1995a] Longerstaey J., Zangari P., Five questions about RiskMetrics. Morgan Guaranty Trust Company, Market Risk Research, JPMorgan.

[1995b] Longerstaey J., More L., Introduction to RiskMetrics. 4th edition, Morgan Guaranty Trust Company, New York.

[1996] Longerstaey J., Spencer M., RiskMetrics: Technical document. Fourth Edition, Morgan Guaranty Trust Company, New York.

[1996] Longin F., The asymptotic distribution of extreme stock market returns. *Journal of Business*, **69**, pp 383–408.

[1994] Loretan M., Phillips P., Testing the covariance stationarity of heavy-tailed time series. *Journal of Empirical Finance*, **1**, pp 211–248.

[2009] Lu C-J., Lee T-S., Chiu C-C., Financial time series forecasting using independent component analysis and support vector regression. *Decision Support Systems*, **47**, (2), pp 115–125.

[2006] Lu Dang Khoa N., Sakakibara K., Nishikawa I., Stock price forecasting using back propagation neural networks with time and profit based adjusted weight factors. *SICE-ICASE International Joint Conference*, Oct., Bexco, Busa, Korea, pp 5484–5488.

[1990] Luede E., Optmization of circuits with a large number of parameters. *Archiv f. Elektr. u. Uebertr.*, Band (44), Heft (2), pp 131-138.

[1984] Luenberger D.G., Linear and nonlinear programming. Addison-Wesley.

[1988] Lukac L., Brorsen B., Irwin S., A test of futures market disequilibrium using twelve different technical trading systems. *Applied Economics*, **20**, (5), pp 623–639.

[2006] Lukosevicius M., Popovici D., Jaeger H., Siewert U., Time warping invariant echo state networks Technical Report No. 2, Jacobs University Bremen.

[2007] Lukosevicius M., Echo state networks with trained feedbacks. Technical Report No. 4, Jacobs University Bremen.

[2009] Lukosevicius M., Jaeger H., Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, **3**, (3), pp 127–149.

[2012] Lukosevicius M., A practical guide to applying echo state networks. Jacobs University Bremen, Germany.

[2012] Lukosevicius M., Jaeger H., Schrauwen B., Reservoir computing trends. *Kunstliche Intelligenz*, Springer, **26**, (4), pp 365–371.

[1996] Lux T., The stable Paretian hypothesis and the frequency of large returns: An examination of major German stocks. *Applied Economics Letters*, **6**, pp 463–475.

[2012] Lye C-T., Hooy C-W., Multifractality and efficiency: Evidence from Malaysian sectoral indices. *Int. Journal of Economics and Management*, **6**, (2), pp 278–294.

[1998] Maass W., Orponen P., On the effect of analog noise in discrete-time analog computations. *Neural Computation*, **10**, pp 1071–1095.

[2002] Maass W., Natschlager T., Markram H., Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, **14**, (11), pp 2531–2560.

[2005a] Maass W., Joshi P., Sontag E.D., Computational aspects of feedback in neural circuits. Working Paper, It was published in 2007 in *PLoS Computational Biology*.

[2005b] Maass W., Joshi P., Sontag E.D., Principles of real-time computing with feedback applied to cortical microcircuit models. Conference Paper in *Advances in neural information processing systems*, January.

[1977] Mackey M.C., Glass L., Oscillation and chaos in physiological control systems. *Science*, **197**, (4300), pp 287–289.

[1993] Maheswaran S., Sims C., Empirical implications of arbitrage-free asset markets. in P.C.B. Phillips, ed., *Models, Methods and Applications of Econometrics*, Cambridge, Basil Blackwell, pp 301–316.

[2011] Maknickiene N., Vytautas Rutkauskas A., Maknickas A., Investigation of financial market prediction by recurrent neural network. *Innovative Infotechmologies for Science, Bussiness and Education*, **2**, (11), pp 3–8.

[1982] Makridakis S., Andersen A., Carbon R., Fildes R., Hibon M., Lewandowski R., Newton J., Parzen R., Winkler R., The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, **1**, pp 111–153.

[1989] Mallat S.G., A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**, (7), pp 674–693.

[1990] Mallat S.G., Hwang W.L., Singularity detection and processing with wavelets. Technical Report No. 549, Computer Science Department, New York University.

[1992] Mallat S.G., Hwang W.L., Singularity detection and processing with wavelets. *IEEE Trans. Inform. Theory*, **38**, pp 617–643.

[1992b] Mallat S.G., Zhong S., Complete signal representation with multiscale edges. *IEEE Trans.*, PAMI **14**, pp 710–732.

[1960] Mandelbrot B.B., The Pareto-Levy law and the distribution of income. *International Economic Revue*, **1**.

[1963a] Mandelbrot B.B., New methods in statistical economics. *The Journal of Political Economy*, **71**.

[1963] Mandelbrot B.B., The variation of certain speculative prices. *Journal of Business*, **36**, (4), pp 394–419.

