Università degli Studi di Torino

Scuola di Management ed Economia

Corso di Laurea in Quantitative Finance and Insurance

Behaviour Simulation and Artificial Neural Networks in Loss Reserving

Candidato:
Luca Piero Aldo Rosazza Gat

Relatore:
Chiar.mo Prof. Pietro Terna

Correlatore:
Chiar.mo Prof. Sergio Margarita

November 2014

Acknowledgements

I would like to express my very great appreciation to Professor Pietro Terna for his guidance. His willingness to give his time and energies so generously has been fundamental.
I would also like to offer my special thanks to Professor Sergio Margarita.
I would like to thank my girlfriend and my friend Dario for their contribute to this work.
I thank my parents for their constructive suggestions.
I am particularly grateful for the help given by my University mates. Special thanks to Cristian.
Many thanks to Alessandro and Alessandro of Ernst & Young for the help and the availability demonstrated.
Many thanks to Gene Sher for the copy of his book he sent me as a gift.

Contents

Introduction

Financial and Insurance businesses are facing a challenging period, due to the amendments introduced by the new European Regulation Solvency II, the Basel III Regulatory Standard, the International Accounting Standards (IAS) and the International Financial Reporting Standards (IFRS). I propose a modelling framework for conducting Loss Reserving analysis in more compliance with the new Regulations. It is unveiling to remark the importance of Insurance Loss Reserving. Instead, I focus on its calculation techniques and the empirical way to support them and to try to overcome their limitations. Mathematical, statistical, financial and economic theories have been used to understand policyholder behaviour and quantify future liabilities and risks. I believe that there is a powerful resource to the Risk Management also from the methods and the techniques of the Complex Systems Science, especially under the new Regulation framework.

Standard reserving methods to evaluate the Loss Reserve are based on aggregated claims data. My work intends to provide a new tool in the Insurance business, which strength resides in the micro-data it analyses. It composes of two independent models:

- an Agent-Based claims simulator;
- an Artificial Neural Network regression.

I propose the thesis that Neural Networks could be able in accurately approximating the relation that links the accident features with the claim amount and development periods, providing a robust estimate of the Ultimate Claim amounts. Think of black-boxes as they have recently entered in the car Insurance business. By monitoring the journey histories and then sharing the driving performances they can provide the Insurer with a big amount of reliable data. An effective tool for modelling these data was the only missing link, which I expect to find in the Neural Network algorithm. Furthermore, I trust that making assumptions on how policyholders are likely to behave in the future is a critical issue to the Insurance industry, as these form a key aspect of the full spectrum of the business, ranging from product design and pricing to reserving and risk management. To this aim, it is valuable to modelling policyholders' behaviours in the Insurance business. Purpose of this study is therefore to simulate an agent *adaptive behaviour* and to use a Neural Network algorithm in attempt to *predict Insurance losses* arising from that behaviour.

This work divides in four Parts. It starts from the review of the Technical Reserves, including their regulation and their calculation methods. In the Second Part it presents the literature on the Methods from the Complex Systems Science that translate my proposal into empirical models, that I develop in the Third Part. In the last Part I suggest further developments and new application fields of what emerged from my analysis.

I introduce the system development methodology in the figure below. The NetLogo Agent-Based Model simulation explores the behaviour of a cognitive agent, which I place in a traffic context. This agent possesses all the reactive, deliberative and affective capabilities describing the human nature. A bidding process makes them to participate in the over-all decision-making process, by trying to impose their own decision, according to instantaneous context the agent faces at a certain moment in time. As a result, dependently on which agency wins the competition, the agent behaves in three ways, respectively:

- calculating the collision possibility using manipulations of standard equations of motion and accordingly acting toward safety;
- targeting optimality from safety, politeness and rule obeying point of view, by following some engineering collision-avoidance models (see Gipp's model, 5.3.3), enforcing safe following distances;
- following the variety of human emotional responses applicable to driver, studied in transportation psychology and behaviour.

While the adaptive behaviour simulation is running, the interactions results are reported on a file, building an Insurance claims dataset, which reports for each claim the date of injury, the number of development periods, the operational time, the amount paid and several features characterising the accident.

At a first stage, several feed-forward multilayer perceptrons having different architectures are fed with part of the dataset as examples, through an in-sample training procedure. Then their predictions on the remaining dataset part are assessed through the sum of squared residual goodness of fit measure. The best neuron-inspired processor is the one that reconciles the highest number of elements in the hidden layer, which determines the number of weights in the regression, with the generalisation capability of the Network. The latter, i.e. the ability to react in a coherent way to not explicitly seen during the learning or imperfect inputs, suffers whether too many neurons are settled in the structure. The neuron in a biological Neural Network can adapt, and change its functionality over time, which too can be done in the biologically inspired device, used for mapping a set of inputs into a set of outputs, through simulated neural plasticity, i.e. the capability to autonomously build a representation of the map from inputs to outputs on the basis of a set of examples. Yet, Neural Network algorithm is consistent, since the estimated regression function is identified by a limited number of components. It has enough flexibility to allow the user to develop the best or most optimal models by varying parameters during the training process, allowing to approximate almost each regression function.

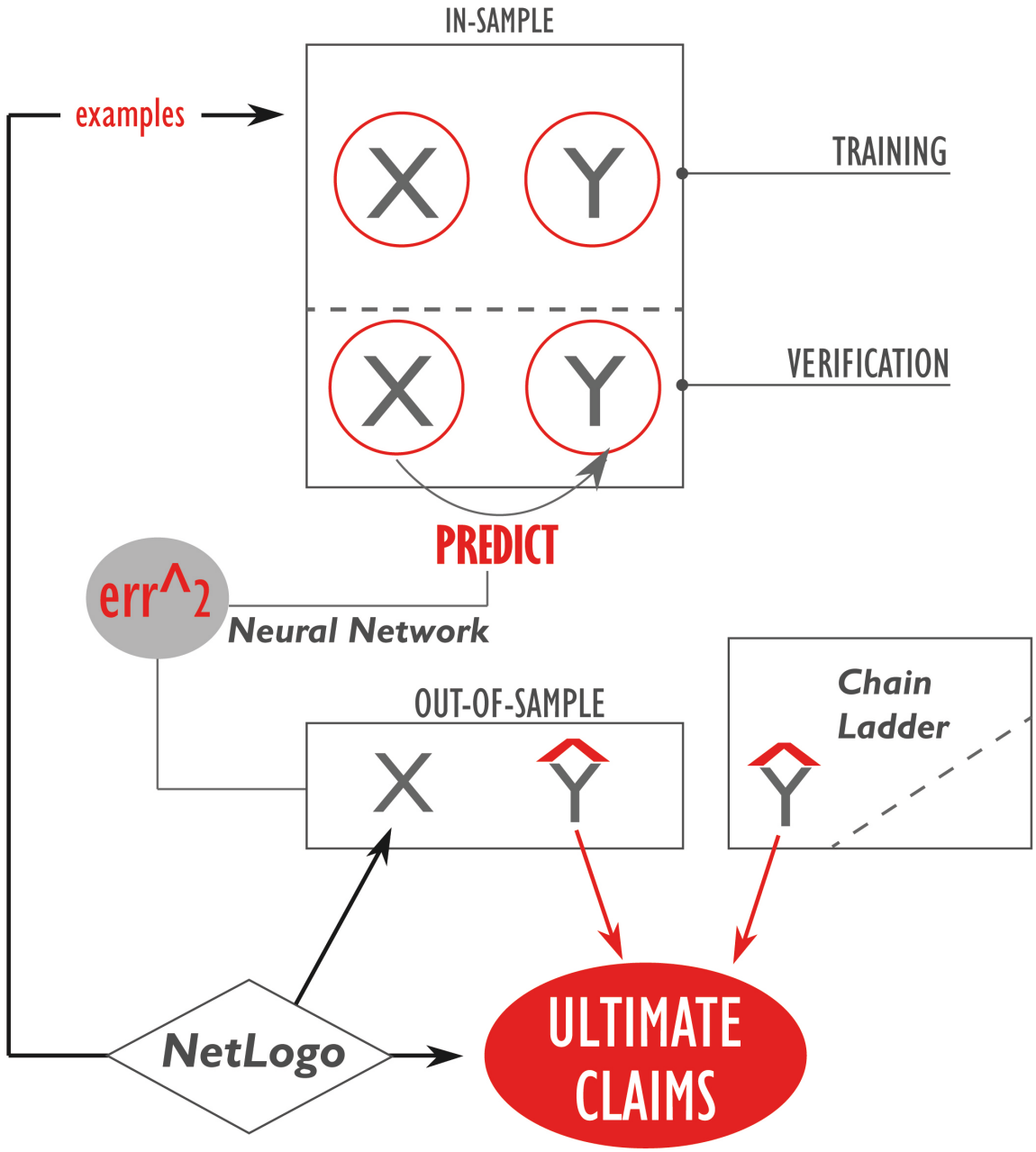

Figure 1: The system development methodology

At a later stage, I use the selected trained Neural Network to perform a regression on the simulated claims dataset to approximate the non-linear relation that links the accident's features to the amount and timing of the claim finalisation. The idea is that, since there is statistical evidence (ACI-ISTAT, 2012) that features characterising an accident concur in the claim amount determination and this in turn concur in the operational time (Taylor and McGuire, 2004) determination, it would be of great value a mechanism that links these behaviour directly to Loss Reserving.

At a last stage, I compare the performance of this powerful self learning mechanisms with that of a classical Chain Ladder method, in order to predict the ultimate dataset claims cost. I provide both the methods with the data in triangular form. Then, I try to implement the innovative procedure of ultimate claims from accident features Neural estimation. This could be an harder task for the regression.

Motivations

I am really concerned in the Risk Management. I am particularly interested in the process of convergence towards the Solvency II framework and I find this could bring great opportunities for Recent Graduates. I really like to learn new and various computational and programming tools to implement in several context. The aim of this Research is that of a business card towards the Risk Management field.

Part I

Reserves

Chapter 1

The Technical Reserves regulation

Insurance is the economic operation that allows you to guarantee against the harmful consequences (in terms of asset damage) of the occurrence of a given risk. This tool operates through the allocation of these harmful consequences among a variety of subjects exposed to the same kind of risk. This mechanism works due to the probability calculus. It allows to predict how many times a particular risk will occur at a certain time within a specified set of subjects. Consequently, it is possible to calculate the average amount that each of the subjects in the group will have to pay in order to compensate those who have actually suffered the damage. This is equivalent to transforming the individual risks in collective ones, so that each insured can be guaranteed against a damaging event bearing a much lower cost than the damage he might suffer in the case of risk verifying.

Technical reserves are specific features of insurance business, linked to its shape. Indeed, the insurance business is characterized by a reversal production cycle. Every other enterprise suffers some production costs and then earns an income by selling its output. Therefore, a regular cycle consists of costs, then income and only at the end, from the difference between income minus costs, of an eventual gain. On the contrary, insurance companies earns the entire income at the beginning of the production cycle, since the legislation established that the customer must pay the premium at the policy subscription or alternatively at the beginning of any insurance period. The premium paid represents the whole activity's price. This led insurance Companies to bear the solvency risk. It follows the necessity that the companies use this income to cover the future costs, avoiding any immediate dividend distribution, which could lead the company to serious solvency problems.

The legislator establishes that any insurance company must write in the statement of financial position under the liabilities an amount equivalent to the one it will have to pay the insured under the insurance contract in addition to another amount equivalent to future day-to-day expenses. Those two amounts constitute the Technical Reserves of the company. They are constituted at the beginning of the period, when it is not possible to forecast with certainty whether the

company will earn a gain or suffer a loss, and they are composed of premiums. Technical reserves are regulated by the Private Insurance Code, which state that technical reserves are compulsory, that they must cover the amount due to the customers whilst they do not have to cover those amount which relates to reinsurance companies. More precise regulation of technical reserves is set under the ISVAP regulations. Both regulations of ISVAP, now IVASS, and Private Insurance Code apply to all the insurance contracts held in the Italian portfolio, which includes all the contracts agreed by insurance companies located in Italy or any European Union country. Thus the only exception to the obligation of technical reserves constitution relates to those contracts signed by secondary companies head-quartered in foreign countries different from the European Union ones. Notice that technical reserves are not the sole reserves admitted in the insurance business. Indeed insurance companies are obliged to settle the legal reserve too, while they can freely constitute company-law reserves as well as free reserves.

Reserving is important because it affects the declared profit of the Insurance Company. Indeed, it depends not only on the actual claims paid, but on the forecasts of the claims which will have to be paid. In this way it can affect the tax paid. Moreover, reserving influences the premium rating and the future profits too. If a Company over estimate the reserves, this affect future profits. If on the contrary a Company under estimate them, this affect the solvency, affecting the quality of the business. Reserves are often the largest item of the balance sheet.

Chapter 2

Life reserves

Technical reserves of Life business differ from those of the Non-Life business since they focus on the specific kind of business run by the company. In the Life sector there are four different kind of technical reserves:

- the mathematical reserve;
- the reserve for amount to pay;
- the reserve for dividend sharing;
- the reserve for future costs.

2.1 The mathematical reserve

The mathematical reserve is the most important one in the Life business. It is calculated through actuarial criteria based on premiums. Recall the definition of actuarial saving premium (Diale, 2013)

$$P_{h+1}^{AS} = P - P_{h+1}^{N} \tag{2.1}$$

that is the sum to accumulate or to withdraw at the beginning of the year for the year of contract $h+1$ by the Company in order to face the future benefit. The saving premiums must be stored in the mathematical reserve, which value at time t correspond to the value of the contract at t. The value of an insurance contract having duration n at time t can be expressed as

$$benefit(t,n) - premiums(t,n) \tag{2.2}$$

where the first term is the actuarial value at time t of the benefits in the time interval (t,n) and the second term is the actuarial value at time t of the premiums in the time interval (t,n).

The mathematical reserve is needed in order to re-establish the actuarial equivalence between benefit and premiums. It represents the average present value of the insurance Company's future debts towards beneficiaries of Life insurance contracts. Mathematical reserve calculation bases on the same elements as the premium calculation: the mortality, disease and disability rates and the interest rate. It can be calculated under two different criteria:

- the perspective method;
- the retrospective method.

Under the perspective method the reserve is equivalent to the difference between the average present value of the Company's debt and the one belonging to the customers

$$benefit[t,n] = premiums[t,n] + V_t^P \tag{2.3}$$

where V_t^P is the perspective mathematical reserve, as it considers the future commitments, and it is an actuarial present value being the difference of two actuarial present values. While

$$benefit[t,n] > premiums[t,n] \Leftrightarrow V_t^P > 0 \tag{2.4}$$

the mathematical reserve represents the Company's debt towards the insured for the future n-t years. Conversely, while

$$benefit[t,n] < premiums[t,n] \Leftrightarrow V_t^P < 0 \tag{2.5}$$

the mathematical reserve represents the Company's credit towards the insured for the future n-t years, which is an unfavourable state for the Company itself. The retrospective method set the reserve equal to the difference between the demographic financial amount of customers' debts and the demographic financial amount of insurance Company's debts. Given x the age of the insured, $\forall t \in [0,n]$ have

$$benefit[0,n] = benefit[0,t] +_t E_x benefit[t,n] \tag{2.6}$$

$$premiums[0,n] = premiums[0,t] +_t E_x premiums[t,n] \tag{2.7}$$

$$V_0^P = benefit[0,n] - premiums[0,n] = 0 \tag{2.8}$$

by subtracting equation (2) from equation (1) get

$$benefit[0,t] - premiums[0,t] +_t E_x\big(benefit[t,n+] - premiums[t,n]\big) = 0 \tag{2.9}$$

and

$$benefit[0,t] - premiums[0,t] = \big(premiums[0,t] - benefit[0,t]\big)\frac{1}{{}_tE_x} \tag{2.10}$$

with the left term being the actuarial capitalization of the difference between past premiums and benefit, $frac1_tE_x$ actuarial capitalization factor, as it is the reciprocal of the discount factor ${}_tE_x$. Can write

$$V_t^R = (premiums[0,t] - benefit[0,t]) \frac{1}{{}_tE_x} \tag{2.11}$$

the retrospective mathematical reserve, taking into account the past commitments. In theory it an be seen as the contract exit price, which the Company has to pay at time t. The actuarial present value at 0 of the retrospective mathematical reserve re-establish the equilibrium between premiums and benefit in the time interval $[0,t]$, as shown by

$$premiums[0,t] = benefit[0,t] + V_t^R {}_tE_x \tag{2.12}$$

The legislator state that technical reserves must be calculated separately for each contract, using the actuarial perspective method that give adequate guarantee of carefulness, taking into account both the future premiums and the future costs. On the contrary, the retrospective method is admitted only if the Company prove that it determines a reserved amount not lower than the one calculated under the perspective method. Retrospective method is also allowed whether the perspective one can not be applied due to the specific nature of the contract.

2.2 The reserve for amount to pay

Second kind of reserve in the Life insurance business is the reserve for amount to pay. It equals the amount necessary to pay:

- capitals and rents due to customers;
- policy redemption amount due to customers;
- amounts due to customers consequently to the accident verification.

The role of the reserve for amount to pay is similar to the one the outstanding claim reserve has in the Non-Life business.

2.3 The additional reserve

The additional reserve is built up with the aim of preserve from the impact of interest rate changing.

$$V_t = V_{t-1} \cdot (1+i) + (profit - loss) \cdot \sqrt{1+i} \tag{2.13}$$

where i is the interest rate, $profit$ corresponds to the premiums and $loss$ represents the benefits.

The concept of required reserve

Required reserve is the amount the company must reserve in order to meet the policyholders, including any contractual guarantees of performance.

The concept of available reserve

Available reserve is the reserve available to the company calculated on the basis of the annual achievable returns reduced by 20%.

There are three methods to estimate the additional reserve amount, they are regulated by ISVAP regulation number 21 art.36:

- without-compensation method;
- compensation between annual periods method;
- compensation between levels of financial guarantee and annual periods method.

without-compensation method

The method without compensation calculate the required reserve making use of the higher interest rate of the guaranteed yield and yield achievable, while it calculate the available reserve using the interest rate achieved by the performance.

compensation between annual periods method

The compensation between annual periods method differs from the method without compensation for the calculation of the available reserve at the end of each year. The available reserve at year end must be set equal to the required reserve.

compensation between levels of financial guarantee and annual periods method

The required reserve has to be determined using the interest rate as at the higher of the guaranteed yield and yield achievable distributed, for each level of financial guarantee contracts. The available reserve should be calculated using the interest rate the performance achieved without realign with the necessary reserve. The present value of the balance sheet must be made with the rate of return achievable.

2.4 The expense reserve

The expense reserve is defined as the balance of the present value of future expenses and the future loadings based on forecast future premiums included future financial income from the expense reserve. The reserve for future costs has to cover all the costs linked to the Company's structure,

including administrative expenses. For segregated funds products the expected financial profit should not be recognised.

2.5 The demographic reserve

The demographic reserve is built up as a cushion against periods with worse than average demographic assumptions, such as longevity increase or mortality increase for Term assurance or other products exposed to death. The reserve for dividend sharing is linked to the dividend amount due to customers. This amount can also be included in the mathematical reserve. Adjunctive reserves are regulated by the legislator or by ISVAP regulations, with the aim of increase reserves global amount.

Chapter 3

Non Life reserves

3.1 Introduction

Unlike other industries the insurance industry does not sell products as such, but promises. An insurance policy is a promise by the insurer to the policyholder to pay for future claims for an upfront received premium (Gesmann *et al.*, 2014).

Thus, it should come to no surprise that the biggest item on the liability side of an insurer's balance sheet is often the provision or reserves for future claims payments.

Those reserves can be broken down in *out-standings claim*, which are losses already reported to the insurance company and *incurred but not reported* (IBNR) claims.

Over the years several methods have been developed to estimate reserves for insurance claims. Changes in regulatory requirements, e.g. Solvency II in Europe, have fostered further research into this topic, with a focus on stochastic and statistical techniques.

In General Insurance (or Non-Life Insurance, e.g. motor, property and casualty insurance) most policies run for a period of 12 months. However, the claims payment process can take years or even decades. Therefore often not even the delivery date of their product is known to insurers.

Sources of uncertainty in the Non Life business include late claims reported, liability disputes, social source, legal sources, general inflation, changes in the nature of risk, changes in the claims handling procedures, growth in account, impact of changes in premiums rates.

Reserves in the Non Life insurance business are several. In Italy they are covered by the Regulation number 16 of ISVAP. The two most important are the premium reserve and the outstanding claims reserve.

3.2 Premium reserve

The premium reserve is set up since at the balance sheet date the risk of a contract is not totally extinct yet. Indeed the premium reserve comprises the amount needed to cover the future claims

payments related to this contract not extinct. It is composed of two parts:

- the unearned premium reserve (UPR);
- the unexpired risk reserve (URR).

3.2.1 The unearned premium reserve

The unearned premium reserve is equal to the amount of premiums written at time t but accrued during the forthcoming years, net of acquisition costs.

Every company is obliged to calculate the unearned premium reserve. ISVAP Regulation 16 introduce two different methods of the reserve estimation:

- the pro rata temporis method;
- the forfeit method.

The pro rata temporis method

The pro rata temporis method assigns to each contract at the balance sheet date the premium quota corresponding to the remaining duration of the contract in a linear way (Pieragnoli, 2013)

$$R_p = \sum_{h=1}^{n} premium_h^T \cdot (1 - \gamma_h) \cdot (1 - t_h) \tag{3.1}$$

where γ is the acquisition cost rate, t_h is the residual duration.

The unearned premium reserve is usually calculated with the pro rata temporis method. Claims frequency are supposed to be uniformly distributed over the coverage period. The probability of claims occurrence during the coverage periods depends only on the length of the period.

The forfeit method

The forfeit method considers all the premiums as if they were equally distributed over the year. However, this assumption should be reviewed according to the line of business position.

$$R_p = premium^T \cdot (1 - \alpha) \cdot \frac{1}{2} \tag{3.2}$$

Alternative way of estimate the unearned premium reserve bases on the assumption of uniform distributed premiums expire dates across the half of the year, the $30th/06/t$. In this case calculations can be made under the forfeit method whether the difference from the pro rata temporis method does not exceed the 2%. This result in holding the unearned premium reserve equal to the 50% of written premiums.

Furthermore, if assume a uniform distribution of premiums expire dates within the mid of each month of a year, calculation can be made using the method of twenty-fourth

$$upr = \frac{1}{24}P_j + \frac{3}{24}P_f + \frac{5}{24}P_m + .. + \frac{23}{24}P_d \tag{3.3}$$

Sufficiency analysis

ISVAP Regulation number 16 art. 6 state that for each line of business the undertakings must check that the premium reserve set up at the previous year end, increased of the instalments accounted during the year corresponding to those contracts for which the premium reserve had been set up, is adequate to cover the cost of all claims incurred during the year related to those contracts. This procedure is named sufficiency analysis.

3.2.2 The unexpired risk reserve

Taking into account the results of the sufficiency analysis, under ISVAP Regulation number 16 art.9, the undertakings are required to assess for each line of business the necessity to set up the unexpired risk reserve if the unearned premium reserve, net of additional premium reserve, is considered to be not sufficient to cover losses arising in the forthcoming years.

ISVAP Regulation number 16 art. 10-11 respectively propose two different deterministic approaches to the unexpired risk reserve calculation:

- the analytical approach;
- the empirical approach.

The analytical approach

ISVAP gives chance of an analytical approach to estimate that excess of costs could generate in the future. Models can be used to estimate the expected claims trend for each homogeneous group of contracts, on the basis of both experience and perspective analysis.

The empirical approach

The empirical approach consists in the use of perspective loss ratio starting from the value of the current year and observed values on adequate retrospective time horizon and other objective elements

$$\max\left\{LossRatio; 1\right\} \cdot \left(UPR + AccruedPremium\right) \tag{3.4}$$

3.3 The outstanding claims reserve

The outstanding claims reserve is an estimate for its nature. This fact constitute a risk for the insurance Company, which could find itself underestimating the amount needed to settle the

contracts in force. The reserve for claims incurred in current and prior years not settled yet has to be sufficient to face all future payments and settlement expenses according to the principle of a careful evaluation based on objective elements. Indeed the reserve equals the total amount needed under the principle of prudence to face the payment of claims incurred in the previous or current years, independently on the reporting date, not yet settled. For this reason in order to assess reasonableness and sufficiency of the claims reserve held statistical and actuarial methods should be used as part of the evaluation process.

There are two reasons for outstanding claims reserving:

- at the balance sheet date there are reported claims that have not been paid yet;
- at the balance sheet date there are claims incurred but not reported yet (IBNR).

The Italian insurance regulation (Dlgs 209/2005 Art.37) identifies the ultimate cost as the criteria for determining the claims reserve, taking into account all the foreseeable future losses on the basis of available reliable historical and perspective data, with particular reference to the specific characteristics of the business. This means that claims reserves should be evaluated under the criteria of the individual assessment of the reserve for each claim incurred and reported, i.e. an analytical estimate of the cost of each reported claim, a claim by claim method, on the basis of the ultimate cost principle.

The value of the ultimate claim cost should be the result of a complex multi-phase technical evaluation, always based on the principle of prudence:

- estimate of the outstanding claims reserve by individual open claim;
- analysis and control of data process used to set up the outstanding claims provision;
- statistical and actuarial methods, especially for long run-off lines of business;
- run-off analyses.

However, estimates based on the ultimate cost value seems to be inadequate for those line of insurance business which are characterized by a longer claims cycle. These problem can find a better solution in an appropriate choice of the evaluation method to apply. Among the statistical and actuarial methods necessary to evaluate the outstanding claims reserve and to verify its adequacy there are two main groups of methodologies:

- the deterministic approaches;
- the stochastic approaches.

The deterministic approaches have their strength in an easy and fast calculation procedure. In particular, they do not require strong constraints on data. However, Solvency II institution has placed the goal of a higher harmonisation of deterministic and stochastic procedures. The reason for this choice has to be found in a more precise estimate the stochastic methods provide, in addition to a probability confidence interval. These come at the cost of more complex fitting procedure and a required data constraints.

3.4 The incurred but not reported reserve

Incurred but not reported losses have occurred but have not been reported to the insurance Company. The estimation of the amount of IBNR claims to reserve in the current year is possible considering the number and the amount of IBNR claims paid in the previous years and those of the outstanding ones in the previous years.

Reserves for IBNR claims has to be separately set up for each line of business, in respect of the ultimate cost principle, on the basis of both past experience and average claim cost of the current year.

3.5 The equalisation and ageing reserve

The legislator establish that the equalisation and ageing reserve can be held by insurance Companies as a cushion with the aim to smooth fluctuations in future periods claims rate with worst than average claims experience or to cover particular risks. In particular it appropriate in all those line of business which experience a high volatility, meaning an high severity with a low frequency claims.

Moreover, such kind of reserve is built over periods when claims experience is better than the average in order to strengthen the balance sheet.

Notice that since the equalisation reserve has the effect of producing higher revenues this could bring to inconsiderate use and abuse. Indeed, despite it is required under some local regulations, the equalisation reserve is not allowed under IFRS and Solvency II framework.

3.6 The Chain Ladder method

Before to explain how the Chain Ladder method works we need to introduce the claims cycle, which describes the process a claim goes through from the time it is submitted by the provider until it is paid by the insurance Company, to determine whether the claim is eligible for payment. It is possible to identify:

- the occurrence date;
- the reporting date, when the transmission of the claim to the insurer takes place;
- the settlement date without payment consists in determining if a claim will not be paid due to factors such as other payers;
- reopening date;
- partial payment date, each claim is given a status of paid, denied or suspended based on the disposition of edits and audits;

- final settlement date, if a claim is paid it has passed through all phases with no resulting issues so that payment is made to the provider.

It is possible, now, to classified the claim based on which event we consider:

- incurred but not reported IBNR;
- opened;
- closed without payment;
- re-opened;
- partially paid and not yet closed;
- paid and closed.

Triangular data

Although the Chain Ladder method is the most elementary, it is also still in practice the most widely used. It is usually assumed that the data are in the form of a triangle, but this is for notational convenience only. Indeed there are no problems in extending the methods to other shapes of data. Claims in the triangle are grouped in cohorts by generations:

- underwriting year/quarter, it is the year in which the policy is written;
- accident year/quarter, it is the accident date;
- reporting year/quarter, it is the claim reporting date.

The elements α_{ij} can represents paid losses, claims number, reserved claims, both incremental or cumulative. Data in the triangle are organised by development period and origin period, which are always in equal number as a square matrix. In conventional form, rows of the triangle represents accident years/quarters, column represents development years/quarters and diagonal represents finalisation years/quarters. The first column will be labelled delay year 0, 1,.., where development quarter 0 coincides with the accident quarter.

The incremental claims payments relating to business year i and delay year j will be denoted P_{ij} in the ij cell, defining the incremental paid losses triangle. Therefore the observed dataset is $\{P_{ij} : i = 1, .., t; j = 1, .., t - i + 1\}$. The statistical approach uses the incremental claims. However the Chain Ladder technique is applied to the cumulative claims. Let C_{ij} denote the cumulative version

$$C_{ij} = \sum_{k=1}^{j} P_{jk} \tag{3.5}$$

defining the incremental paid losses triangle. In the same way P_{ij}^F and C_{ij}^F denote the corresponding quantities in respect of just finalised claims. Each cell of the triangle contains the paid

losses, whether paid in that year/quarter or earlier, in respect of claims finalised in the cell. Let F_{ij} denote the number of claims finalised in the ij cell. Let G_{ij} denote their cumulative version. Define average sizes of finalised claims, incremental and cumulative, respectively

$$S_{ij} = \frac{P_{ij}^F}{F_{ij}} \tag{3.6}$$

$$T_{ij} = \frac{C_{ij}^F}{G_{ij}} \tag{3.7}$$

Age-to-age factors

The assignment is to forecast outstanding claims on the basis of past experience, i.e. to fill in the lower right hand side of the claims triangle. In its classical form the Chain Ladder method assumes complete accident year homogeneity, i.e. payments will arise in the future based on the past. Thus it is possible to use historic data to predict the future, calculating development factors (age-to-age factors) as ratios of sums of cumulative claims with the same delay index. Notice that distortions to the triangle such as

- catastrophes;
- large losses;
- unusual contracts;

are against the homogeneity of the data, therefore they need to be separately evaluated. The estimate of the development factor m_j for column j is

$$m_j = \frac{\sum_{i=0}^{T-j} C_{ij}}{\sum_{i=0}^{T-j} C_{ij-1}} \tag{3.8}$$

and the model on which it is based is

$$E(C_{ij}|C_{i1}, C_{i2}, .., C_{i,j-1}) = m_j C_{i,j-1} \tag{3.9}$$

with $j = 2, .., t$. $E(C_{ij})$ is the expected ultimate loss, estimated by multiplying the latest loss $C_{i,t-i+1}$) by the appropriate estimated value of m_j

$$\hat{E}(C_{ij}) = (\prod_{j=t-i+2}^{t} \tilde{\lambda}_j) \cdot C_{i,t-j+1} \tag{3.10}$$

The statistical Chain Ladder

The Chain Ladder statistical model is a two way model (Zehnwirth, 1994) where accident years and development years are two factors at various levels. The Chain Ladder statistical model

$$y(i,d) = \alpha_i + \sum_{j=1}^{d} \gamma_j \tag{3.11}$$

where α_i corresponding to the accident year i represents the effect of accident year i and the parameter $\gamma_j - \gamma_{j-1}$, difference in trends, represents the effect of development year j. The number of parameters in the model is $2s-1$. This model is the direct statistical extension of the standard age-to-age development factor technique.

Limitations of the Chain Ladder method

Traditional Chain Ladder models are based on a few cells in an upper triangle and often give inaccurate projections of the reserve. Indeed, accident compensation data are often characterized by features that make loss reserving and pricing difficult when using traditional actuarial techniques such as the Chain Ladder method. Many problems may arise with the Chain Ladder implementation Verrall (1994):

- firstly, not enough connection is made between the accident years, resulting in an over-parametrised model and unstable forecasts;
- secondly, the development pattern is assumed to be the same for all accident years. No allowance is made by the Chain Ladder technique for any change in the rate of claim finalisation, or for any other factors which may change the shape of the run-off pattern.
- legislative changes affecting the claims and the social attitude;
- seasonality and issues of the season;
- changes in the rules governing the payment of claims
- superimposed inflation.

These complex features of the data could eventually violate the conditions for the Chain Ladder method implementation (Taylor and McGuire, 2004). The Chain Ladder is based on a very restrictive model whose conditions are likely to be breached quite commonly in practice. Whenever this happens, the Chain Ladder is liable to material errors in the output it generates, the loss reserves amounts. In order to find solutions to the errors must apply some kind of corrective actions the model itself. However, this procedure could be long and difficult on two scores (Taylor and McGuire, 2004):

- The Chain Ladder method falls within a category named "phenomenological" by Taylor, McGuire and Greenfield (2003). This definition has to be taken in the sense that the Chain Ladder reflects just a little of the underlying mechanism of claim payments, having as a consequence that the required form of correction may not be readily apparent;
- Even when the required form of correction can be identified, the perseverance in using the Chain Ladder method might become more tedious and less reliable than its desertion in favour of a fundamentally different approach.

Traditional stochastic models have the same problem (Larsen, 2007). They are based on the same few summaries of the Chain Ladder and in addition are based on the often unrealistic assumption of independence between the aggregate incremental values.

Part II

The Method

Chapter 4

Complex systems

> "In the existing sciences much of the emphasis over the past century or so has been on breaking systems down to find their underlying parts, then trying to analyse these parts in as much detail as possible. And particularly in physics this approach has been sufficiently successful that the basic components of everyday systems are by now completely known. But just how these components act together to produce even some of the most obvious features of the overall behaviour we see has in the past remained an almost complete mystery." (Wolfram, 2002)

Some things are simple and uninteresting. Other appear random and are also uninteresting, but some are absorbingly interesting. These are complex systems. Complex systems are financial markets with dramatically fluctuating prices, companies that come and go, health care systems with unpredictable expenditure trends, and governments that rise and fall (Mills, 2010). Complex systems are set of related objects having interesting evolutionary patterns within randomness and simplicity. We call the objects of complex systems *agents*. They have relationships and behaviour rules. They move in the *environment*, with which they interact. Agents, relationships, behaviour rules, and an environment: these are the basic elements of all complex systems.

A Special subset of complex systems found particularly in social settings are the so called *complex adaptive systems*. Such complex systems change their behaviour to adapt to changes in their environment.

To model the behaviour of complex systems we can employ many tools. Some common tools that complexity scientists use to organize behaviour rules are *Game Theory*, *Genetic Algorithms*, *Heuristics*, *Agent-Based models* and *Artificial Neural Networks*. Most have been assimilated from physics and mathematics. Some tools are used to set up the model structure, including the agents relationships behaviour rules. Others are for real-world data analysis to develop agent attributes or behaviour rules, or to validate model results (Mills, 2010).

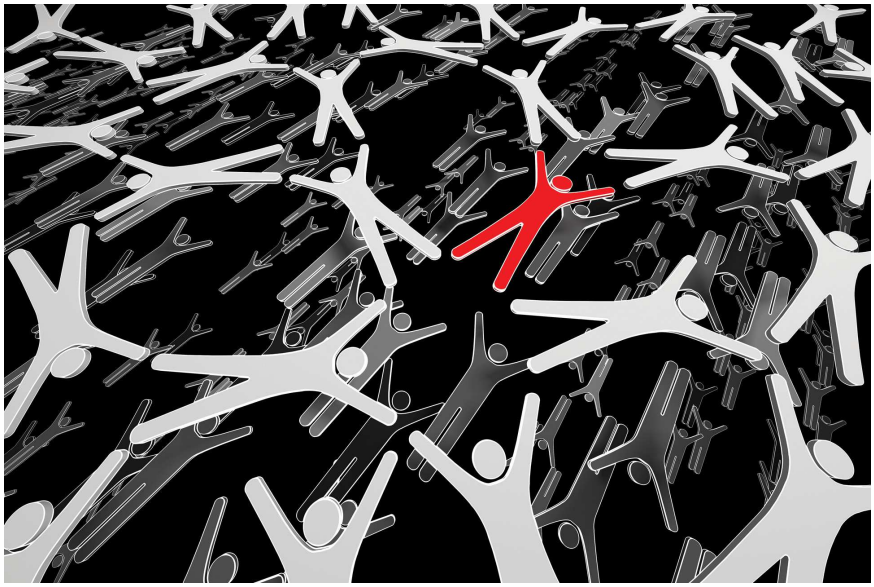

Figure 4.1: Complex system

4.1 Simulation

Simulation in general, and Agent-Based modelling in particular, is a third way of doing science in addition to deduction and induction (Axelrod and Tesfatsion, 2006). Scientists use deduction to derive theorems from assumptions, and induction to find patterns in empirical data. Simulation, like deduction, starts with a set of explicit assumptions. But unlike deduction, simulation does not prove theorems with generality. Instead, simulation generates data suitable for analysis by induction (Axelrod and Tesfatsion, 2006). The modeller makes those assumptions thought most relevant to the situation at hand and then watches phenomena emerge from the agents' interactions. Sometimes that result is an equilibrium. Sometimes it is an emergent pattern. Sometimes, however, it is an unintelligible mangle. Nevertheless, unlike typical induction, the simulated data come from a rigorously specified set of assumptions regarding an actual or proposed system of interest measurements of the real world. Consequently, simulation differs from standard deduction and induction in both its implementation and its goals. Simulation permits increased understanding of systems through controlled computational experiments (Axelrod and Tesfatsion, 2006).

4.2 Agent-Based Models

> "As multi agent systems become more established in the collective consciousness of the computer science community, we might expect to see increasing effort devoted to devising methodologies to support the development of agent systems.

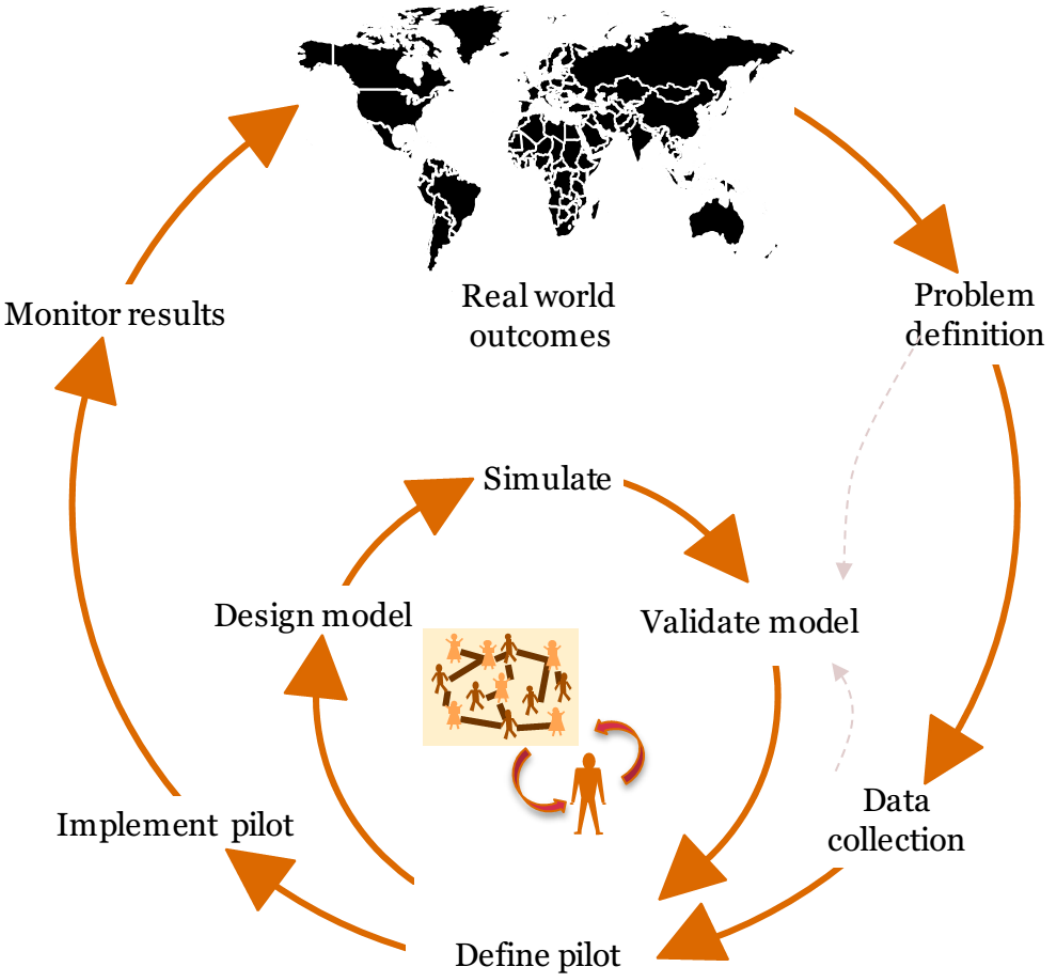

Figure 4.2: Agent-Based Simulation

Such methodologies have been highly successful in the object-oriented community." (Michael Wooldridge)

4.2.1 Agent-Based modelling

Agent-Based modelling (ABM) is a relatively new computational method (Castiglione, 2009) for studying complex systems composed of a large number of interacting entities with many degrees of freedom, exhibiting the following two properties (Axelrod and Tesfatsion, 2006):

- the system is composed of interacting agents;
- the system exhibits emergent properties, i.e. properties arising from the interactions of the agents that cannot be deduced simply by aggregating the properties of the agents.

When the interaction of the agents is contingent on past experience, and especially when the agents continually adapt to that experience, mathematical analysis is typically very limited in its ability to derive the dynamic consequences. In this case, ABM might be the only practical method of analysis (Axelrod and Tesfatsion, 2006).

ABM begins with assumptions about agents and their interactions and then uses computer simulation to generate *histories* that can reveal the dynamic consequences of these assumptions (Axelrod and Tesfatsion, 2006).

ABM are generally identified as theoretical exercises aimed at investigating the (unexpected) macro effects arising from the interaction of many individuals — each following possibly simple rules of behaviour — or the (unknown) individual routines/strategies underlying some observed macro phenomenon (Richiardi, 2012). As such, the typical ABM is a relatively small *toy* model (Richiardi, 2014), which can be used to understand relevant mechanisms of social interaction.

4.2.2 ABM goals

The specific goals pursued by ABM researchers take four forms (Axelrod and Tesfatsion, 2006):

Empirical understanding Why have particular large-scale regularities evolved and persisted, even when there is little top-down control? ABM researchers seek causal explanations grounded in the repeated interactions of agents operating in specified environments. In particular, they ask whether particular types of observed global regularities can be reliably generated from particular types of agent-based models.

Normative How can agent-based models be used as laboratories for the discovery of good designs? ABM researchers pursuing this objective are interested in evaluating whether designs proposed for social policies, institutions, or processes will result in socially desirable system performance over time.

Heuristic How can greater insight be attained about the fundamental causal mechanisms in social systems? Even if the assumptions used to model a social system are simple, the

consequences can be far from obvious if the system is composed of many interacting agents. The large-scale effects of interacting agents are often surprising because it can be hard to anticipate the full consequences of even simple forms of interaction.

Methodological How best to provide ABM researchers with the methods and tools they need to undertake the rigorous study of social systems through controlled computational experiments?

4.2.3 The Agents

The basic idea of ABM is to construct the computational counterpart of a conceptual model of a system under study on the basis of discrete entities (agents) with some properties and behavioural rules, and then to simulate them in a computer to mimic the real phenomena. The agent is an autonomous entity having its own internal state reflecting its perception of the environment and interacting with other entities according to more or less sophisticated rules. (Castiglione, 2009). Agents can represent people (say drivers, as in my simulation), but they can also represent social groupings such as Insurance Companies. Roughly speaking, an entity is an agent if it has some degree of autonomy, that is, if it is distinguishable from its environment by some kind of spatial, temporal, or functional attribute: an agent must be identifiable. Moreover, it is usually required that an agent must have some autonomy of action and that it must be able to engage in tasks in an environment without direct external control (Castiglione, 2009). The Society of Mind Cognitive agent that I present later indeed owns this autonomy of action.

Typically agents are situated in space and time and reside in networks or in lattice-like neighbourhoods. The location of the agents and their responsive behaviour are encoded in algorithmic form in computer programs.

Agents can be identified on the basis of a set of properties that must characterize an entity (Castiglione, 2009):

Autonomy the capability of operating without intervention by humans, and a certain degree of control over its own state. ABM is characterised as a bottom-up approach, opposed to traditional modelling which is usually based on a top-down approach (Ulbinaite and Le Moullec, 2010).

Social ability the capability of interacting by employing some kind of agent communication language.

Reactivity the ability to perceive an environment in which it is situated and respond to perceived changes.

Pro-activeness the ability to take the initiative, starting some activity according to internal goals rather than as a reaction to an external stimulus.

In ABM, agents typically do not simultaneously perform actions at constant time-steps. Rather, their actions follow discrete-event cues or a sequential schedule of interactions. The discrete-event setup allows for the cohabitation of agents with different environmental experiences (Castiglione, 2009).

Physics investigation is based on building models of reality. It is a common experience that, even using simple "building blocks", one usually obtains systems whose behaviour is quite complex. This is the reason why Cellular Automata-like, and therefore agent-based models, have been used extensively among physicists to investigate experimentally (that is, on a computer) the essential components of a complex phenomenon, which constitute artificial worlds (Castiglione, 2009). An agent may represent a particle, a financial trader, a cell in a living organism, a predator in a complex ecosystem, a power plant, an atom belonging to a certain material, a buyer in a closed economy, customers in a market model, forest trees, cars in large traffic vehicle system, etc (Castiglione, 2009).

Agent actions

Actions are the elements at the basis of agent behaviour. Agent actions can cause modifications in their environment or in other agents that constitutes the ABM (Bandini *et al.*, 2009).

Agent Architecture

The term architecture (Bandini *et al.*, 2009) refers to the model of agent internal structure that is responsible of effectively selecting the actions to be carried out, according to the perceptions and internal state of an agent. Different architectures have been proposed in order to obtain specific agent behaviours and they are generally classified into deliberative and reactive (we will see more specifically in the next section).

Agent Interaction

Interaction is a key aspect in ABM. The essence of a ABM is the fact that the global system dynamics emerges from the local behaviours and interactions among its composing parts.

Environment

The environment, besides enabling perceptions, can regulate agents' interactions and constraint their actions (Bandini *et al.*, 2009).

Design philosophy

- a key notion is that simple behavioural rules generate complex behaviour. This principle, known as K.I.S.S. (keep it simple, stupid) is extensively adopted in the modelling community;

- design model around available data;
- standardized interface so multiple groups can contribute;
- industrial code, modern software standards, open source;

4.2.4 Some examples of what ABM have already accomplished in Economics

- Firm size: Axtell;
- Financial markets: LeBaron, Lux, SFI stock mkt, ...;
- Credit markets: Gallegati, Delligati, ...;
- Labor market: Clower and Howitt;
- Mortgage prepayment (Geankoplos et al.);
- Leverage in real estate: Khandahani, Lo, Merton;
- Energy markets: Tesfatsion;
- Whole economy: EURACE project.

4.2.5 ABM and Economics

Two features might be regarded as the main building blocks of the Artificial Economics (Leitner and Wall, 2009): agent-based models and the use of computational techniques to "solve" them. In particular, artificial markets or social systems, artificial networks or artificial organizations consisting of interacting, heterogeneous agents are modelled, computationally represented and simulated. The behaviour of the artificial system – whatever it might be – is "observed" over time and analysed by the researcher.

Economics is a field where ABM provides a very interesting and valuable instrument of research. Indeed, mainstream economic models typically make the assumption that an entire group of agents, for example, investors, can be modelled with a *single rational representative agent*. While this assumption has proven extremely useful in advancing the science of economics by yielding analytically tractable models, it is clear that the assumption is not realistic: people differ and as many psychological studies have shown, they often deviate from rationality in systematic ways (Castiglione, 2009). For the same reason ABM results a useful tool to Insurance Risk Management as well.

When specialized to computational economic modelling, ABM reduces to Agent-Based Computational Economics (ACE). This consists in the computational study of economies modelled as

evolving systems of autonomous interacting agents (Tesfatsion and Judd, 2006). ACE complements the traditional analytical approach and is gradually becoming a standard tool in economic analysis (Castiglione, 2009).

Because the paradigm of agent-based modelling and simulation can handle richness of detail in the agent's description and behaviour (Castiglione, 2009), this methodology is very appealing for the study and simulation of social systems like traffic system, where the behaviour and the heterogeneity of the interacting components are not safely reducible to some stylized or simple mechanism. It concerns the emulation of the individual behaviour of a group of social entities, typically including their cognition, actions and interaction.

Since it is difficult to formally analyse complex multi agent systems, they are mainly studied through computer simulations. However, results obtained through simulations do not formally validate the observed behaviour. Indeed, there is a compelling need for a mathematical framework which one can use to represent ABM systems and formally establish their properties.

4.3 Society of Mind Cognitive Agent

4.3.1 Agent architectures

The Agent-Based model I simulated presents an agent architecture based on *"Society of Mind"* (SoM) metaphor on human mind and cognition of Marvin Minsky, an American cognitive scientist in the field of artificial intelligence (AI) and co-founder of Massachusetts Institute of Technology's AI laboratory.

Due to the features implicitly generated by their design the SoM agents allow to represent human heterogeneity and variety of behaviours. Indeed, the Society of Mind cognitive agent possesses all reactive, deliberative and affective capabilities, outside the hierarchical paradigm (Leu *et al.*, 2013). This particular agent architecture allows us to build an *Adaptive Behaviour Simulation Model*.

Agent-Based models are nowadays indispensable tools for solving various problems that involve adaptation and behavioural heterogeneity. Agents were categorised over time in numerous classes and types (Caballero and Botia, 2012), the fundamental architectural approaches from a cognitive perspective being *reactive*, *rational* and *affective* agents (Leu *et al.*, 2013):

Reactive agents can actuate simple response, such as adaptation of position, velocity, orientation, etc. to local and most of the time primary stimuli such as vision, tactile perception, etc.

Rational and Affective agents own an internal representation of the environment being able to generate action plans and intentions based on either rational or affective decision making processes respectively.

Since each approach is concerned with a different set of cognitive–behavioural capabilities, the resulting agent designs are also limited to solving the corresponding set of problems. Efforts to com-

bine architectures into hybrid designs for covering a wider range of human cognitive–behavioural capabilities exist and they consider hierarchically layered designs with reactive parts at lower levels of cognitive capabilities being commanded by rational and / or affective parts situated towards the higher cognitive levels (Leu *et al.*, 2013). However, these designs proved to become rapidly logically and computationally complex when the range of cognitive capabilities widens, fact that limits their usability (Caballero and Botia, 2012).

The Minsky's approach The approach I implemented in my model was introduced by Marvin Minsky in his "Society of Mind" (SoM) view on human mind and cognition (Minsky, 1988). It is an architectural approach for designing cognitive agents, i.e. agents with all reactive, deliberative and affective capabilities, outside the hierarchical paradigm. This means that reactive, deliberative and affective components coexist *at the same level* in the cognitive capabilities pyramid while they are active in different environmental contexts (Eluyode and Akomolafe, 2013). The *bidding-process* procedure in the model makes them to participate in the over-all decision-making process, by trying to impose their own decision. In other words their importance rises and falls according to instantaneous context the agent faces at a certain moment in time.

In Minsky's approach, human mind is not a singular entity organised in a fixed hierarchical construction of cognitive capabilities but rather an entire system of competing agencies, which creates a dynamic, permanently evolving society-like edifice (Eluyode and Akomolafe, 2013). An agent architecture starting from this idea would potentially offer a better representation of the variety of cognitive capabilities with a simpler and less computationally complex architectural design than conventional hierarchical approaches, describing the human nature in a more comprehensive manner. In my model, following the work Eluyode and Akomolafe (2013), I built an SoM agent architecture that can successfully model human innate behavioural diversity as well as real-time adaptation. Then I placed those SoM agents in various road traffic situations, a cognitively demanding environment.

The perception–action cycle

Agents have been defined in many ways over the years and have been categorised in many formal taxonomies. From a definition point of view, most definitions place agents in a *perception–action cycle* (Smith and Hancock, 1995), in which individual agents perceive surrounding environment through sensors and based on that generate the appropriate behaviour. From a taxonomy point of view, two are of major concern for my simulation model:

the perception–action cycle , concerned with what major steps the agent takes within the cycle;

the cognitive capability architectural categorisation , which shows how individual agents are decomposed in component modules and how these modules interact to generate perception–action cycles

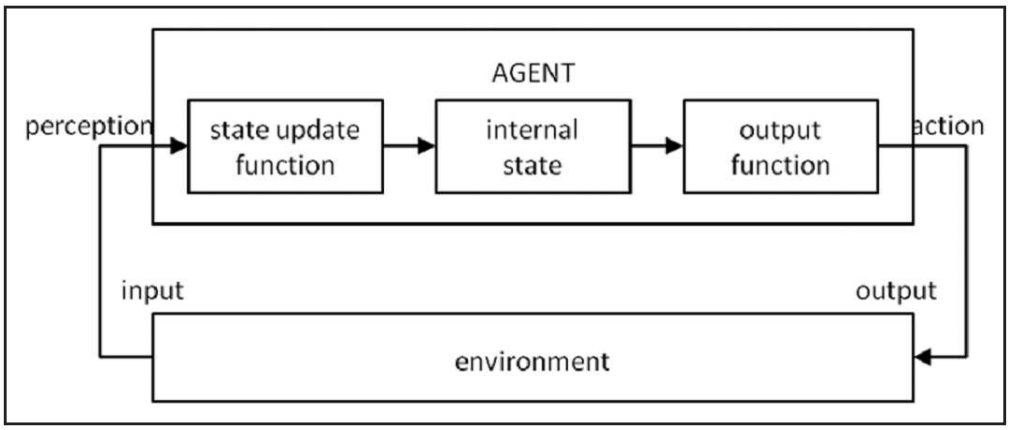

Figure 4.3: Perception-action cycle

Fundamental formulation of perception–action cycle is the three-step Sense–Decide–Act (SDA) cycle (Wooldridge, 2002):

sense data from the world are obtained through sensor;

decide processing to those data is applied for establishing what to do;

act actuators are commanded to produce behaviour.

SDA cycles involve an autonomous context-based decision making mechanism, which can function based on any kind of cognitive processing abilities.

Cognitive capability architectural design

Reactive agents Reactive architectures aim at building agents capable of fast reactions to changes detected in the immediate environment (Eluyode and Akomolafe, 2013). Such agents have no or very simple internal representation of the environment. They are built in a behaviour-based paradigm (Russell and Norvig, 1995) providing very tight coupling between perception and action. For this reason reactive agents belong to a different implementation of the decisional component, the Sense–React cycle (Russello *et al.*, 2011), since autonomous decision making (intelligence) is not an intrinsic attribute, but the product of the interaction between agent and environment. They are ideal in very dynamic and unpredictable environments and their implementation is simple leading to low computational complexity (Eluyode and Akomolafe, 2013). Moreover, reactive agents have a high degree of adaptability and flexibility due to the fact that they have no central planning component. However, this simplicity comes at the cost of inherent limitations. Indeed, they need sufficient information available from the local environment, since they do not possess an internal representation of the world. This leads reactive agents having short-term view, without long-term planning capabilities or decision based on global environmental context.

From a cognitive perspective, reactive agents are not considered intelligent per se but rather the multi-agent systems they are part of show emergent collective intelligence (Eluyode and Akomolafe, 2013).

Rational agents Rational architectures, such as Believe–Desire–Intention (**?**) or the Procedural Reasoning System (PRS) (Ingrand *et al.*, 1992) concentrate on long-term planning of actions, centred on sets of goals. They are capable of deciding and acting in more than just a reactive manner, based on their own reasoning and view on the surrounding world and considering a set of alternative courses of action (Eluyode and Akomolafe, 2013). Environment is represented internally as an explicit symbolic model of the world, and decisions are made through logical reasoning based on pattern matching and symbolic manipulation. Given this architectural approach, rational agents fall into the SDA paradigm when looked at from perception–action perspective (Eluyode and Akomolafe, 2013). The advantage of rational agents is they can cover more real-world problems through the fact decision-making mechanisms are closer to human reasoning processes. However, they are also more computationally complex than reactive counterparts since generation of alternate plans and processing of choice requires more complex implementation and higher computational effort (Eluyode and Akomolafe, 2013). Furthermore, they can only model rational cognitive processes and hence generate rational behaviour, with decision-making based on choice and involving certain profit functions (Eluyode and Akomolafe, 2013).

Affective agents In affective architectures, agents have components that in connection with other internal or external components can instantiate affective states (Eluyode and Akomolafe, 2013). Thus, affective agents contain explicit or implicit representations of various affective features such as desires, emotions, moods etc. (Broekens *et al.*, 2007). Like rational agents, affective agents also fall into SDA paradigm. However, internal decision-making mechanisms in the decisional stage are different, since affective agents appeared as a response to rational agents' inability to deal with more complex cognitive contexts, which were beyond the usability range of rational decision theory tools (Eluyode and Akomolafe, 2013).

In the case of humans theory of somatic markers normal operation of human decision-making requires an emotional mechanism, which regulates rational reasoning by creating biased affective forecasts of potential consequences of an action (Bechara *et al.*, 1994). Without this emotional signal, the brain only uses rational reasoning, which slows down or even jeopardise the decision-making because of the many possible conflicting options to be considered (Eluyode and Akomolafe, 2013). However, not only humans, but even simple organisms with no rational capabilities were found to have certain affective underpinnings, which generate various behavioural patterns such as attraction, aversion etc., whereas in a general consideration any organism/agent of a certain complexity possess affective states that act in relation with the other cognitive functions and capabilities (Scheutz and Logan, 2001), (Zadeh *et al.*, 2006).

Hybrid agents None of the main architectural approaches discussed above is suitable for building complex cognitive agents (Eluyode and Akomolafe, 2013), which ultimate goal is to identify and replicate the mechanisms underlying human cognition and behaviour. Indeed, very little can be done by trying to work on them separately. (Gershenson, 2004) concluded that there is no best approach than cognitive paradigms, since each view treats different aspects of cognition in different contexts. Hence, most of the complex cognitive agents/theories are trying to find unified representations of these mechanisms (Clark and Grush, 1999), (Froese, 2012), (Newell, 1990). The most notable cognitive architectures proposed over time, such as (Laird, 2012), (Anderson, 1996), (Sun, 2006) or (Sloman, 2009), use some or all three types of cognitive capabilities (reactive, rational, affective) as sub-components in hierarchically layered designs (Eluyode and Akomolafe, 2013). Agent's control subsystems are arranged into a hierarchy. Higher layers deal with information at increasing levels of abstraction. Thus, the reactive component is considered representing lower level cognitive processes, providing fast response to events without complex reasoning (Eluyode and Akomolafe, 2013). This is controlled by either a rational or an affective component (or both) situated at a higher level of cognitive capabilities, which contains a world model and makes decisions according to rational or affective reasoning. The problem in such architectures is how to model interactions between hierarchical layers and control mechanism. Two important control frameworks have been proposed (Müller *et al.*, 1995). They classify such architectures into *horizontally* and *vertically layered*:

horizontally layered architectures each layer has access to sensing and acting, making a potential decomposition into subagents possible. Each layer is connected to the sensory input and action output, and so it produces suggestions as to what actions to be taken. These suggestions are *approved*, *altered* or *dismissed* by the higher hierarchical layers;

vertically layered architectures sensory input and action output are connected to the lowest layer. Hence, only the lowest layer is involved in sensing and actuating while higher layers are involved in complex processing and decision making.

Sloman (2003) addressed Hierarchical aspect of layered designs. He discusses a dominance dimension of architectures consisting of the amount of control exercised by higher-level cognitive processes on the lower levels. The more and stricter control higher levels have on lower levels, the stronger the dominance is, while in the opposite direction the dominance decreases as lower levels are allowed other types of interactions apart from the subordination ones (Eluyode and Akomolafe, 2013). Moreover, higher cognitive processes could directly turn on and off various inferior processes, they can indirectly influence their operation acting as modulators or, in the less direct type of control, they can facilitate their training and/or evolution.

Existing hybrid architectures have been criticised though for the lack of methodologies for guiding the design. Existing designs are very specific, application dependent and hence difficult to be categorised or included in certain theoretical approaches (Eluyode and Akomolafe, 2013). Furthermore, since fundamental sciences have not yet found a consensus regarding which cogni-

tive processes are positioned at which abstraction level, existing architectures and the way they describe the interactions between various agencies (i.e., reactive, deliberative, affective) are not entirely supported by formal theories (Eluyode and Akomolafe, 2013).

4.3.2 Society of Mind agent architecture

Minsky (1988) may contain answers for the difficult task of modelling interactions between various classes of cognitive processes, in order to design complex cognitive agents (Eluyode and Akomolafe, 2013).

Some authors disapprove of the lack of scientific validation of many concepts and ideas presented in Minsky's approach (Ginsberg, 1991), (Reeke Jr, 1991) (Smoliar, 1991). However they also acknowledge that any single claim made by SoM is so true and undeniable in its common sense that Minsky's approach, despite not qualifying as a theory, is also virtually unarguable with scientific methods, coming rather from an extremely pertinent observation and very deep understanding of human nature and human action formation mechanisms (Dyer, 1991), (Stefik and Smoliar, 1991) (Thagard, 1993), (Eluyode and Akomolafe, 2013).

Among the many concepts and assumptions, two aspects of the Minsky's theory are fundamental for the model construction:

- the principle of non-compromise;
- the K-line theory of memory.

Principle of non-compromise According to Minsky, the human mind contains several agencies, which compete at any moment in time for imposing their own decision about the action to be taken (Eluyode and Akomolafe, 2013). Depending on instantaneous internal and external contextual factors, these agencies rise and fall in terms of their strength in the competing process. As the name suggests the compromise between two agencies is impossible.

K-line theory of memory K-line theory of memory assumes that memory of past actions forms the knowledge base from which entities (subagents) of the agencies in the Society (of Mind) are made of (Eluyode and Akomolafe, 2013). In Minksy's approach K-line selection is viewed only as a step within the decision-making process, i.e. re-membering mechanism feeds multiple agents and agencies of the mind in order to support their state update and their position within the sub-agent competition process (Eluyode and Akomolafe, 2013).

The SoM agent architectural schema

The agent (Eluyode and Akomolafe, 2013) consists of a number n of sub agencies A_i, with $i = 1, n$, each of them containing a set of internal entities (basic agents) e. First, an agency A_i senses the environment by gathering from the larger input dataset x_i, with $i = 1, m$ only that information that is relevant for it. Then its entities interact and update their states and

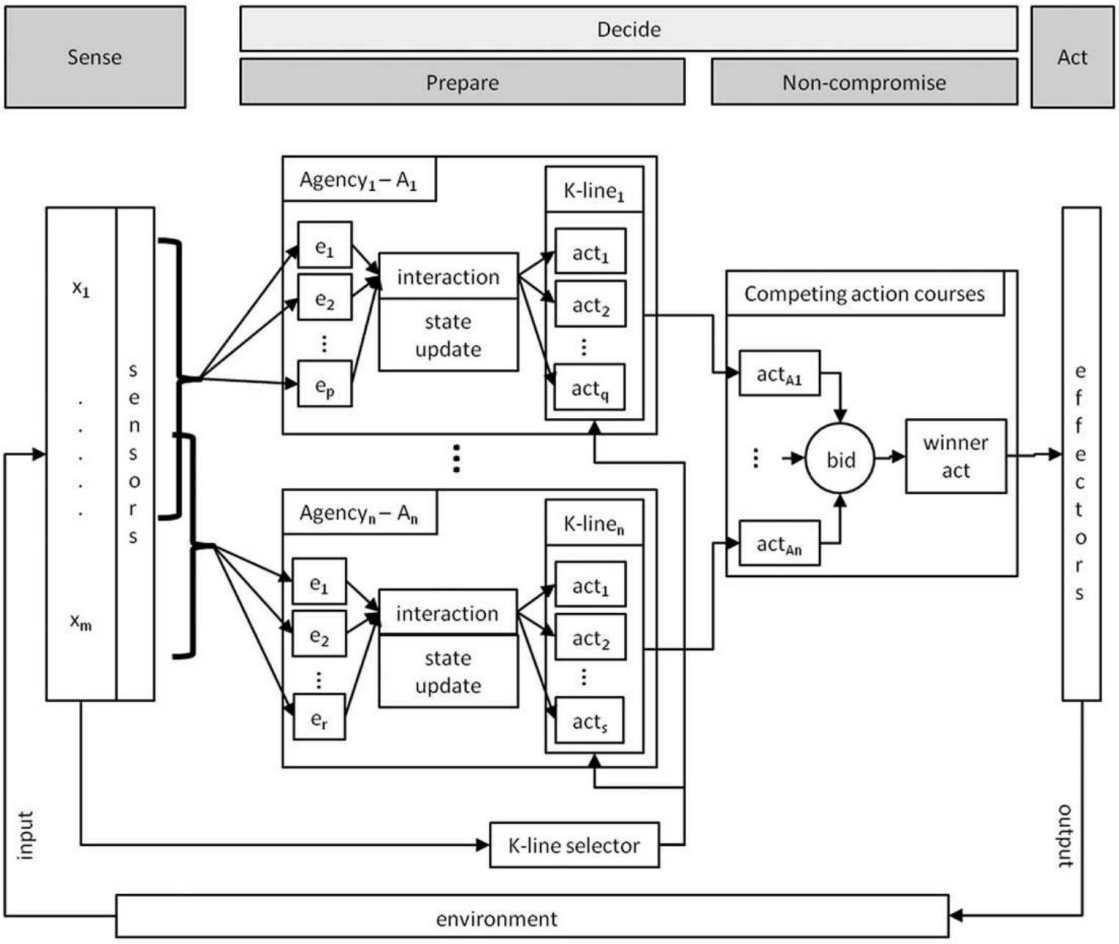

Figure 4.4: The figure shows the architectural framework of an SoM agent describing its potential internal dynamics

through that the state of the entire agency. As a result of interaction of its internal entities, the agency A_i is able to propose a set $K-line_i$ of actions act, which are usually the result of that particular interaction of internal entities e. However, not all these potential actions are relevant to the current context. Therefore, a $K-line$ selector chooses from the $K-line_i$ set of potential actions only that action (or subset of actions) act_{Ai} that is strictly related to the current context in order to participate in the competition with the actions proposed by other sub-agencies. Once all agencies finish their preparation and the K-line selector selects the candidate action for each agency, these actions participate in a bid in which, according to a certain bidding strategy, only one course of action will win and will be transferred to effectors to be actuated in the environment other sub-agencies. Once all agencies finish their preparation and the K-line selector selects the candidate action for each agency, these actions participate in a bid in which, according to a certain bidding strategy, only one course of action will win and will be transferred to effectors to be actuated in the environment (Eluyode and Akomolafe, 2013).

The SoM agent architecture is still in a classic SDA loop, with the SoM agent sensing the environment through its agencies, deciding about a course of action and acting upon the environment through effectors (Eluyode and Akomolafe, 2013). Once the agent is aware of the multiple courses of action, the next stage of the decisional process starts: the competition.

Apart from the fact an RRA SoM agent is assumed to be able to cope with complex problems that combine all three types of cognitive processes, it also can reduce its capabilities when needed by inhibiting one or more of its agencies if current tasks do not involve certain cognitive processes (Eluyode and Akomolafe, 2013).

4.3.3 Society of Mind applied in traffic behaviour

I built an SoM driver Agent-Based on Eluyode and Akomolafe (2013). As shown in fig. 5.3, noncontextual behavioural features are assigned to individual drivers and bias their traffic behaviour, while instantaneous environmental inputs and dynamics of internal agencies account for building real-time adaptation. In order to do so, following Eluyode and Akomolafe (2013), a driver takes three types of decision standing for the three types of agencies:

Affective agency a driver takes decisions based on the current emotional state regardless safety, annoyance produced to other drivers and traffic regulations;

Rational agency a driver takes rational decisions in order to drive safely, to avoid annoying other drivers and to obey traffic regulations;

Reactive agency a driver MUST take avoiding actions (full brake, lane change) in order to escape from an immediate danger, such as a collision.

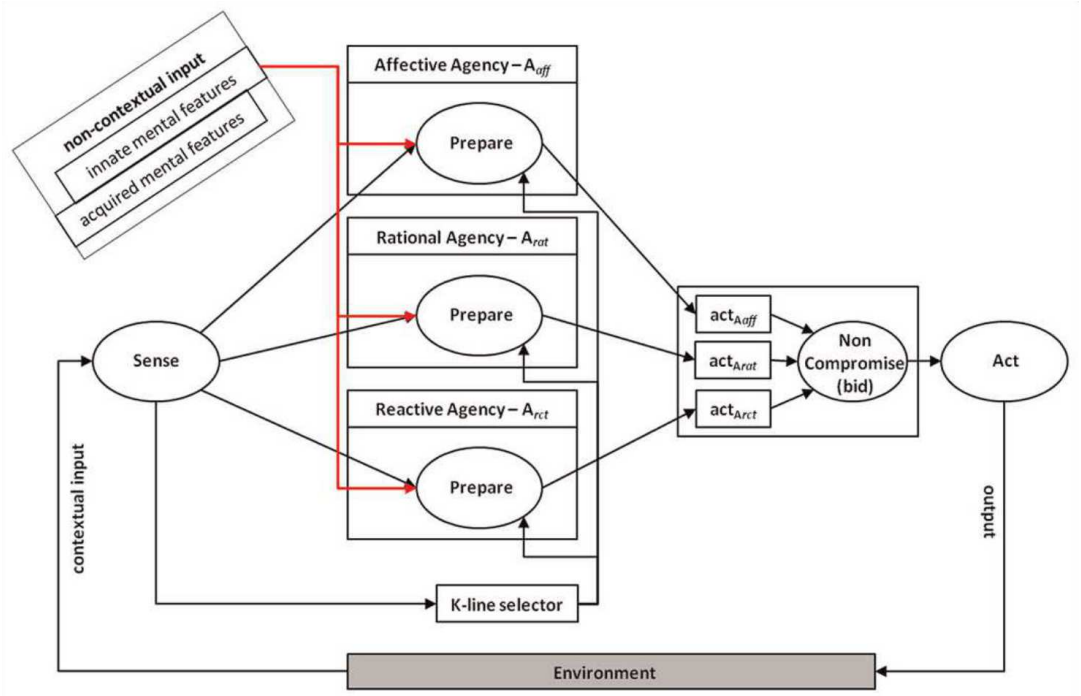

Figure 4.5: Non-hierarchical hybrid Reactive–Rational–Affective (RRA) hybrid Society of Mind (SoM) architectural schema

Behavioural propensities and innate personality features

We have seen that one of the important aspects of the proposed RRA SoM architectural framework is the capability of handling behavioural propensities. In my model, as in Eluyode and Akomolafe (2013), only the innate behavioural features (personality) are considered.

Implementation of personality, as the expression of innate behavioural propensities, is based on Goldberg (1990) five-factor dimensional model of personality are represented as a five-dimensional tuple $P(P_O, P_C, P_E, P_A, P_N)$ with $P_i \in (-\infty, +\infty)$ where P_i are the personality factors:

- Openness;
- Conscientiousness;
- Extraversion;
- Agreeableness;
- Neuroticism.

In order to obtain a computational representation, values of personality factors P_iE are limited to the finite real interval $P_i \in (-P_{max}; P_{max})$ with $P_{max} = 1$. Values of personality traits are considered system constants and are assigned to agent at the beginning of simulation. They are fed to internal agencies during the simulation together with contextual information coming from sensors and they participate in the state update process implemented in each agency (Eluyode and Akomolafe, 2013).

Contextual input scaling

At simulation time-step t_n internal agencies prepare for potential actions at t_{n+1} based on distances to front and back neighbours at tn, based on the unchangeable values of personality traits, and based on their internal state at t_{n-1} (Eluyode and Akomolafe, 2013). However, inputs come from personality factors and are situated, as explained above, in the $(-1, 1)$ interval, while distances are expressed in metres. From here the need to scale distance inputs taken from physical environment to normalised values, which can be used in conjunction with values of personality factors and with those of internal entities (Eluyode and Akomolafe, 2013). However, as intervals of variation are not constant and hence their limits are not fixed values throughout the simulation a simple normalisation process would not be sufficient. Deviation of real distance to neighbours from a minimum safe distance is therefore used.

fig. 5.4 shows the *input scaling* procedure as from Eluyode and Akomolafe (2013). Vehicle of interest c running from left to right and situated between two neighbours. Distances to these neighbours d_{1r}, d_{4r} vary in time and for this reason they cannot be placed in a fixed variation range. Hence, normalising them is a difficult issue since limits of the variation interval are not

constant. fig. 5.4 (b) and (c) shows the scaling procedure for distance between current vehicle c and front neighbour 1, the procedure for all other neighbours being similar.

In this car-following situation (Eluyode and Akomolafe, 2013), an additional point s is considered in the collision-avoidance (CA) car-following model described in (5.17). This point is situated at the minimum safety distance d_{1s} to leading vehicle (neighbour 1) within which a collision would be unavoidable in the eventuality that driver of the vehicle in front would act unpredictably, and given an assumed maximum deceleration capability. Distance d_{1s} is also calculated according to (5.17.).

However, in real traffic situations, the distance between the current and leading vehicle is not always d_{1s} as in the ideal CA case, but it can be either below or above the safe distance, generating a measurable deviation Δ_1

$$\Delta_1 = d_{1s} - d_{1r} \tag{4.1}$$

and subsequently a normalised deviation x_1 which can fall into two cases

$$x_1 = \begin{cases} \dfrac{\Delta}{d_{1s}}; d_{1s} > d_{1r} \\ \\ \dfrac{\Delta}{d_{4r}}; d_{1s} < d_{1r} \end{cases} \tag{4.2}$$

The first case (fig. 5.4.b) is when real distance d_{1r} is below the safe distance, i.e. $d_{1r} < d_{1s}$. In this case, deviation from an ideal point is towards unsafety, and the range of this deviation is between zero – when vehicle c keeps a safe distance $d_{1s} = d_{1r}$ to leading vehicle 1 – and a maximum deviation of d_{1s}– when c follows vehicle 1 at a hypothetical $d_{1r} = 0$. In this case, deviation from a safe distance can vary in the interval $(0, d_{1s})$. Hence the value used as denominator for normalised deviation x is d_{1s} with resultant values of x_1 situated in interval $(0, 1)$

$$x_1 = \frac{\Delta}{d_{1s}} = \frac{d_{1s} - d_{1r}}{d_{1s}} \tag{4.3}$$

The second case (fig. 5.4.c) is when the real distance d_{1r} is above the safe distance $d_{1r} > d_{1s}$. In this case, deviation Δ from the ideal point is towards safety. Hence, deviation range is between zero – when $d_{1r} = d_{1s}$– and a maximum hypothetical deviation of d_{4r} when the real following distance is $d_{1r} = d_{4r} + d_{1s}$. In such a case deviation from the safe distance can vary in the interval $(-d_{4r}, 0)$. Therefore the denominator used for calculating the normalised value x is d_{4r} with resultant values of x_1 in interval $(-1, 0)$.

$$x_1 = \frac{\Delta}{d_{4r}} = \frac{d_{1s} - d_{1r}}{d_{4r}} \tag{4.4}$$

Overall, the input scaling process for vehicle c and neighbour 1 has a resultant scaled (normalised)

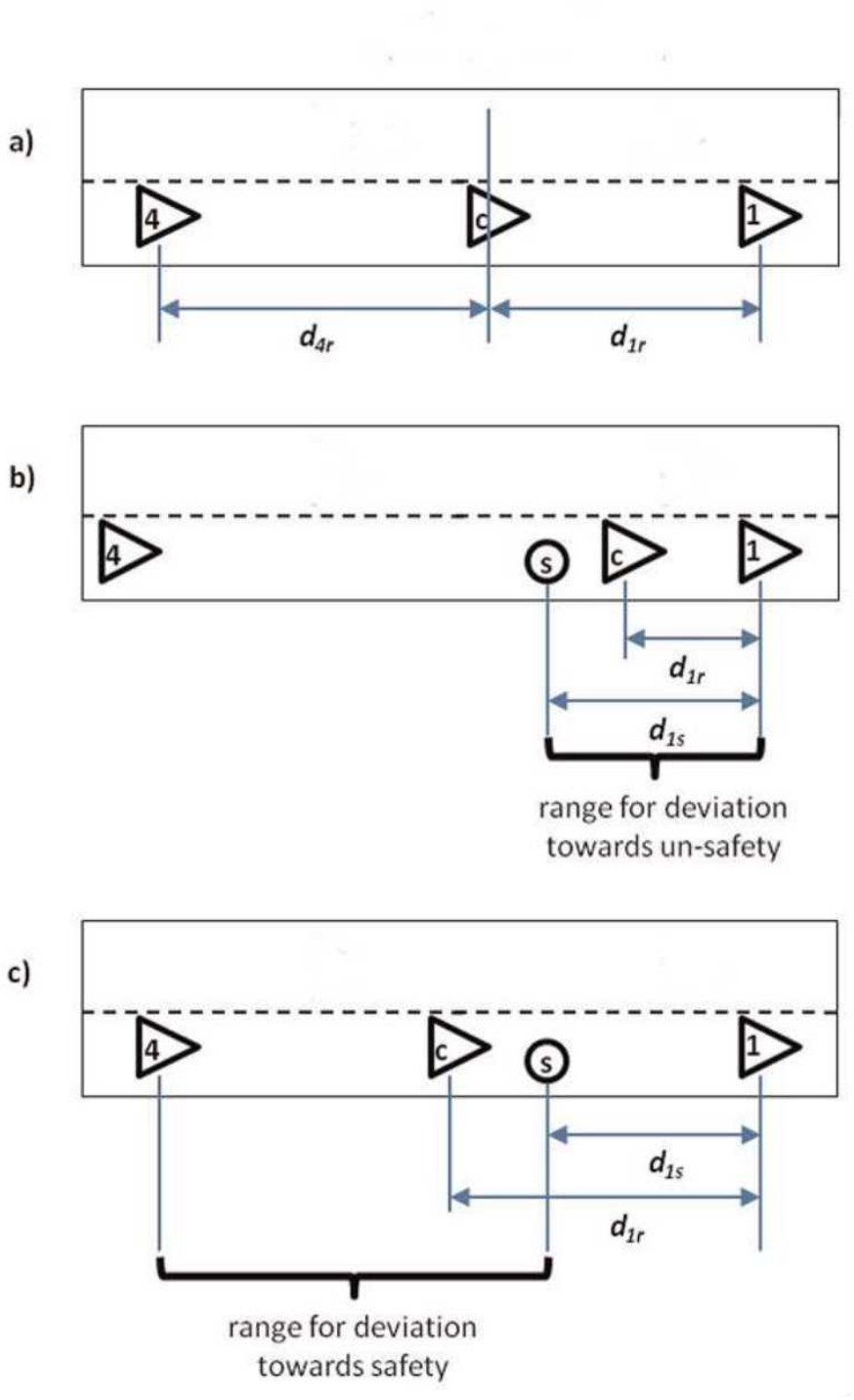

Figure 4.6: Input scaling process

input $x_1 \in (1, 1)$, which can be used in conjunction with the other relevant variables and constants in the process of agency state update.

Affective agency

Studies in transportation psychology and behaviour found that the space of emotions applicable to drivers consists of three independent emotions (Grimm *et al.*, 2007):

- happiness (E_H);
- drowsiness (E_S);
- anger (E_A).

The affective system assumed in my model (as Eluyode and Akomolafe (2013) did) uses a dimensional approach on these emotions which represent the variety of human emotional responses.

Emotional space can be represented as a three-dimensional tuple $E(E_H, E_S, E_A)$ with $E_i \in (-\infty, +\infty)$, where E_i are primitive emotions (Eluyode and Akomolafe, 2013). The problem generated by infinite emotional intervals was solved following the same approach used for personality, i.e. infinite values of emotions Ei were conveniently limited to finite real intervals $E_i \in [-E_{max}; E_{max}]$ with $E_{max} = 1$. Internal dynamics of the SoM affective agency is an adaptation to SoM driver agent implementation of the general influence diagram, and consists of two main elements:

- emotional update function;
- strength update function.

Emotion update function The three emotions considered in the model, happiness, drowsiness and anger (Grimm *et al.*, 2007), increase or decrease at each update cycle as follows

$$\begin{cases} E_H(t_n) = E_H(t_{n-1}) + \dfrac{1}{4}\displaystyle\sum_{i=1}^{4} x_i(t_n) \\ E_D(t_n) = E_H(t_{n-1}) + \dfrac{1}{4}\displaystyle\sum_{i=1}^{4} \dfrac{x_i(t_n)}{2} \\ E_A(t_n) = E_A(t_{n-1}) + P_N + \dfrac{1}{4}\displaystyle\sum_{i=1}^{4} x_i(t_n) \end{cases} \tag{4.5}$$

where x_i are the scaled inputs corresponding to instantaneous distance–speed relation to the neighbours, as seen before, and the values of the emotions are limited in $(-1, 1)$.

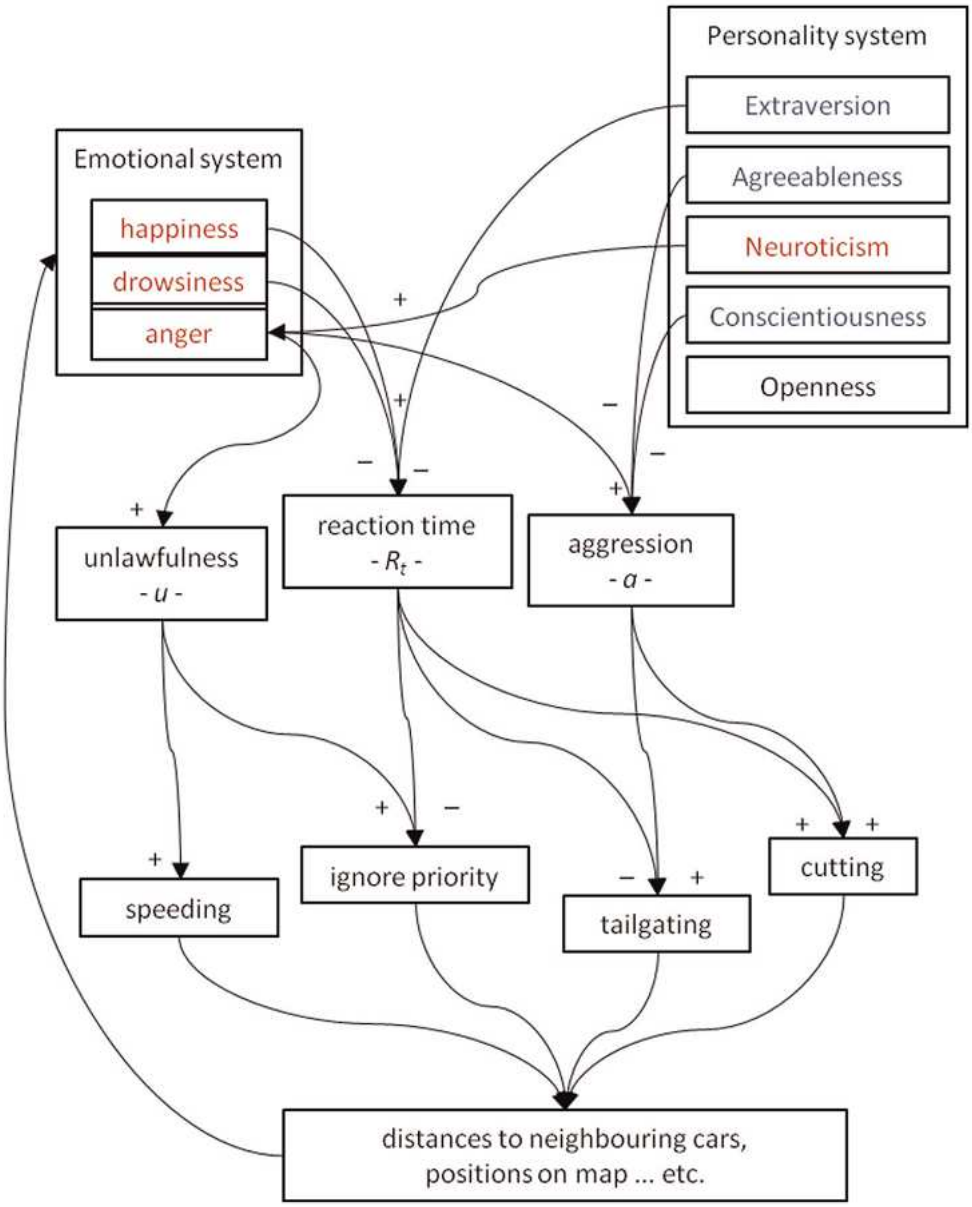

Figure 4.7: Influence diagram of action formation. It explains the transfer of driver's affective features into traffic-related actions process.