[1971] Mandelbrot B.B., When can price be arbitraged efficiently? A limit to the validity of the random walk and martingale models. *Re. Econom. Statist.*, **53**, pp 225–236.

[1979] Mandelbrot B.B., Taqqu M., Robust R/S analysis of long-run serial correlation. in *Proceedings of the 42nd Session of the International Statistical Institute*, , pp 69–104, Manila, Bulletin of the I.S.I..

[1982] Mandelbrot B.B., The fractal geometry of nature. W.H. Freeman and Company, New York.

[1997] Mandelbrot B.B., Fisher A., Calvet L., A multifractal model of asset returns. Cowles Foundation Discussion Paper No. 1164.

[2005] Manimaran P., Panigrahi P.K., Parikh J.C., Wavelet analysis and scaling properties of time series. *Phys. Rev. E*, **72**, 046120, pp 1–5.

[2009] Manimaran P., Panigrahi P.K., Parikh J.C., Multiresolution analysis of fluctuations in non-stationary time series through descrete wavelets. *Physica A*, **388**, pp 2306–2314.

[2000] Mantegna R.N., Stanley H.E., An introduction to econophysics: Correlation and complexity in finance. Cambridge University Press, Cambridge.

[2003] Matia K., Ashkenazy Y., Stanley H.E., Multifractal properties of price fluctuations of stocks and commodities. *Europhysics Letters*, **61**, (3), pp 422–428.

[2004] Matos J.A.O., Gama S.M.A., Ruskin H., Duarte J., An econophysics approach to the Portuguese stock index-psi-20. *Physica A*, **342**, (3-4), pp 665–676.

[2008] Matos J.A.O., Gama S.M.A., Ruskin H.J., Al Sharkasi A., Crane M., Time and scale Hurst exponent analysis for financial markets. *Physica A*, **387**, pp 3910–3915.

[1943] McCulloch W., Pitts W., A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, pp 115–133.

[1996] McCulloch J.H., Financial applications of stable distributions. in G.S. Maddala, C.R. Rao, eds., *Handbook of Statistics*, **14**, Elsevier, pp 393–425.

[1997] McCulloch J.H., Measuring tail thickness to estimate the stable index α: A critique. *Journal of Business & Economic Statistics*, **15**, (1), pp 74–81.

[2008] McKay B., Evolutionary Algorithms. Encyclopedia of Ecology, Seoul National University, Seoul, Republic of Korea, pp 1464–1472.

[2010] Menkhoff L., The use of technical analysis by fund managers: International evidence. *Journal of Banking & Finance*, **34**, (11), pp 2573–2586.

[1981] Merton R.C., On market timing and investment oerformance: An equilibrium theory of value for market forecasts. *Journal of Business*, **54** (3), pp 363–406.

[2007] Metghalchi M., Glasure Y., Garza-Gomez X., Chen C., Profitable technical trading rules for the Austrian stock market. *International Business & Economics Research Journal*, **6**, (9), pp 49–58.

[2012] Metghalchi M., Marcucci J., Chang Y-H., Are moving average trading rules profitable? Evidence from the European stock markets. *Applied Economics*, **44**, (12), pp 1239–1559.

[2015] Metghalchi M., Chen C., Hayes L.A., History of share prices and market efficiency of the Madrid general stock index. *International Review of Financial Analysis*, **40**, pp 178–184

[2018] Metghalchi M., Hajilee M., Hayes L.A., Return predictability and efficient market hypothesis: Evidence from Iceland. *The Journal of Alternative Investments*, **21**, (1), pp 68–78.

[2004a] Mezura-Montes E., Coello Coello C.A., Tun-Morales E.I., Simple feassibility rules and differential evolution for constrained optimization. *Third Mexican International Conference on Artificial Intelligence, MICAI, Lecture Notes in Artificial Intelligence*, pp 707–716.

[2004] Mezura-Montes E., Coello Coello C.A., A study of mechanisms to handle constraints in evolutionary algorithms. Workshop at the Genetic and Evolutionary Computation Conference, Seattle, Washington, ISGEC.

[2006] Mezura-Montes E., Velazquez-Reyes J., Coello Coello C.A., Modified differential evolution for constrained optimization. *IEEE Congress on Evolutionary Computation*, IEEE Press, pp 332–339.

[2006b] Mezura-Montes E., Coello Coello C.A., Velazquez-Reyes J., Munoz-Davila L., Multiple offspring in differential evolution for engineering design. *Engineering Optimization*, **00**, pp 1–33.

[1996] Michalewicz Z., Genetic algorithms + data structures = evolution programs. Springer Verlag.

[2003] Mikosch T., Starica C., Long-range dependence effects and ARCH modeling. In *Theory and Applications of Long-range Dependence*, Birkhauser Boston, Boston, pp 439–459.

[2009] Mingguang L., Gaoyang L., Artificial neural network co-optimisation algorithm based on differential evolution. In *Second International Symposium on Computational Intelligence and Design*, pp 256–559.

[1997] Mitchell T.M., Machine learning. McGraw-Hill.

[2003] Mittelmann H.D., An independent benchmarking of SDP and SOCP solvers. *Mathematical Programming*, **95**, (2), pp 407–430.

[1999] Mladenov V.M., Maratos N.G., Neural networks for solving constrained optimization problems. Working Paper, Technical University of Sofia.

[1989] Montana D.J., Davis L., Training feedforward neural networks using genetic algorithms. *Int Jt Conf Artif Intell*, **89**, pp 762–767.

[2006] Moyano L.G., de Souza J., Duarte Queiros S.M., Multi-fractal structure of traded volume in financial markets. *Physica A*, **371**, pp 118–121.

[1990] Muller U.A., Dacorogna M.M., Olsen R.B., Pictet O.V., Schwarz M., Morgenegg C., Statistical study of foreign exchange rates, empirical evidence of a price change scaling law, and intraday analysis. *Journal of Banking and Finance*, **14**, pp 1189–1208.

[1997] Muller U.A., Dacorogna M.M., Dave R.D., Olsen R.B., Pictet O.V., von Weizsacker J.E., Volatilities of different time resolutions: Analyzing the dynamics of market components. *Journal of Empirical Finance*, **4**, pp 213–239.

[2018] Murdoch W.J., Liu P.J., Yu B., Beyond word importance: Contextual decomposition to extract interactions from lstms. International Conference on Learning Representations.

[2009] Murguia J.S., Perez-Terrazas J.E., Rosu H.C., Multifractal properties of elementary cellular automata in a discret wavelet approach of MF-DFA. *EPL Journal*, **87**, pp 2803–2808.

[1999] Murphy J.J., Technical analysis of the financial markets: A comprehensive guide to trading methods and applications. Prentice Hall Press.

[2003] Murtagh F., Stark J.L., Renaud O., On neuro-wavelet modeling. Working Paper, School of Computer Science, Queen's University Belfast.

[1991] Muzy J.F., Bacry E., Arneodo A., Wavelets and multifractal formalism for singular signals: Application to turbulence data. *Phys. Rev. Lett.*, **67**, (25), pp 3515–3518.

[1993] Muzy J.F., Bacry E., Arneodo A., Multifractal formalism for fractal signals: The structure-function approach versus the wavelet-transform modulus-maxima method. *Phys. Rev. E*, **47**, (2), pp 875–884.

[2011] Nair B.B., Sai S.G., Naveen A., Lakshmi A., Venkatesh G., Mohandas V., A GA-artificial neural network hybrid system for financial time series forecasting. *Inf Techno Mob Commun.*, **147**, pp 499–506.

[2014] Nanculef R., Frandi E., Sartori C., Allendea H., A novel Frank-Wolfe algorithm. Analysis and applications to large-scale SVM training *Inf. Sci.*, **285**, pp 66–99 .

[1990] Nelson D.B., ARCH models as diffusion approximations. *Journal of Econometrics*, **45**, pp 7–38.

[1991] Nelson D.B., Conditional heteroskedasticity in asset returns: A new approach. *Econometrica*, **59**, pp 347–370.

[2013] Niere H.M., A multifractality measure of stock market efficiency in Asean region. *European Journal of Business and Management*, **5**, (22), pp 13–19.

[1999] Nocedal J., Wright S.J., Numerical optimization. Sringer Verlag.

[2011] Noman N., Bollegala D., Iba H., An adaptive differential evolution algorithm. *IEEE*, pp 2229–2236.

[2005] Norouzzadeh P., Jafari G.R., Application of multifractal measures to Tehran price index. *Physica A*, **356**, pp 609–627.

[2006] Norouzzadeh P., Rahmani B., A multifractal detrended fluctuation description of Iranian rial-US dollar exchange rate. *Physica A*, **367**, pp 328–336.

[2012] Oh G., Eom C., Havlin S., Jung W-S., Wang F., Stanley H.E., Kim S., A multifractal analysis of Asian foreign exchange markets. *European Physical Journal B*, pp 85–214.

[1993] Ohlsson M., Peterson C., Soderberg B., Neural networks for optimization problems with inequality constraints: The knapsack problem. *Neural Computation*, **5**, (2), pp 331–339.

[2003] Olson D., Mossman C., Neural network forecasts of Canadian stock returns using accounting ratios. *International Journal of Forecasting*, **19**, pp 453–465.

[2004] Onwubolu G.C., Differential evolution for the flow shop scheduling problem. in *New Optimization Techniques in Engineering*, G.C. Onwubolu and B. Babu, Eds., Springer-Verlag, Berlin, pp 585–611.

[2013] Fernando F.E.B., Freitas A.A., Rice j., Johnson C.G., New sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions on Evolutionary Computation*, **17**, (1), pp 64–76.