Strength update function After the emotional states are updated they are used together with personality traits and scaled contextual inputs for transferring the updated affective state into a strength level, which further participates in the bidding process included in a non-compromise stage of decision (Eluyode and Akomolafe, 2013). The *affective strength update function* is

$$aff_{str}(t_n) = \frac{a(t_n) + u(t_n)}{2} \tag{4.6}$$

with a representing the aggression level computed as

$$a(t_n) = \frac{E_A(t_n) - P_A - P_C}{3} \tag{4.7}$$

and u representing the unlawfulness level computed as

$$u(t_n) = E_A(t_n) \tag{4.8}$$

Affective strength is also in $(-1, 1)$.

Rational agency

This agency is targeting optimality from safety, politeness and rule obeying point of view, by enforcing safe following distances correlated with vehicle speed (Eluyode and Akomolafe, 2013). This behavioural pattern is achieved through one of the existing well-established CA car-following models, which generate a prescribed safe behaviour based on manipulation of the standard Newtonian equations of motion. (Eluyode and Akomolafe, 2013).

CA models assume a minimum safety distance within which a collision would be unavoidable in the eventuality that the driver of the vehicle in front would act unpredictably, and given an assumed maximum deceleration capability (Eluyode and Akomolafe, 2013)

$$\Delta x(t_n - T) = \alpha\phi^2(t_n - T) + \beta_l\psi^2(t_n) + \beta\psi(t_n) + b_0 \tag{4.9}$$

where ϕ stands for follower's velocity, ψ stands for leader's velocity, Δx is the minimum safe distance between leading and following vehicles. Empirical parameters of the CA model are those presented in Brackstone and McDonald (1999):

- unique reaction time $T = 0.75 seconds$;
- $a = 0.00028$;
- $\beta_l = 20.0084$;
- $b = 0.784$;
- $b_0 = 4.1$.

My model, as in Eluyode and Akomolafe (2013), uses the Gipps model, an enhanced version of the classic CA approach.

Safe Distance Model - Gipps' Model Kometani and Sasaki (1959) studied the first model based on safe distance concept. They stated (Ranjitkar *et al.*, 2005) that the driver of following vehicle chooses his speed based on a safe following distance to avoid possible collision with the vehicle ahead.

In the late-1970s Peter G. Gipps developed an enhanced car-following model that can be calibrated using common sense assumptions on driving behaviour. These include acceleration, deceleration, maximum speed, etc. Gipps' model is widely preferred for simulation purpose. The parameters 2.5 and 0.025 are arbitrarily chosen as proposed by its developer, while acceleration rate a, jammed spacing s can be assigned with some fixed values but the parameters like response time T, braking rate b, maximum desired speed V and assumed braking rate need to be optimized. Gibbs model based on driver behaviour and expectancy for vehicles in a stream of traffic. Limitations on driver and vehicle parameters for safety purposes mimic the traits of vehicles following vehicles in the front of the traffic stream. Difference in Gipps' model by other models consists in that Gipps uses a time step within the function, τ, to reduce the computation required for numerical analysis.

The reason for modelling vehicles in a stream of traffic and their interactions comes from the need to analyse changes to roadway parameters. Indeed, many factors including driver, traffic flow and roadway conditions, affect how traffic behaves. Gipps' model general form current to that time is:

$$a_n(t+\tau) = l_n \frac{[v_{n-1}(t) - v_n(t)]^k}{[x_{n-1}(t) - x_n(t)]^m} \tag{4.10}$$

which is defined primarily by one vehicle (noted by subscript n) following another (noted by subscript $n-1$, reaction time of the following vehicle, the location, speed and acceleration of the following vehicle and finally model constants, needed to adjust the model to real-life conditions.

Gipps' new and improved model should reflect the following properties:

1. The model should reflect real conditions;
2. Model parameters should correspond to observable driver characteristics without undue calculation;
3. The model should behave as expected when the interval between successive recalculations of speed and position is the same as driver reaction time.

Model notation:

- a_n is the maximum acceleration the driver of vehicle n wishes to undertake;
- b_n is the most severe braking the driver of vehicle n wishes to undertake, with $b_n < 0$;

- s_n is the effective size of vehicle n, that is, the physical length plus a margin into which the following vehicle is not willing to intrude, even when at rest;
- V_n is the speed at which the driver of vehicle n wishes to travel;
- $x_n(t)$ is the location of the front of vehicle n at time t;
- $v_n(t)$ is the speed of vehicle n at time t;
- τ is the apparent reaction time, a constant for all vehicles.

Gipps' model comprises a set of limitations. The following vehicle is limited by the constraints:

1. It will not exceed its driver's desired speed;
2. Its free acceleration should first increase with speed as engine torque increases then decrease to zero as the desired speed is reached;

$$v_n(t+\tau) \leq v_n(t) + 2.5a_n\tau\left(1 - \frac{v_n}{V_n}\right)\sqrt{0.025 + \frac{v_n(t)}{V_n}} \tag{4.11}$$

3. Braking is given by

$$x^*_{n-1} = x_{n-1}(t) - \frac{v_{n-1}(t)^2}{2b_{n-1}} \tag{4.12}$$

for vehicle $n-1$ at point x^*_{n-1}, where x^*_n for vehicle n is given by

$$x^*_n = x_n(t) + [v_n(t) + v_n(t+\tau)]\frac{\tau}{2} - \frac{v_n(t+\tau)^2}{2b_n} \tag{4.13}$$

at time $t+\tau$.

For safety reasons, driver of vehicle n (the following vehicle) must ensure that the difference between point where vehicle $n-1$ stops, x^*_{n-1}, and the effective size of vehicle $n-1$, s_{n-1}, is greater than the point where vehicle n stops, x^*_n. However, Gipps finds the driver of vehicle n allows for an additional buffer and introduces a safety margin, of delay θ when driver n is travelling at speed $v_n(t+\tau)$. Thus the braking limitation is given by

$$x_{n-1}(t) - \frac{v_{n-1}(t)^2}{2b_{n-1}} - s_{n-1} \leq x_n(t) + [v_n(t) - v_n(t-\tau)]\frac{\tau}{2} + v_n(t+\tau)\theta - \frac{v_n(t+\tau)^2}{2b_n} \tag{4.14}$$

Because a driver in traffic cannot estimate b_{n-1}, it is replaced by an estimated value $\hat{b}$. Therefore, the above after replacement yields,

$$-\frac{v_n(t+\tau)^2}{2b_n} + v_n(t+\tau)\frac{\tau}{2+\theta} - [x_{n-1}(t) - s_{n-1} - x_n(t)] + v_n(t)\frac{\tau}{2} + \frac{v_{n-1}(t)^2}{2\hat{b}} \leq 0 \tag{4.15}$$

If the introduced delay, θ, is equal to half of the reaction time, $\frac{\tau}{2}$, and the driver is willing to brake hard, a model system can continue without disruption to flow. Thus, the previous equation can be rewritten with this in mind to yield

$$v_n(t+\tau) \leq b_n\tau + \sqrt{b_n^2\tau^2 - b_n\left(2\left[x_{n-1}(t) - s_{n-1} - x_n(t)\right] - v_n(t)\tau - \frac{v_{n-1}(t)^2}{\hat{b}}\right)} \tag{4.16}$$

If the final assumption is true, i.e. the driver travels as fast and safely as possible, the new speed of the driver's vehicle is given by the final equation being Gipps' Model:

$$v_n(t+\tau) = \min\Bigg\{v_n(t) + 2.5a_n\tau\left(1 - \frac{v_n(t)}{V_n}\right)\sqrt{0.025 + \frac{v_n(t)}{V_n}} \\ ; b_n\tau + \sqrt{b_n^2\tau^2 - b_n\left[2\left[x_{n-1}(t) - s_{n-1} - x_n(t)\right] - v_n(t)\tau - \frac{v_{n-1}(t)^2}{b_{n-1}}\right]}\Bigg\} \tag{4.17}$$

where the first argument of the minimization regimes describes an uncongested roadway and headways are large, and the second argument describes congested conditions where headways are small and speeds are limited by followed vehicles. These two equations used to determine the velocity of a vehicle in the next time step represent free-flow and congested conditions, respectively. If the vehicle is in free-flow, the free-flow branch of the equation indicates that the speed of the vehicle will increase as a function of its current speed, the speed at which the driver intends to travel, and the acceleration of the vehicle. Analysing the variables in these two equations, it becomes apparent that as the gap between two vehicles decreases (i.e. a following vehicle approaches a leading vehicle) the velocity given by the congested branch of the equation will decrease and is more likely to prevail.

Recently, Kruass (1997) proposed a variant of the Gipps model. It is a stochastic model as it includes a stochastic term which is set to zero to unify the comparison. All other parameters to be optimized are same as of Gipps model i.e. response time T, braking rate b and maximum desired speed V.

Rational update function According to (5.9), as for Eluyode and Akomolafe (2013), the rational agency computes the minimum distance d_{1s} that vehicle c must keep from leading vehicle (neighbour 1) given current speeds of the two vehicles (avoid tailgating). This is further used in the Gipps model to adjust the speed of current car c with regard to the safety distance, as (5.17). Thus, the output of rational agency is a potential set of actions, which, depending on the physical situation on the road, consists of current vehicle's speed at next time step t_{n+1}.

Strength update function Since the rational agency mainly acts in a prescribed manner based on a car-following model, the strength of this agency can be assumed to depend on contextual input from neighbour 1 only, and also on values of personality traits (Eluyode and Akomolafe,

2013). Indeed, the need for rationality increases as the following distance falls below the safe distance and decreases otherwise. Also, rationality is expected to increase as personality traits are on the positive side and decrease otherwise. The *rational strength update function* is

$$rat_{str}(t_n) = x_1(t_n) + \frac{P_C + P_A - P_N}{3} \tag{4.18}$$

Reactive agency

Reactive agency implements the driver's physiological reaction (reflex) when a collision with the vehicle in front is unavoidable given the deceleration capability of the current vehicle, the speed difference between the two vehicles, and the distance between them.

Reactive update function Reactive agency is implemented, as in Eluyode and Akomolafe (2013), in a rule-based approach in which the collision possibility is calculated using manipulations of standard equations of motion

$$\Delta x_{danger} = \frac{(\phi - \psi)^2}{d_M} \tag{4.19}$$

where, as before, ϕ stands for follower's velocity, ψ stands for leader's velocity, d_M is the vehicle's maximum deceleration capability

Strength update function Unlike the other two agencies, reactive agency has a binary strength update function, which signals a dangerous situation with regard to the relation between the current vehicle c and the leading vehicle (neighbour 1)

$$rct_{str}(t_n) = \begin{cases} 1, d_{1r} \leq x_{danger} \\ 0, d_{1r} > x_{danger} \end{cases} \tag{4.20}$$

If distance d_{1r} between c and neighbour 1 is equal or lower than Δx_{danger}, reactive agency generates the maximum strength, which translates into 100% chances of winning the competition with other agencies and the agency proposes full deceleration capability. If distance d_{1r} between c and neighbour 1 is higher than Δx_{danger}, danger does not exist. Thus reactive agency generates a disqualifying value of strength, which will translate into 100% chances of losing the competition with other agencies (Eluyode and Akomolafe, 2013)

Bidding strategy

The bidding process represents the computational instantiation of SoM principle of non-compromise, with internal agencies of SoM agent participating in a competition with only one winner (Eluyode and Akomolafe, 2013). Thus, a bidding process must be implemented instead of a compromise-based conflict resolution mechanism. Internal agencies bid with their

true value of strength in the interval $(0, 1)$ and winner is the agency with the highest strength (Eluyode and Akomolafe, 2013).

4.4 Artificial Neural Networks

> "One day in 1961, the meteorologist Edward Lorenz was using a computer to model weather with a set of twelve equations. He had run the equations to evolve a weather pattern for a particular time period, but wanted to see the end of the pattern again. To save time, rather than start the simulation at the beginning of the time period, he started it midway through, using parameters from his previous printout for the starting point. An hour later, when he returned, he was surprised: rather than reproduce the prior results, the new results ended up wildly different. He found the reason: the parameters he entered for the second run were accurate only to three decimal places, whereas the initial run had results accurate to six decimal places. That small difference in initial conditions led to wildly different weather. This effect, sensitive dependence on initial conditions, became known as the butterfly effect: a butterfly's flight in Texas today can produce a tiny atmospheric change that a month later produces a storm in Thailand. This result led to the conclusion that it is impossible to predict the weather for more than a few days, and to chaos theory. Chaos theory is the mathematical study of dynamic systems that are highly sensitive to initial conditions (chaotic systems). Many complex systems - such as the weather, populations, neurons, and economic systems - can also be chaotic systems." (Mills, 2010)

4.4.1 Background of the Study

Today's computers can be traced back at least to Blaise Pascal's 1642 mechanical calculator. The modern era in computing started with the unveiling of ENIAC on February 15^{th}, 1946. The development of the transistor in 1948 enabled the creation of integrated circuits in 1958, which, in turn, enabled the first microprocessor in 1971. Since then the clock frequency of the microprocessors has increased 1,000-fold. As remarkable as this evolution is, it has been headed in a direction diametrically opposite to the computing paradigm of the brain. Consequently, today's microprocessors are eight orders of magnitude faster (in terms of clock rate) and four orders of magnitude hotter (in terms of power per unit cortical area) than the brain. Considering overall energy consumption underscores the divergence between the brain and today's computers even more starkly. However, the exact workings of the human brain are still a mystery. In short, it is nothing like the currently available electronic computers, or even Artificial Neural Networks. These Artificial Neural Networks (Anderson and McNeill, 1992) try to replicate only the most basic elements of this complicated, versatile, and powerful organism. They do it in a primitive

way. But for the software engineer who is trying to solve problems, neural computing was never about replicating human brains. It is about machines and a new way to solve problems.

Human system and computing system have as their nerve centre brain and computer (Central Processing Unit, CPU) respectively (Eluyode and Akomolafe, 2013). The computer is capable of executing efficiently arithmetic and logic operations while the brain is efficient in executing operations that involve pattern recognition and matching (Chen *et al.*, 1986). The brain also recognizes and acts on constantly changing patterns of input stimuli. In the past years, the computing industry has attempted to make a computer that treats information as patterns in the same way that the brain does.

The computing system of today emphasizes the efficient and intelligent performance of the computer users. It therefore facilitates the interactive processing of information in a manner that is intelligent, menu and dialogue driven (Bezdek, 1993; Akinyokun, 2002). This has consequently brought about a challenge on the study of Neural Networks. A Network inspired by the structure and function of biological neural system. A Neural Network that attempts to mimic the functions of the biological central nervous system and some of the sensory organs attached to it.

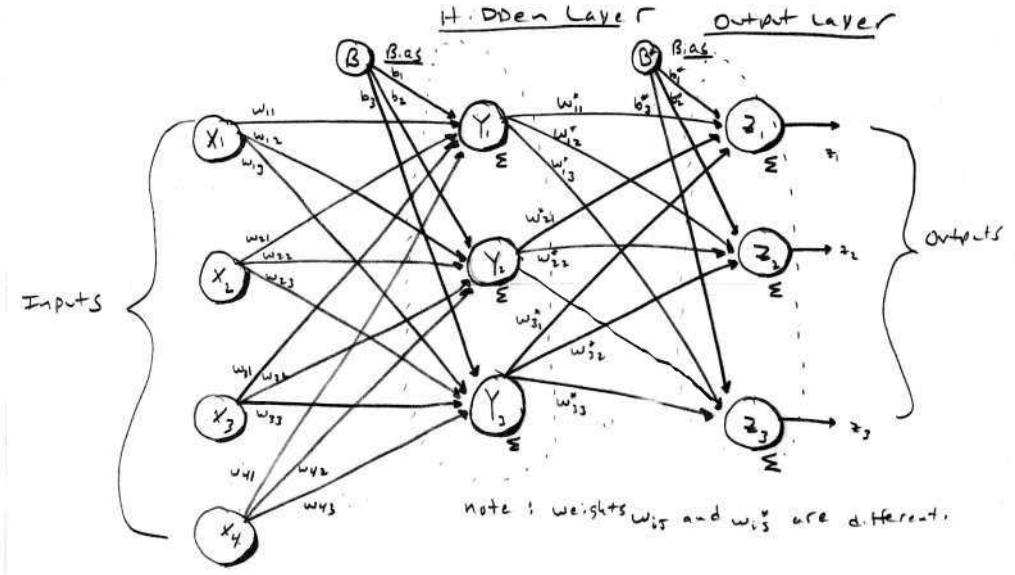

Figure 4.8: Feed-forward Artificial Neural Network

The trends of the developments in the computing industry over the years are as follows:

1950 - 1970: The emphasis was on the science of computing and the major development was the processing of numeric data.

1971 - 1980: The emphasis was on the engineering of computing and the major breakthrough was the processing of text data. Computers were being used for word processing, type setting and desk top publishing.

1981 - 1990: The emphasis was on the re-engineering of computing and the major development was the processing of image and graphic data. During this period, computers were being used to draw multidimensional graphs, scan, store, verify and validate images, signatures and classified documents.

1991 to date: The emphasis has been on the technology of computing and the major development has been audio and video processing. During this period, a class of computing system described as multimedia system emerged and the development brought about the situation whereby the dividing line between a digital computer and an analog computer no longer exist.

The Brain-inspired Computer

Along the way, progressing through Phase 0, Phase 1, Phase 2, and Phase 3, IBM have journeyed from neuroscience to supercomputing, to a new computer architecture, to a new programming language, to algorithms, applications, and now to a new chip, the "TrueNorth" (`http://www.research.ibm.com/articles/brain-chip.shtml`). Unlike the prevailing Von Neumann architecture (but like the brain) TrueNorth has a parallel, distributed, modular, scalable, fault-tolerant, flexible architecture that integrates computation, communication, and memory and has no clock. It is fair to say that TrueNorth completely redefines what is now possible in the field of brain-inspired computers, in terms of size, architecture, efficiency, scalability, and chip design techniques.

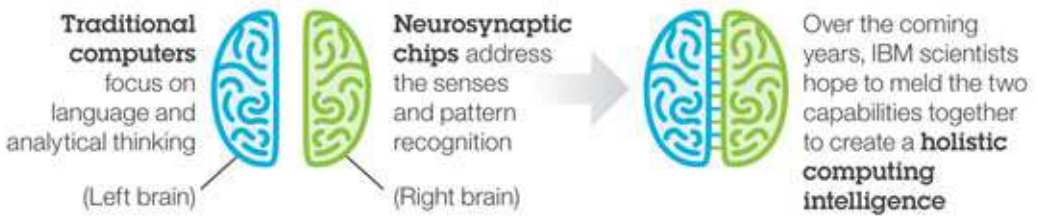

Figure 4.9: Holistic intelligence

There are two factors: technology and architecture. Unlike today's inorganic silicon technology, the brain uses biophysical, biochemical, organic wetware. While future enabling nanotechnology is underway, IBM focused on the second factor: architecture innovation-specifically, on minimizing the product of power, area, and delay in a system that could be implemented in today's state-of-the-art technology.

The important development in the computing industry between 1950 and 1990 has John Von Neumann architecture as their platform. The major characteristics of the architecture are as follows:

1. computing system that operates on discrete signals;

2. memory system that records discrete signals to be processed, a sequence of specific instructions that serially processes the signals and produces the output reports;

3. computing system that operates by a continuous cycle of fetching an instruction from the memory, executing the instruction and storing the result of the instruction in the memory;

4. computing system which emphasizes the efficient and primitive performance of the computer rather than the efficient and intelligent performance of the brain of the human being which uses it. Computing systems were used to automate the operations of human experts, as such, it was considered as an alternative rather than a partner to human experts (Akinyokun, 2002).

Recent development is aimed at building computing system that will operate intelligently like the human brain. Six years ago, IBM and their university partners embarked on a quest: to build a brain-inspired machine that at the time appeared impossible (`http://www.research.ibm.com/articles/brain-chip.shtml`). Today, in an article published in Science, IBM delivers on the DARPA SyNAPSE metric of a one million neuron brain-inspired processor. The chip consumes merely 70 milliwatts, and is capable of 46 billion synaptic operations per second, per watt. Literally a synaptic supercomputer in your palm.

4.4.2 BNN (Biological Neural Network)

Our brains (Sher, 2012) are biological parallel computers, composed of roughly 100.000.000.000 (one hundred billion) signal processing elements called *Neurons*. Like a vast graph, these neurons are connected with each other in complex topological patterns. Each neuron in this vast processing graph accepts *signals* from thousands of other neurons, processes those signals, and then outputs a *frequency encoded signal* and passes it onwards to thousands of other neurons. Though each neuron on its own is relatively easy to understand, when you connect together a few billion of them, it becomes incredibly difficult to predict the outcome given some specific input. If you are careful and connect these biological signal processing elements in some particular pattern, the final output of this vast graph might even be something useful, an *intelligent system* for example.

A neuron is an electrically excitable cell that processes and transmits information through electrical and chemical signals. These signals between neurons occur via synapses, specialized connections with other cells. Neurons can connect to each other to form neural networks *(NNs)*.

A typical neuron, as shown in fig. 4.2, is a cell composed of three main parts: the *soma* (cell body), the *dendrites*, and the *axon*.

Synaptic signals from other neurons are received by the soma and dendrites; signals to other neurons are transmitted by the axon. A typical *synapse*, then, is a contact between the axon of one neuron and a dendrite or soma of another. Synaptic signals may be *excitatory* or *inhibitory*. If the net excitation received by a neuron over a short period of time is large enough, the neuron

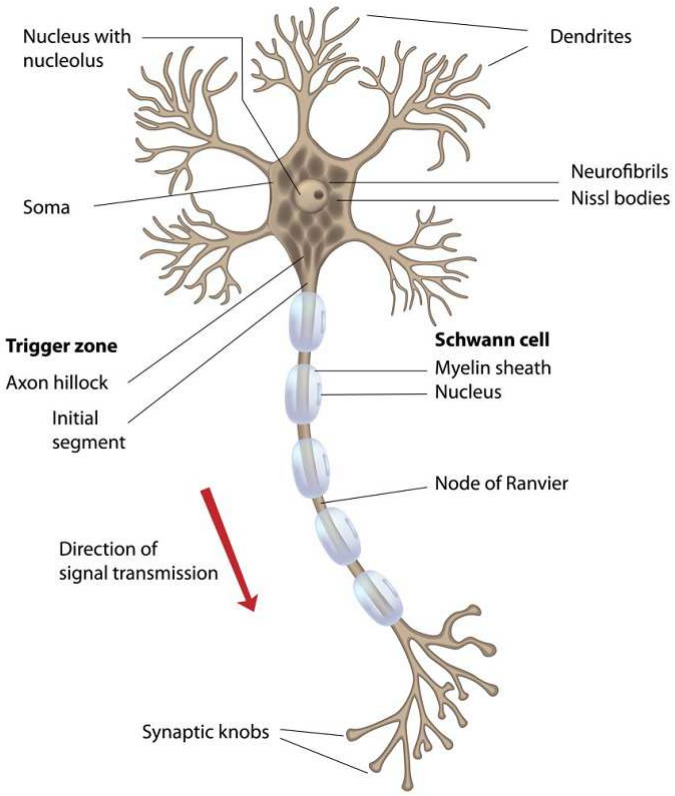

Figure 4.10: Biological neuron

generates a brief pulse called an *action potential*, which originates at the soma and propagates rapidly along the axon, activating synapses onto other neurons as it goes (fig. 4.3).

The **soma** is the central part of the neuron. It contains the nucleus of the cell, and therefore is where most protein synthesis occurs. The **dendrites** of a neuron are cellular extensions with many branches, and metaphorically this overall shape and structure is referred to as a *dendritic tree*. This is where the majority of input to the neuron occurs. The **axon** is a finer, cable-like projection that can extend tens, hundreds, or even tens of thousands of times the diameter of the soma in length. The axon carries nerve signals away from the soma (and also carries some types of information back to it). Many neurons have only one axon, but this axon may (and usually will) undergo extensive branching, enabling communication with many target cells.

Biological neurons output frequency encoded signals, and they process the incoming frequency encoded signals through spatiotemporal integration of those signals. If the excitation level at a given time surpasses its threshold, an action potential is generated and passed down the axon.

The neurons don't just accept incoming signals and produce outgoing signals, the neurons also change over time based on the signals they process. This change in the way neurons respond to signals by adding more receptors to the dendrites, or subtracting receptors from the dendrites, or

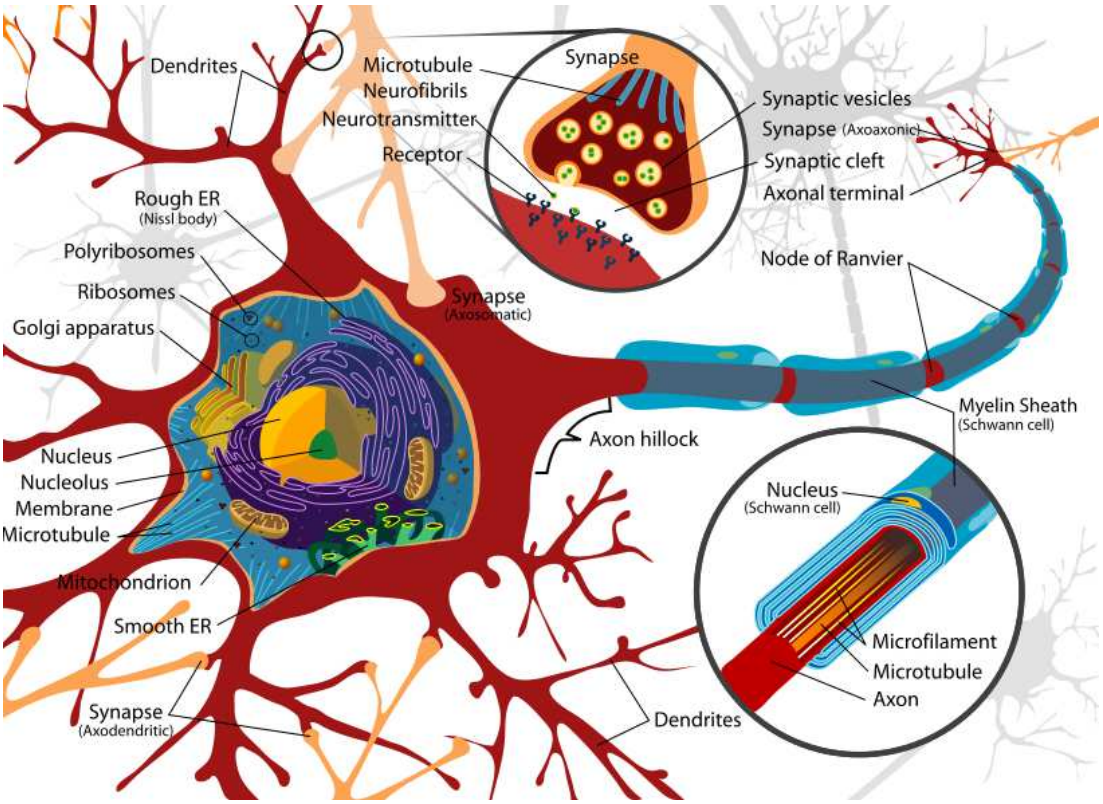

Figure 4.11: Synapse

modifying the way their receptors work, is one of the processes by which a neural network *learns* and changes its excitability towards certain signals, it is how we accumulate experience and form memories. Other ways by which a neural network learns is through the axons branching and making new connections, or breaking old connections. And finally the neural network changes in the way it processes signals through having the very fluid in which the neurons are bathed changed and chemically modified, through drugs or other means for example.

Neurons can change over time based on the signals they process, the neurons change biologically, they change their information processing strategies, and they can form new connections to other neurons, and break old ones. This process is called *neuronal plasticity*.

4.4.3 ANN (Artificial Neural Network)

Artificial Neural Networks are self learning mechanisms. Writers have hyped that these neuron-inspired processors can do almost anything. These exaggerations have created disappointments for some potential users who have tried, and failed, to solve their problems with Neural Networks (Anderson and McNeill, 1992). These application builders have often come to the conclusion that Neural Networks are complicated and confusing. Unfortunately, that confusion has come

from the industry itself. An avalanche of articles have appeared touting a large assortment of different Neural Networks, all with unique claims and specific examples. Currently, only a few of these neuron-based structures, paradigms actually, are being used commercially. One particular structure, the *feedforward, Backpropagation network*, is by far and away the most popular. Most of the other Neural Network structures represent models for "thinking" that are still being evolved in the laboratories. Yet, all of these Networks are simply tools and as such the only real demand they make is that they require the Network architect to learn how to use them.

Artificial Neural Networks (ANNs) are *biologically inspired devices* used for mapping a set of inputs into a set of outputs (Beltratti *et al.*, 1996). The mapping is carried out by the processing elements, called *artificial neurons*, which are interconnected to form a network divided into layers (usually three): like a biological neuron, an artificial one accepts signals (*inputs*) through its artificial dendrites (*input layer*), processes those signals in its artificial soma (*intermediate layers*), and outputs the processed signals to other neurons it is connected with (*output layer*). It is a concise representation of what a biological neuron does.

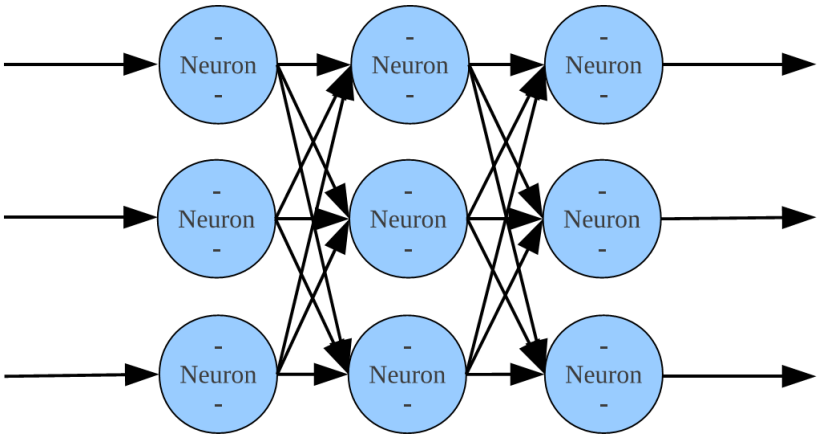

Figure 4.12: Artificial Neural Network

Therefore, the biological neural network is a vast graph of parallel processing simple biological signal integrators (Sher, 2012), and the Artificial Neural Networks too is a vast graph of parallel processing simple signal integrators. The neurons in a biological neural network can adapt, and change its functionality over time, which too can be done in Artificial Neural Network through simulated neural plasticity.

However, there are of course differences (Sher, 2012). We abstract the functionality undertaken by the receptors on the dendrites with simple weights, nevertheless, each incoming signal is weighted, and depending on whether the weight is positive or negative, each incoming signal can act as an excitatory or inhibitory one, respectively. We abstract spatiotemporal signal integration that occurs at the axon hillock with an activation function (which can be anything,

and as complex as the researcher desires), nevertheless, the weighted signals are integrated at the output point of the artificial neuron to produce the final output vector, which is then passed onwards to other neurons. And finally, we abstract the functionality undertaken by the axon with simple signal message passing, nevertheless, the final output signal is propagated, diligently, to all postsynaptic artificial neurons.

One thing though that differs significantly in the typical artificial neural networks, and the biological neural networks is that the neurons in a biological neural network *frequency encode* their signals, whereas in the Artificial Neural Networks, the neurons *amplitude encode* their signals. What has more flexibility? Frequency encoded neural networks systems or the amplitude encoded ones? It is difficult to say, but we do know that both, biological and artificial neural networks are *Turing complete*, which means that both possess the same amount of flexibility.

In the 1930s, while thinking about Gödel's incompleteness theorem, Alan Turing wondered if he could construct a machine that not only would perform simple arithmetic, but also would help investigate the limits of what can be computed. In 1937, he described such a machine, now known as a Turing Machine. It consists of three parts:

1. A tape that is infinitely long and divided into successive cells, each of which has either a 0, a 1, or a blank.

2. A read/write head that can move to a particular cell and either read its symbol or write a symbol to it. Associated with the read/write head is a current state.

3. A transition table with a set of rules that tells the read/write head its next state and its next action (write a symbol, or move one cell left or right), based on its current state and the symbol at the head's current position.

With an appropriate transition table, a Turing machine can perform any arithmetic or logical function, even one that is infinitely long. Modern computers are finite implementations of a Turing Machine.

Remarkably, Turing proved that there exists a Turing Machine that can reproduce the behaviour of any other Turing Machine, including itself. This is a Universal Turing Machine.

The implications of the fact that both systems are universal Turing machines is that even if a single artificial neuron does not do as much, or perform as a complex computation as a single biological neuron, we could put a few artificial neurons together into an artificial neural circuit, and this artificial neural circuit will have the same processing power and flexibility as a biological neuron. On the other hand, note that frequency encoding signals takes more time, because it will at least take the amount of time between multiple spikes in the spike train of the signal for the message to be forwarded (since it is the frequency, the time between the spikes that is important), whereas in an amplitude encoded message, the single spike, its amplitude, carries all the information needed.

We do know one thing though, the limits of speed, signal propagation, neural plasticity, life span of the neuron, integration of new neural systems over the organism's lifetime, are all

limited in wetware by biology. None of these limitations are present in hardware, the only speed limit of signal propagation is that of light in a hardware based neural computing system. The non biological neural computer can add new neural circuits to itself over lifetime, and that lifetime span is unlimited, given that hardware upkeep is possible. A typical artificial neuron does not simulate a biological neuron at the atomic, or even molecular level. Artificial neurons are abstractions of biological neurons, they represent the essentials of biological neurons, their nonlinear signal integration, plasticity, and concurrency.

Properties of Neural Networks

Basic properties of neural networks, independent of the specific structure (number of layers, number of neurons) are the following Beltratti *et al.* (1996):

Learning: the capability of the network to adapt its behaviour to the environment, or in other words to autonomously build a representation of the map from inputs to outputs on the basis of a set of examples;

Generalization: the ability to react in a coherent way to imperfect inputs or to inputs not explicitly seen during learning;

Soft degradation: the alteration or elimination of some elements of the network does not prevent it from working but only induces a smooth degradation in performance.

Flexibility: the ability to approximate almost each regression function $f(x)$;

Consistency: the estimated regression function is identified by a limited number of components.

Description of the Neuron

The artificial neuron (fig. 4.5) is a processing element that:

1. receives some signals as inputs
$$[I_0, I_1, .., I_k] \tag{4.21}$$

2. computes the weighted sum of all the signals that are received as inputs to generate a global net input
$$net_j = \sum_{i=0}^{k} I_j w_{ij} \tag{4.22}$$

3. yields as output O_j a transformation of the net input by means of an activation function f
$$O_j = f(net_j) \tag{4.23}$$

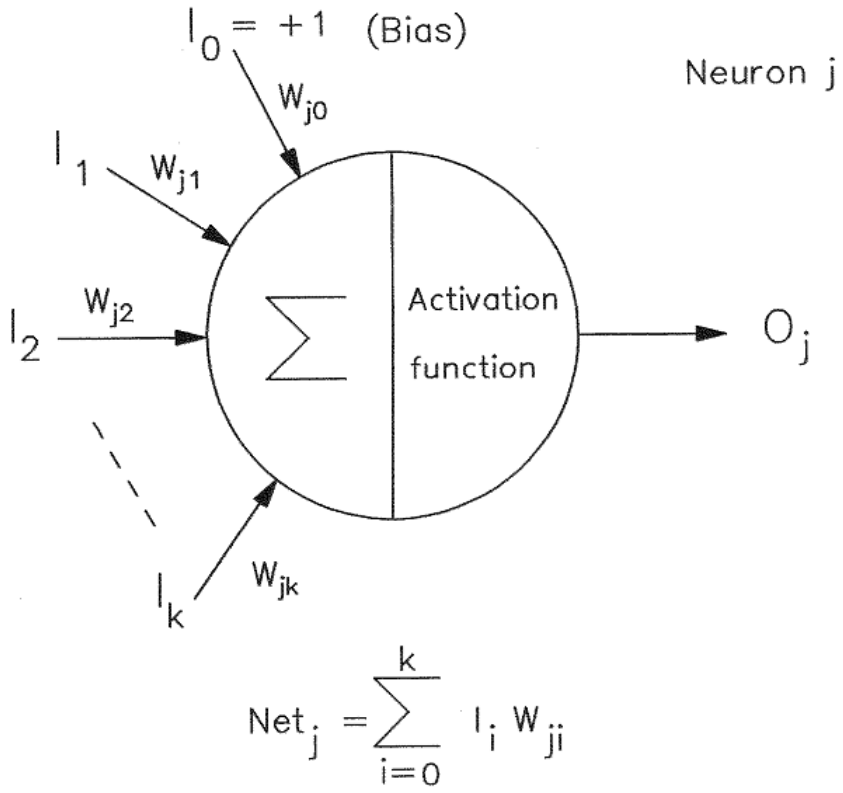

Figure 4.13: Artificial Neuron's structure

Analogously to natural neurons, the *activation function* has two main characteristics (Beltratti *et al.*, 1996):

- the existence of a threshold;
- upper and lower boundedness.

A natural neuron fires only when its net input reaches a given threshold θ, and its activation level never surpasses a given saturation level. In order to apply the most widely used learning algorithms, based on the derivatives of the activation function, the threshold function is often approximated by a differentiable one such as the *logistic* or the *arctan*. In other applications a linear function is used. The existence of the threshold is replicated by introducing a *bias* among the inputs when the logistic is used as an activation function. This is equivalent to considering another input signal at a constant value of 1.

The effects of the various input signals on the neuron are not homogeneous, but depend on the strength of their connection with the source of the signals. Such a connection is represented by a weight w_{ij} going from source neuron i to destination neuron j. The absolute value of this weight represents the strength of the connection: a positive (negative) sign reflects an excitatory (inhibitory) link.

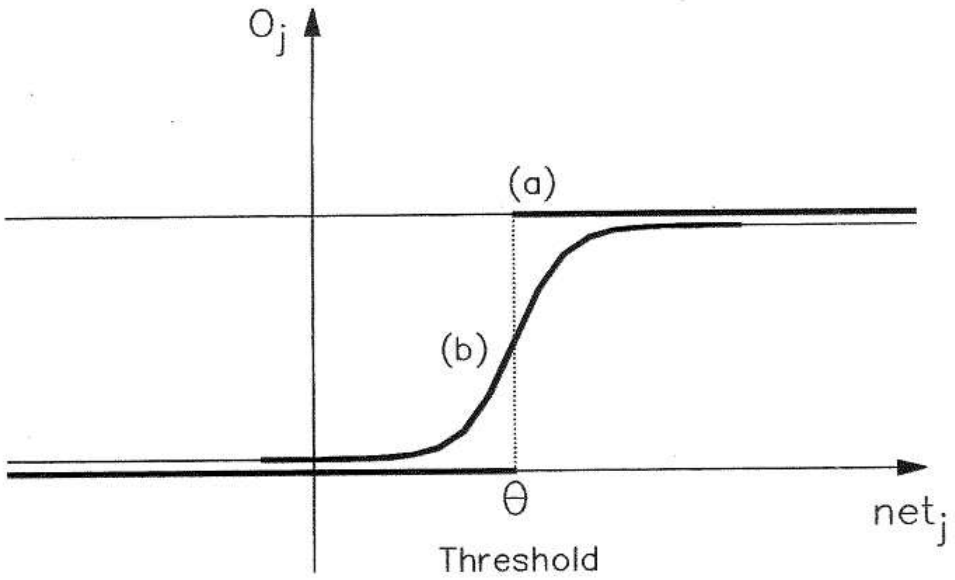

Figure 4.14: Threshold (1) and Logistic (2) activation functions

The structure of the Network

A neuron by itself is a simple processing element. It is when we interconnect these neurons together, in parallel and in series, when we form a neural network, that true computational power emerges. The way these connections are made (*Network's architecture*), the way weights of the connections are determined (*learning algorithm*) and the *activation functions* that determines the behaviour of the neurons are the features that characterize a Neural Network.

The *depth* of a Neural Network is the number of layers that compose it.

1. One input layer which receives signals from outside. Then it transmit them to the subsequent layer of the network. The number of input neurons is problem-specific and depends on the amount and type of information which is processed by the network;

2. One or more intermediate (or hidden) layers whose neurons perform the processing described at point 1. The number of hidden layers determines the complexity of the processing done by the network: while most applications are based on ANNs with one hidden layer, complex problems may require two or three layers;

3. One output layer which returns signals to the outside, once the network has processed the inputs. Output neurons perform the same signal processing as described at 1.

The way neurons are connected fully determines the structure of the network. Assigning each neuron a layer allows us to see whether the connections from one neuron to another are *feed-forward* or *recurrent* (Sher, 2012):

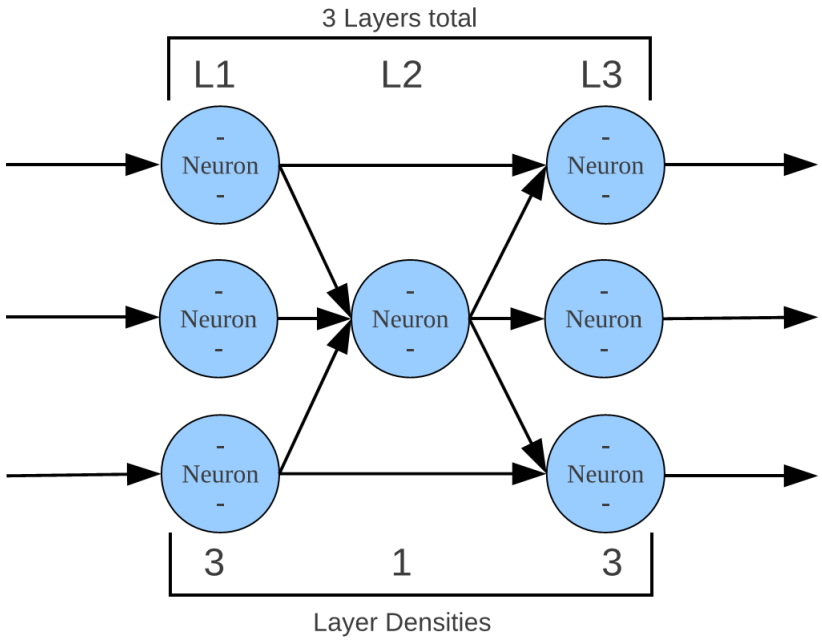

Figure 4.15: Three-layered Neural Network. The first layer has 3 neurons, the second has 1, the third has 3.