[2002] Palmer A., Montano J.J., Redes neuronales artificiales aplicadas al analisis de datos. Doctoral Dissertation. University of Palma de Mallorca.

[1998] Papageorgiou G., Likas A., Stafylopatis A., A hybrid neural optimization scheme based on parallel updates. *International Journal of Computer Mathematics*, **67**, pp 223–237.

[1985] Parker D.B., Learning logic report TR-47. MIT Press

[1997] Partridge D., Krzanowski W.J., Software diversity: Practical statistics for its measurement and exploitation. *Information & Software Technology*, **39**, (10), pp 707–717.

[2000] Pasquini M., Serva M., Clustering of volatility as a multiscale phenomenon. *European Physical Journal B*, **16**, (1), pp 195–201.

[2015] Patel J., Shah S., Thakkar P., Kotecha K., Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques. *Expert Systems with Applications*, **42**, pp 259–268.

[2008a] Pedersen M.E.H., Chipper

eld A.J., Parameter tuning versus adaptation: Proof of principle study on differential evolution. Technical Report HL0802, Hvass Laboratories.

[2008b] Pedersen M.E.H., Chipper

eld A.J., Tuning differential evolution for artificial neural networks. Hvass Laboratories Technical Report no. HL0803.

[2010] Pedersen M.E.H., Tuning and simplifying heuristical optimization. PhD thesis, Computational Engineering and Design Group, School of Engineering Sciences, University of Southampton.

[1994] Peng C.K., Buldyrev S.V., Havlin S., Simons M., Stanley H.E., Goldberger A.L., Mosaic organization of DNA nucleotides. *Physical Review E*, **49**, (2), pp 1685–1689.

[1997] Polak E., Optimization: Algorithms and consistent approximations.

[2015] Polydoros A.S., Nalpantidis L., Kruger V., Advantages and limitations of reservoir computing on model learning for robot control. IROS Workshop on Machine Learning in Planning and Control of Robot Motion

[1993] Pomerleau D.A., Knowledge-based training of artificial neural networks for autonomous robot driving. in J. Connell and S. Mahadevan, (eds.), *Robot Learning*, pp 19–43, Kluwe Academic Publishers.

[1992] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P., Numerical recipes. 2nd ed. Cambridge, Cambridge University Press.

[2005] Price K., Storn R., Lampinen J., Differential evolution: A practical approach to global optimization. Springer, Heidelberg.

[2001] Qi M., Predicting US recessions with leading indicators vianeural network models. *International Journal of Forecasting*, **17**, pp 383–401.

[2016] Qiu M., Song Y., Predicting the direction of stock market index movement using an optimized artificial neural network model. PLoS ONE **11** ,(5): e0155133. doi:10.1371/journal.

[1993] Quinlan J.R., C4.5: Programs for machine Learning. Morgan Kaufmann, San Francisco, CA.

[1998] Quinlan J.R., Induction of decision trees. *Machine Learning*, **1**, (1), pp 81–106.

[1993] Rabemananjara R., Zakoian J., Threshold ARCH models and asymmetries in volatility. *Journal of Applied Econometrics*, **8**, pp 31–49.

[1999] Ramsey J.B., The contribution of wavelets to the analysis of economic and financial data. *Phil. Trans. R. Soc.*, **357**, pp 2593–2606.

[2017] Rani T., Kumar N., *International Journal of Advanced Research in Computer and Communication Engineering*, **6**, (5), pp 616–623.

[1995] Refenes A.N., Neural networks in capital markets. Wiley.

[1994] Refenes A.N., Zapranis A.D., Francis G., Stock performance modeling using neural network: A comparative study with regression models. *Neural Network*, **5**, pp 961–970.

[1997] Refenes A.N., Bentz Y., Bunn D.W., Burgess A.N., Zapranis A.D., Financial time series modeling with discounted least squares backpropagation. *Neurocomputing*, **14**, (2), pp 123–138.

[2008] Reinhart R.F., Steil J.J., Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant PA-10 robot. In Learning and Adaptive Behaviors for Robotic Systems (LAB-RS), pp 35–40.

[2009] Reinhart R.F., Steil J.J., Attractor-based computation with reservoirs for online learning of inverse kinematics. Research Institute for Cognition and Robotics (CoR-Lab), Bielefeld University.

[2009b] Reinhart R.F., Steil J.J., Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot icub. In IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp 323–330.

[2011] Reinhart R.F., Reservoir computing with output feedback. Technische Fakultät, Universität Bielefeld.

[2011] Reinhart R.F., Steil J.J., A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, **19**, pp 27–46.

[1999] Resnick S., Samorodnitsky G., Xue F., How misleading can sample ACFs of stable MAs be? (Very!). *Ann. Appl. Prob.*, **9**, pp 797–817.

[2000] Resnick S., Van Den Berg E., Sample correlation behavior for the heavy tailed general bilinear process. *Comm. Statist. Stochastic Models*, **16**, pp 233–258.

[2016] Ribeiro M.T., Singh S., Guestrin C., Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 1135–1144.

[1995] Rice J.A., Mathematical statistics and data analysis. Thomson Information, Second Edition.

[1987] Robinson A.J., Fallside F., The utility driven dynamic error propagation network. Cambridge University Engineering Department

[2012] Rodan A., Tiňo P., Simple deterministically constructed cycle reservoirs with regular jumps. The University of Birmingham.

[2010] Rokach L., Pattern classification using ensemble methods. World Scientific, Singapore.

[1962] Rosenblatt F., Principles of perceptrons. Spartan, Washington, DC.

[1999] Ross B.J., A lamarckian evolution strategy for genetic algorithms. In Lance D. Chambers, editor, *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, **3**, pp 1–16. CRC Press, Boca Raton, Florida.

[2017] Ross A.S., Hughes M.C., Doshi-Velez F., Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI-17, pp 2662–2670.

[1986] Rumelhart D., McClelland J., Parallel distributed processing. MIT Press, Cambridge, Mass.

[1986b] Rumelhart D.E., Hinton G.E., Williams R.J., Learning internal representations by error propagation. in Parallel distributed processing: explorations in the microstructure of cognition, **1**, MIT Press, Cambridge, Mass, pp 318–362.

[1994] Rumelhart D., Widrow B., Lehr M., The basic ideas in neural networks *Communications of the ACM*, **37**, (3), pp 87–92.

[1959] Samuel A.L., Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**, (3), pp 535–554.

[2015] Sangwan K.S., Saxena S., Kant G., Optimization of machining parameters to minimize surface roughness using integrated ANN-GA approach. in *Proceedings of the 22nd CIRP Conference on Life Cycle Engineering*, **29**, Sydney, Australia, pp 305–310.

[2005] Santana-Quintero L.V., Coello Coello C.A., An algorithm based on differential evolution for multi-objective problems. *International Journal of Computational Intelligence Research*, **1**, ISSN 0973-1873, pp 151–169.

[1990] Schapire R.E., The strength of weak learnability. *Machine Learning*, **5**, (2), pp 197–227.

[1992] Schmidhuber J., A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running network. *Neural Computation*, **4**, (2), pp 243–248.

[2007] Schmidhuber J., Wierstra D., Gagliolo M., Gomez F.J., Training recurrent networks by Evolino. *Neural Computation*, **19**, (3), pp 757–779.

[1999] Schmitt F., Schertzer D., Lovejoy S., Multifractal analysis of foreign exchange data. *Applied Stochastic Models and Data Analysis*, **15**, pp 29–53.

[2005] Schulmeister S., Components of the profitability of technical currency trading. WIFO Working Papers, No. 263, December.

[1996] Schwager W.F., Technical analysis. Wiley.

[1995] Schwefel H.P., Evolution and optimum seeking. John Wiley & Sons.

[2008] Senol D., Ozturan M., Stock price direction prediction using artificial neural network approach: The case of Turkey. *Journal Artif Intell*, **1**, (2), pp 70–77.

[2008] Sherrod P.H., Dtreg: predictive modeling software.

[1991] Shin Y., Ghosh J., The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. *International Joint Conference on Neural Networks*.

[1998] Shin K-S., Kim K-J., Han I., Financial data mining using genetic algorithms technique: Application to KOSPI 200. Korea Advanced Institute of Science and Technology

[2000] Shin T., Han I., Optimal signal multi-resolution by genetic algorithms to support financial neural networks for exchange-rate forecasting. *Expert Syst. Appl.*, **18**, pp 257–269.

[2002] Shipp C.A., Kuncheva L.I., Relationships between combination methods and measures of diversity in combining classifiers. *Information Fusion*, **3**, (2), pp 135–148.

[1985] Shor N.Z., Minimization methods for non-differentiable functions. Springer-Verlag.

[2012] Shynkevich A., Performance of technical analysis in growth and small cap segments of the US equity market. *Journal of Banking & Finance*, **36**, pp 193–208.

[2012] Si T., Hazra S., Jana N.D., Artificial neural network training using differential evolutionary algorithm for classification. Department of Information Technology National Institute of Technology, Durgapur West Bengal, India.

[1991] Siegelmann H.T., Sontag E.D., Turing computability with neural nets. *Applied Mathematics Letters*, **4**, (6), pp 77–80.

[2007] Siewert U., Wustlich W., Echo-state networks with bandpass neurons: Towards generic time-scale-independent reservoir structures. Internal Status Report, *PLANET Intelligent Systems GmbH*.

[1996] Skalak D.B., The sources of increased accuracy for two proposed boosting algorithms. In *Working Notes of the AAAI'96 Workshop on Integrating Multiple Learned Models*, Portland, OR.

[2008] Slowik A., Bialko M., Training of artificial neural networks using differential evolution algorithm. Department of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland.