Feed-forward Networks: Fully connected Neural Network is one of the most widely used structure, where all the neurons of each layer are connected with (and only with) all the neurons of the subsequent layer: input neurons with hidden neurons, and hidden neurons with output neurons. Within a single layer, there is no connection from one neuron to another. When connections are oriented and the direction of information flow goes only from input to hidden and from hidden to output, the network is called *feed-forward* network (fig. 4.8). Their mechanism is described in the following example. In the network of fig. 4.8, the neurons are grouped in three layers: input, hidden and output. There are four input neurons (denoted 1, 2, 3, 4), three hidden neurons (5, 6, 7) and two output neurons (8, 9). The structure is fully interconnected because each neuron in the input layer is connected with each neuron in the hidden layer, which in turns communicates directly with each neuron in the output layer. Each connection has an associated weight w_{ij}, the weight from neuron i to neuron j, which defines the strength of the connection. The information received by the neurons flows only in the forward direction, from input to output, without feedback. Bias is not considered for the sake of simplicity. If the four inputs have values I_1, I_2, I_3 and I_4, then the overall signal passed to neuron 5 is equal to:

$$net_5 = w_{51}I_1 + w_{52}I_2 + w_{53}I_3 + w_{54}I_4 \tag{4.24}$$

where net_i denotes the input to the $i - th$ neuron. Neuron 5 then transforms this linear

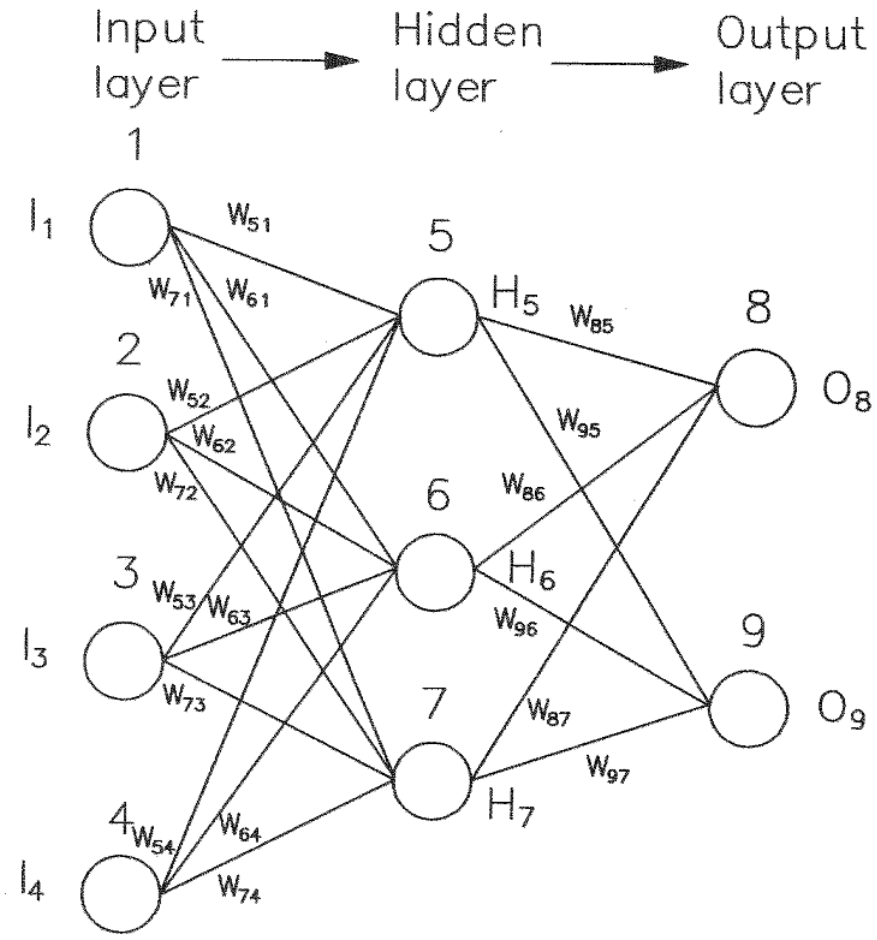

Figure 4.16: Feed-forward network

combination of inputs by means of the logistic activation function and yields an output equal to:

$$H_5 = [1 + exp(-net_5)]^{-1} \tag{4.25}$$

Together with the output of neurons 6 and 7, this output then forms a linear combination which is passed on to neuron 8 and 9. The signal received by neuron 8 is:

$$net_8 = w_{85}H_5 + w_{86}H_6 + w_{87}H_7 \tag{4.26}$$

These neurons are again transformed by the logistic to become the final output of the neural network. The output of neuron 8 is therefore:

$$O_8 = [1 + exp(-net_8)]^{-1}$$

Matrix notation can be used for a concise description. If A and$B$ are matrices that collect the values of the weights from input to hidden and from hidden to output respectively, O is the output vector of the network, I is the input vector and f is the activation function (defined over appropriate vectors), then the network can be described as:

$$O = f(f(IA)B)$$

For the example in fig. 4.8, the two matrices are respectively:

$$A_{(4,3)} = \begin{bmatrix} w_{51} & w_{61} & w_{71} \\ w_{52} & w_{62} & w_{72} \\ w_{53} & w_{63} & w_{73} \\ w_{54} & w_{64} & w_{74} \end{bmatrix} \tag{4.27}$$

$$B_{(3,2)} = \begin{bmatrix} w_{85} & w_{95} \\ w_{86} & w_{96} \\ w_{87} & w_{97} \end{bmatrix} \tag{4.28}$$

The weights are the devices encoding the information into the network. So, for a given structure, the behaviour of the network strictly depends on the set of weights. Their value must be properly set within the training phase.

Recurrent Networks: Other architectures exist which differ from the one we have described here by allowing intra-layer connections (links among neurons belonging to the same layer) or backward information flow, as in *recurrent* network (Sher, 2012), meaning some neuron A sends a signal to neuron B which itself is behind A, and whose original output signal is either fed directly to neuron A, or was forwarded to other neurons and then eventually got to neuron A before it itself produced its output signal (the recurrent signal that it sent back to neuron B). Indeed in recurrent Neural Networks, one can have *feed-forward* and *feedback* loop based neural circuits, and a neuron B could have sent a signal to neuron A, which then processed it and sent its output back to neuron B and so on (fig. 4.9). As can be seen in the recurrent Neural Network example, neuron A receives a signal from somewhere, processes it, sends a signal to neuron B, which processes the signals sent to it and then sends an output signal to neuron C, D, but also a recurrent signal back to A and itself.

What is significant about recurrent neural networks is that they can form *memory circuits*. For example, fig. 4.10 shows the neuron sends a signal back to itself. This means that at every moment, it is aware of its previous output, and that output is taken into account when producing a new output. The neuron has memory of its previous action, and depending on the weight for that recurrent connection, its previous signal at time step T can play a large or a small part in its output at a time step $T + 1$.

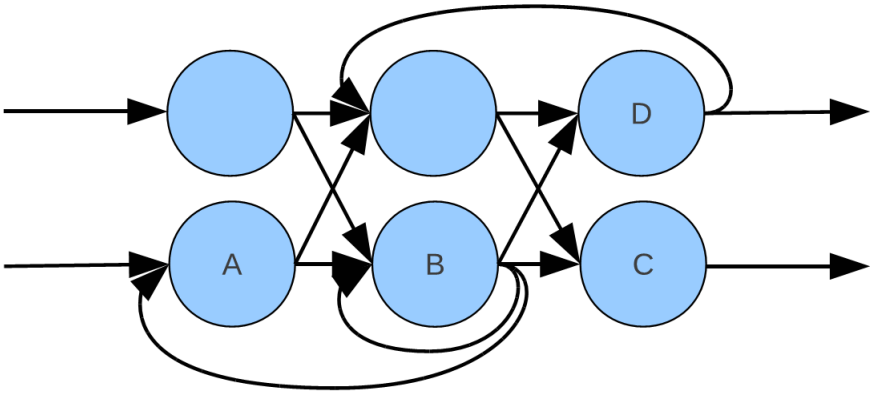

Figure 4.17: Recurrent Network

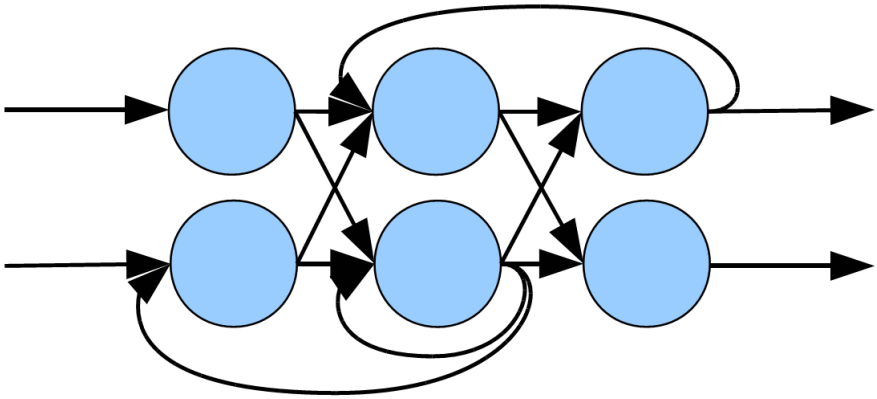

Figure 4.18: Three-layers recurrent Neural Network, with 3 recurrent connections

In fig. 4.11 neuron 1 has a recurrent connection to neuron 2, which outputs a signal back to neuron 1. This neural circuit too forms a memory system, because this circuit does not simply process signals, but takes into account the information from time step $T-2$, when making a decision with regards to the output at time step T. Why $T-2$? Because at $T-2$ neuron 1 outputs a signal to 2 rather than itself, it is then at $T-1$ that 2 outputs a signal to 1, and it is only at time T that 1 outputs a signal after processing an input from some other element, and a signal it output at $T-2$, which was processed by 2 before coming back to 1 again. Thus this memory loop is deeper, and more involved. Even more complex systems can of course be easily evolved, or engineered by hand.

ANNs can be seen as nonlinear models (Lee *et al.*, 1993). The specific functional forms used in nonlinear models imply of course that in general the function that generates the data is different from the one implied by Artificial Neural Networks (Beltratti *et al.*, 1996), so that the

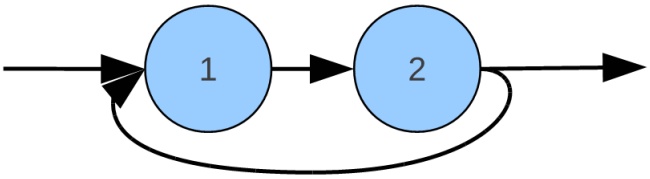

Figure 4.19: Two-layers recurrent Neural Network

econometric theory that needs to be used for Artificial Neural Networks is the one for misspecified nonlinear models. Their pre-specified structure notwithstanding, Artificial Neural Networks have the ability to approximate arbitrarily well any continuous function and its derivatives, and for this reason have been called *universal approximators* (Hornik *et al.*, 1989)

Many machine learning algorithms, such as logistic regression and nonlinear logistic regression can be implemented using neural networks.

Where does the input signals discussed until now come from? In real implementations the Neural Networks have to interact with the real or simulated world (Sher, 2012), and the signals they produce need to be somehow used to accomplish useful tasks and act upon those real or simulated worlds. For example, our own brain accepts signals from the outside world, and signals from our own body through the various sensory organs, and the embedded sensory neurons within those organs. For example our eyes, our skin, our nose are all sensory organs with large concentrations of sensory elements that feed the signals to the vast neural network we call our brain. These sensory organs, these sensors, encode the signals in a form that can be forwarded to, and understood by, the brain. The output signals produced by our brains also have no action without some actuators to interpret those signals, and then use those signals to act upon the world. The output signals are made sense of by the actuators, our muscles for example evolved to know how to respond when receiving signals from the motor neurons, and it is our muscles that perform actions upon the world based on the signals coming from the biological Neural Network. Thus, though it is the Neural Network that thinks, it is the Neural Network with sensors and actuators that forms the whole system. Without our sensory organs, our brain is in the dark, and without our muscles, it does not matter what we think, because we can have no affect on, and no way to interact with, the world. It is the same with Artificial Neural Networks. They require sensors, and actuators. A sensor can be a camera, which can package its output signals in a way that can be understood by the Neural Network, for example by representing the sensory signals as vectors. An actuator can be a motor, with a function that can translate the Neural Network's output vector into electrical signals that controls the actual motor. Thus it is the whole thing, the sensors connected to and sending the sensory signals to the Neural Network,

and the Neural Network connected to and sending its output signals to the actuators, that forms the full system.

Weaknesses of the Network

Arbitrariness: there are no strong criteria affecting the choice of the number of hidden layers nor goodness of fit measures;

Complexity in the estimation procedure;

Inference: there are not standard errors associated to the coefficients;

Interpretation: as long as the hidden layers number increase the interpretation of the results becomes more complex and difficult.

The training of the Network

As in the brain, a perceptron learns by adjusting the weights of its links until the output fits the underlying data. The way weights are set, the *training*, represents a distinctive feature of a Network in respect of another.

There is difference between learning and training processes. In true learning, the Neural Networks are able to change on their own and adapting through experience (Sher, 2012) due to neural plasticity. Neural plasticity is indeed the ability of the neuron to change due to experience. This is *Unsupervised Learning*. Conversely, under the training process (*Supervised Learning*), the Neural Network has a supervisor that tells it whether its answers are right or wrong, and it is the supervisor (an external algorithm) that modifies the Neural Network so that the next time it will hopefully produce a better answer. It consists in setting a Neural Network and observing how it behaves. Then verifying whether its behaviour is appropriate, i.e it represents the answer to some question, we change its weights to realize the desired mapping from inputs to outputs. Therefore, in the training process the synaptic weights of the neurons are modified and optimized by an outside supervisor. We can classify the following processes:

- Supervised learning;
- Unsupervised learning.

A kind of Neural Network trained to associate a set of input vectors to a set of output ones is called *associative memory*.

Supervised Learning Supervised Learning is a machine learning approach to inferring a target function from a training dataset composed of a set of training examples (Sher, 2012). Any such series of examples can be represented by time series (Beltratti *et al.*, 1996), i.e. a set of observations on variables at different time periods. Each training example is composed of an input vector and an expected output vector.

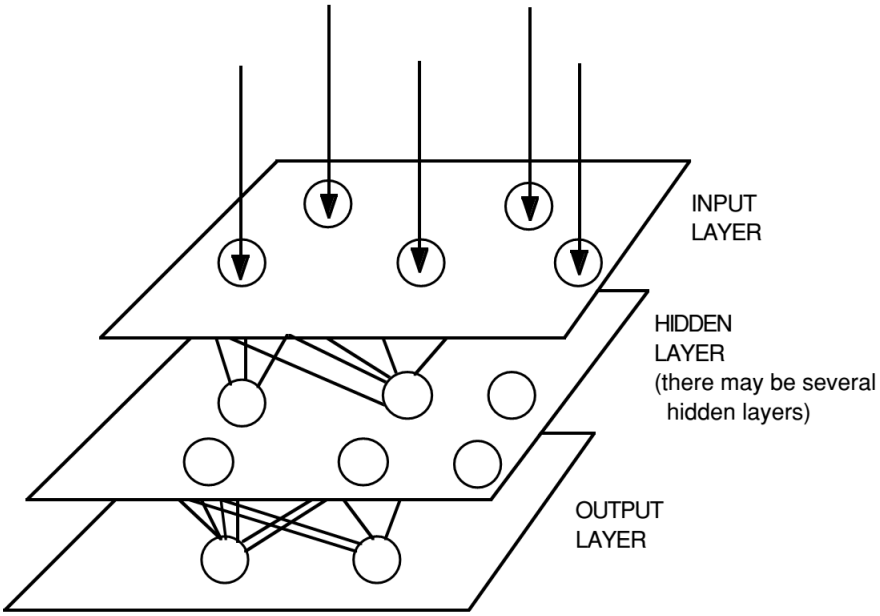

Figure 4.20: Artificial Neural Network diagram

When applying the supervised learning, i.e. the training process, a supervisor (or external system) compares the Neural Network's output to a correct, pre-calculated one. Then, based on the difference between the two, modifies the weights of the neurons based on some optimization algorithm. Therefore, a supervised "learning" algorithm can only applies when the answers is already known, i.e. it is possible to build a training set. Then it will be possible to use it with input signals it has not yet seen.

The most widely used of such supervised algorithms is the *Error Backpropagation* (McClelland *et al.*, 1986). It allows the Network to choose its weights in order to minimize a performance function defined over the output of the Network and some targets (Beltratti *et al.*, 1996). This performance function is generally the *sum of squared errors*, where the sum is taken with respect to both the number of available pattens and the number of output neurons. The error for pattern t and output neuron j is the difference between the target T_{tj} and the output of the network O_{tj}.

The Backpropagation algorithm is a supervised process. Consider a problem with T patterns, and a Network structure that includes K inputs, H hidden and J outputs. The objective function of the problem is (Beltratti *et al.*, 1996)

$$E(W) = \frac{1}{2} \sum_{t=1}^{T} \sum_{j=1}^{J} (T_{tj} - O_{tj})^2 \tag{4.29}$$

where W is a vector collecting the set of weights, multiplication by $\frac{1}{2}$ is introduced only for simplifying the expression of the derivatives computed in the learning algorithm.

In summary, the training of the Neural Network through the Backpropagation works as follows:

1. Create a multi-layered feed forward Neural Network, where each neuron has initial weights of the vector W drawn from a uniform distribution, generally set at $(-0.5, 0.5)$.

2. A pattern t is provided to the Network. The pattern is propagated forward through the network as

$$net_{th} = w_{h0} + w_{h1} I_{t1} + w_{h2} I_{t2} + .. + w_{hK} I_{tK} \tag{4.30}$$

with $h = 1, 2, .., H$, describes how signals I_t received from outside are transmitted to the hidden neurons.

$$H_{th} = [1 + exp\,(-net_{th})]^{-1} \tag{4.31}$$

with $h = 1, 2, .., H$, represents the non-linear transformation performed by hidden neurons through the logistic function.

$$net_{tj} = w_{j0} + w_{j1} H_{t1} + w_{j2} H_{t2} + .. + w_{jH} H_{tH} \tag{4.32}$$

with $j = 1, 2, .., J$, shows how the Network linearly combines the output of the hidden neurons which are then transmitted to the output neurons.

$$O_{tj} = [1 + exp\,(-net_{tj})]^{-1} \tag{4.33}$$

with $j = 1, 2, .., J$, shows the way outputs are computed by the output neurons with the same non-linear transformation through the logistic function.

3. A set of J errors $(T_{tj} - O_{tj})$ is computed by comparing the target and the output of step 2. for pattern t:

$$E_t\,(W) = \frac{1}{2} \sum_{t=1}^{T} (T_{tj} - O_{tj})^2 \tag{4.34}$$

The error term is used to modify weights connecting all the various layers.

4. The errors are propagated back to the neurons, and weights of the neurons are updated based on their contribution to that error. The rule for weight modification is

$$\Delta_t w_{ji} = -\alpha \frac{\delta E_t(W)}{\delta w_{ji}} \tag{4.35}$$

where α is the learning rate, regulating the learning speed.

5. When equation (4.15) is used for the modification of the weight connecting hidden and output layers, the global error is derived with respect to the weights connecting hidden to

output. The derivative is

$$\frac{\delta E_t(W)}{\delta w_{ji}} = \frac{\delta E_t(W)}{\delta O_{tj}} \cdot \frac{\delta O_{tj}}{\delta w_{ji}} \tag{4.36}$$

where $i = 0, 1, ..., H; j = 1, 2, ..., J$. Index i begins at 0 to take into account the presence of a bias for each output neuron. Consider now the two factors on the right hand side of equation (4.16). From (4.14) the first factor equals

$$\frac{\delta E_t(W)}{\delta O_{tj}} = -(T_{tj} - O_{tj}) \tag{4.37}$$

From (4.12)-(4.13), due to the logistic function, the second factor equals

$$\frac{\delta O_{tj}}{\delta w_{ji}} = \frac{\delta O_{tj}}{\delta net_{tj}} \cdot \frac{\delta net_{tj}}{\delta w_{ji}} = O_{tj}\,(1 - O_{tj})\,H_{ti} \tag{4.38}$$

6. When (4.15) is used for the modification of the weights connecting input and hidden layers, the global error is derived with respect to the weighs connecting input to hidden. The derivative can be expressed as

$$\frac{\delta E_t(W)}{\delta w_{ji}} = \frac{\delta E_t(W)}{\delta net_{tj}} \cdot \frac{\delta net_{tj}}{\delta w_{ji}} \tag{4.39}$$

with $i = 0, 1, ..., K; j = 1, 2, ..., H$. Index i begins at 0 to take into account the presence of a bias for each hidden neuron. From equation (1.2) the first factor is equal to

$$\frac{\delta E_t(W)}{\delta net_{tj}} = H_{tj}(1 - H_{tj}) \tag{4.40}$$

The second factor is equal to

$$\frac{\delta net_{tj}}{\delta w_{ji}} = I_{tj} \sum_{k=1}^{j} (T_{tk} - O_{tk}) O_{tk}\,(1 - O_{tk})\,w_{kj} \tag{4.41}$$

7. The cycle is repeated from step 2. by considering a different pattern, until all the patterns have been examined by the network (index t run from 1 to T).

8. The T squared errors are summed in order to obtain a global error over all the patterns.

9. Steps 2. to 7. are repeated until the global error reaches a specified value.

The steps of recursively updating the synaptic weights of all the neurons in the feedforward Neural Network based on the error between the Neural Network's output and the expected output is demonstrated by fig. 4.13 (Sher, 2012). Starting with step 1., the Neural Network's output is O while the expected output is X:

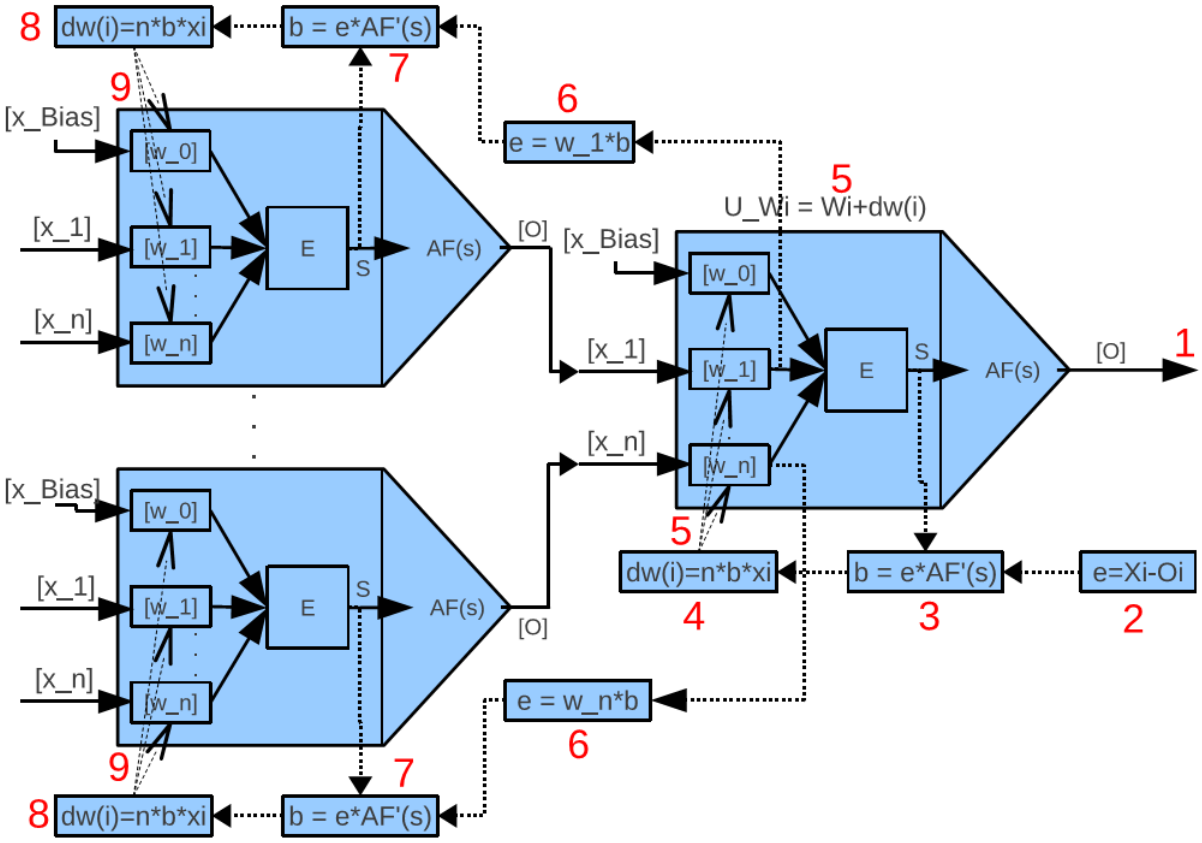

Figure 4.21: Backpropagation learning algorithm

1. The neuron in the output layer produces an output O;
2. e is the error of the neuron's output as compared to the expected output;
3. b (β) is calculated by multiplying the derivative of the activation function by e;
4. the δ (change in) weight for each weight i is calculated;
5. every synaptic weight i of the neuron is updated using the appropriate $dw(i)$ for each W_i.
6. The next e error is recursively calculated for every presynaptic neuron;
7. b (β) is calculated;
8. delta (δ) weight is calculated for each weight i;
9. every synaptic weight i of the neuron is updated using the appropriate $dw(i)$.

   This procedure is continued recursively to the other presynaptic neurons, all the way to, and including, the first layer neurons.

Therefore, whenever a training set is available, Neural Network's weights optimization for some particular task can be performed through the Backpropagation algorithm, running it through the training set multiple times, until the total error between the Neural Network's output and the expected output is small enough to consider the Neural Network's synaptic weights a solution (Sher, 2012). At this point Neural Network can be applied to the real problem for which we have been training it. Notice, again, it is necessary that the answer to the problem is already known. Indeed, once the Neural Network is trained, its weight will remain static, and the neural circuit is used as a program, unchanging for the remainder of its life (Sher, 2012).

Some aspects of the Backpropagation algorithm deserve comment (Beltratti *et al.*, 1996):

Learning rate equation (4.15) is derived from a general gradient descent algorithm. Generally, the coefficient that transforms the information in the first derivatives into a change in the estimated parameters is time-varying, whereas the learning rate α used in the Backpropagation is fixed. In order to ensure the network reaches a solution that coefficient is often set to a small value, having negative consequence of increasing the number of iterations necessary to get a solution. Learning rate may also be modified as learning take place. Indeed, (White, 1989) to ensure the algorithm convergence is necessary to decrease α at a certain rate.

Momentum equation (4.15) is often modified to

$$\Delta_t w_{ji} = -\alpha \frac{\delta E_t(W)}{\delta w_{ji}} + \beta \Delta_{t-1} w_{ji} \tag{4.42}$$

where β is called *Momentum*. This formulation introduces some degree of persistence in the weights modification, since each change depends on the previous one. The inclusion of this term allows keeping α's value larger, in order to speed up convergence, since β is able to provide some stability for the search process. The Momentum inclusion allows avoiding weights excessive oscillations, being justified as an approximation to a more general conjugate gradient (Battiti, 1992).

Backpropagation and maximization algorithms Backpropagation can be considered a very special case of more general algorithms used for non linear problems, such as Newton-Raphson or Berndt. These are more complicated from computational aspect and time-consuming, given that they usually involve many weights. Therefore they are not suitable to Neural Networks-based problems.

Local minima The backpropagation algorithm does not guarantee finding a global minimum, i.e. a Network whose weights produce the lowest classification error among all possible combinations and values of weights. Indeed, it treats learning as an optimization problem,

performing a deterministic local search based on gradient descent to (hopefully) approximate the best solution. But gradient descent has inherent limitations that prevent this from happening. For this reason it may drive the algorithm to a local minimum of the error function. fig. 4.15 well explain this concept. A Neural Network trapped into a local minimum during the learning process is likely to exhibit bad performance in terms of learning and generalization capabilities. Rules to escape from local minima are the following:

- adding random noise to patterns;
- reducing the learning rate;
- taking into account the momentum term;
- adding more hidden units;
- changing the initial random values of the weights.

Therefore, Backpropagation is only a local optimization algorithm. To genuinely find the best neural network, one would have to use a global optimization algorithm, one that has the potential to traverse the entire search space, while remaining time-efficient.

One of the algorithms vaunted for this property is genetic algorithm (GA). I will not enter in merit here.

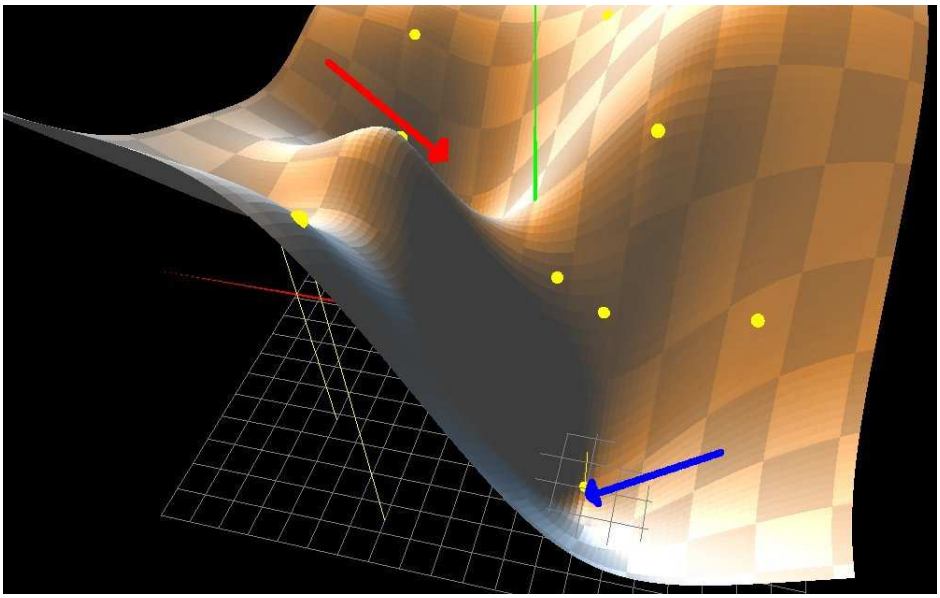

Figure 4.22: Overfitting in machine learning. The yellow dots represent training set data. The green line represents the true functional relationship. The blue line indicate the global minimum, while the red line shows the learned function, which has fallen victim to overfitting.

Initial conditions In order to check for robustness of the final solution, the algorithm may be initialised with different sets of initial weights, the check whether the solutions are very different.

Neuron threshold and bias For each hidden and output neuron the threshold θ is represented by the value of the weight connecting the bias level I_0, set equal to 1, and the neuron itself: this value is added to the net input of the neuron. Random initialisation and subsequent learning extend to the whole set of weights, including weights relating to the bias.

On-line and Off-line learning In on-line learning weight are changed after the presentation of each pattern, while in off-line learning the change takes place after the presentation of all the patterns of the training set. In the first case the signal updating takes place after the presentation of data for each time period, without re-elaborating the information contained in the dataset for the time periods before t. In off-line Backpropagation several iterations are generally necessary before obtaining a good estimate of the set of weights. on line learning possess some randomness that may be useful in escaping local minima, but for the same reason may also miss some good local minimum that was being explored.

Tuning of parameters main parameters of the Backpropagation algorithm are the learning rate and the momentum (or the rate of change), and the distribution of initial random weights. Their values have to be empirically determined as there are no fixed rule for finding optimal values. Since a good set of parameters for the specific problem under consideration requires many trials to be found, this is a very time-consuming process.

Choice between architectures The choice between different architectures does not follow simple rules, but is instead the result of a process of trials and errors. Within feed-forward Network the choice of the number of layers is an important task. While theoretical results have shown that standard feed-forward Networks with only a single hidden layer are universal approximators (Hornik *et al.*, 1989), many applications use the Artificial Neural Networks with more than one hidden layer.

Generally, the larger the number of neurons, the better the ability to learn a specific mapping through the observation of a set of examples. However, in presence of noisy data, learning *in-sample* often takes place at the expense of a lower ability to generalise out of the set of examples that has been seen during the learning process. After a certain point the Neural Network starts using the extra neurons to fit the noise in the data (*overfitting*). This is confusing for *out-of-sample* forecasting. The *weight-elimination* is a technique for starting from a large Neural Network and then lowering its dimensions by assigning a cost to Network complexity. (White, 1990) provides growth rates for network complexity that asymptotically avoid the dangers of both overfitting and underfitting the training set.

In general, the neuron activation function depends on parameters. The architecture of the network and the parameters have to be selected as to maximize the generalization

capability, that is the ability of gaining reliable predictions over new dataset which has never been used to train the network.

The complex nonlinear functional form of the network makes it very difficult to interpret the estimated Network weights ("black box" problem). In linear regression models, the values of the estimated coefficients provide a direct measure of the contribution of each variable to the model's output. In the case of Neural Networks, it is very complicated to analytically identify the impact of an input on the estimated output value. Even in the simplest case of Networks, each input is fed through a nonlinear activation function and it is also affected by different weights w_{ij}. For this reasons Neural Networks are sometimes called "a black box" (Gonzalez *et al.*, 2000): the Network uses the inputs to calculate the output, but the researcher does not clearly understand why a given value is forecasted. This problem could be greatly alleviated by applying the sensitivity analysis (Nicholas Refenes *et al.*, 1994).

Weights interpretation It is sometimes useful to perform statistical inference on a subset of weights, for instance to perform a nonlinearity test or to study causality. Standard inference procedures can be used for these purposes through the mis-specified nonlinear models theory. Notice it is in general impossible to assign specific interpretations to individual weights.

Evaluation criteria Trained Network's performance test is very important test of Network design. It generally based on some statistical indicators such as the coefficient of determination computed over the targets and the outputs of the Network. This evaluation is performed under two ways:

- in-sample measure of performance, based on the data used during the training and expresses the effectiveness of the learning;
- out-of-sample measure of performance, based on the data not used during the training, i.e. not belonging to the training set. This is a measure of the generalization capability of the Network.

Knowledge representation Representation of the knowledge acquired during the training is distributed, in the sense that each concept is a pattern of activity over all the neurons. In this way each neuron is involved in representing many concepts. This feature confers robustness on the network but makes interpretation of self-made rules very difficult.

Unsupervised learning Conversely to the supervised learning, no target vectors set but only an input one is provided here. The Network modifies weights such that similar input vectors are assigned to the same output unit (*cluster*). There is no error or reward signals which can be used to guide the modification process of neural weights based on the difference between the output and the expected output (Sher, 2012). Instead, the Neural Network self modifies its parameters

based on the inputs and its own outputs through some algorithm. The Neural Network will output a vector for each determined cluster. It consists, for example, in giving each neurode the functionality which allows it to change its information processing strategy based on the signals it processes.

Our brains (Sher, 2012) do not have an external supervisor. The biological neurons that compose brain use different types of unsupervised learning, in a sense that they have plasticity and they change based on their experience.

Generalisation error

One of the most relevant problems of the Neural Network is to obtain high performances in response to out-of-sample input values. This problem is called *generalisation*. It consists in the choice of the optimal number of elements in the hidden layers, aiming at:

- capture the training data path;
- get reliable estimates when out-of-sample data are presented, i.e. different from those of the training.

One might think the estimate to be more precise as long as the number of neurons increase. However, data are usually affected by noise. Therefore, a Neural Network representing almost perfectly the training examples would learn the noise too. This would translate in poor out-of sample estimates. Such a problem is named *"overfitting"*.

The problem of overfitting

The Artificial Neural Networks could yield a regression function with a large amount of flexibility. However, if the regression function is equipped with a sufficient number of inputs and parameters, it is possible for the regression function to model the observed response exactly (Mulquiney and Actuaries, 2004). In this case one has modelled not only the underlying features of the data but also the noise inherent in the data, i.e. the model has been *overfitted*. Choosing a regression function which has not been overfitted is a problem the method has to deal with. The solution requires to constraint the fitting such that the model only describes the underlying features of the data.

This problem is similar to the problem of choosing the degree of smoothing in nonparametric estimation (Sarle, 1995). But overfitting can have much more serious consequences.

It is relevant to notice that when applying neural networks no assumption is made (usually) on the statistical distribution of the response. This means statistical tests to protect against overfitting are not available. Moreover, for same reason it is not possible to estimate parameters by Maximum Likelihood estimation. These are instead estimated by specifying a loss function, the sum of squares, that needs to be minimized.

Given these considerations, several methods to prevent against overfitting exists. Two main approaches to controlling the complexity of a Neural Network are

model selection;

regularisation.

Model selection for a Neural Network requires choosing the number of hidden units and connections thereof. Regularisation involves constraining or penalising the solution of the estimation problem to improve generalisation by smoothing the predictions (Sarle, 1995).

Common Neural Network methods for regularisation include the following.

Weight decay Decay parameter prevents overfitting by adding a penalty function to the sum of squares error function which becomes larger as the regression function becomes less smooth

$$RSS + \lambda \left(\sum_m W_m^2 + \sum_m \sum_p w_{mp}^2 \right) \tag{4.43}$$

where W_m and w_{mp} are the weight parameters from the neural network regression function. The weight decay parameter λ controls the magnitude of the penalty, so by choosing it larger the fitted regression function becomes smoother. We choose λ by cross-validation, i.e. we randomly divide the dataset into a training one and a test one. We then fit a number of neural network models to the training data using a number of values of λ. Then for each models we determine the sum of squares in the test dataset. We finally choose the value of λ that minimises the sum of squares in the test dataset. As the value of λ gets smaller, the regression function becomes less smooth, starting to fit the underlying features of the data. Since these should be common to both the training and the test datasets, the sum of squares in both sets will decrease. However, as the value of λ still decreases, the function starts modelling the noise in the training dataset. Because the latter will be different in both the training and test datasets, the sum of squares in the test dataset will start to increase. At this point we have begun overfitting the data.

Cross-validation This method allows to fit both parameters and the number of units in the hidden layer of a trained Neural Network, giving chance of understanding whether the new Network is better than the previous one. Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent dataset. It applies well in evaluating and comparing learning algorithms (Refaeilzadeh *et al.*, 2009) cross by dividing data into two segments:

- one used to learn or train a model;
- the other used to validate the model.

In typical cross-validation, the training and validation sets must cross-over in successive rounds such that each data point has a chance of being validated against. The basic form of cross-validation is k-fold cross-validation. Other forms of cross-validation are special cases of k-fold cross-validation or involve repeated rounds of k-fold cross-validation.

In k-fold cross-validation data are first partitioned into k equally (or nearly equally) sized segments or folds. Subsequently k iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for validation while the remaining $k-1$ folds are used for learning. In fig. 4.15 an example with k=3 is shown. The darker sections

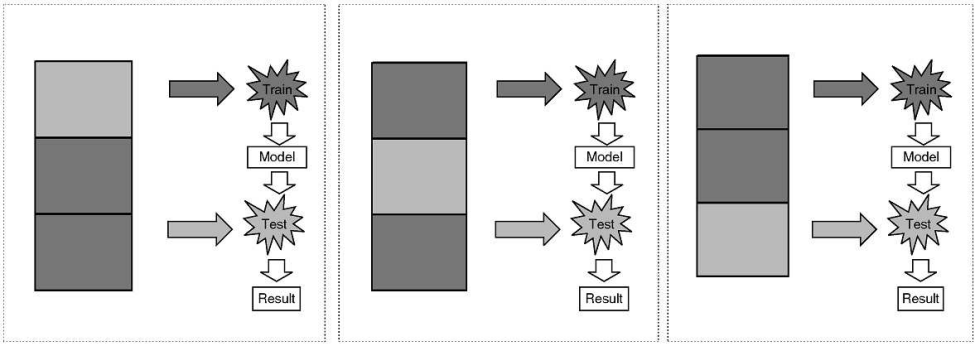

Figure 4.23: Three-fold cross-validation procedure

of the data are used for training while the lighter sections are used for validation. Most common validation in data mining and machine learning is 10-fold-cross-validation (k=10).

Cross-validation is used to evaluate or compare learning algorithms. In each iteration, one or more learning algorithms use $k-1$ folds of data to learn one or more models, and subsequently the learned models are asked to make predictions about the data in the validation fold. The performance of each learning algorithm on each fold can be tracked using some predetermined performance metric like accuracy. Upon completion, k samples of the performance metric will be available for each algorithm. Different methodologies such as averaging can be used to obtain an aggregate measure from these sample, or these sample can be used in a statistical hypothesis test to show that one algorithm is superior to another.

Since this procedure is a kind of trial and error, it could require long time before to yield a satisfactory model. From here the necessity to hold guidelines for the model selection.

Early stopping based on cross validation is the most common solution to the overfitting problem in Neural Network training (Sarle, 1995). It proceeds as follows:

1. Split the available data into two separate training and validation sets, e.g. in a 2-to-1 proportion.

2. Train only the training set. Use a large number of hidden elements, a small random initial values and a slow learning rate.

3. Evaluate the per-example error on the validation set (validation error) once in a while, periodically during the training, e.g. after every fifth epoch.