[1989] Smith M.J., Portmann C.L., Practical design and analysis of a simple neural optimization circuit. *IEEE Trans. Circuit and Systems*, **36**, pp 42–50.

[1973] Sneath P.H.A., Sokal R.R., Numerical taxonomy: The principles and practice of numerical classification. W.H. Freeman, San Francisco, CA.

[2015] Sondwale P., Overview of predictive and descriptive data mining Techniques. *International Journal of Advanced Research in Computer Science and Software*, **5**, (4), pp. 262–265.

[2008] Sorokina D., Caruana R., Riedewald M., Fink D., Detecting statistical interactions with additive groves of trees. In *Proceedings of the 25th international conference on Machine learning*, pp 1000–1007

[2004] Steil J.J., Backpropagation-decorrelation: Online recurrent learning with O(N) complexity. Neuroinformatics Group, Faculty of Technology University of Bielefeld. In *Proceedings of the IEEE International Joint Conference on Artificial Neural Networks*, **2**, pp 843–848.

[2005] Steil J.J., Memory in backpropagation-decorrelation $O(N)$ efficient online recurrent learning. In *Proceedings of the 15th International Conference on Artificial Neural Networks*, **3697** of LNCS, pp 649–654.

[2012] Stepanek J., Stovicek J., Cimler R., Application of genetic algorithms in stock market simulation. Cyprus International Conference on Educational Research (CY-ICER-2012) North Cyprus, US08-10.

[1995] Storn R., Price K., Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. International Computer Science Institute, Berkeley, **TR-95-012**.

[1996] Storn R., System design by constraint adaptation and differential evolution. International Computer Science Institute, Berkeley, **TR-96-039**.

[1997] Storn R., Price K., Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, **11**, (4), pp 341–359.

[2000] Storn R., Digital filter design program. FIWIZ.

[2008] Storn R., Differential evolution research: Trends and open questions. in U.K. Chakraborty, editor, *Advances in Differential Evolution*, **1**, pp 1–31, Springer-Verlag, Berlin Heidelberg.

[2014] Stosic D., Stosic D., Stosic T., Stanley H.E., Multifractal analysis of managed and independent float exchange rates. Department of Physics, Boston University. Published in 2015 in *Physica A: Statistical Mechanics and its Applications*, **428**, pp 13–18.

[1998] Struzik Z.R., Removing divergence in the negative moments of the multi-fractal partition function with the wavelet transformation. Working Paper INS-R9803, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[1999] Struzik Z.R., Local effective Holder exponent estimation on the wavelet transform maxima tree. in *Fractals: Theory and Applications in Engineering*, eds., M. Dekking, J. Levy Vehel, E. Lutton, C. Tricot, Springer Verlag, pp 93–112.

[2000] Struzik Z.R., Determining local singularity strengths and their spectra with the wavelet transform. *Fractals*, **8**, (2), pp 163–179.

[2002] Struzik Z.R., Siebes A., Wavelet transform based multifractal formalism in outlier detection and localisation for financial time series. *Physica A*, **309**, pp 388–402.

[2003] Struzik Z.R., Econophysics vs cardiophysics: The dual face of multifractality. Working Paper, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[1999] Sun X., The Lasso and its implementation for neural networks. Department of Statistics, University of Toronto.

[1999] Sun W., Yuan Y-X., Optimization theory and methods: Nonlinear programming. Springer.

[2009] Sussillo D., Abbott L.F., Generating coherent patterns of activity from chaotic neural networks. *Neuron*, **63**, (4), pp 544–557.

[1986] Sweeney R., Beating the foreign exchange market. *The Journal of Finance*, **41**, (1), pp 163–182.

[1996] Takefuji Y., Wang J., Neural computing for optimization and combinatorics. World Scientific Publishing Co.

[1995] Tan H., Neural network model for stock forecasting. Master's Thesis, Texas Tech University.

[2009] Tan C., Financial time series forecasting using improved wavelet neural network. Master's Thesis, Aarhus Universitet.

[2007] Tanaka-Yamawaki M., Tokuoka S., Adaptive use of technical indicators for the prediction of intra-day stock prices. *Physica A*, **383**, pp 125–133.

[2014] Tang F., Tiňo P., Chen H., Learning the deterministically constructed echo state networks. International Joint Conference on Neural Networks.

[1986] Tank D.W., Hopfield J.J., Simple neural optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Trans. Circuits and Systems*, CAS-33, pp 533–541.

[1981] Taqqu M.S., Self-similar processes and related ultraviolet and infrared catastrophes. In Random Fields : Rigorous Results in Statistical Mechanics and Quantum Field Theory, *Colloquia Mathematica Societatis Janos Bolya*, Vol. 27, Book 2, pp 1027–1096.

[1995] Taqqu M.S., Teverovsky V., Willinger W., Estimators for long-range dependence: An empirical study. *Fractals*, **3**, (4), pp 785–798.