4. Stop training as soon as the error on the validation set starts growing.

5. Use the weights the network had in that previous step as the result of the training run.

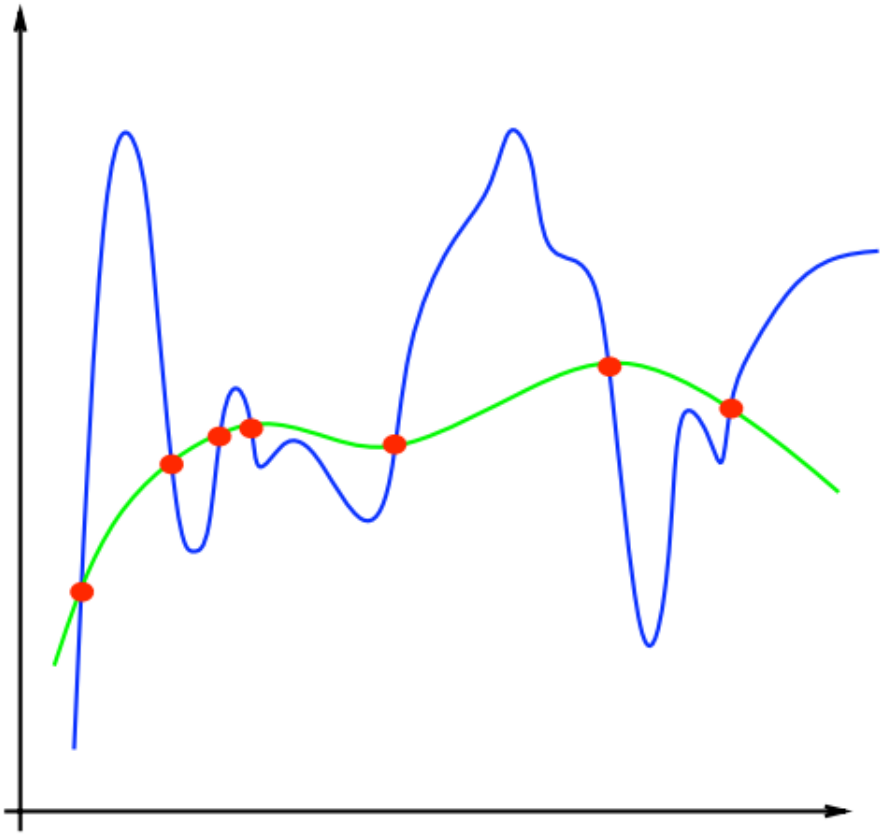

Figure 4.24: Overfitting in machine learning. The red dots represent training set data. The green line represents the true functional relationship, while the blue line shows the learned function, which has fallen victim to overfitting

Neural networkers usually divide their sample into two separate data sets:

the *training set* is used by the algorithm to estimate the network weights;

the *test set* is used to evaluate the forecasting accuracy of the network.

Since the test set is not used during the estimation of the network weights, the forecasts made from the test set amount to an ex post *out-of-sample* forecast (Gonzalez *et al.*, 2000). The neural networker aims at minimizing the forecasting error in the training set using a criterion such as the mean squared error (MSE).

Part III

The Models

Chapter 5

NetLogo AB-Model

> "No matter how sophisticated their behaviour rules, agents in a complex system (such as humans) can never be perfectly rational." (Mills, 2010)

NetLogo is an Agent-Based modelling language used for the implementation of Agent-Based models. I explain here the source code of the model. In order to make the comprehension more easy I start describing the interface and the list of instructions the user should follow to implement the behavioural simulation. Then I explain what action correspond to the related setting. Finally I presents some behavioural simulations and the related results.

5.1 The source code

5.1.1 Model's settings - the *SETUP*

Fig. 5.1 shows the interface as it presents to the model's user. The modeller should implement the following initialize procedure steps before to start the simulation:

1. set the desired number of Society of Mind cognitive agents. This can be done through the slider *#Society-of-Mind-cognitive-agents*;

2. set the percentage of SoM agents that will be generated with personality traits and emotional state required by the user (*%sample-SoMs*). The remaining agents will have randomly generated personality traits and emotional state. Moreover, the user can easily choose all the agents to have identical personality by switching on the *Identical-personality* switcher;

3. whether the user desires not set the agents' personality, while leaving it randomly generated by the program, he should skip this step. On the contrary, having correctly operated the previous step, he can set the related personality (horizontal) and emotional (vertical) sliders (explanation will be provided later);

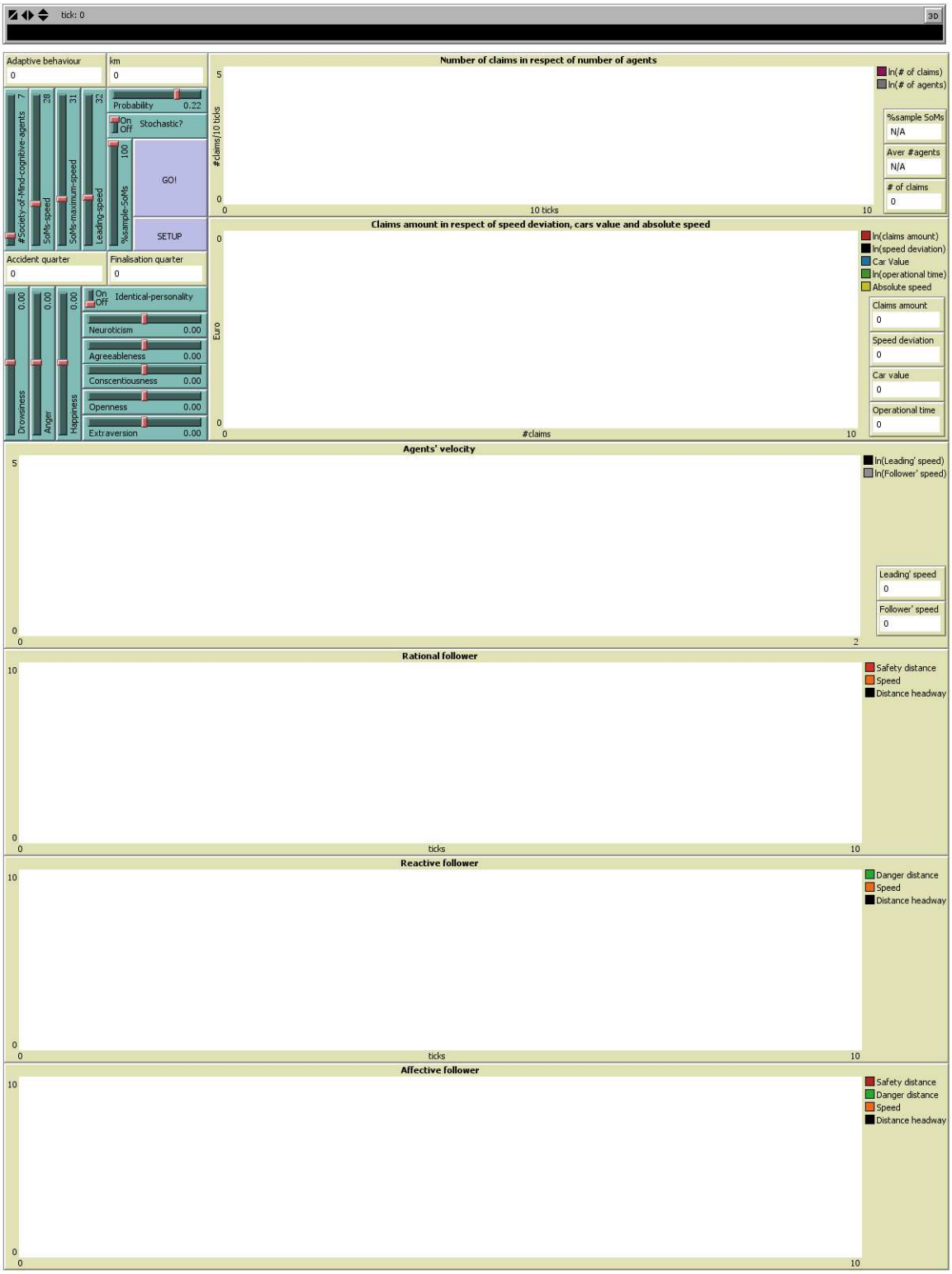

Figure 5.1: The model interface

4. set the SoM agents' default speed (*SoMs-speed*), the SoM agents' maximum speed (*SoMs-maximum-speed*) and the Leading vehicle's speed (*Leading-speed*);
5. switch on / off the stochastic component affecting the claims finalisation procedure;
6. whether the user requires the program to include the stochastic component, he could slide the probability of this component to affect the simulation (*Probability*);
7. press the *SETUP* button.

Once the parameters have been settled, the user can run the simulation by pressing the *GO!* button. All the control settings still can be modified during the simulation. However, in order to understand the results of a particular setting, I suggest not change them through a simulation. This way it will be possible to compare the effect of the different settings on the agent's behaviour, which is definitely the aim of the simulation research.

The *breeds* I defined three breeds in the model:

- the *drivers* agents;
- the *SoMs* agents;
- the *sample-SoMs* agents.

```
breed [drivers driver]
breed [soms som]
breed [sample-soms sample-som]
```

The keyword *breed* [< *breeds* >< *breed* >] can only be used at the beginning of the Code tab, before any procedure definitions. The first input defines the name of the agentset associated with the breed. The second input defines the name of a single member of the breed.

Breed built-in variables Any turtle of the given breed has its breed built-in variable set to that agentset

```
drivers-own [v_n car-value]
soms-own [Po Pc Pex Pa Pn Eh Ed Ea x aggression unlawfulness
  d1r d4r front-turtles next-turtle back-turtles coming-turtle
  leading-altro-lato follower-altro-lato v_n max-v_n a_n
  braking_n aff_strength rat_strength react_strength d_m
  danger-distance ds car-value]
sample-soms-own [Po Pc Pex Pa Pn Eh Ed Ea x aggression unlawfulness
  d1r d4r front-turtles next-turtle back-turtles coming-turtle
  leading-altro-lato follower-altro-lato v_n max-v_n a_n
  braking_n aff_strength rat_strength react_strength d_m
  danger-distance ds car-value]
```

It can be seen that both SoMs and sample-SoMs agents own the same built-in variable. Indeed they are equal at all, with the only exception of the personality settings. SoMs agents have random settled personality, while sample-SoMs agents have their personality settled by the user as in step 3. above.

v_n is the velocity of the current agent n, the *follower vehicle*. The agent that has moved before is the *leading vehicle*, identified with subscript $n-1$, as it precedes the follower n. Consequently, its velocity is v_{n-1}.

Each agent knows the velocity of the front vehicle, therefore each owns v_{n-1}.

There is always one and only one driver alive in every simulation. He is always the leading vehicle. Therefore, he do not own the variable v_{n-1}, since nobody has moved before.

car-value is one more very important variable that every agentset owns. It can assume integer value in (1,3) and it represents the agent's car value.

Variables described until now are held by both the drivers and the SoMs breeds. The following variables rather belong to the Society of Mind agents only:

Po constitutes the SoM's personality factor *openness*;

Pc constitutes the SoM's personality factor *conscientiousness*;

Pex constitutes the SoM's personality factor *extraversion*;

Pa constitutes the SoM's personality factor *agreeableness*;

Pn constitutes the SoM's personality factor *neuroticism*;

Pn constitutes the SoM's personality factor *neuroticism*;

Eh corresponds to the SoM's emotional state *happiness*;

Ed corresponds to the SoM's emotional state *drowsiness*;

Ea corresponds to the SoM's emotional state *anger*.

The above features are fundamental and they form the model core, characterizing the SoM agents' personality traits and emotional states, as in the underlying theory explained in Part II. The model's user can set them as in step 2. and 3. above.

The *SETUP* button When the user passes through the step 7. of the setting procedure, the *setup* procedure is called:

```
to setup
  ca
  setup-globals
  reset-ticks
end
```

The *setup-globals* instruction simply set some global variables

```
to setup-globals
  set size_n-1 0
  set braking_n-1  -3.5
  set tau 0.75
  set theta tau / 2
end
```

which constitute some parameters used in the agents' speed calculation we will see later.

The global variables declared at the beginning of the source code are

```
globals [size_n-1 theta tau eq_1 eq_2 braking_n-1 v_n-1 abs-speed
  x_n-1 #meters #claims sample-som #created-agents km claim-amount
  accident-quarter finalisation-quarter record-v_n-1 delta_v
  car-value_n-1 strategy total-car-value #claims-per-ticks #ticks
  operational-time #agents speed tik v_n-1-forMonitor]
```

The other setting instructions are called by the go procedure, in order to make them still modifiable during the simulation, as already seen.

5.1.2 Run the simulation - the *GO!*

The simulation starts by pressing the *GO!* button in the interface, which calls the *go* procedure:

```
to go
  tick
  set tik tik + 1
  set #ticks #ticks + 1
  if ticks = 1
  [
    setup-driver
    ask drivers
    [
      fd v_n
      set km km + v_n / 1000
      leave-track
      record-car
    ]
  ]
  if ticks > 1
  [
    ifelse ( count sample-soms / count turtles ) * 100 < %sample-SoMs
```

```
  [
    setup-sample-som
  ]
  [
    setup-soms
  ]
  foreach reverse sort-on [xcor] turtles
  [
    ask ?
    [
      ifelse breed = drivers
      [
        set v_n leading-speed
        fd v_n
        set km km + v_n / 1000
        leave-track
      ]
      [
        rational-update
        real-distance
        input-scaling
        rational-strenght-update
        emotion-update
        affective-strength-update
        reactive-update
        reactive-strength-update
        bidding-process
        plot-speed
        leave-track
        record-car
        move
      ]
    ]
  ]
]
if tik / 10 = 1825
[
  user-message (word "We have considered "
    (ticks / 10 / 365)
```

```
      " years. The training dataset has been generated.
      Now please press the go! button to generate the verification dataset.
      An error will be generated. Ignore it by pressing ok and Go! again.")
    reset-ticks
    stop
  ]
  if tik / 10 = 3650
  [
    user-message (word "We have considered "
      (ticks / 10 / 365)
      " years. The verification dataset has been generated.")
    stop
  ]
end
```

The *tick* command advances the tick counter by one and updates all plots. It represents the simulation time-step. If tick equals 1, i.e. the *go* procedure is running for the first time after a setup, the program call the setup-driver procedure

```
to setup-driver
  create-drivers 1
  [
    set heading 90
    set size 2
    set v_n leading-speed
    set car-value 1 + floor random 3
  ]
end
```

where one driver is created. He has velocity v_n equal to the *Leading-speed* slider in the interface settled by the user through the initializing procedure step 4. Moreover, the driver agent has a *car-value* randomly assigned by the program in the interval (1,3). The *size* parameter in this case is only for graphical representation.

After the driver has been generated, the *go* procedure ask him to move forward of V_n patches. A patch is a point in the simulated world having x and y coordinates. In my model one patch correspond to one meter.

Once the driver has moved, he is asked to leave a track of its movement through the *leave-track* procedure. This will be recalled every time by all the agents after they move. Since the agent's speed variable v_n is agent-specific, it can not be seen by the other agents. However, each agent need to know the front vehicle speed in order to compute a speed consistent with both the environment and his internal states. For this reason, after an agent has computed his speed, after

he has moved, his speed is reported in a "global box" v_{n-1}, so that the following agent can take into account during his speed calculation procedure. Moreover, also the leading agent's position on the x axis x_{n-1} and the leading's car value *car-value* live the same matter. The solution is the same as for the velocity, through the *leave-track* procedure for the x_{n-1} case

```
to leave-track
  set record-v_n-1 v_n-1
  set v_n-1-forMonitor v_n-1
  set v_n-1 v_n
  set x_n-1 xcor
end
```

while the *record-car* procedure is used for the *car-value*

```
to record-car
  set car-value_n-1 car-value
end
```

The first simulation time-step has gone. From now on, i.e. *if ticks > 1*, each simulation time-step starts verifying whether the percentage of SoM agents having user-settled personality (sample-soms) is lower than the one reported by the *%sample-SoMs* slider in the interface, which can be settled at any time by the user. If this is the case, the *setup-sample-som* procedure is recalled, increasing by one unit per time-step the number of *sample-soms*

```
to setup-sample-som
  if #Society-of-Mind-cognitive-agents > count turtles - 1
  [
    set #created-agents #created-agents + 1
    create-sample-soms 1
    [
      set heading 90
      set max-v_n soms-maximum-speed
      set v_n soms-speed
      setup-my-personality-my-emotions
      setup-accel-brak
      set car-value 1 + floor random 3
    ]
  ]
end
```

They have initial speed, personality and emotion settings defined in the interface by the user (or by default if no settings has been made) before the simulation has started

```
to setup-my-personality-my-emotions
  set Po Openness
  set Pc Conscientiousness
  set Pex Extraversion
  set Pa Agreeableness
  set Pn Neuroticism
  set Eh happiness
  set Ed drowsiness
  set Ea anger
end
```

Conversely, if the percentage of sample-soms alive in the world equals that settled in the interface, the *setup-soms* procedure is recalled, through which one *som* agent per time-step is created

```
to setup-soms
  if #Society-of-Mind-cognitive-agents > count turtles - 1
  [
    set #created-agents #created-agents + 1
    create-soms 1
    [
      set heading 90
      set max-v_n  soms-maximum-speed
      set v_n soms-speed
      setup-personality
      setup-emotions
      setup-accel-brak
      set car-value 1 + floor random 3
    ]
  ]
end
```

He has initial speed as user-settled in the interface, but randomly generated personality and emotion settings

```
to setup-personality
  ifelse Identical-personality = true
  [
    setup-my-personality-my-emotions
  ]
  [
    set Po random-float -1 + random-float 1    ;Openness
    set Pc random-float -1 + random-float 1    ;Conscientiousness
```

```
    set Pex random-float -1 + random-float 1   ;Extraversion
    set Pa random-float -1 + random-float 1    ;Agreeableness
    set Pn random-float -1 + random-float 1    ;Neuroticism
  ]
end

to setup-emotions
  set Eh random-float -1 + random-float 1   ;Happiness
  set Ed random-float -1 + random-float 1   ;Drowsiness
  set Ea random-float -1 + random-float 1   ;Anger
end
```

Maximum acceleration and braking capability parameters are equal for all the vehicles

```
to setup-accel-brak
  set a_n 10
  set braking_n -3.5
  set d_m 9.81
end
```

Summarizing, the driver has been created and has moved. A SoM or a sample-SoM has also been created. Now, starting from the one standing on the highest x coordinate to the one on the lowest, each agent is asked to implement a list of instructions. The reader can refer to Chapter 4 for the underlying theory. If the agent belongs to the drivers breed, the instructions he must follow are the same as before. Conversely, if the agent is a SoM, he is asked to

```
  rational-update
  real-distance
  input-scaling
  rational-strenght-update
  emotion-update
  affective-strength-update
  reactive-update
  reactive-strength-update
  bidding-process
  plot-speed
  leave-track
  move
  record-car
```

Rational agency The rational, affective and reactive agencies are explained in Part II. Following the SDA (Sense Decide Act) theory, the rational agency behaves through the *rational*

update function

```
to rational-update
  set ds ( 0.00028 * v_n ^ 2 ) + ( - 0.0084 * v_n-1 ^ 2 ) + ( 0.784 * v_n-1 ) +
  + 4.1
end
```

setting a value of safety distance *ds* as in equation (4.9) described in Chapter 4, where the parameters of the equation are given from theory.

After that, the agent implements the *real-distance* procedure, in order to measure the distance from the forward and the backward vehicles, d_{1r} and d_{4r} respectively

```
to real-distance
  set front-turtles [xcor] of turtles with [xcor > [xcor] of myself]
  set back-turtles [xcor] of turtles with [xcor < [xcor] of myself]
  ifelse front-turtles = []
  [
    die
  ]
  [
    set next-turtle min front-turtles
    set d1r next-turtle - xcor
  ]
  ifelse back-turtles = []
  [
    set d4r 10
  ]
  [
    set coming-turtle max back-turtles
    set d4r xcor - coming-turtle
  ]
end
```

Then the distances so computed are scaled in (0,1), so that they can be jointly used with the personality traits, and an output $x \in (-1, 1)$ is computed

```
to input-scaling
  ifelse d1r < ds
  [
    set x (ds - d1r) / ds
  ]
  [
```

```
    ifelse d4r = 0
    [
      set x d4r
    ]
    [
      set x (ds - d1r) / d4r
    ]
  ]
  if x > 1
  [
    set x 1
  ]
  if x < -1
  [
    set x -1
  ]
end
```

Finally, as the last step of the rational agency, the *rational strenght update function* is computed, which will participate in the bidding process

```
to rational-strenght-update
  set rat_strength x + (Pc + Pa - Pn) / 3
end
```

Affective agency The SoM jumps now to the affective agency implementation procedure. He starts by updating his emotional state making use of the x value computed above

```
to emotion-update
  set Eh Eh + 0.25 * x
  if Eh > 1 [set Eh 1]
  if Eh < -1 [set Eh -1]
  set Ed Ed + 0.25 * x * 0.5
  if Ed > 1 [set Ed 1]
  if Ed < -1 [set Ed -1]
  set Ea Ea + Pn + 0.25 * x
  if Ea > 1 [set Ea 1]
  if Ea < -1 [set Ea -1]
end
```

Then he updates the affective strength function taking in consideration his emotional state

```
to affective-strength-update
  set aggression ( Ea - Pa - Pc ) * ( 1 / 3 )
  set unlawfulness Ea
  set aff_strength ( aggression + unlawfulness ) * 0.5
end
```

Reactive agency The SoM update the reactive agency and compute reactive strength update function, as in the theory, as for the other two agencies

```
to reactive-update
  set danger-distance ((v_n - v_n-1) ^ 2) / d_m
end


to reactive-strength-update
  ifelse d1r < danger-distance [set react_strength 1][set react_strength 0]
end
```

The reactive agency case differs from the other two agencies in that the *reactive strength* is a dummy variable, as explained in Chapter 4.

The bidding process The three agencies have their true value of strength to bid with. They now participate in a competition with only one winner (non-compromise)

```
to bidding-process
  ifelse react_strength = 1
  [
    reduce-speed
    set strategy "reduce speed"
    plot-reactive
  ]
  [
    ifelse aff_strength > rat_strength
    [
      speeding
      set strategy "speeding"
      plot-affective
    ]
    [
      gipps-speed
      set strategy "Gipps speed"
      plot-rational
```

```
    ]
  ]
end
```

If the winner is the reactive agency, the strategy implemented by the SoM will be the speed reduction. If, conversely, the affective agency will win, SoM will choose to speeding his vehicle. Yet, if the winner will be the rational agency, SoM will adopt a safe behaviour, following the Collision Avoidance CA model.

Finally, the features characterizing the decision process of the winner agency, mainly velocities and distances, are represented in a plot, in order to compare them and to draw conclusions.

Reduce speed The agent adopting the speed reduction strategy decrease his velocity v_n by the vehicle maximum deceleration capability d_m.

```
to reduce-speed
  set v_n v_n - d_m
end
```

Speeding If the bidding process has been awarded to the affective agency, the SoM increases his speed as to the maximum allowed by the user in the interface

```
to speeding
  set v_n max-v_n
end
```

Gipps speed When the SoM has a positive personality, we expect him to set his speed following a rational behaviour. Indeed, he set his speed as in Gipps' model defined in Chapter 4

```
to gipps-speed
  set eq_1 v_n + 2.5 * a_n * tau * (1 - v_n / max-v_n) * (0.025 + v_n /
  / max-v_n) ^ 0.5
  set eq_2 braking_n * (tau / 2 + theta) + ( (braking_n ^ 2) * (tau / 2 +
  + theta) ^ 2 - braking_n * (2 * (x_n-1 - size_n-1 - xcor) - v_n * tau -
  - (v_n-1 ^ 2) / braking_n-1)) ^ 0.5
  ifelse eq_1 < eq_2
  [set v_n eq_1]
  [set v_n eq_2]
end
```

Now that speed has been computed by each agent, the program plots both the current vehicle's speed v_n and the previous vehicle's speed v_{n-1}.

Then the current agent is asked to leave track of his movements and of his car value, through the *leave-track* and the *record-car* procedures seen before.

Move The last procedure of the go list of instruction makes the SoM agent moving based on the deciding process developed above

```
to move
  set #meters 0
  while [#meters != floor v_n]
  [
    fd 1
    set #meters #meters + 1
    if any? other turtles-here
    [
      set-delta-speed
      set-abs-speed
      set-claim
      claimsFeatures-procedure
      amountQuarter-procedure
      die
    ]
    if xcor = max-pxcor
    [
      die
      stop
      ]
    ]
end
```

The agent decided to move of v_n meters. However, if the behaviour was not a good one, it may happen the SoM collides the front vehicle. In such a case he end the simulation and the program records the δ of the two vehicles' speed

```
to set-delta-speed
  set delta_v (- 1 * (record-v_n-1 - v_n))
end
```

Then the program also evaluates the absolute speed

```
to set-abs-speed
  set abs-speed v_n / 50
end
```

which will generate a larger claim amount as long as the follower's speed increases over 50 km/h, and vice versa as it decrease below 50 km/h.

The claim If a claim has happened, the *move* procedure calls the *set-claim* procedure in order to set the claim amount

```
to set-claim
  set total-car-value car-value_n-1 + car-value
  set claim-amount (delta_v * total-car-value * abs-speed * 100)
  if stochastic? = true
  [
    let Pstochastic random-float 1
    if Pstochastic <= probability
    [
      set claim-amount claim-amount + claim-amount * 0.10
    ]
  ]
  ifelse claim-amount < 0
  [
    set claim-amount 0
  ]
  [
    set #claims #claims + 1
    plot-claims-#
  ]
  plot-behaviour
end
```

The claim amount can take two different values, depending on whether the stochastic component is set by the user to affecting the simulation or not:

- In the case the claim amount is purely deterministic it is set to

$$A = \Delta_V \cdot C \cdot V \cdot 100 \tag{5.1}$$

  with A the claim amount, Δ_V the difference between the leading and the follower speed, C the total car value, obtained as the sum of the follower's C_n and the leading's C_{n-1} car values, V the absolute speed described above.

- When the stochastic component is set to take part into the claim amount evaluation, a random number is drawn in (0,1). If this is smaller or equal to the probability value assigned through the slider in the interface, which can take value in (0.01,0.3), then the stochastic component will affect the claim amount A_{stoch}, which will equal

$$A_{stoch} = A \cdot (1 + 0,10) \tag{5.2}$$

The *plot-behaviour* procedure graphically represents the number of claims in respect of the number of agents.

The data set procedures At this point everything regarding the agents has been performed. The *claimFeatures* and *amountQuarter* procedures are now implemented in order to build the data set object of the Artificial Neural Network model.

The *claimFeatures* procedure allows to record on a text file some claims features:

- the follower's speed v_n;
- the leading's speed v_{n-1};
- the difference between the two Δ_V;
- the follower's car value C_n;
- the leading's car value C_{n-1};
- the sum of these two C;
- the absolute speed V.

```
to claimsFeatures-procedure
  if claim-amount > 0
  [
    ifelse tik / 10 < 1825
    [
      file-open "claimsFeatures-training.txt"
      file-write v_n
      file-write record-v_n-1
      file-write delta_v
      file-write car-value_n-1
      file-write car-value
      file-write total-car-value
      file-write abs-speed
      file-close
    ]
    [
      file-open "claimsFeatures-verification.txt"
      file-write v_n
      file-write record-v_n-1
      file-write delta_v
      file-write car-value_n-1
```

```
      file-write car-value
      file-write total-car-value
      file-write abs-speed
      file-close
    ]
  ]
end

to amountQuarter-procedure
  if claim-amount > 0
  [
    ifelse tik / 10 < 1825
    [
      file-open "amountQuarter-training.txt"
      file-write claim-amount
      set accident-quarter floor ((ticks / 10) / 90) + 1
      set finalisation-quarter floor (claim-amount / 1000)
      set operational-time finalisation-quarter + accident-quarter
      if finalisation-quarter < 1
      [
        set finalisation-quarter 1
      ]
      if stochastic? = true
      [
        let Pstochastic random-float 1
        if Pstochastic <= probability
        [
          set finalisation-quarter finalisation-quarter + 1
        ]
      ]
      file-write finalisation-quarter
      file-close
    ]
    [
      file-open "amountQuarter-verification.txt"
      file-write claim-amount
      set accident-quarter floor ((ticks / 10) / 90) + 1
      set finalisation-quarter floor (claim-amount / 1000)
      set operational-time finalisation-quarter + accident-quarter
      if finalisation-quarter < 1
```

```
      [
        set finalisation-quarter 1
      ]
      if stochastic? = true
      [
        let Pstochastic random-float 1
        if Pstochastic <= probability
        [
          set finalisation-quarter finalisation-quarter + 1
        ]
      ]
      file-write finalisation-quarter
      file-close
    ]
  ]
end
```

The *amountQuarter* procedure is necessary to report on a text file the *claim amount* seen above and the claim *finalisation quarter*. The stochastic component can affect also the claim finalisation quarter, as it was for the claim amount. Indeed, the claim finalisation quarter is calculated by the program simply as the largest integer less than or equal to F, where

$$F = \frac{A}{1000} \tag{5.3}$$

However, when the user has required the stochastic component to take part in the calculation, the drawing procedure involving the probability is the same we have seen for the claim amount before. Therefore, if the drawn falls in the case of stochastic affection case, the finalisation quarter is postponed by one quarter, becoming

$$F_{stoch} = F + 1 \tag{5.4}$$

The simulation We have seen the list of instruction a SoM agent must follow in a simulation time-step. After he concluded his task, the same list must be implemented by all the other followers alive. At this point a simulation time-step has gone. Hence a new one can start, having as actors all the SoM still alive, which have not collided other agents, and some new one, in order to respect the agents number settled by the user in the interface.

		BALANCED PERSONALITY A		NEGATIVE PERSONALITY B		POSITIVE PERSONALITY B	
		max-SoM-V	Driver_V	max-SoM-V	Driver_V	max-SoM-V	Driver_V
homogeneous personalities	Sim 1	30	40	30	40	30	40
	Sim 2	50	40	50	40	50	40

Figure 5.2: Homogeneous personalities - Simulations summary table

5.2 Cognitive Behaviour Simulations

I simulated the Society of Mind agents cognitive behaviour on a 9000 meters single-lane road segment, in order to allow the mental features of the SoM agents to evolve through the simulation. The leading car travelled with a prescribed speed, while the following vehicles were driven by SoM agents.

5.2.1 Homogeneous personalities simulations

I report in fig. 5.2 a table which summarizes the main homogeneous personalities simulations I ran, where the reader can find the major simulations settings.

Simulation 1 - A

Affective agency The affective agency won the bidding competition the majority of times, hence the speed is kept to the vehicle's maximum, as shown in the figure. The plot presents a positive increasing trend in the distance headway. Moreover, it can be clearly seen that due to the fact that all the agents had the same balanced personality, they all kept the same distance from the vehicle ahead. Indeed, the black line displays a flat pattern with fractures. These fractures represent the distance of the second vehicle, a SoM, from the first vehicle, the driver. The latter had an higher speed, so it constantly increased its distance from the follower at each simulation time step.

Reactive agency There were no danger, thus no reason for the reactive agency to enter the competition.

Rational agency There were no reason to travel at a reduced speed. Therefore neither the rational speed was computed.

The claims In such a scenario no claims happened.

The speeds The plot of the agents' velocity in the figure shows a stable trend during the simulation. SoMs travelled at a speed of 30 km/h, the maximum their vehicles allowed them.

The driver travelled at 40 km/h. Therefore each SoM could follow the vehicle ahead at the maximum desired speed without any risk of collision.

The number of SoMs I tried to increase the number of agents. The results did not changed.

Simulation 2 - A

Affective agency Since SoM agents had a higher maximum speed than the leading vehicle, it can be see from the figure that when the rational agency won the competition, the distance headway (black line) has a flat trend, while whenever the affective agency won, i.e. the SoM speeded its vehicle, it falls towards the danger distance. Moreover, whenever the speed (orange line) equals the distance headway, a claim happened. 7 claims happened. Indeed, the two lines touch 7 times in the figure.

Reactive agency Reactive never won.

Rational agency The figure presents the rational follower behaviour. It shows that The SoM always kept a speed of 40 km/h, which is lower than it desired, but safe enough to stay below the distance headway, never causing accidents.

The claims The claims verified, as in the figure. Each SoM collided with the driver, because of their speed difference, when the affective agency won the bidding process.

Simulation 1 - B Results are the same identical as in Simulation 1 - A, as expected, due to the fact that the SoMs' maximum speed was lower than the leading speed.

Simulation 2 - B Results are the same identical as in Simulation 2 - A. However, the concept is slightly different. Indeed, in Simulation 2 - A the claims happened because some agents had negative personality. Conversely, the claims of this simulation were expected, since all the agents had negative personality.

Simulation 1 - C Only the rational agent was involved here, as expected. Claims did not verify.

Simulation 2 - C Again, whenever all the SoM are rational, the rational agency always wins, without any accident.

5.2.2 Heterogeneous personalities simulations

Now I present some simulations where SoM had heterogeneous personality traits.

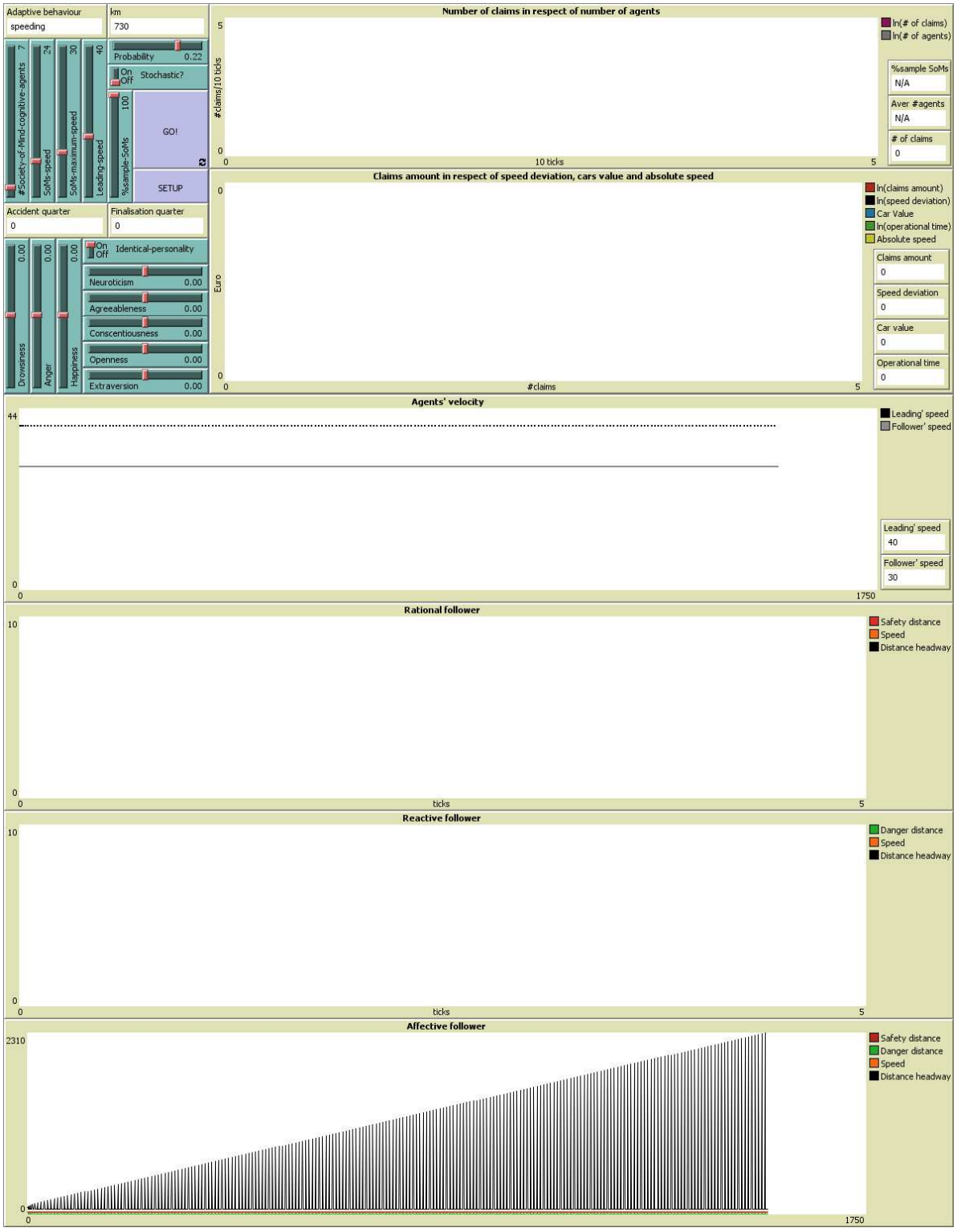

Figure 5.3: Simulation 1 - A

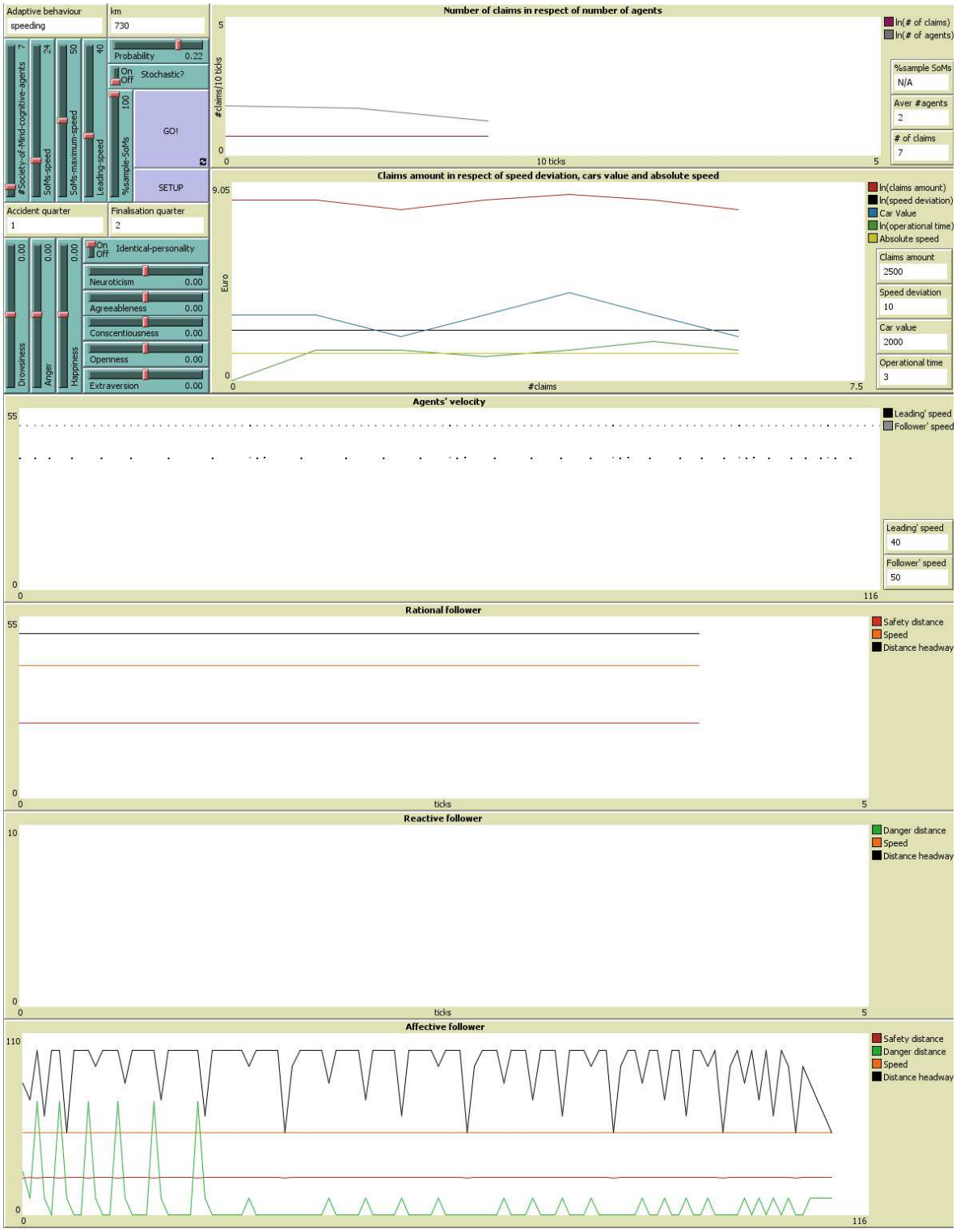

Figure 5.4: Simulation 2 - A

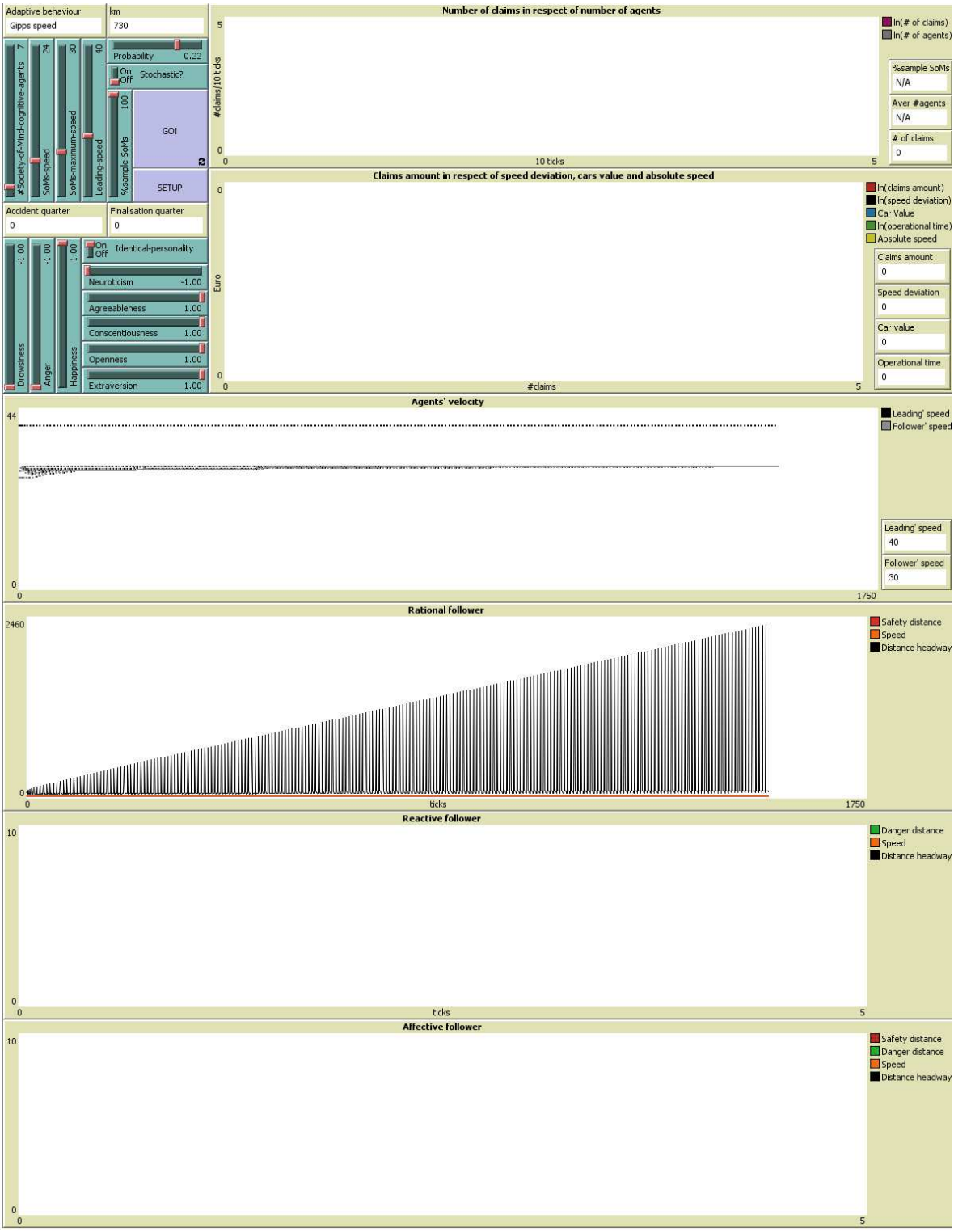

Figure 5.5: Simulation 1 - C

Simulation 3

Affective agency Affective agency shows an identical pattern as in simulation 1 - A.

Reactive agency Equal to simulation 1 - A.

Rational agency The rational agency, as shown in figure, reduced the SoM's speed when the distance headway fell.

The claims Although the rational agency reduced the speed, one claim happened.

Simulation 3 bis I want to show a different case. Since agents had randomly assigned personality, each simulation provided different results. Indeed, I ran another simulation maintaining the same setting, gaining different results.

Affective agency In this case the affective agent speeded its car colliding with another SoM, which were probably decreasing in order to avoid collision with the vehicle ahead. Figure clearly show the approaching and the collision.

Rational agency Figure shows a pattern which remember the one of the affective agency seen above. Indeed, the reason of such a trend is the same.

Simulation 4 The 46% of the SoMs here were characterised by quite balance emotions and quite negative personality traits.

The affective agency The affective agency won the competition just few times. Only at the simulation start the rational agent caused an accident.

The reactive agency It can be seen from the figure that any decrease in the danger and headway distances is followed by a speed reduction. Indeed, the orange line segment before the two flat segments represents the vehicle's speed before the agent brakes. The orange line segment in the trait where the two distances remain constant (flat) represents the speed decrease. The speed was too high with respect to the vehicle maximum brake capability to allow collision avoidance. Indeed, 18 accidents happened.

Rational agency The rational agency kept a rational constant speed around 40 km/h.

Simulation 5

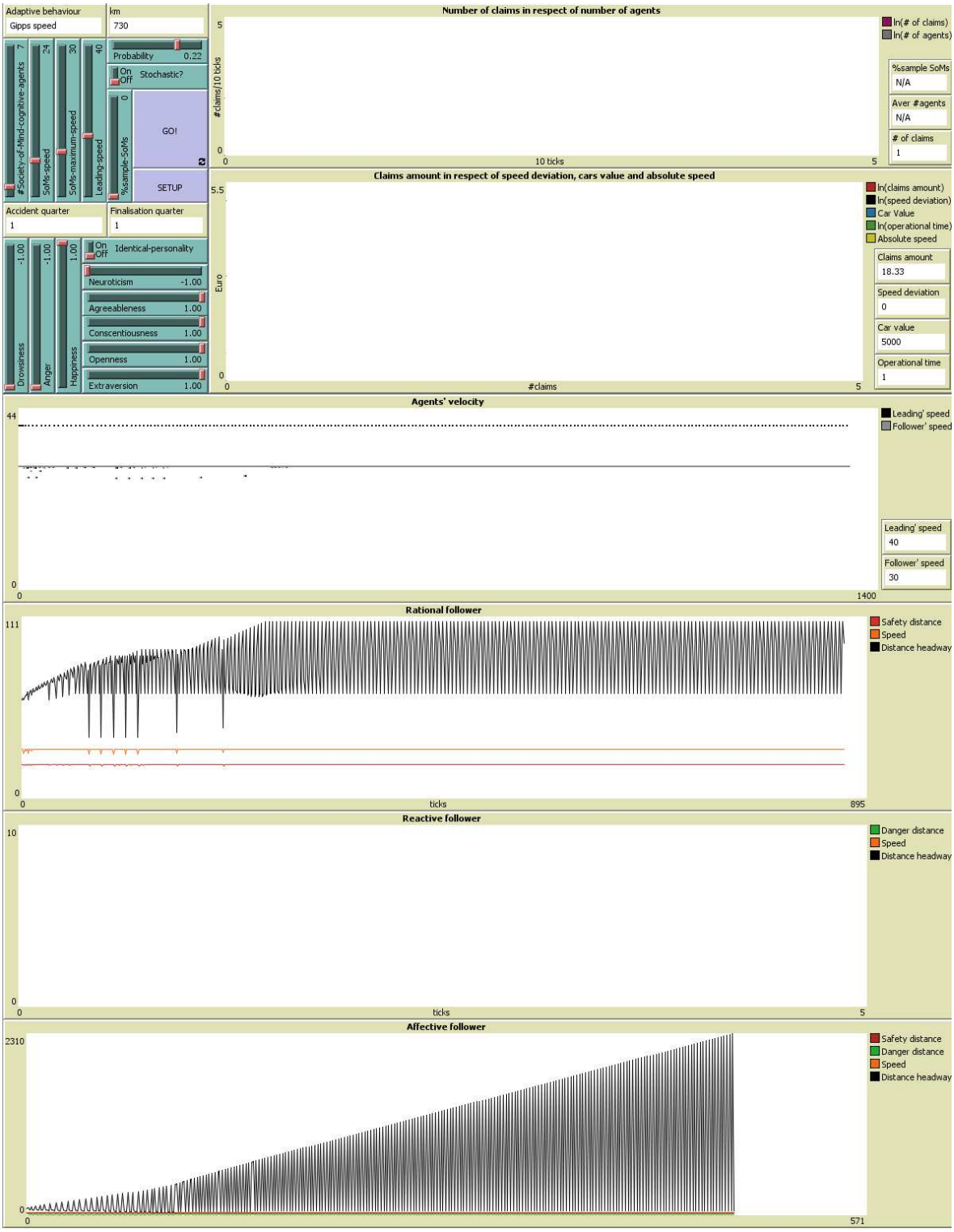

Figure 5.6: Simulation 3

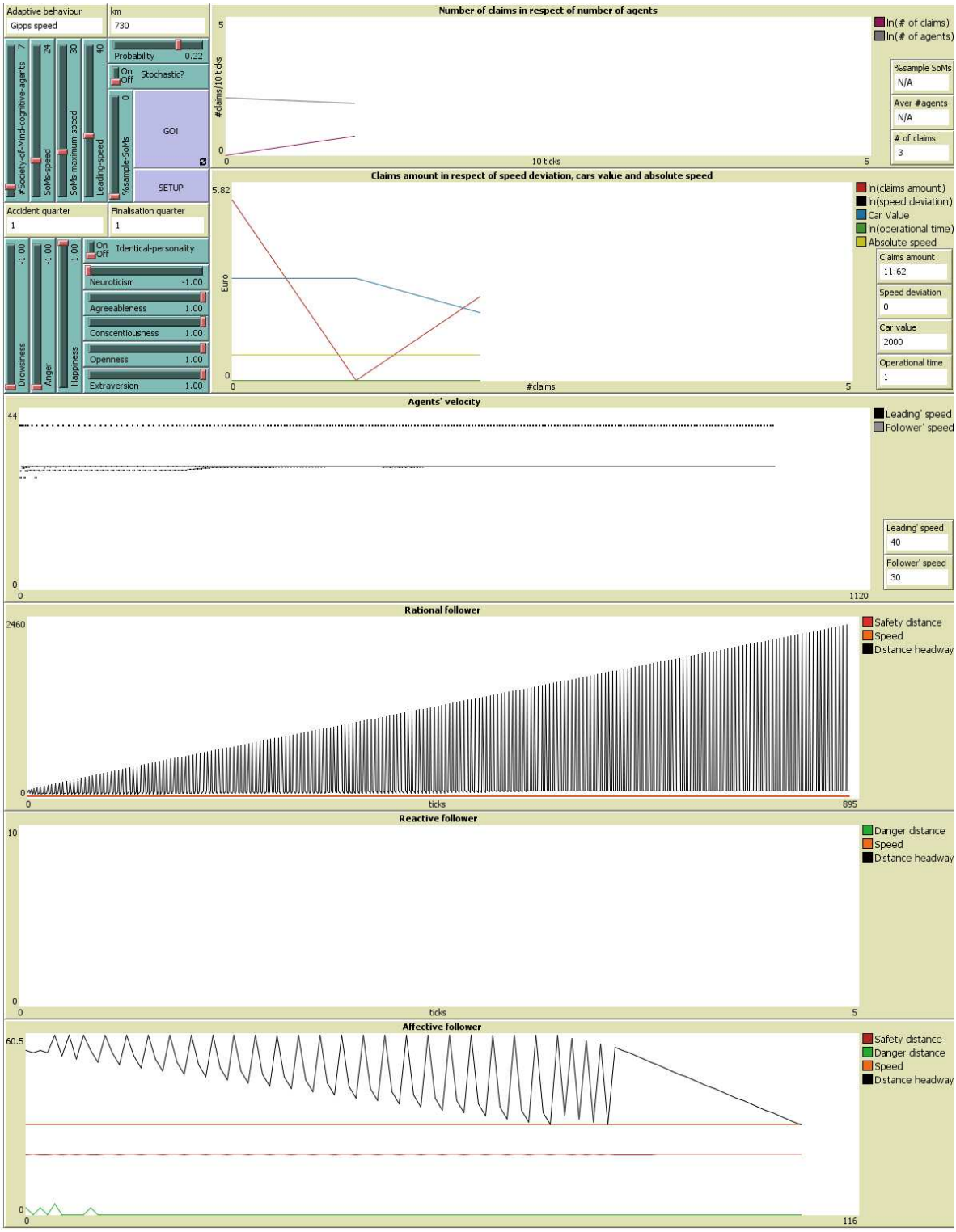

Figure 5.7: Simulation 3 bis

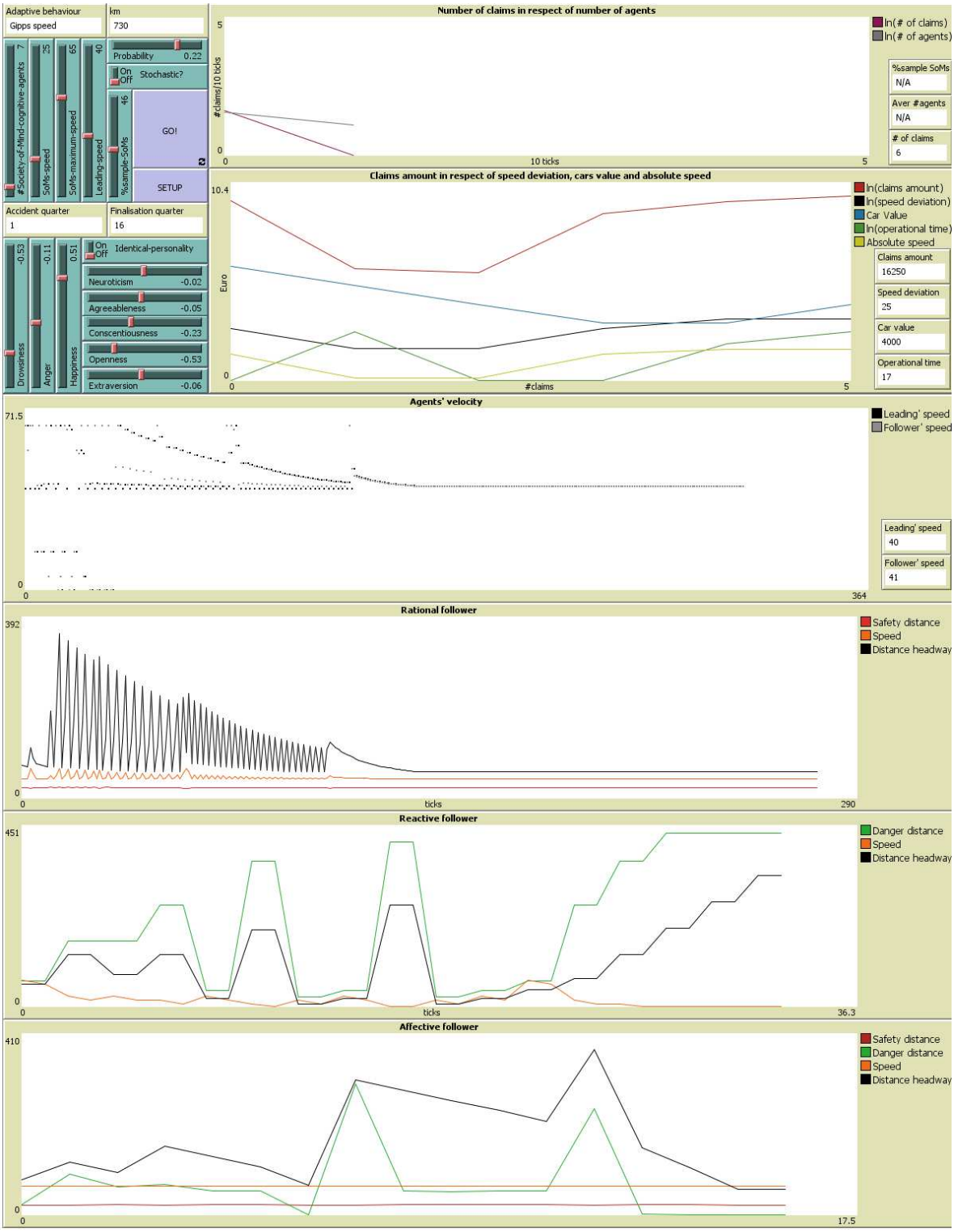

Figure 5.8: Simulation 4

The claim What is interesting here is the positive linear relation between the claim amount, the cars value and the speed deviation clearly shown in the figure, as from the claim amount formula, when the stochastic component is left out.

Simulation 6

The rational agency In this simulation the rational agency provided a good result. After the agents have all been created, they adjusted their speed holding it at the desired level (SoMs-maximum-speed), while still keeping a constant distance headway as well.

Affective agency Affective agent behaved as before, speeding its vehicle. Nevertheless, it did not collide other vehicles.

Reactive agency Typical reactive follower behaviour. It shows that as the distance headway decreased, the SoM decreased its speed as well, so that the distance towards safety decreased as a consequence. Whenever the distance headway increased, the SoM increased its vehicle's speed, while still keeping safe the distance.

5.2.3 High traffic density simulations

Simulation 7 Here 100 homogeneous positive personality SoMs travelled through the 9000 meters road segment at an average speed of 85 km/h, lower than maximum desired of 90 km/h. Due to this speed reduction no claims verified.

Simulation 8 100 homogeneous negative personalities SoMs travelled through the 9000 meters road segment at the desired speed of 100 km/h. 11 accident happened. The worst result.

Simulation 9 100 heterogeneous personality SoMs travelled through the 9000 meters road segment at an average speed of 97 km/h, very close to the desired one of 100 km/h. However, 11 accident happened, lower than simulation 7 number where all SoMs had negative personalities, but good enough reason to travel at a lower speed, as in simulation 7.

5.2.4 Results

The SoM driver agent was tested in a set of traffic contexts in order to demonstrate its usability.

The SoM agent was evaluated in simple car-following situations. Results proved consistency with traffic expectations from a standard car-following point of view. The agent's internal agency activation behaviour (adaptation) demonstrated consistency of its response with both contextual and non-contextual inputs.

Then the SoM agent was evaluated in a multi-agent environment for various traffic densities and behavioural mixes.

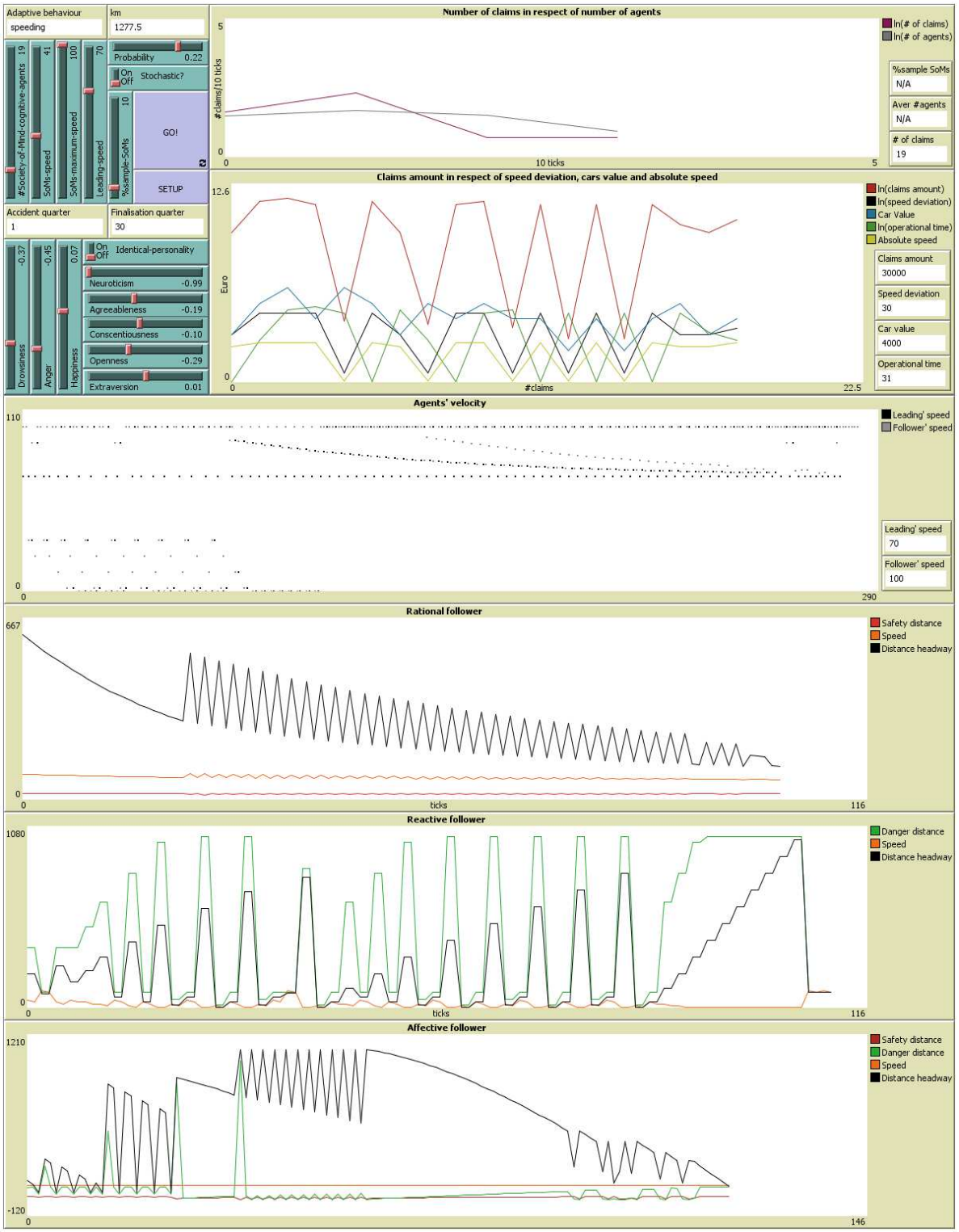

Figure 5.9: Simulation 5

Figure 5.10: Simulation 6

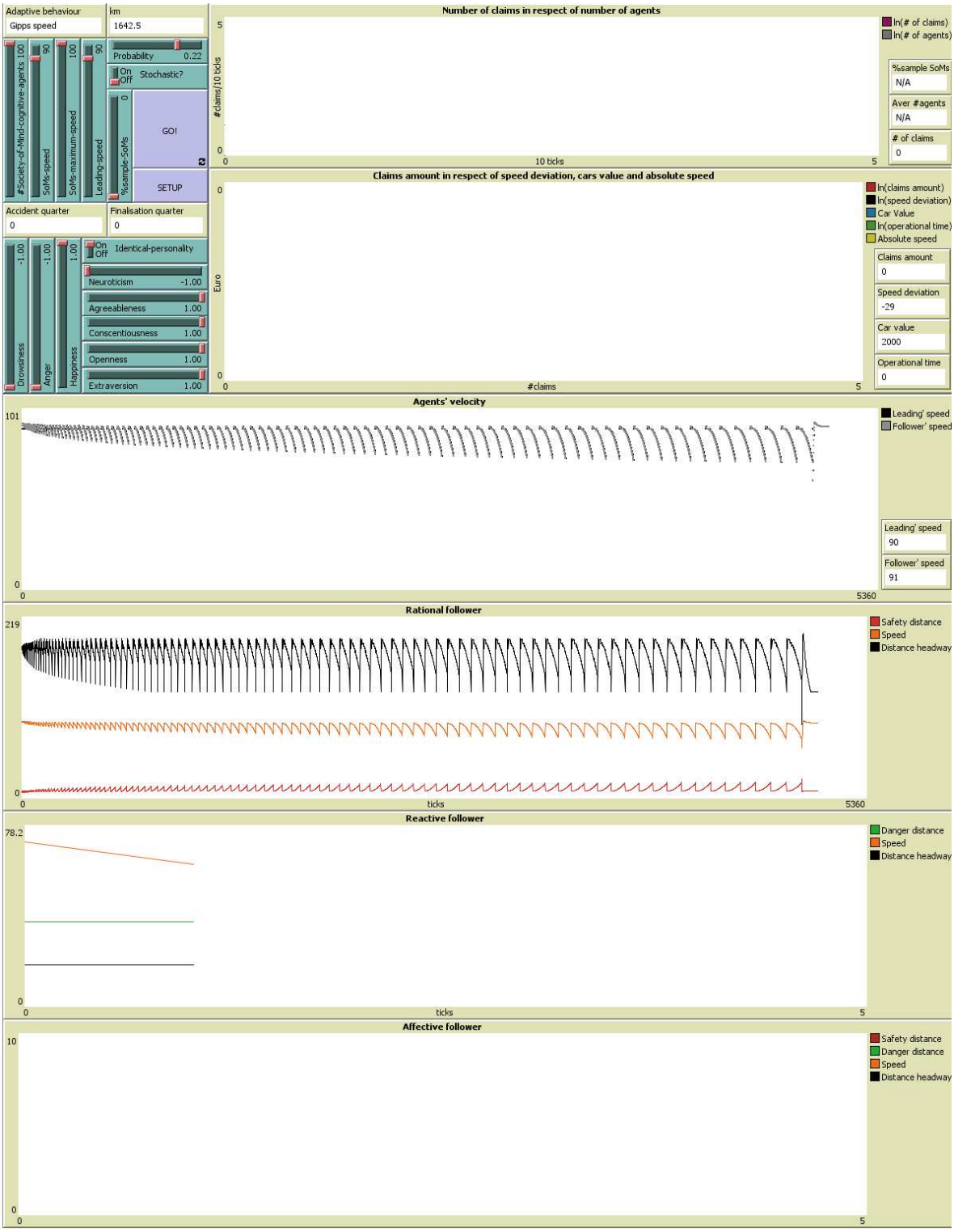

Figure 5.11: Simulation 7

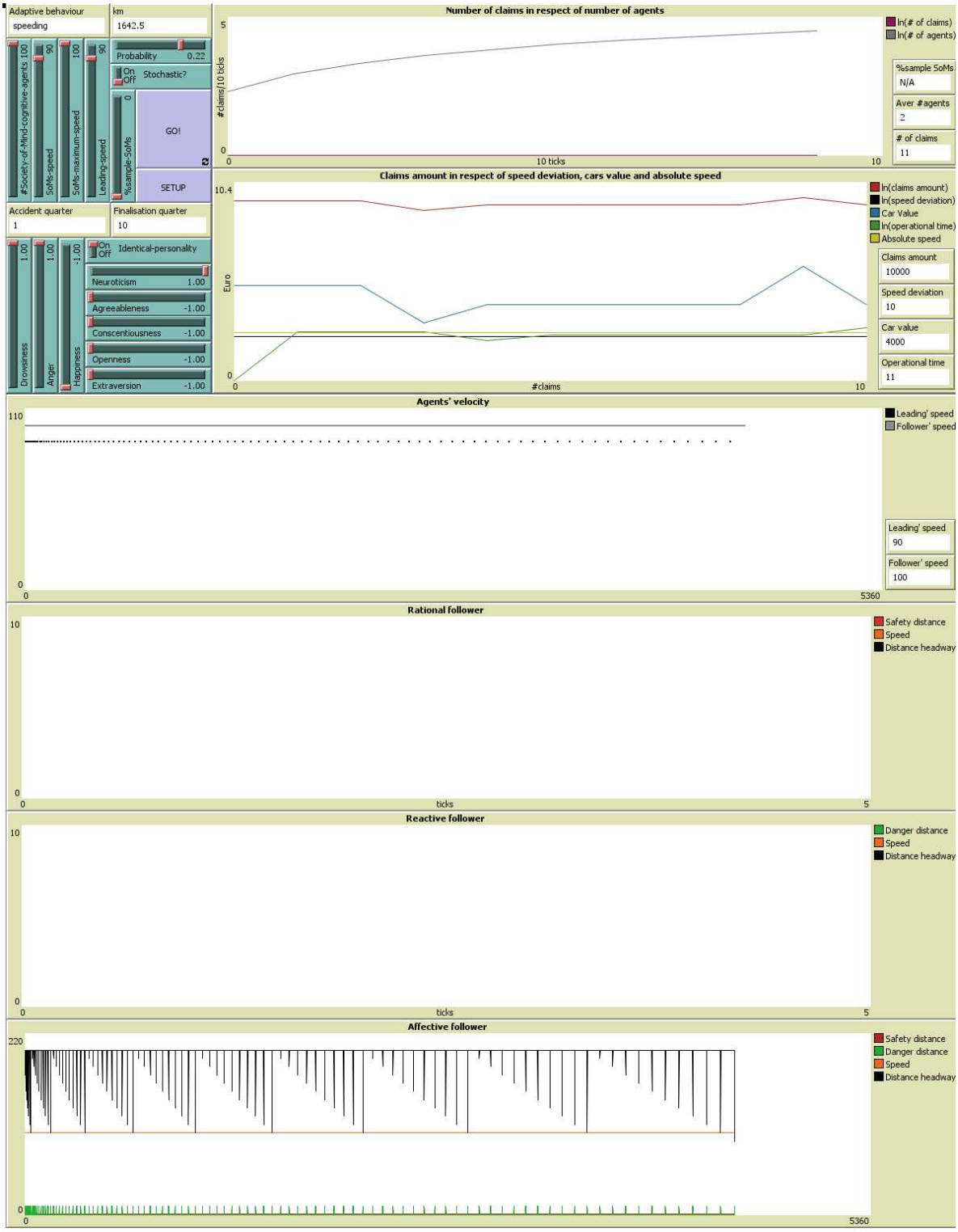

Figure 5.12: Simulation 8

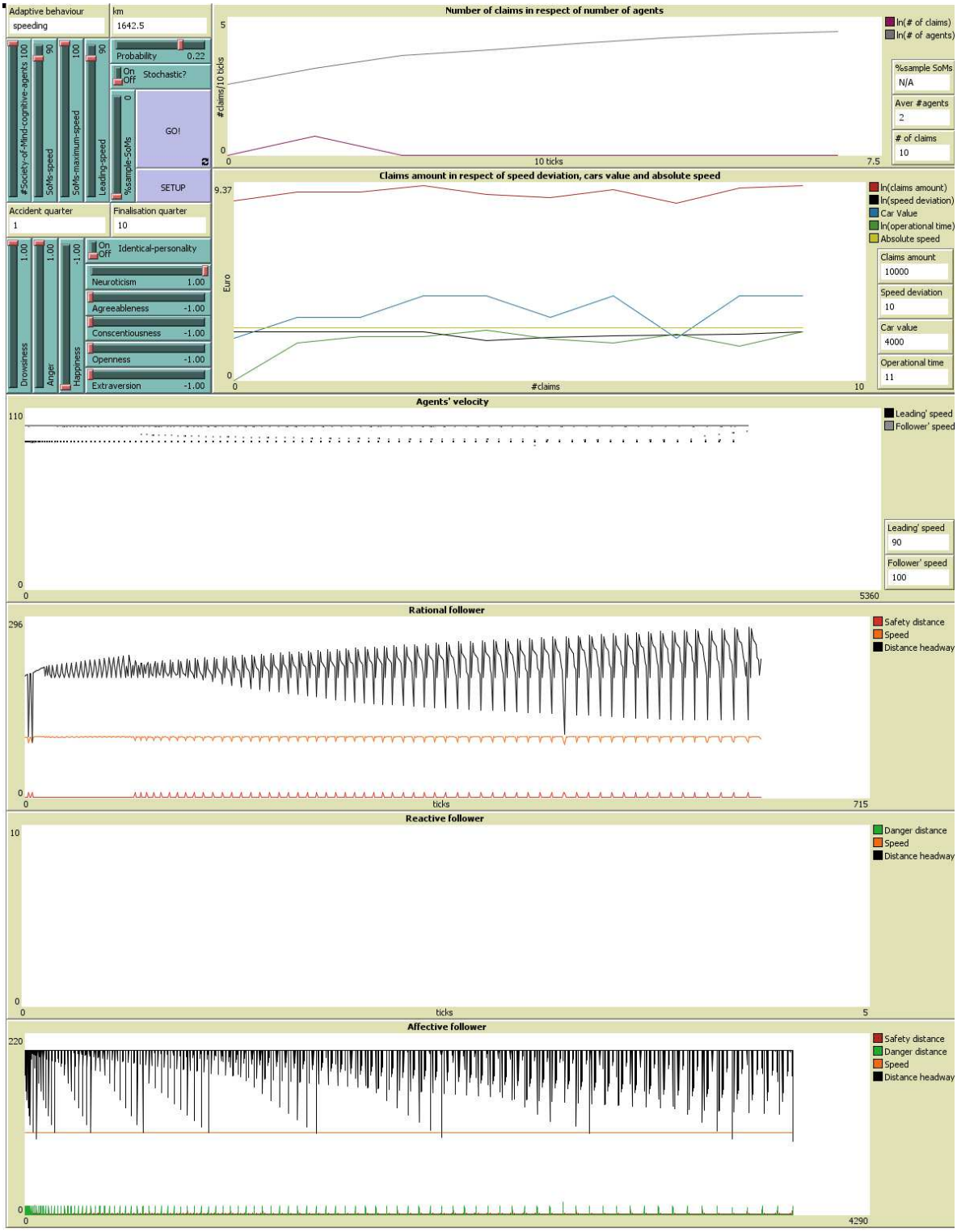

Figure 5.13: Simulation 9

Whenever the identical-personality variable in the model was set true, all the agents showed identical behaviour, i.e. identical personality traits. This is equivalent to considering populations of drivers that are purely homogeneous from a personality point of view.

Many times the simulated SoM's maximum speed was lower than the leading's speed. In such cases, despite personality factors participating in the emotional update function, the resultant decision was mostly rational even for the most negative personalities.

With a high traffic densities the positive rational agent demonstrated the correct behaviour, as expected, travelling at the highest possible safe speed without collide any other vehicle. A very efficient behaviour.

5.2.5 Conclusions

I found very interesting results, confirming the activation rate of each internal agency to be consistent with both innate and contextual inputs.

I showed that the resultant Society of Mind cognitive driver agent design is capable of producing the expected driving behaviours for driver agents with a variety of personality features in a wide range of traffic situations. The adaptive capability demonstrated by the SoM agent architecture confirms the appropriateness of non-hierarchical hybrid agents adoption in many other problems of interest involving multiple cognitive decisions. From here it follows the robustness and the appropriateness of the SoM model application to the Insurance empirical case, proving the realism and the validity of the present thesis.

However, this comes at a cost. Indeed, it is quite difficult to make the agent mental features correctly evolving. In other words, an ability of the modeller in the parameters setting is required. Environmental starting conditions are fundamental in order to allow the agent cognitively behaves. If they were not properly settled, what seems an interesting result actually provides a false truth.

Further studies could evolve for assessment of traffic performance and safety measures, where Agent-Based Models with the capability of recreating the human behavioural variety are required.

Chapter 6

ANNs in Insurance Loss Reserving

In this chapter I am going to explain how Artificial Neural Networks can apply to some aspects of Loss Reserving and Pricing for a motor bodily injury portfolio, which I simulated based on the Agent-Based Model framework in the NetLogo environment.

6.1 The ANN-Model

6.1.1 R internals and packages

R has a few packages for creating neural network models (neuralnet, nnet, RSNNS). Brian Ripley and William Venables (Ripley, 2011) are the developer of *nnet* package, that I used for the construction of my Neural Network.

The *nnet* package

The *nnet* package is a software for feed-forward neural networks with a single hidden layer and for multinomial log-linear models. It has been created to fit single-hidden-layer neural network, possibly with skip-layer connections. It should be noted that the built-in *nnet* implementation is written using C.

nnet(x,...)

```
nnet(x, y, weights, size, Wts, mask,
linout = FALSE, entropy = FALSE, softmax = FALSE,
censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Arguments

x matrix or data frame of x values for examples.

y matrix or data frame of target values for examples.

weights (case) weights for each example – if missing defaults to 1.

size number of units in the hidden layer. Can be zero if there are skip-layer units.

Wts initial parameter vector. If missing chosen at random.

lineout switch for linear output units. Default logistic output units.

entropy switch for entropy (= maximum conditional likelihood) fitting. Default by least-squares.

rang Initial random weights on [-rang, rang]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that rang * max(|x|) is about 1.

decay parameter for weight decay. Default 0.

maxit maximum number of iterations. Default 100

MaxNWts The maximum allowable number of weights. There is no intrinsic limit in the code, but increasing MaxNWts will probably allow fits that are very slow and time-consuming.

abstol Stop if the fit criterion falls below abstol, indicating an essentially perfect fit.

reltol Stop if the optimizer is unable to reduce the fit criterion by a factor of at least 1 - reltol.

Value Object of class "nnet" or "nnet.formula". Mostly internal structure, but has components:

wts the best set of weights found.

value value of fitting criterion plus weight decay term.

fitted.values the fitted values for the training data.

residuals the residuals for the training data.

convergence 1 if the maximum number of iterations was reached, otherwise 0.

The model predictions function *predict* {*stats*}

predict is a generic function for predictions from the results of various model fitting functions (`http://cran.r-project.org/`). The function invokes particular methods which depend on the class of the first argument.

Usage

```
predict (object, ...)
```

Arguments

object is a model object for which prediction is desired;

... are additional arguments affecting the predictions produced.

Most prediction methods which are similar to those for linear models have an argument newdata specifying the first place to look for explanatory variables to be used for prediction. Some considerable attempts are made to match up the columns in newdata to those used for fitting, for example that they are of comparable types and that any factors have the same level set in the same order (or can be transformed to be so).

The form of the value returned by predict depends on the class of its argument.

predict.nnet {*stats*}

predict.nnet predicts new examples by a trained neural net. This function is a method for the generic function *predict()* for class "nnet". It can be invoked by calling *predict(x)* for an object$x$ of the appropriate class, or directly by calling *predict.nnet(x)* regardless of the class of the object.

Usage

```
predict(object, newdata, type = c("raw","class"), ...)
```

Arguments

object is an object of the class nnet as returned by nnet;

newdata is a matrix or a data frame of test examples. A vector is considered to be a row vector comprising a single case;

type is the type of output. If *type* = *"class"*, value is the corresponding class (which is probably only useful if the net was generated by nnet.formula);

... are arguments passed to or from other methods.If *type* = *"raw"*, value is the matrix of values returned by the trained network.

plot.nnet{}

The functions in *nnet* package allow to develop and validate the most common type of neural network model, i.e the feed-forward multi-layer perceptron. The functions have enough flexibility to allow the user to develop the best or most optimal models by varying parameters during

the training process. One major disadvantage is an inability to visualize the models. In fact, neural networks are commonly criticized as "black-boxes" that offer little insight into causative relationships among variables.

plot.nnet is a function for plotting neural networks from the nnet package. This function allows the user to plot the network as a neural interpretation diagram, with the option to plot without colour-coding or shading of weights. The black lines are positive weights and the grey lines are negative weights. Line thickness is in proportion to magnitude of the weight relative to all others.

fitted{*stats*}

fitted is a generic function which extracts fitted values from objects returned by modelling functions. *fitted.values* is an alias for it. All object classes which are returned by model fitting functions should provide a fitted method.

Usage

```
fitted(object, ...)
fitted.values(object, ...)
```

Arguments

object an object for which the extraction of model fitted values is meaningful.

... other arguments.

6.1.2 The dataset

The dataset I generated relates to a scheme of Auto Bodily Injury insurance, which is a compulsory form of insurance. It does not include any component of property coverage.

Cognitive Agents in the simulation model behave similarly to human people. Following their own personality traits and their internal agencies interactions they control car's movements in respect to neighbouring agents. When a car accident happens, the program keeps track of its features, generating a dataset which composes of two different subsets:

- the claims' features dataset *"claimsFeatures"*;
- the claims' amount and finalisation quarter dataset *"amountQuarter"*

The *claimsFeatures* dataset contains all the relevant accident informations:

1. the leading's speed V_n;
2. the follower's speed V_{n-1};

3. the delta-speed $V_n - V_{n-1}$;
4. the value of follower vehicle, in the interval (1,3);
5. the value of leading vehicle, in the interval (1,3);
6. the sum of the two car values, i.e. the total value of cars involved in the accident;
7. the absolute speed, an amount that captures the entity of the claim in respect of the speed the cars were travel. This is relevant to place a difference between low entity claims, happened when cars were probably travelling at a low speed, and high entity claims, when cars were conversely travelling at a high speed;

The *amountQuarter* dataset includes all the relevant claim informations:

1. the claim amount, calculated through a very simple linear equation in addition to a stochastic component to better fit the reality. Actually, this stochastic component enters in the claim's amount calculation only if the user require it to the program;
2. the quarter of claim finalisation, which is linearly determined from the claim amount, in addition again to a stochastic component when specifically required.

We have seen the *Early Stopping* method based on cross validation. In order to apply this method I made NetLogo to divide the dataset composed by claimsFeatures and amountQuarter in two parts, the *training* and the *verification*. While the simulation is running, when half of the simulation time has gone the NetLogo program cuts the claimsFeatures-training and the amountQuarter-training set of data. Accidents and claims features generated from this point are recorded on two new files, the *claimsFeatures-verification* and the *amountQuarter-verification* respectively.

6.1.3 The Neural Network construction procedure

The Neural Network construction starts with the *training procedure*, in which I used the -training dataset in order to train several Neural Networks having different structures. It ends with the *verification procedure*, where the -verification dataset has been used with the aim of testing the Networks generated during the training and finding which one provide better forecast of the verification data subset.

The training

The training starts by defining the X and Y matrices, which contain the data sets *claimsFeatures-training* and *amountQuarter-training* respectively. Figure 6.1 shows the R console passed through this step.

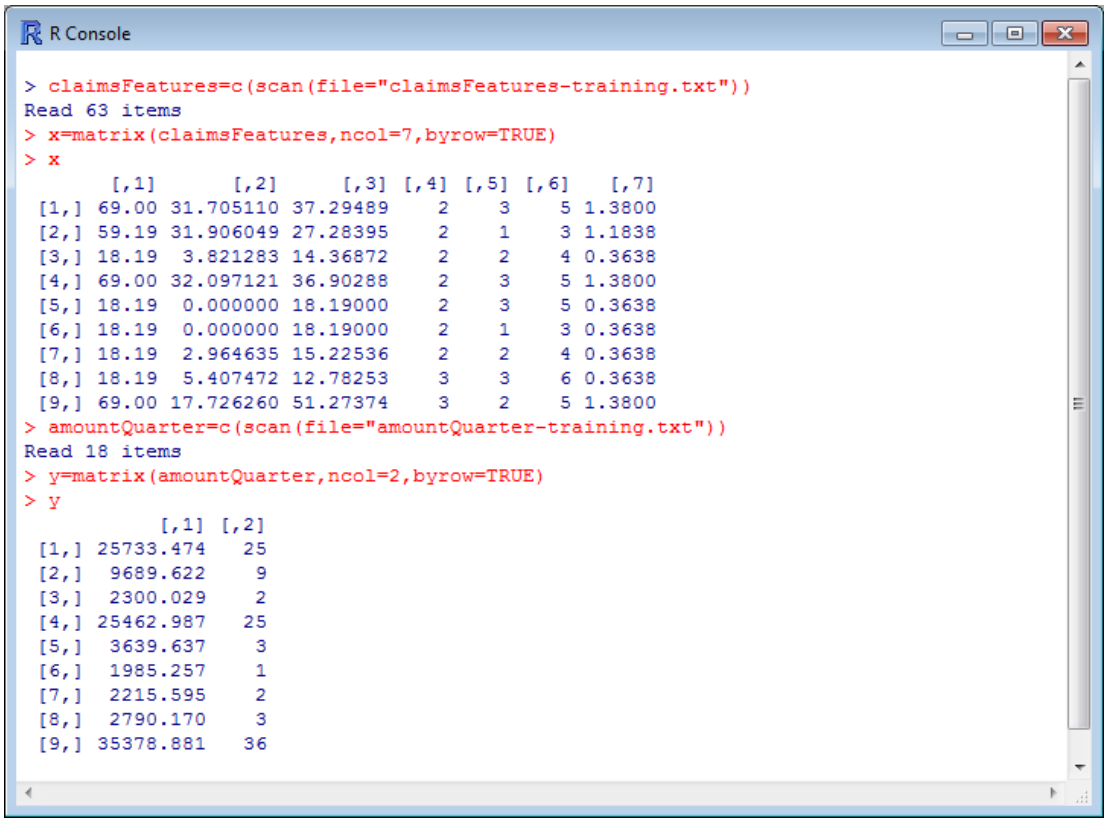

Figure 6.1: Data sets *claimsFeatures-training* and *amountQuarter-training* on the R console

The dataset scaling The R software has created two objects, we have seen, X and Y, containing only the training subsets.

These two matrices are now used to feed the Neural Network, as they constitute a set of examples for the learning algorithm.

We are not required to normalize our data. However, standardizing the inputs we both make the learning process faster and reduce the chances of getting stuck in local optima. Therefore, an adjust function has been defined, which scales both the vector of claims features (claimsFeatures-training) and that of amount and quarter of finalization (amountQuarter-training). Here the code:

```
\# SCALING FUNCTIONS ADJUST TO SCALE X & Y IN (0,1)
adjustX=function(x,xmin,xmax) {(x-xmin)/(xmax-xmin)}
adjustY=function(y,ymin,ymax) {(y-ymin)/(ymax-ymin)}
```

myNet Now that the X and Y matrices containing the claimsFeatures-training and the amountQuarter-training data have been scaled, I assigned them to the *nnet* package, building the Neural Network that I named *myNet*. In the code example reported below I set the Network

to have 8 neurons in the hidden layer, and the maximum number of iterations allowed while trying to converge to 90000.