[1986] Taylor S.J., Modelling financial time series. New York, John Wiley & Sons.

[2007] Taylor G.W., Hinton G.E., Roweis S., Modeling human motion using binary latent variables. in *Advances in Neural Information Processing Systems*, **19**, pp 1345–1352, MIT Press, Cambridge.

[2008] Terman D.H., Izhikevich E.M., State space. Scholarpedia, http://www.scholarpedia.org/article/Phase space, Ohio State University.

[2004] Timmermann A., Granger C.W.J., Efficient market hypothesis and forecasting. *International Journal of Forecasting*, **20**, pp 15–27.

[2018] Tsang M., Cheng D., Liu Y., Detecting statistical interactions from neural network weights. Conference Paper at ICLR 2018, Department of Computer Science, University of Southern California.

[2002] Tsay R.S., Analysis of financial time series. John Wiley & Sons, Hoboken, New Jersey.

[2006] Turiel A., Perez-Vicente C.J., Grazzini J., Numerical methods for the estimation of multifractal singularity spectra on sampled data: A comparative study. *Journal of Computational Physics*, **216**, pp 362–390.

[2016] Van den Poel D., Chesterman C., Koppen M., Ballings M., Equity price direction prediction for day trading: Ensemble classification using technical analysis indicators with interaction effects. in *2016 IEEE Congress on Evolutionary Computation* (CEC).

[1997] Vandewalle N., Ausloos M., Coherent and random sequences in financial fluctuations. *Physica A*, **246**, (3), pp 454–459.

[1998] Vandewalle N., Ausloos M., Crossing of two mobile averages: A method for measuring the robustness exponent. *Phys. Rev.*, **58**, pp 177–188.

[1998b] Vandewalle N., Ausloos M., Multi-affine analysis of typical currency exchange rates. *European Physical Journal B.*, **4**, (2), pp 257–261.

[1998c] Vandewalle N., Ausloos M., Boveroux PH., Detrended fluctuation analysis of the foreign exchange market. Working Paper.

[2017] Villarrubia G., De Paz J.F., Chamoso P., De la Prieta F., Artificial neural networks used in optimization problems. *Neurocomputing*, **272**, pp 10–16.

[2010] Vivekanandan P., Nedunchezhian R., A fast genetic algorithm for mining classification rules in large datasets. *International Journal on Soft Computing*, **1**, (1), pp 10–20.

[2008] Wang W., Nie S., The performance of several combining forecasts for stock index. in *2008 International Seminar on Future Information Technology and Management Engineering*, pp 450–455.

[2009] Wang Y., Liu L., Gu R., Analysis of efficiency for Shenzhen stock market based on multifractal detrended fluctuation analysis. *International Review of Financial Analysis*, **18**, pp 271–276.

[2011] Wang Y., Wei Y., Wu C., Analysis of the efficiency and multifractality of gold markets based on multifractal detrended fluctuation analysis. *Physica A: Statistical Mechanics and its Applications*, **390**, pp 817–827.

[2011b] Wang Y., Wu C., Pan Z., Multifractal detrending moving average analysis on the US Dollar exchange rates. *Physica A*, **390**, pp 3512–3523.

[2008] Wendt H., Contributions of wavelet leaders and bootstrap to multifractal analysis: Images, estimation performance, dependence structure and vanishing moments. Confidence intervals and hypothesis tests. Docteur de l'Universite de Lyon, Ecole Normale Superieure de Lyon, Traitement du Signal - Physique.

[1974] Werbos P., Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University.

[1990] Werbos P., Backpropagation through time: What it does and how to do it. in *Proceedings of the IEEE*, **78**, (10), pp 1550–1560.

[2000] White H., A reality check for data snooping. *Econometrica*, **68**, pp 1097–1127.

[1990] Whitley D., Starkweather T., Bogart C., Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, **14**, (3), pp 347–361.

[2005] Wiestra D., Gomez F., Schmidhuber J., Modeling systems with internal state Evolino. In GECCO, pp 1795–1802.

[1978] Wilder W., New concepts in technical trading systems. Trend Research, Greensboro, NC.

[1989] Williams R.J., Zipser D., A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, pp 270–280.

[1990] Williams R.J., Peng J., An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural Computation*, **2**, (4), pp 490–501.

[1992] Williams R.J., Zipser D., Gradient-based learning algorithms for recurrent networks and their computational complexity. in Back-propagation: Theory, architectures and applications. NJ:Erlbaum, edt. Y.Chauvin and D.E.Rumelhart, chapter 13, pp 433–486.

[2005] Witten J.H., Frank E., Data mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2 edition.

[1992] Wolpert D.H., Original contribution: Stacked generalization. *Neural Networks*, **5**, (2), pp 241–259.