```
#NEURAL NETWORK'S ARCHITECTURE
myNet=nnet(x=x,y=y,size=8,maxit=90000)
```

Figure 6.2 show a summary of the Neural Network *myNet* in the example. In order to find the

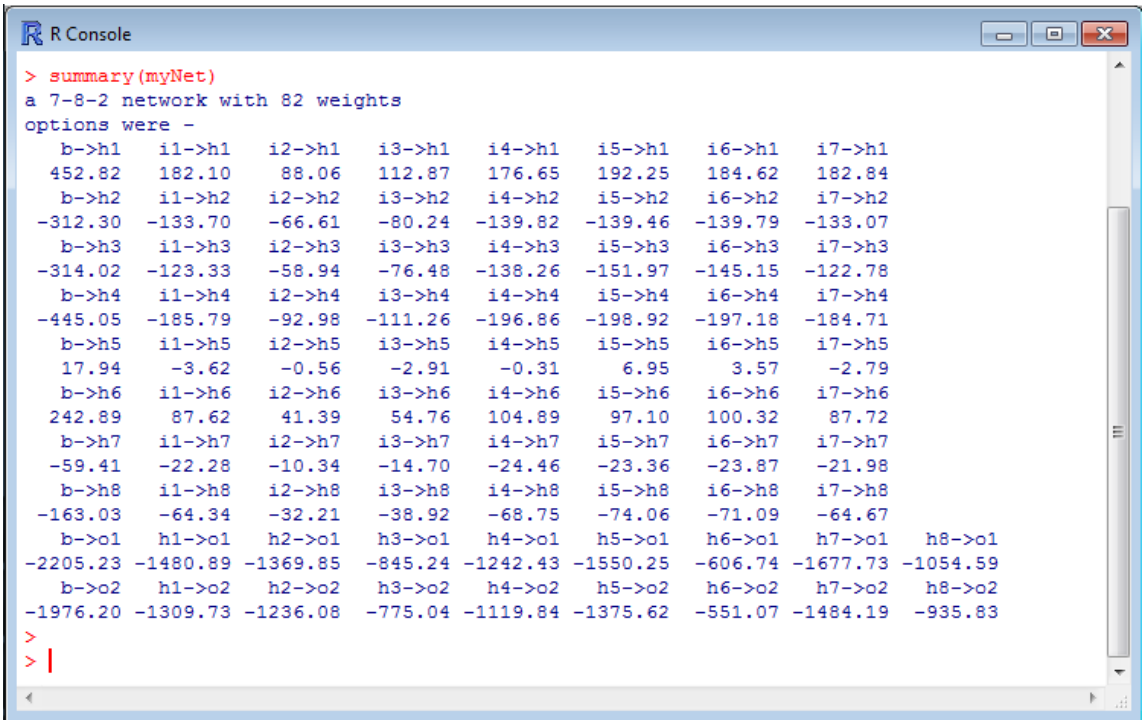

Figure 6.2: Summary of the *myNet* Neural Network

best number of neurons I ran several trials with different configurations of the *size* parameter, which indeed regulates the number of elements in the hidden layer. For each trial I calculated the Sum of Squared Residuals (SSR):

```
#SUM OF "myNet" SQUARED RESIDUALS
sum(myNet$residuals**2)
```

Every time the program starts from a different point on the error function, yielding a different SSR. I let the program to choose initial random values of the weights. As long as I increased the number of neurons the SSR reduced, i.e. the ability to learn the specific mapping through the observation of the set of examples increases. It seems to be a good news. However, as we I am going to show in the verification procedure, this negatively affects the Neural Network's estimation performance, compromising its generalisation capability as already explained in Part II.

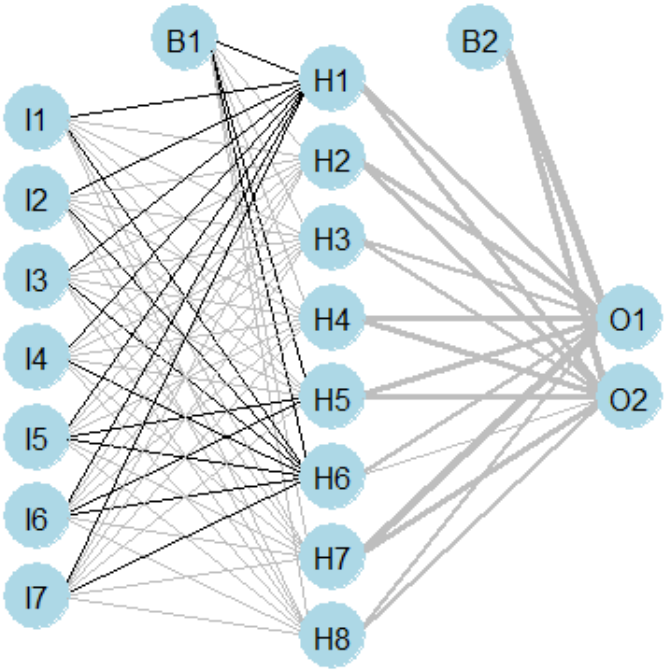

Figure 6.3: Plot of the *myNet* Neural Network in the example. Black lines are positive weights and grey lines are negative weights. Line thickness is in proportion to magnitude of the weight relative to all others.

The verification

Before to feed the Network with the input subset for the testing, the verification procedure develops through the following steps:

1. Create a matrix Y from the amountQuarter-verification set of values

```
#MATRIX OF OBSERVED CLAIMS AMOUNT AND FINALISATION QUARTER FOR VERIFICATION
#amountQuarter=c(scan(file=file.choose()))
amountQuarter=c(scan(file="amountQuarter-verification.txt"))
y=matrix(amountQuarter,ncol=2,byrow=TRUE)
```

   Y contains the observations of claims amount and finalisation quarter associated to the claims features we are going to feed the network with. It constitutes the data subset I separated at the beginning and kept as expected prediction for the verification procedure.

2. Load one of the trained Neural Networks, saved in the training session, having its own number of elements in the hidden layer

```
#LOAD THE NEURAL NETWORK
load("ann8")
```

   In the example above the Neural Network loaded is *ann8*.

3. Build the X matrix containing the claims features subset isolated from the whole dataset explicitly for the verification, i.e. the claimsFeatures-verification. Notice that the Neural Network has never seen these data before

```
#INPUT MATRIX OF CLAIMS FEATURES FOR VERIFICATION
#claimsFeatures=c(scan(file=file.choose()))
claimsFeatures=c(scan(file="claimsFeatures-verification.txt"))
x=matrix(claimsFeatures,ncol=7,byrow=TRUE)
```

4. The items of the X matrix need to be scaled, as we have already seen when I described the training. The function is identical

```
#SCALING FUNCTIONS ADJUST TO SCALE X & Y IN (0,1)
adjustX=function(x,xmin,xmax) {(x-xmin)/(xmax-xmin)}
```

myPrediction Everything is ready for the verification. We got our list of trained Neural Networks to be tested, verifying how proper they are in *out-of-sample* estimation. Indeed, since the test set is not used during the training, the forecast made from the test subset is an out-of the training sample one, despite the realizations of the dataset we are testing are already known.

Therefore, in order to predict from the results of the Neural Network fitting, I assigned the X matrix containing the claims features to the *predict* function

```
#PREDICTION FUNCTION
myPrediction=predict(myNet,x)
```

The *predict* function performs the forecast of the $\hat{Y}$ matrix, that I named "myPrediction", through the trained Neural Network *myNet*. The result is a two columns matrix, representing the estimate of the claims *amount* and *finalisation quarter* resulting from the claims features we have fed the Network with.

myPrediction matrix is still scaled in (0,1). The next step is therefore to *descale* it. The function is the opposite as it was before

```
#DESCALE THE PREDICTED OUTPUT MATRIX "myPrediction"
descale=function(myPrediction,yMin,yMax) {myPrediction*(yMax-yMin)+yMin}
yMin1=min(y[,1])
```

```
yMax1=max(y[,1])
myPrediction[,1]=descale(myPrediction[,1],yMin1,yMax1)
yMin2=min(y[,2])
yMax2=max(y[,2])
myPrediction[,2]=descale(myPrediction[,2],yMin2,yMax2)
```

Now that we hold both the observed values, i.e. the Y matrix, and the estimated values, i.e. the *myPrediction* matrix, we can finally compare them in order to gain the estimation error and verify the goodness of fit.

Model selection We have seen each Neural Network has a Sum of Squared Residual value. Since it belongs to the Network, it still remains the same during the verification procedure. Therefore it can not be used to check whether a Network would better fit our dataset than one other.

What can be used to compare the precision of a Network in respect of another are some Goodness of Fit measures. To this aim I used the R packages *base* and *hydroGOF*, where GOF stands for Goodness Of Fit

```
#LOAD PACKAGES
library(base)
library(hydroGOF)
```

I personally retain the following statistics, listed in the R code, would give a proper criteria for model selection and forecasting performance evaluation:

```
#SQUARED SUM OF DIFFERENCES
sum((y[,1]-myPrediction[,1])**2)
sum((y[,2]-myPrediction[,2])**2)

#MEAN SQUARE ERROR
mse(myPrediction,y)

#ROOT MEAN SQUARE ERROR
rmse(myPrediction,y)

#MEAN ABSOLUTE ERROR
mae(myPrediction,y)

#MEDIAN ABSOLUTE DEVIATION
mad(myPrediction[,1])
mad(y[,1])
mad(myPrediction[,2])
```

```
mad(y[,2])
```

I computed these statistics for each of the Neural Networks that I trained, for both the deterministic and the stochastic data sets, as reported in fig 6.5. Then I compared the results. The table illustrates the trials that I made to evaluate the Neural Networks' Goodness of Fit. The heading *ann* in the table corresponds to the name I assigned to the Networks on my local data. The heading *size* express the value I set the parameter *size* of the *nnet* package in R, corresponding to the number of elements in the hidden layer. Trials have been performed on two different datasets, dataset 1 and dataset 2. Each one has been generated in two different versions, keeping equal all agents' parameters in NetLogo, except for the stochastic component. Doing this way it has been generated the dataset without the stochastic component (deterministic) and the one with it (stochastic). Each one composes of:

- claimsFeatures-training generated without the stochastic component;
- amountQuarter-training generated without the stochastic component;
- claimsFeatures-verification generated without the stochastic component;
- amountQuarter-verification generated without the stochastic component;
- claimsFeatures-training generated with the stochastic component;
- amountQuarter-training generated with the stochastic component;
- claimsFeatures-verification generated with the stochastic component;
- amountQuarter-verification generated with the stochastic component;

A Neural Network trained on the training deterministic subset of the deterministic dataset tested on the verification deterministic subset of the same deterministic dataset is reported in the table as ann_i, *deter* under the training heading, *deter* under the verification heading. The stochastic case therefore corresponds to ann_i, *stoch*, *stoch*. I also tried to forecast a stochastic dataset using a Network that has never seen stochastic examples before. This case is reported in the table as ann_i, *deter*, *stoch*. I also made the opposite case.

The resulting Goodness of Fit Measures confirm what was my expectation. In general, a higher number of neurons produces a lower SSR value during the training procedure (*SSR-training* in the table). Whereas in the case of deterministic dataset this translate in a performance increase, it does not in the stochastic dataset case. As a matter of fact, increasing the size parameter over 8 when forecasting a stochastic scenario will determine a poor estimate, causing a loss of accuracy.

I interpret this result as problem of *generalisation* of the Neural Network when the data underlying relation is nonlinear.

dataset	ann	train	verif	size	decay	maxit	SSR-train	SSD-1	SSD-2	MSE-1	MSE-2	RMSE-1	RMSE-2	MAE-1	MAE-2
1	1	stoch	stoch	90	0	90000	1,0675	2,31E+09	3,09E+03	4,64E+05	6,19E-01	681,0969	0,7868	546,3691	0,5111
1	2	stoch	stoch	90	0	90000	1,0449	1,62E+09	5,11E+03	3,24E+05	1,03E+00	569,6311	1,0126	445,6181	0,6399
1	3	stoch	stoch	40	0	90000	1,0756	3,48E+09	4,00E+03	6,98E+05	8,02E-01	835,3585	0,8958	730,5082	0,5990
1	4	stoch	stoch	10	0	90000	1,1317	1,39E+09	2,91E+03	2,80E+05	5,84E-01	528,8342	0,7643	304,4984	0,4952
1	5	stoch	stoch	10	0	90000	1,1595	1,25E+09	2,65E+03	2,52E+05	5,32E-01	501,5884	0,7296	281,7249	0,4769
1	6	stoch	stoch	9	0	90000	1,2053	1,70E+09	1,75E+03	3,41E+05	3,51E-01	584,2905	0,5922	468,6253	0,3789
1	7	stoch	stoch	8	0	90000	1,1474	1,55E+09	3,11E+03	3,12E+05	6,23E-01	558,3429	0,7895	309,0919	0,5235
1	8	stoch	stoch	8	0	90000	1,1397	1,23E+09	2,66E+03	2,46E+05	5,34E-01	496,4596	0,7308	267,5376	0,4717
1	9	stoch	stoch	7	0	90000	1,1580	1,99E+09	1,72E+03	3,98E+05	3,44E-01	630,9610	71,9061	558,7801	0,3858
1	10	stoch	stoch	7	0	90000	1,1455	1,79E+09	1,67E+03	3,59E+05	3,34E-01	599,0779	0,5781	524,4983	0,3756
1	11	stoch	stoch	5	0	90000	1,1817	1,82E+09	3,73E+03	3,65E+05	7,48E-01	604,1177	0,8648	389,0418	0,5910
1	12	stoch	stoch	5	0	90000	2,1040	2,63E+09	3,00E+03	5,28E+05	6,02E-01	726,7783	0,7761	504,2465	0,4904
2	13	stoch	stoch	29	0	90000	2,6706	2,80E+10	1,72E+04	5,76E+06	3,55E+00	2400,2409	1,8842	1941,0794	1,4842
2	14	stoch	stoch	10	0	90000	2,7315	3,84E+10	2,81E+04	7,90E+06	5,78E+00	2811,3624	2,4041	2306,7007	1,9455
2	15	stoch	stoch	8	0	90000	3,0046	1,58E+10	1,05E+04	3,26E+06	2,16E+00	1805,9254	1,4693	1406,7281	1,0832
2	16	stoch	stoch	7	0	90000	2,7444	2,30E+10	1,55E+04	4,73E+06	3,19E+00	2174,7954	1,7853	1751,6131	1,4059
2	17	stoch	stoch	5	0	90000	2,7850	4,29E+10	3,19E+04	8,84E+06	6,58E+00	2973,9682	2,5642	2482,5171	2,1172
2	18	stoch	stoch	3	0	90000	3,8068	1,09E+10	1,47E+04	2,25E+06	3,03E+00	1501,3701	1,7400	1122,5692	1,3980
2	18	stoch	deter	3	0	90000	3,8068	1,09E+10	1,47E+04	1,66E+06	1,39E+00	1289,8583	1,1811	850,6279	0,8743
2	15	stoch	deter	8	0	90000	3,0046	1,58E+10	1,05E+04	3,26E+06	2,16E+00	1805,9254	1,4693	1406,7281	1,0832
1 to 2	8	stoch	deter	8	0	90000	1,1397	4,56E+08	1,97E+03	8,70E+04	3,77E-01	295,0040	0,6142	240,0455	0,4476
1	19	deter	deter	90	0	90000	0,0754	2,95E+09	8,72E+03	6,11E+05	1,81E+00	781,9829	1,3445	651,2157	0,7440
1	20	deter	deter	30	0	90000	0,1063	6,05E+08	5,57E+02	1,25E+05	1,15E-01	354,2306	0,3397	287,0350	0,2057
1	21	deter	deter	10	0	90000	0,1624	1,07E+09	1,24E+03	2,21E+05	2,58E-01	470,0077	0,5081	370,9521	0,3866
1	22	deter	deter	5	0	90000	0,4460	4,58E+08	8,38E+02	9,49E+04	1,74E-01	308,0983	0,4168	231,4311	0,3334
1	23	deter	deter	3	0	90000	1,7546	1,56E+10	1,57E+04	3,24E+06	3,25E+00	1799,1893	1,8025	1481,9193	1,4577
1	24	deter	deter	1	0	90000	7,3693	8,20E+09	8,01E+03	1,70E+06	1,66E+00	1304,0227	1,2890	1001,8692	1,0424
2	25	deter	deter	50	0	90000	0,0836	8,20E+09	8,01E+03	2,33E+05	9,44E-01	482,4947	0,9717	385,3502	0,4709
2	26	deter	deter	20	0	90000	0,1493	1,44E+09	2,78E+03	2,75E+05	5,32E-01	524,7059	0,7292	426,4733	0,5587
2	27	deter	deter	3	0	90000	4,6772	4,72E+09	5,00E+03	9,78E+05	1,04E+00	988,9470	1,0186	701,5676	0,7369
2	28	deter	stoch	50	0	90000	0,0836	8,87E+09	5,02E+03	2,82E+06	5,29E+00	1677,8384	2,3002	1308,3653	1,5343
2	29	deter	stoch	5	0	90000	0,4460	3,58E+09	4,16E+03	7,37E+05	8,57E-01	858,7356	0,9255	442,6346	0,6198

Figure 6.4: Neural Networks trials

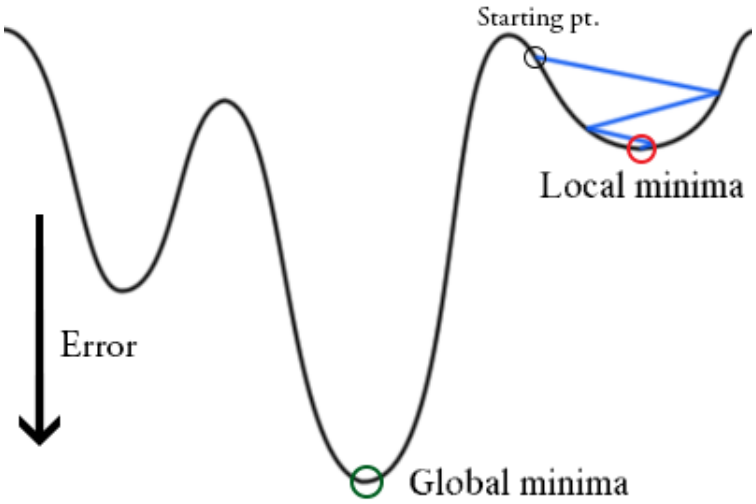

Figure 6.5: If the starting point for gradient descent was chosen inappropriately, more iterations of the algorithm will only make it approach a local minimum, never reaching the global one.

Subsequent considerations concern the model selection. Looking at the GOF measures, I would prefer Neural Network 8, since it performs very well both when stochastic and deterministic dataset has given. Most interesting results in the table are highlighted in grey.

Finally, the stochastic component affects the finalisation quarter determination. This result comes out from the estimation error. Goodness of Fit measures display higher value for finalisation quarter in the case of stochastic data. This is due to the more difficulties the Networks encounter in learning examples having an intrinsic randomness. Nevertheless it must be noted that despite one would expect predictions on deterministic dataset to be slightly more accurate, Neural Networks have surprisingly well performed also on stochastic dataset, confirming their worthiness.

The plotting I want to show here the graphical representation of some interesting Neural Networks fittings, which are listed in the table too.

6.1.4 Out-of-sample forecasting

I created several Neural Networks. Through the verification procedure I then chose the one with the lowest forecast error. The aim was to get a powerful tool to perform regression on a given set of claims whose amount and finalisation are still unknown. Indeed an important part of any reserving or pricing analysis is to project estimates into future periods.

6.2 Neural Network vs. Chain Ladder

In the present section I model an out-of-sample Neural Network forecast. Then I implement a classical deterministic Chain Ladder algorithm on the same dataset. The aim is to asses the Neural Network created in the previous section by comparing it with a standard Chain Ladder method.

6.2.1 The *ChainLadder* package

The *ChainLadder* package created by Markus Gesmann provides various statistical methods which are typically used for the estimation of outstanding claims reserves in general insurance. The package has implementations of the Mach, Munich, Bootstrap, multivariate and Chain Ladder factor models (CLFM), as well as the loss development factor curve fitting methods of Dave Clark and Generalised Linear Model based reserving models.

Historical insurance data is often presented in form of a triangle structure, showing the development of claims over time for each exposure (origin) period. An origin period could be the year the policy was sold, or the accident year.

Most reserving methods of the ChainLadder package expect triangles as input data sets with development periods along the columns and the origin period in rows. I already presented literature explaining the method.

chainladder

```
chainladder(Triangle, weights = 1, delta = 1)
```

Basic Chain Ladder function to estimate age-to-age factors for a given cumulative run-off triangle. This function is used by Mack- and MunichChainLadder.

Arguments

Triangle cumulative claims triangle.

weights sets the weights for all triangle entries to 1.

delta 'weighting' parameters.

The key idea is to see the Chain Ladder algorithm as a weighted linear regression through the origin applied to each development period.

Value chainladder returns a list with the following elements:

Models linear regression models for each development period;

Triangle input triangle of cumulative claims;

weights weights used;

delta deltas used.

6.2.2 The dataset

I generated and modelled a dataset recording claims for 20 accident years. This dataset reports two additional items with respect to the one I presented before:

- Date of injury;
- Development periods.

These are necessary for the Chain Ladder model.

Moreover, the dataset displays some clear trends that I specifically created such as:

- changes in the rate of claim finalisation;
- the average size of finalised claims increases with operational time;
- seasonality.

These features are not uncommon in accident compensation. It has been demonstrated that the traditional Chain Ladder has difficulty in coping with these features (Taylor and McGuire, 2004).

I try to demonstrate here how the architecture of the Neural Network provides an effective framework for dealing with these features.

6.2.3 The Neural Network prediction

In order to be consistent with the work already done I loaded the Neural Network *8*, which had well performed during the training and verification procedures. I also modelled the dataset through the Neural Networks *3* and *9*, in order to give proof that, also in out-of-sample estimation, the best Neural Network was indeed the *8* (which I remind, has 8 neurons in the hidden layer), selected at the end of the verification.

The R code and the procedure for the Neural Network forecasting are exactly the same identical that I explained for the verification.

When the prediction stopped, the estimated ultimate claims were summed up and compared with the true observed ultimate claims from the dataset. I show the results at the end of the section.

6.2.4 The Chain Ladder prediction

I wrote my own code to transform an mxn incremental matrix into a triangle, an incremental one first and then in a cumulative C_{ik} one, which was filled for $k \leq n+1-i; i = 1, .., m; m \geq n$. Alternatively, one could adopt the ChainLadder package's built-in functions.

```
TRIANGLE<-matrix(nrow=20,ncol=20)
amountQuarter=c(scan(file="amountQuarter-training.txt"))
y=matrix(amountQuarter,ncol=4,byrow=TRUE)

#INCREMENTAL TO CUMULATIVE PAID CLAIMS MATRIX##############################
fit<-function(m,n){y[y[,2]==m & y[,3]==n]}
i<-1
j<-1
m<-1
n<-0
while(i!=21){
while(j!=21){
temporary<-fit(m,n)
temporary<-matrix(ncol=4,temporary)
TRIANGLE[i,j]<-sum(temporary[,1])
j<-j+1
n<-n+1
}
i<-i+1
m<-m+1
j<-1
n<-0
}
save(TRIANGLE,file="completeTriangle")

#PAID CLAIMS TRIANGLE#########################################################
j=2
while(j!=21){
TRIANGLE[,j]<-TRIANGLE[,j-1]+TRIANGLE[,j]
j=j+1
}
i<-20
j<-20
k<-1
while(i!=1){
while(j!=k){
TRIANGLE[i,j]<-NA
j<-j-1
}
```

```
i<-i-1
j<-20
k<-k+1
}
```

When the triangle was ready, I implemented a standard chain latter without tail factor. It can be seen in the code below that the parameter *tail* is set equal to 1.

```
##STANDARD CHAIN LADDER without TAIL####################################
chainladder(TRIANGLE, weights = 1, delta = 1)
linkratios <- c(attr(ata(TRIANGLE), "vwtd"), tail = 1)
round(linkratios, 3)
LDF <- rev(cumprod(rev(linkratios)))
names(LDF) <- colnames(TRIANGLE)
round(LDF, 3)
currentEval <- getLatestCumulative(TRIANGLE)
EstdUlt <- currentEval * rev(LDF)
Exhibit <- data.frame(currentEval, LDF = round(rev(LDF), 3), EstdUlt)
Exhibit <- rbind(Exhibit,data.frame(currentEval=sum(currentEval), LDF=NA,
 EstdUlt=sum(EstdUlt),row.names = "Total"))
Exhibit
plot(TRIANGLE)
graphics.off()
```

The estimated ultimate claims *EstdUlt* were then summed up, and compared with the true observed ultimate claims from the dataset, as before.

I also run a Chain Ladder with tail factor at 1.05.

Results

The results of the soft-computing model fitting exercises are shown in the following two tables. They are surprising. In all the three trials that I made the Neural Networks outperformed the Chain Ladder.

6.2.5 The Mack-Chain-Ladder

We have seen the Ultimate Claims estimation. It is also interesting to forecast the IBNR (Incurred But Not Reported) claims based on the cumulative claims development triangle and to estimate the standard error around it. This goal can be achieved only through a stochastic evaluation model of the time series of claims. Indeed, despite a major complexity and the need for restrictions on the underlying data set, only stochastic models allow, in addition to a point estimate, also the reserve range of variation according to a certain probability level. Conversely,

deterministic techniques provide only a point estimate, disregarding any analysis on the probability of its verification. To this aim I implemented the Mack Chain Ladder on the simulated data. The modelling results are shown below.

6.2.6 Conclusions

The first model fitting ended with the Neural Network underestimating the observed dataset with a relative error of 0,121% only. This wonderful result repeated in the second trial.

In general, we can say that the Chain Ladder without tail factor systematically under estimated the dataset. Moreover, Neural Network *9* was the most inaccurate in each trial.

In the third trial, due to its generalisation capability, the Neural Network *3* outperformed the both the Neural Network *2* and the chain ladder. This time the Chain ladder with tail made better forecast than the Neural Network *3*.

This illustrates that the success of a particular method depends to a large extent on how appropriate the method's architecture is to the problem. This will not always be apparent at the outset and it often desirable to try a number of different methods. Moreover, the soft computing methods were not completely automated. I found that some skill/experimentation was required to get optimal performance out of each algorithm. The ability of soft-computing methods to automatically model the complex features of a data set mean that soft-computing methods may play important roles in model verification and checking.

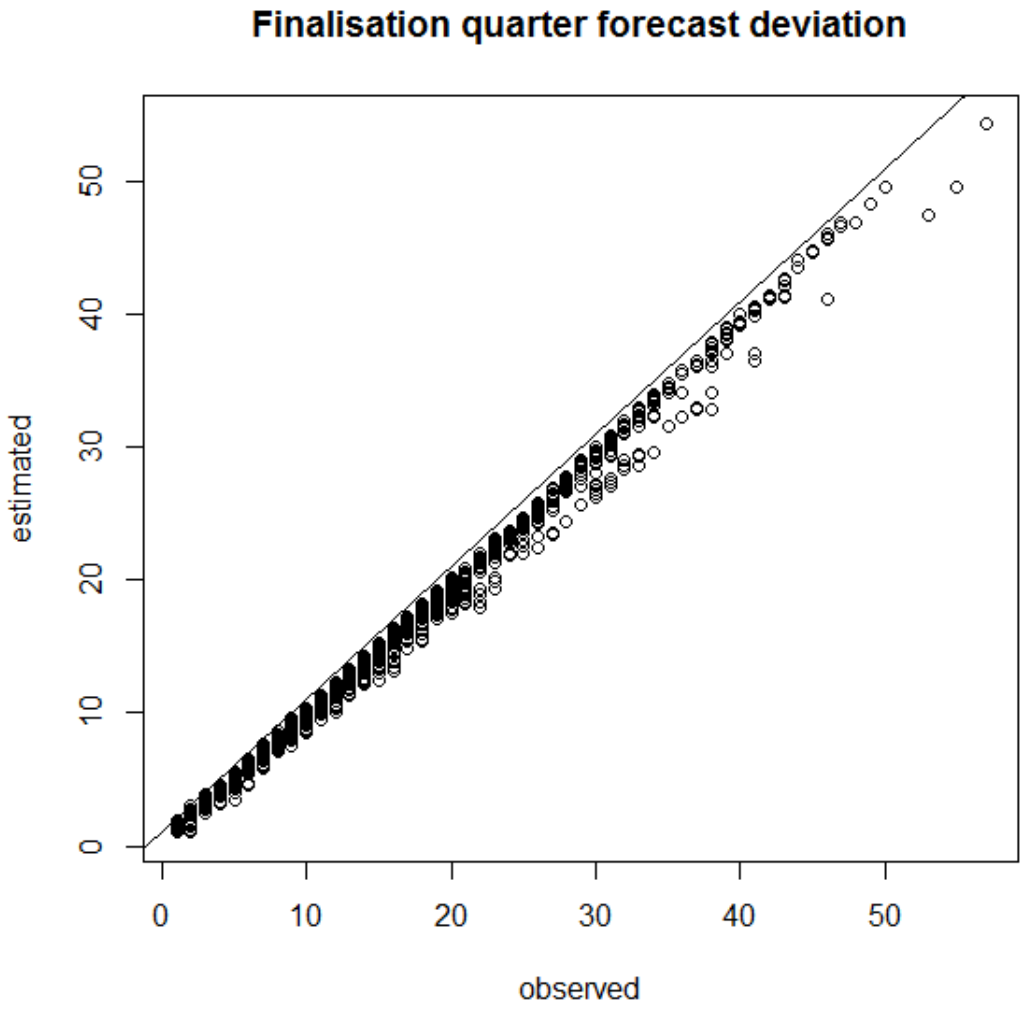

Figure 6.6: myNet *8*, a good fit. The figure graphically give confirm that Artificial Neural Network *8* provided a good performance, as seen before.

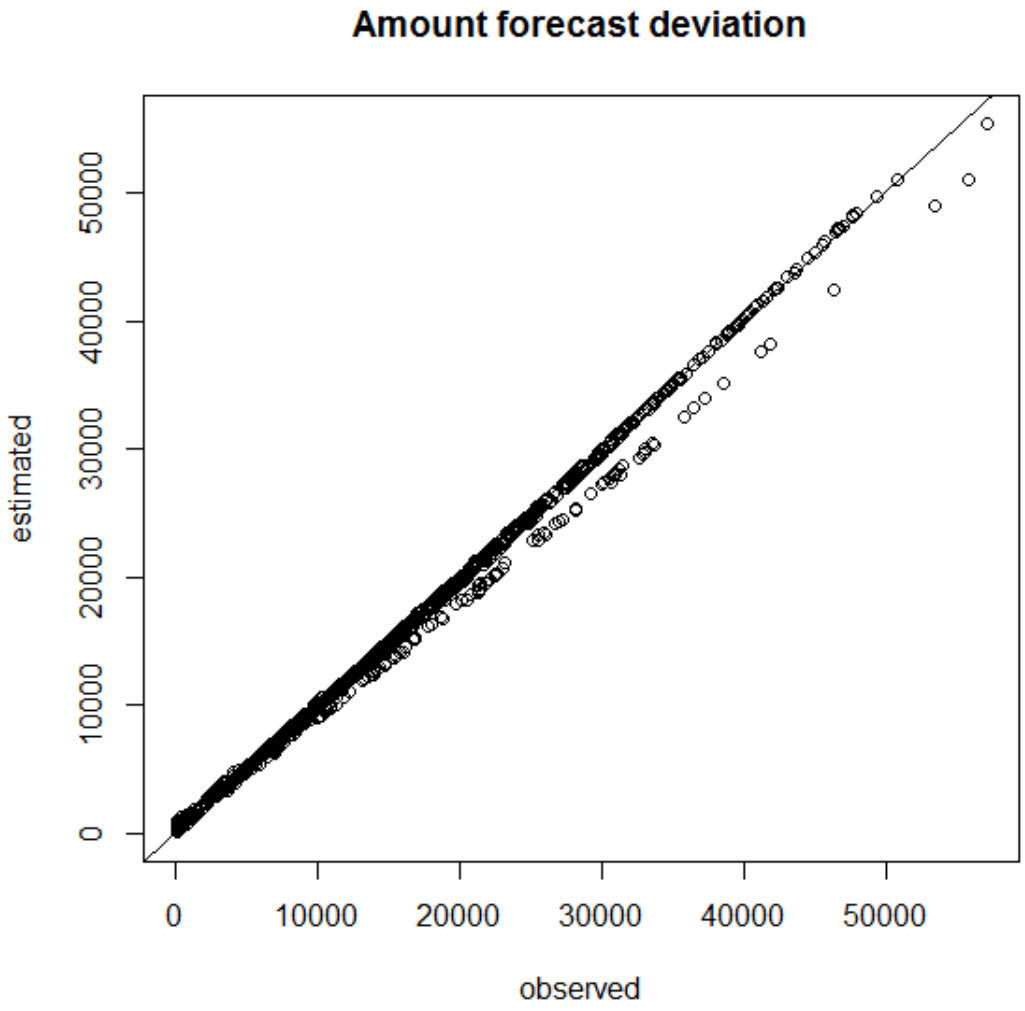

Figure 6.7: myNet *8*, a good fit. myNet *8*, a good fit. The figure graphically gives confirm that Neural Network *8* provided a good performance, as seen before.

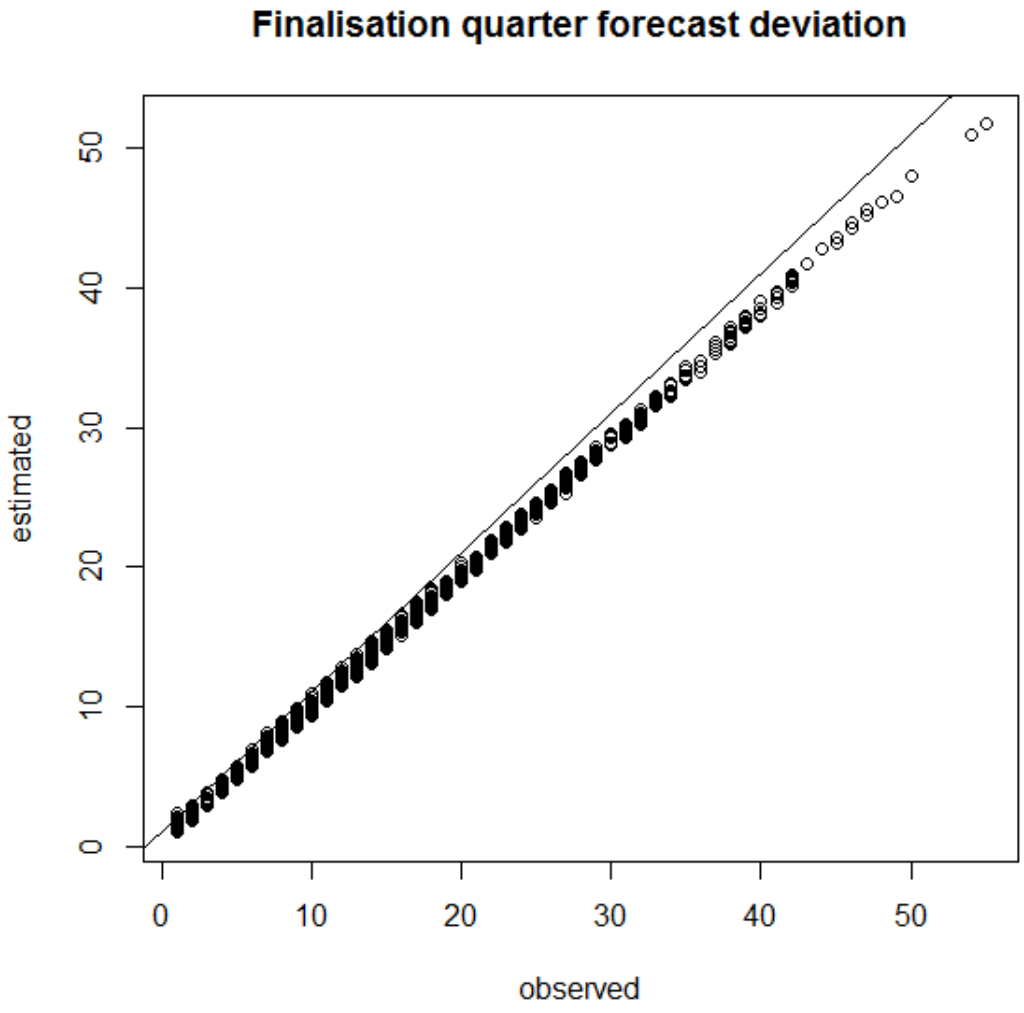

Figure 6.8: myNet *8*. Also when tested on a deterministic dataset, despite it has been trained on a stochastic one, Neural Network *8* fit very well the underlying data.

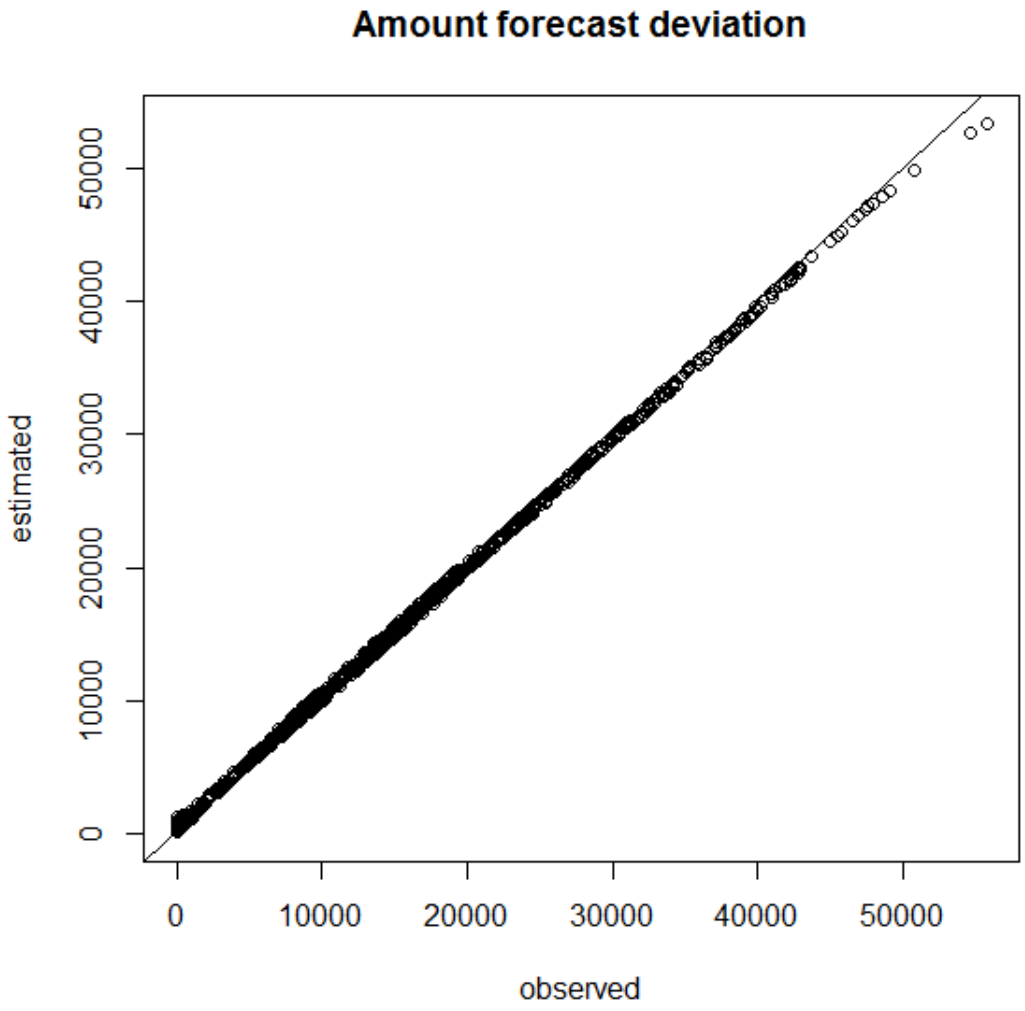

Figure 6.9: myNet *8*. Also when tested on a deterministic dataset, despite it has been trained on a stochastic one, Neural Network *8* fit very well the underlying data.

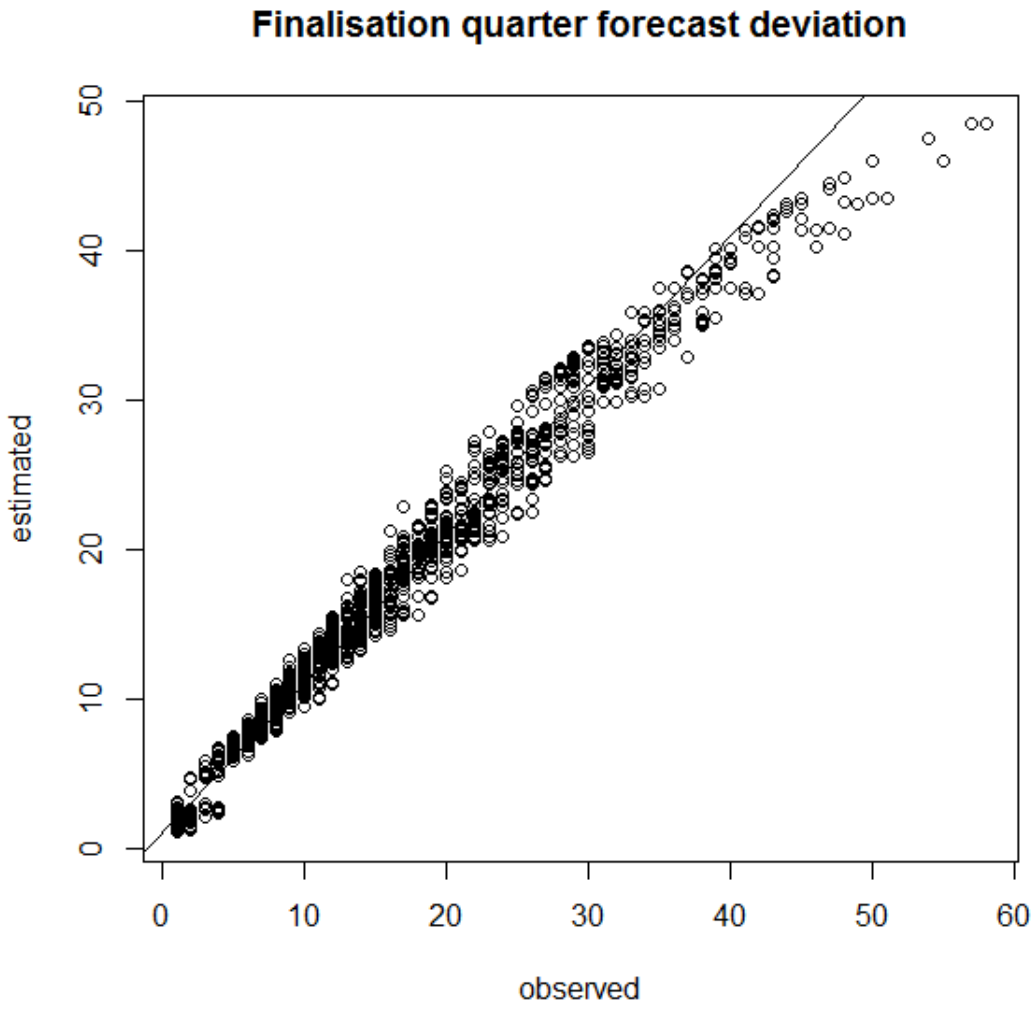

Figure 6.10: myNet 18, not an accurate prediction. It can be seen from the figure how the values predicted by Neural Network 18 differ from the true observations.

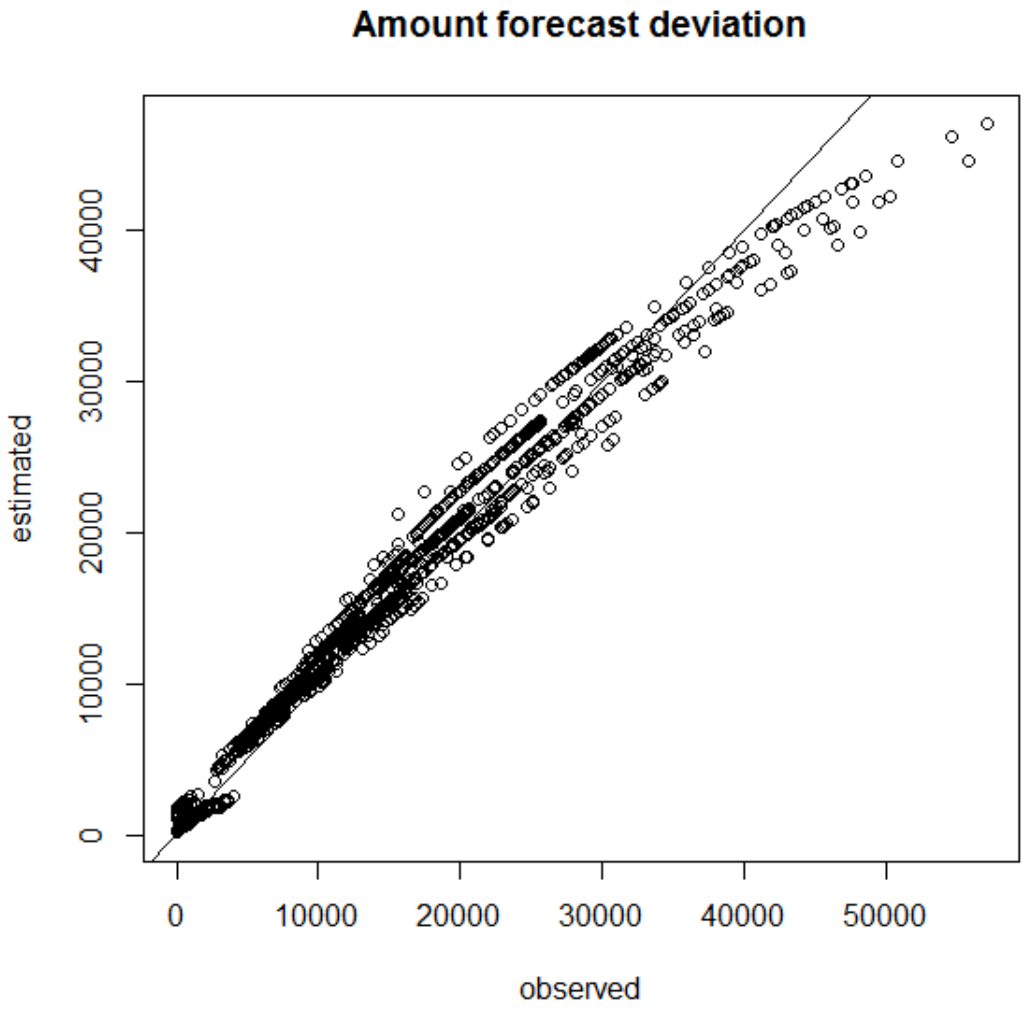

Figure 6.11: myNet 18, not an accurate prediction. It can be seen from the figure how the values predicted by Neural Network 18 differ from the true observations.

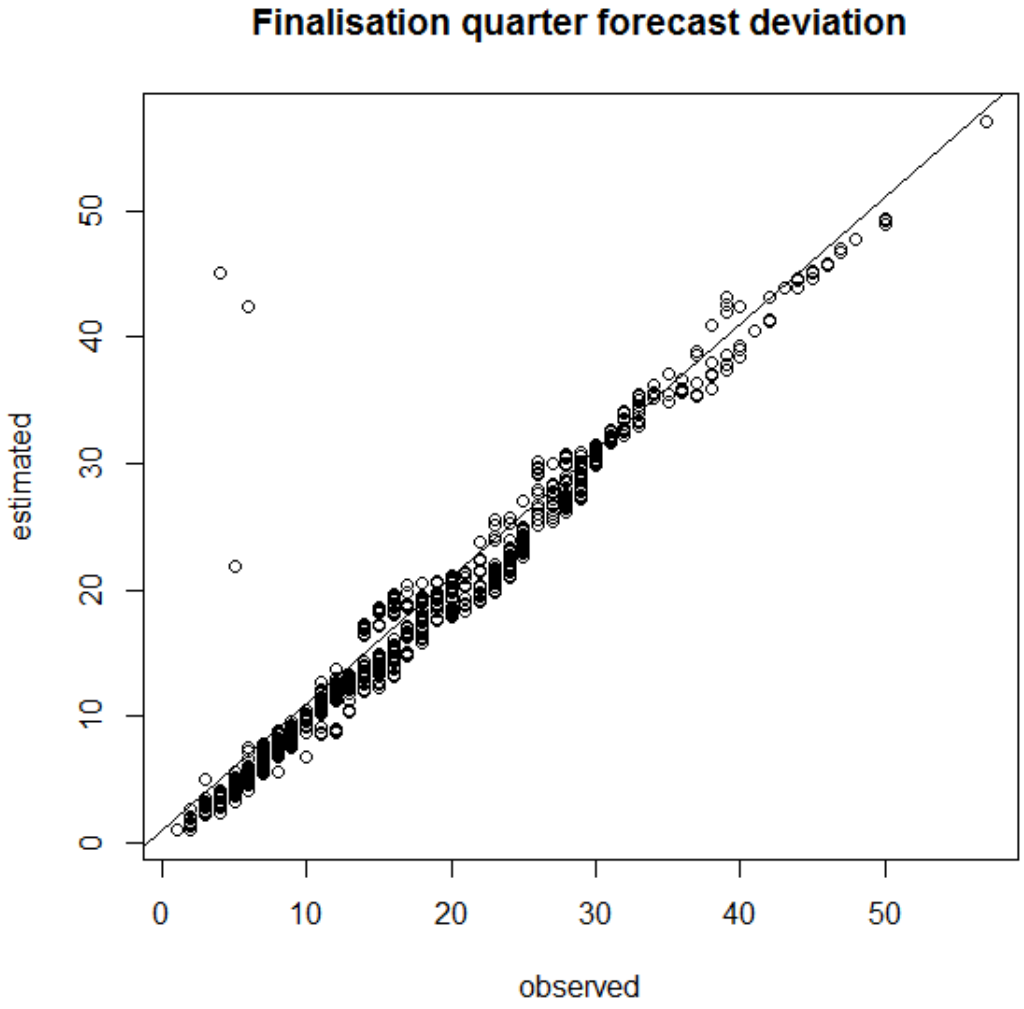

Figure 6.12: The prediction of the figure is good for the finalisation quarter while it not for the claim amount, which is affected by the stochastic component. Indeed it refers to Neural Network 15, which results reported in the table presents the same pattern.

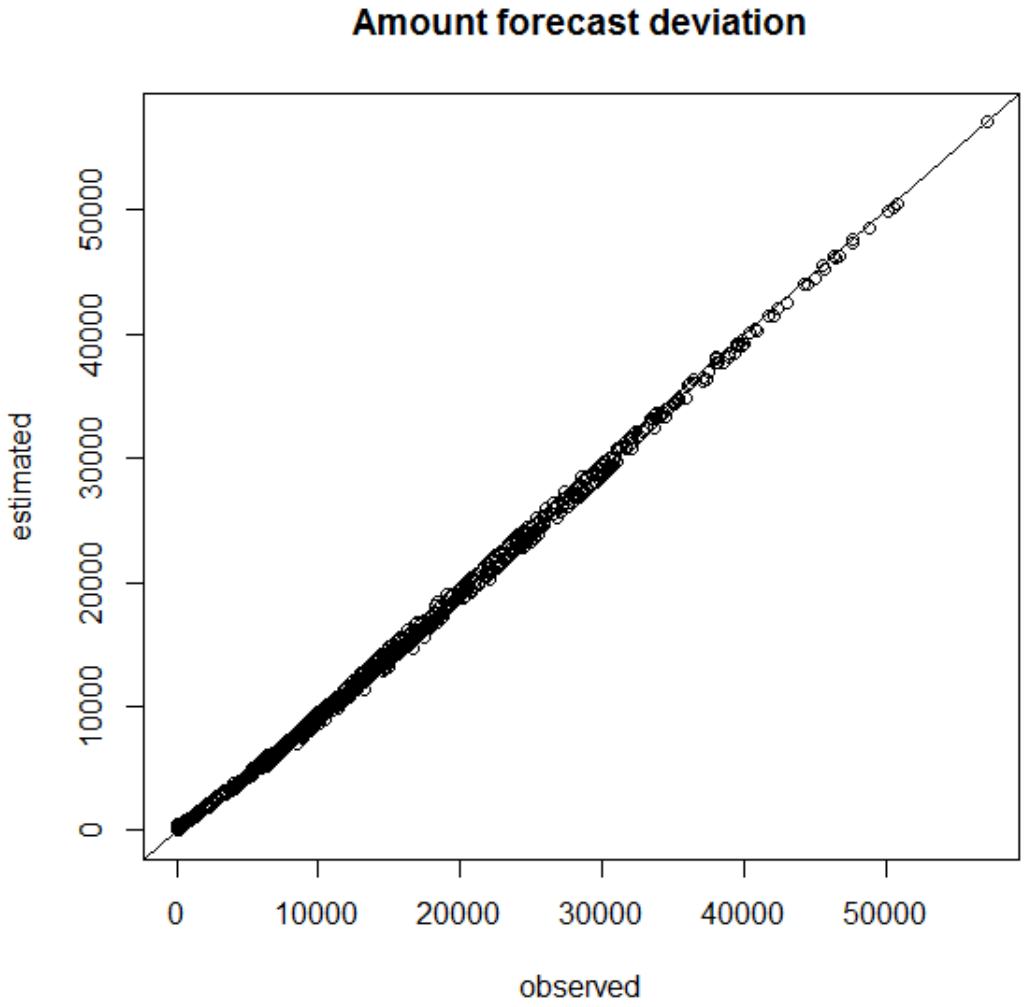

Figure 6.13: The prediction of the figure is good for the finalisation quarter while it not for the claim amount, which is affected by the stochastic component. Indeed it refers to Neural Network 15, which results reported in the table presents the same pattern.

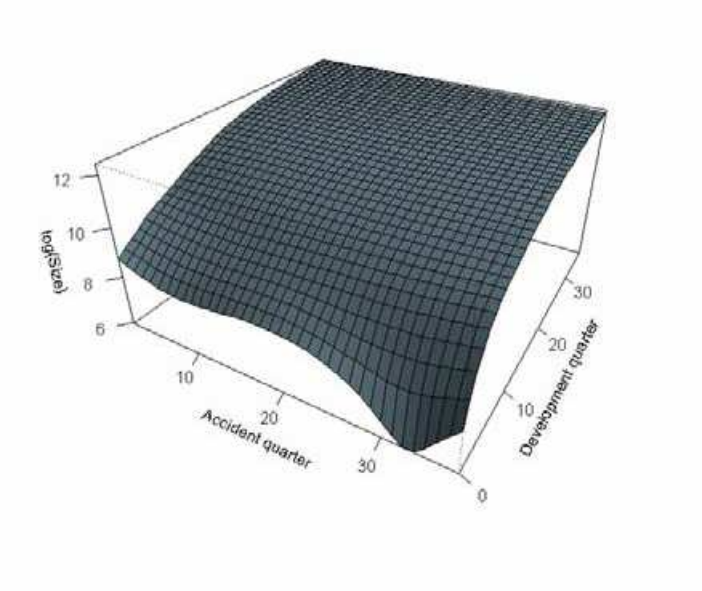

Figure 6.14: A Neural Network fitted with the insurance claims dataset-log(average size of finalised claims)

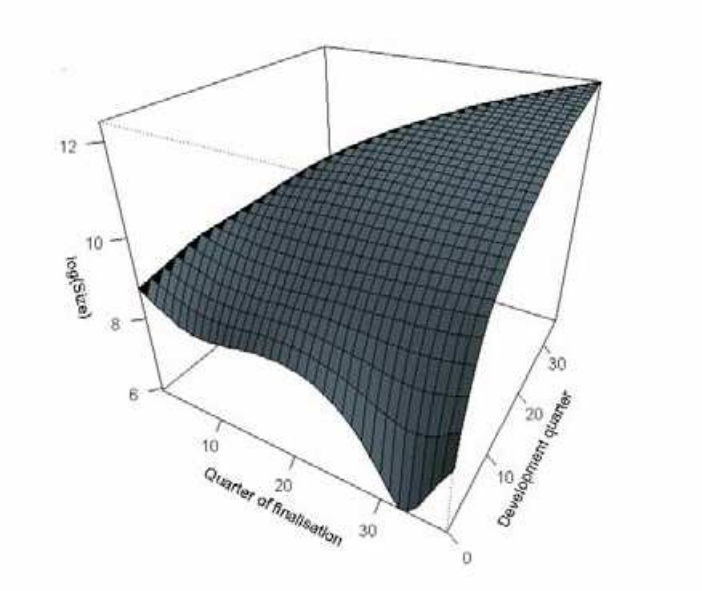

Figure 6.15: A Neural Network fitted with an insurance claims dataset-log(average size of finalised claims)