[1991] Wright A.H., Genetic algorithms for real parameter optimization. in Foundation of Genetic Algorithms, ed. G. Rawlins, *First Workshop on the Foundation of Gen. Alg. and Classified Systems*, Los Altos, CA, pp 205-218.

[2008] Wyffels F., Schrauwen B., Verstraeten D., Stroobandt D., Band-pass reservoir computing. In 2008 *IEEE International Joint Conference on Neural Networks*, pp 3204–3209.

[2009] Wyffels F., Schrauwen B., Design of a central pattern generator using reservoir computing for learning human motion. Electronics and Information Systems Department, Ghent University, Belgium.

[1995] Xu L., Jordan M.I., Hinton G.E., An alternative model for mixtures of experts. In G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems 7*, MIT Press, pp 633–640. Cambridge, MA, 1995.

[1996] Xu L., Jordan M.I., On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, **8**, (1), pp 129–151.

[2007] Xu Y., Yang L., Haykin S., Decoupled echo state networks with lateral inhibition. *Neural Networks*, **20**, (3), pp 365–376.

[1996] Yao J.T., Poh H.L., Equity forecasting: A case study on the KLSE index. *Neural Networks in Financial Engineering, Proceedings of 3rd International Conference on Neural Networks in the Capital Markets*, Oct 1995, London, A-P.N. Refenes, Y. Abu-Mostafa, J. Moody and A. Weigend, (eds.), World Scientific, pp 341–353.

[1998] Yao J.T., Tan C.L., A study on training criteria for financial time series forecasting. Working Paper.

[2000] Yao J.T., Tan C.L., Time dependent directional profit model for financial time series forecasting. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, July 2000, Como, Italy, **5**, pp 291–296.

[2006] Yasuda T., Nakamura K., Kawahara A., Tanaka K., Neural network with variable type connection weights for autonomous obstacle avoidance on a prototype of six-wheel type intelligent wheelchair. *International Journal of Innovative Computing, Information and Control*, **2**, (5), pp 1165–1177.

[2015] Yosinski J., Clune J., Nguyen A., Fuchs T., Lipson H., Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579.

[2009] Yuan Y., Zhuang X-T., Jin X., Measuring multifractality of stock price fluctuation using multifractal detrended fluctuation analysis. *Physica A*, **388**, (11), pp 2189–2197.

[1900] Yule G., On the association of attributes in statistics. *Philosophical Transactions of the Royal Society of London*, **194**, pp 257–319.

[1994] Zadeh L.A., Fuzzy logic, neural networks, and soft computing. *Communications of the ACM*, **37**, (3), pp 77–84.

[2002] Zaharie D., Critical values for the control parameters of differential evolution algorithms. in R. Matousek, P. Osmera, eds., *Proceedings of MENDEL 2002, 8th International Conference on Soft Computing*, Brno University of Technology, Faculty of Mechanical Engineering, pp 62–67, Institute of Automation and Computer Science.

[2006] Zeng F., Zhang Y., Stock index prediction based on the analytical center of version space. *Advances in Neural Networks*, **3973**, pp 458–463.

[2007] Zhang X., Chen Y., Yang J.Y., Stock index forecasting using pso based selective neural network ensemble. in *International Conference on Artificial Intelligence*, pp 260–264.

[2012] Zhou Z-H., Ensemble methods: Foundations and algorithms. Chapman & Hall, Machine Learning & Pattern Recognition Series.

[2008] Zhu X., Wang H., Xu L., Li H., Predicting stock index increments by neural networks: The role of trading volume under different horizons. *Expert Syst. Appl.*, **34**, (4), pp 3043–3054.

[2005] Zielinski K., Peters D., Laur R., Stopping criteria for single-objective optimization. In Proceedings of the Third International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore.

[2006] Zielinski K., Laur R., Constrained single-objective optimization using differential evolution. in *2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp 223–230.

[2007] Zielinski K., Laur R., Stopping criteria for a constrained single-objective particle swarm optimization algorithm. *Informatica*, **31**, pp 51–59.

[2008] Zielinski K., Laur R., Stopping criteria for differential evolution in constrained single-objective optimization. in Chakrabotry, Ed., Advances in Differential Evolution, **143**, Springer-Verlag, Berlin, pp 111–138.

[2006] Zimmermann H.G., Grothmann R., Schaefer A.M., Tietz C., Identification and forecasting of large dynamical systems by dynamical consistent neural networks. in *New Directions in Statistical Signal Processing: From systems to brain*. MIT Press, edt S.Haykin, J.Principe, T.Sejnowski, J.McWhirter, pp 203–242.

[2007] Zunino L., Tabak B.M., Perez D.G., Garavaglia M., Rosso O.A., Inefficiency in Latin-American market indices. *The European Physical Journal B*, **60**, (1), pp 111–121.

[2008] Zunino L., Tabak B.M., Figlioa A., Perez D.G., Garavaglia M., Rosso O.A., A multifractal approach for stock market inefficiency. *Physica A*, **387**, pp 6558–6566.