```
$Triangle
      dev
origin        1        2        3        4        5        6        7         8        9      10
  2001 3474.601 34907.77 72724.53 117272.8 243134.7 419855.5 700992.6 1030815.5 1357235 1646096
  2002 6610.579 19791.14 57010.34 128697.7 273126.5 483620.7 737932.4 1021415.4 1308546 1557483
  2003 8960.830 30646.07 84656.94 171722.0 304237.2 512715.4 757363.6 1081554.0 1345232 1682258
  2004 5602.319 20112.60 66881.37 147221.3 240221.0 448358.9 712041.7  946131.1 1127129 1397052
  2005 8465.441 23863.38 63738.65 137236.0 256666.5 492983.8 740177.6  939734.5 1277964 1589148
  2006 4164.825 25168.75 73568.92 169300.9 327489.5 535610.4 774354.1 1046580.1 1313117 1588460
  2007 7299.046 25645.16 67519.38 174208.7 400764.4 607967.0 813590.6 1020312.0 1368165 1739139
  2008 6539.524 21682.68 86434.45 174492.5 317931.1 489637.7 686908.7  931534.0 1259290 1561609
  2009 8029.132 16544.83 61566.06 126274.5 257344.1 419072.2 663538.5  868566.8 1166961 1407498
  2010 5751.072 28556.94 78942.73 161229.4 283408.5 495074.8 795386.3 1068360.8 1294813 1633054
  2011 6311.553 25437.75 47881.26 138904.4 354860.5 601929.8 913880.1 1140991.3 1370019 1692787
  2012 6437.361 15011.02 56484.96 164244.3 322188.5 488380.3 719854.9  965542.6 1189151       NA
  2013 4272.099 26459.28 63914.65 136076.4 305604.0 479314.2 737381.2  928126.9       NA       NA
  2014 7491.752 29600.44 82167.71 165632.9 341025.4 538499.2 812193.3        NA       NA       NA
  2015 5346.922 38121.04 86032.60 192440.8 352697.6 515163.5       NA        NA       NA       NA
  2016 5704.470 19547.66 86101.78 180439.3 351109.7       NA       NA        NA       NA       NA
  2017 2687.766 19276.58 70379.77 142586.7       NA       NA       NA        NA       NA       NA
  2018 7000.636 20685.23 55380.92       NA       NA       NA       NA        NA       NA       NA
  2019 5224.780 22722.94       NA       NA       NA       NA       NA        NA       NA       NA
  2020 7013.282       NA       NA       NA       NA       NA       NA        NA       NA       NA
      dev
origin      11      12      13      14      15      16      17      18      19      20
  2001 1949649 2238650 2530033 2819182 3083578 3278497 3422852 3509356 3653946 3768631
  2002 1810543 2099069 2394222 2566036 2865008 3137414 3234305 3404839 3550196      NA
  2003 1975924 2200066 2346257 2544682 2714013 3002388 3128348 3350631      NA      NA
  2004 1725349 2022727 2273735 2527994 2729853 2895872 3168536      NA      NA      NA
  2005 1933355 2165872 2394861 2647676 2916327 3161551      NA      NA      NA      NA
  2006 1906688 2114504 2389024 2649218 2834997      NA      NA      NA      NA      NA
  2007 2128795 2386980 2666139 2954070      NA      NA      NA      NA      NA      NA
  2008 1818077 2119897 2470970      NA      NA      NA      NA      NA      NA      NA
  2009 1706361 2072296      NA      NA      NA      NA      NA      NA      NA      NA
  2010 1926880      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2011      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2012      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2013      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2014      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2015      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2016      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2017      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2018      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2019      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
  2020      NA      NA      NA      NA      NA      NA      NA      NA      NA      NA
```

Figure 6.16: Data in triangular form

```
$delta
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

$weights
      dev
origin 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
  2001 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  2002 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA
  2003 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA NA
  2004 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA NA NA
  2005 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA NA NA NA
  2006 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA NA NA NA NA
  2007 1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA NA NA NA NA NA
  2008 1  1  1  1  1  1  1  1  1  1  1  1  1 NA NA NA NA NA NA NA
  2009 1  1  1  1  1  1  1  1  1  1  1  1 NA NA NA NA NA NA NA NA
  2010 1  1  1  1  1  1  1  1  1  1  1 NA NA NA NA NA NA NA NA NA
  2011 1  1  1  1  1  1  1  1  1  1 NA NA NA NA NA NA NA NA NA NA
  2012 1  1  1  1  1  1  1  1  1 NA NA NA NA NA NA NA NA NA NA NA
  2013 1  1  1  1  1  1  1  1 NA NA NA NA NA NA NA NA NA NA NA NA
  2014 1  1  1  1  1  1  1 NA NA NA NA NA NA NA NA NA NA NA NA NA
  2015 1  1  1  1  1  1 NA NA NA NA NA NA NA NA NA NA NA NA NA NA
  2016 1  1  1  1  1 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
  2017 1  1  1  1 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
  2018 1  1  1 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
  2019 1  1 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
  2020 1 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

Figure 6.17: Default weights and delta

```
attr(,"class")
[1] "ChainLadder"   "TriangleModel" "list"
> linkratios <- c(attr(ata(TRIANGLE), "vwtd"), tail = 1)
> round(linkratios, 3)
 <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
4.020 2.860 2.179 1.984 1.643 1.507 1.332 1.275 1.233 1.195 1.145 1.122 1.101 1.088 1.082 1.052
 <NA>  <NA>  <NA>  tail
1.049 1.042 1.031 1.000
> LDF <- rev(cumprod(rev(linkratios)))
> names(LDF) <- colnames(TRIANGLE)
> round(LDF, 3)
 [1] 608.007 151.254  52.888  24.271  12.231   7.442   4.940   3.709   2.909   2.360   1.975
[12]   1.724   1.536   1.396   1.283   1.186   1.127   1.075   1.031   1.000
> currentEval <- getLatestCumulative(TRIANGLE)
> EstdUlt <- currentEval * rev(LDF)
> Exhibit <- data.frame(currentEval, LDF = round(rev(LDF), 3), EstdUlt)
> Exhibit <- rbind(Exhibit,data.frame(currentEval=sum(currentEval), LDF=NA, EstdUlt=sum(EstdUlt)
> Exhibit
       currentEval     LDF  EstdUlt
2001   3768631.238   1.000  3768631
2002   3550195.557   1.031  3661624
2003   3350631.200   1.075  3600715
2004   3168536.272   1.127  3571817
2005   3161551.310   1.186  3749133
2006   2834996.922   1.283  3636064
2007   2954070.195   1.396  4122813
2008   2470969.702   1.536  3796514
2009   2072295.657   1.724  3572610
2010   1926879.945   1.975  3804941
2011   1692787.060   2.360  3994187
2012   1189150.648   2.909  3459640
2013    928126.920   3.709  3442611
2014    812193.296   4.940  4012193
2015    515163.533   7.442  3834038
2016    351109.737  12.231  4294500
2017    142586.651  24.271  3460665
2018     55380.919  52.888  2928961
2019     22722.937 151.254  3436926
2020      7013.282 608.007  4264122
Total 34974992.983      NA 74412705
```

Figure 6.18: Link ratios and estimation results

	OBS	ANN8	err	ANN3	err	ANN9	err	OBS
1	€ 84.616.362	€ 84.717.977	0,120%	€ 83.182.125	-1,693%	€ 88.827.875	4,851%	€ 76.818.727
2	€ 85.709.889	€ 85.918.320	0,243%	€ 84.566.636	-1,331%	€ 91.289.612	6,252%	€ 77.245.456
3	€ 84.566.436	€ 86.501.569	2,288%	€ 85.154.006	0,695%	€ 91.120.028	7,750%	€ 75.717.371

Figure 6.19: Neural Networks trials

	CL			
	no tail	err	tail 1,05	err
1	€ 70.205.242	-8,609%	€ 73.715.504	-4,040%
2	€ 74.412.705	-3,667%	€ 78.133.340	1,149%
3	€ 71.121.141	-6,070%	€ 74.677.199	-1,374%

Figure 6.20: Chain Ladder trials

```
MackChainLadder(Triangle = TRIANGLE, est.sigma = "Mack")

        Latest Dev.To.Date  Ultimate      IBNR  Mack.S.E CV(IBNR)
2001 3,768,631     1.00000 3,768,631         0         0      NaN
2002 3,550,196     0.96957 3,661,624   111,429       243  0.00218
2003 3,350,631     0.93055 3,600,715   250,084     4,511  0.01804
2004 3,168,536     0.88709 3,571,817   403,281    92,096  0.22837
2005 3,161,551     0.84328 3,749,133   587,581   144,525  0.24597
2006 2,834,997     0.77969 3,636,064   801,067   159,330  0.19890
2007 2,954,070     0.71652 4,122,813 1,168,743   187,569  0.16049
2008 2,470,970     0.65085 3,796,514 1,325,545   187,768  0.14165
2009 2,072,296     0.58005 3,572,610 1,500,314   206,268  0.13748
2010 1,926,880     0.50642 3,804,941 1,878,061   244,527  0.13020
2011 1,692,787     0.42381 3,994,187 2,301,400   266,113  0.11563
2012 1,189,151     0.34372 3,459,640 2,270,489   256,729  0.11307
2013   928,127     0.26960 3,442,611 2,514,484   314,143  0.12493
2014   812,193     0.20243 4,012,193 3,199,999   399,195  0.12475
2015   515,164     0.13437 3,834,038 3,318,874   448,421  0.13511
2016   351,110     0.08176 4,294,500 3,943,390   579,852  0.14704
2017   142,587     0.04120 3,460,665 3,318,078   670,779  0.20216
2018    55,381     0.01891 2,928,961 2,873,580   771,124  0.26835
2019    22,723     0.00661 3,436,926 3,414,203 1,193,665  0.34962
2020     7,013     0.00164 4,264,122 4,257,109 2,227,434  0.52323

                  Totals
Latest:     34,974,992.98
Dev:                 0.47
Ultimate:   74,412,704.60
IBNR:       39,437,711.62
Mack S.E.:   3,367,436.85
CV(IBNR):            0.09
```

Figure 6.21: Mack Chain Ladder results

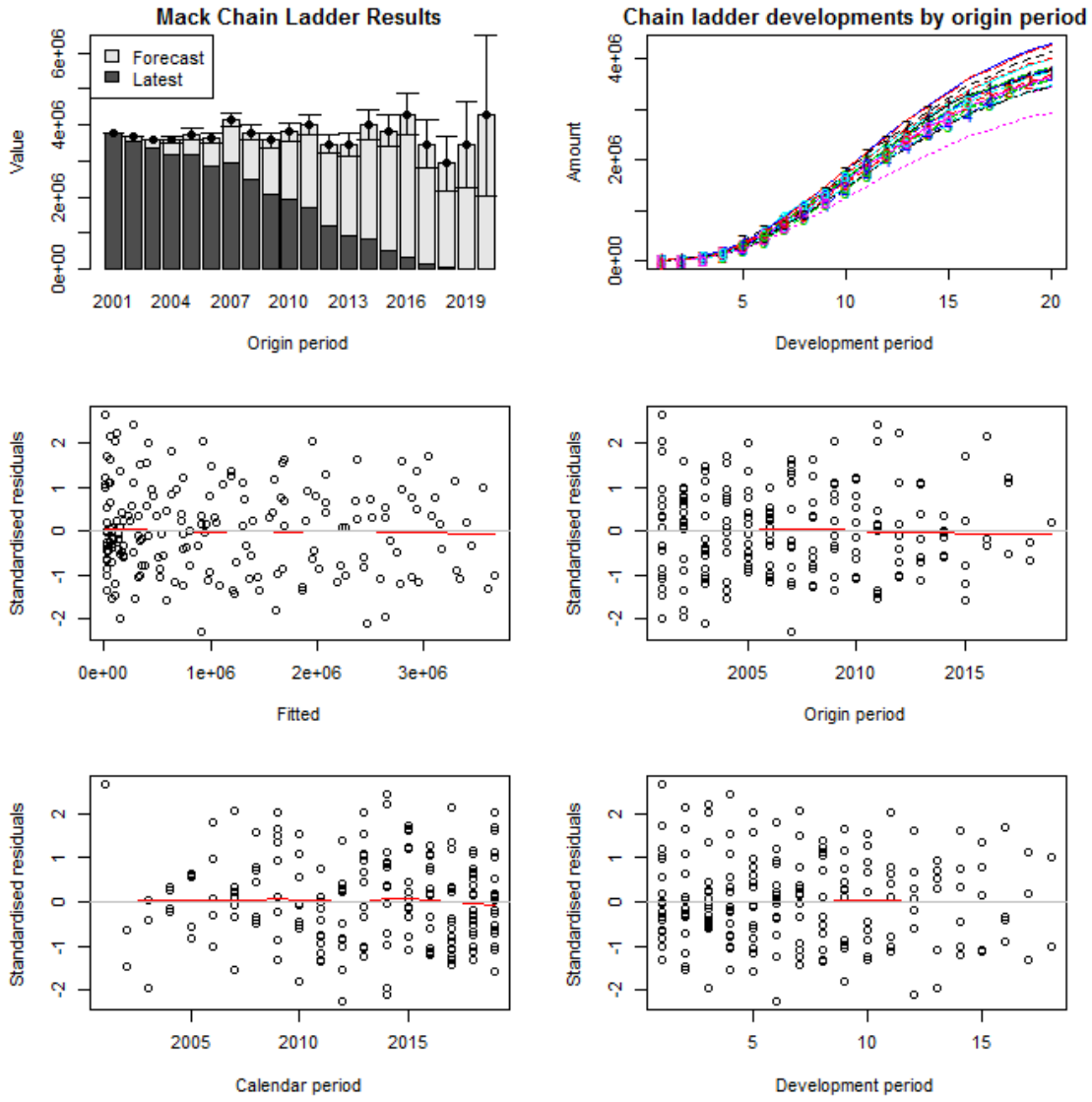

Figure 6.22: Mack Chain Ladder results

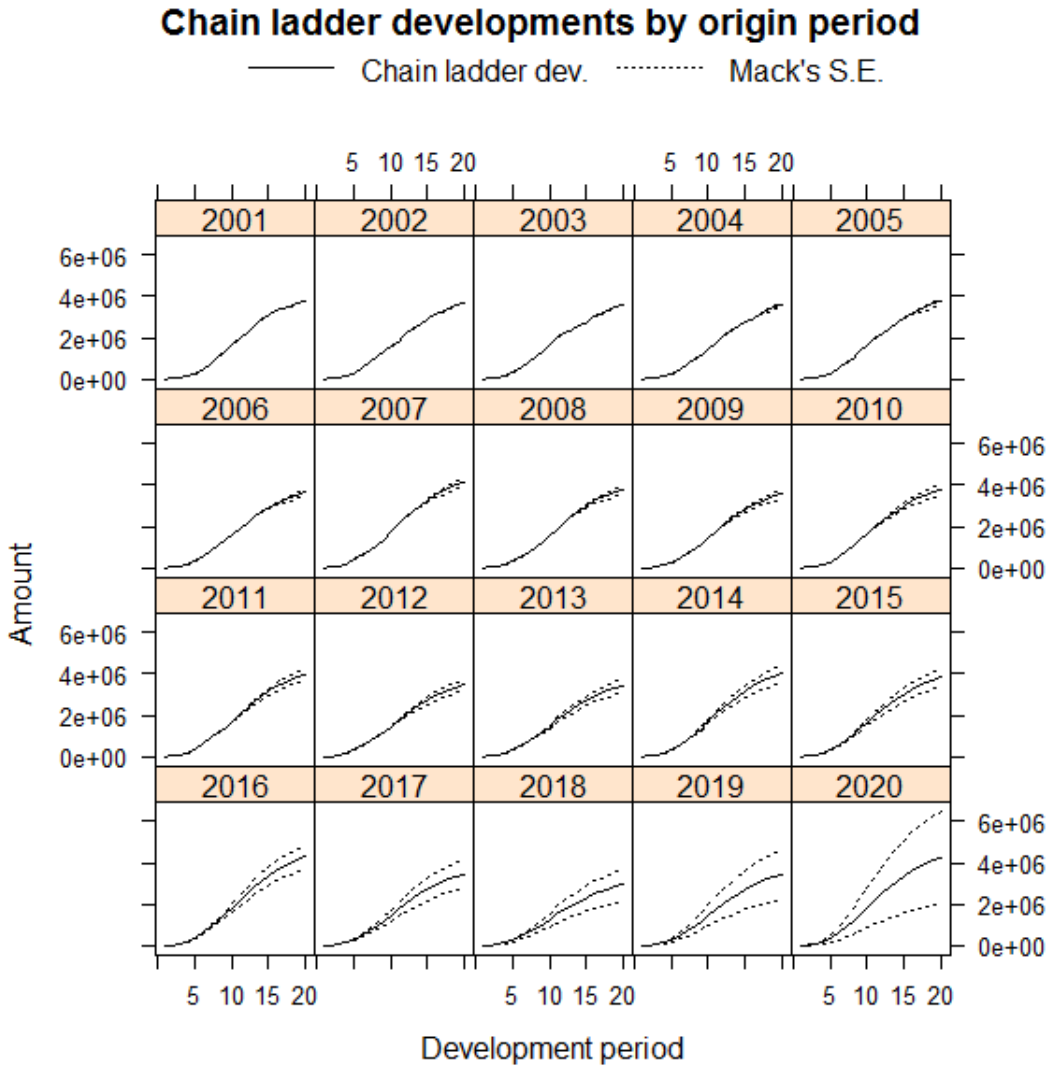

Figure 6.23: Mack Chain Ladder results

Part IV

Conclusions

Chapter 7

Conclusions

When I started this thesis my idea was to apply Soft-Computing techniques to the Insurance business. However, I would have never imagined results were so good.

I created two independent programs:

- an Adaptive Behaviour simulation model;
- a Neural Network algorithm.

Then I applied them to:

- analyse the adaptive behaviour of a cognitive agent in a traffic context;
- make forecasting on loss reserving and claims development by performing a Neural Network regression on the inputs from the Agent-Based simulation;
- compare the prediction accuracy of the neuron-inspired processor against that of a standard Chain Ladder method.

Working on micro-data analysis, as in my innovative approach, demonstrated much more accurate than working on aggregated data, as from the classical approach. We have seen that the new Regulation requires Insurance Companies to involve also a measure of uncertainty around the estimate of the Loss Reserve. In this framework such a technological tool I proposed could be really valuable.

The Agent-Based Model proved its ability in explaining real cognitive decision processes, representing human heterogeneity. Indeed, due to the features generated by its design, the cognitive agent behaves in a manner that perfectly mimic human variety of behaviours. Also, a stochastic components can be included to take part in the simulation model, bringing that complex component of reality through which the phenomena under study evolves. These results confer my model a reliable authority. Notice also that, due to the model's flexibility, the cognitive agent can be placed in every context where cognitive capabilities concur in determining the

results. It follows that by simply changing the context in which we place the cognitive agent, the Adaptive-Behaviour simulation tool can give truthful understandings on policyholders, and more in general on consumers, cognitive behaviours. It is a powerful engine that allows us to study whichever behavioural complex system that evolves following cognitive rules.

Despite its clear interface, the program takes sometimes a user's specific ability in setting the model parameters. Indeed, *unexpected results*, which are desirable and of major interest, always come from a skilled user.

A fundamental aspect of any model is the way it outputs the results. My program exhibits the simulation results in a clear form, providing all the informations and graphical representations that are needed to study the phenomena. Furthermore, the results are of easy interpretation and comprehension to every one.

In what regards the second Model, results were also excellent. The Neural Network that I selected during the training procedure, due to its low sum of squared residuals value and to its high generalisation capability, performed a regression on the simulated claims dataset to approximate the non-linear relation that links the accident's features to the amount and timing of the claim finalisation, providing an accurate forecast. Yet, when the performance of this powerful self learning mechanisms was compared with that of a classical Chain Ladder method, in order to predict the ultimate dataset claims cost, results were embarrassing, the Neural Network brutally outperformed the Chain Ladder. From here it has emerged that, in order to better comply with the Solvency II Regulation, classical reserving methods, such as the chain ladder, need severely to be supported with conventional statistical modelling techniques, such as Generalised Linear Modelling (GLM), and other more sophisticated techniques such as Neural Networks, that allow to identify and to model non-linear relationships almost automatically.

One major disadvantage of this powerful tool, however, is an inability to visualize the models. In fact, Neural Networks are commonly criticized as "black-boxes" that offer little insight into causative relationships among variables. Yet, they require the Network architect to learn how to use them, which is not an easy task.

Advantages, in addition to the more precise estimate of the ultimate claims amount, come from the use of the innovative way of proceed. Indeed, the Neural Network regression does not require to restrict the dataset to a square matrix, providing this way opportunity for assessing also *long tail* businesses. Furthermore, this way doing we overcome the *data quality* problem, as we consider a larger amount of data. Yet, differently from the case of parametric models, Neural Networks does not require to specify, in advance, the functional relationship between dependent and independent variables, and to identify expected interaction effects among independent variables.

I found some difficulties in well understanding the exact steps necessary to build, to train and to evaluate Neural Networks in practice. For this reason, I try in the present work to be the simplest and the most exhaustive as in my abilities. I hope this will be helpful to newcomers.

Chapter 8

New fields of Cognitive Agents application

From the previous considerations and because their ability to model dynamic and heterogeneous systems, I suggest the Agent-Based Model application also in the Insurance field, where they could play a central role in the design, distribution and risk management of insurance products. They promote a more sophisticated understanding and evaluation of product design, pricing, valuation, reserving and hedging.

Either through deterministic approach or stochastic modelling, earlier attempts at modelling the base and dynamic behaviour of policyholders present two major drawbacks. They have been at an aggregate level with little or no differentiation of policyholder behaviour based on different socio-demographic, attitudinal or behavioural factors. Such an aggregate level analysis fails to account for the value that different policyholders place on certain features, and accordingly the aggregate level results are not refined. Moreover, they have assumed a classical rational expectations approach, focusing primarily on the financial drivers while they do not account for how strongly social, cognitive and emotional factors influence consumers' financial decisions.

Agent-based Modelling has proven powerful and efficient for a large number of applications, having recently established in financial computing, which has a powerful means for the analysis and the prediction of financial products and mechanisms such as risk management, trading strategies and derivatives pricing.

The new European Regulation Solvency II calls for new tools that would enable Insurance Companies and Service Providers to model, analyse, and predict the evolution of the Insurance Risk in order to better match and comply with that Regulation. In this context, having the right tools to understand insured behaviour and risk/claims evolution is highly desirable. I remember that, due to the model's flexibility, the cognitive agent model can be placed in every context where cognitive capabilities concur in determining the results.

More in general, Institutions could make use of the proposed Behaviour Simulation Model

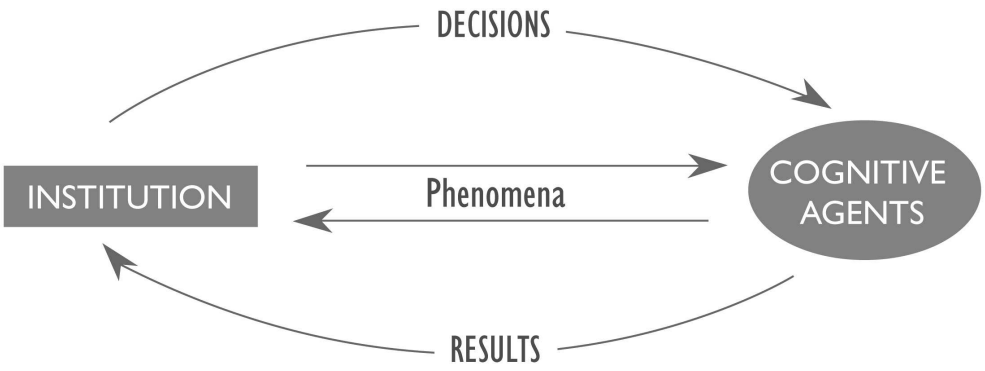

Figure 8.1: Phenomena studying through ABM

whenever they run a decision process whose end affect the cognitive behaviours, in order to reach a more deep understanding of the social phenomena underlying the assumptions on which their decision processes rely. I show this concept in the figure below.

Chapter 9

New fields of Neural Networks application

9.1 The black-box car Insurance application

Recently, black-boxes have entered in the car Insurance business. An insurer installs a box under the bonnet of the car, which contains a global positioning system (GPS) and transmits information back to the Insurance Company via a satellite. Insurers can than monitor the driving performance, and then calculate the premium based on how risky a driver they think that you are. The data is usually shared with the policyholder through a web-based portal that records journey histories and shows your latest "driver score", which is then used to calculate your premium. To calculate your driver score, black box car insurers will monitor things like:

- the time of day that you drive;
- average speeds on different types of roads;
- how many miles you drive;
- acceleration, braking and cornering.

Black-boxes can contribute to construct safer vehicles (Sekhar Reddy, 2009), improving the treatment for crash victims, helping insurance companies with their vehicle crash investigations, and enhancing road status in order to decrease the death rate.

I think that if the Neural Network would be fed with data from those black-boxes, it would be certainly find the same underlying relation between the accident features and the claim amount that I analysed here. Moreover, since we have seen that there exists a relation between the operational time and the claim amount, when an accident happens we could forecast, directly from the black-box data, the claim amount and the development periods. Input data would be,

in such a case, no more seven, as it was the case of my thesis, but many more. This would not represent a matter, as I already said that the only speed limit of signal propagation in hardware-based neural computing system is that of light. The non biological neural computer can add new neural circuits to itself unlimited.

9.2 Neural Networks to Insurance underwriting

Kitchens (2009) suggested that there may be room for improvement in the actuarial and underwriting process. Neural Networks could attempt to predict car Insurance losses using the information available to the underwriters when a policy is accepted. I share the though that Neural Network would more accurately asses the risk level of each individual policy holder, rather than a class of policyholders. Ideally, each insured would contribute a premium equivalent to their contribution to the overall group's risk.

9.3 Understanding Reserving Risk

Introduction

The estimation of future claims liabilities has been an increasing key issue in insurance over the time and many deterministic methods have been developed, in the past, in order to obtain an estimated (point) value of provision for outstanding claims. Recently, under the forthcoming Solvency II and new international accounting principles in insurance, this valuation is nowadays requested to involve also a measure of uncertainty around the (point) estimate of claim reserves. At this regard, stochastic models for outstanding claims valuation have been recently carried on with the aim to assess at least a variability coefficient related to the point estimate of the reserve (i.e. central/best estimate). Bootstrapping methodologies and closed formulae (e.g. Mack formula) are two alternative ways to obtain the prediction error of the Chain-Ladder method. Furthermore the probability distribution of claims reserve could be directly derived by the sampling and the following Monte-Carlo procedure, while Mack formula allow to derive that only under some distributional assumptions (e.g. assuming a log Normal distribution with mean and standard deviation equal to the best estimate of the reserve and the Mack prediction error respectively). Moreover, many actuarial studies are investigating this method in order to propose further developments. Other deterministic methods, as Bornhuetter-Ferguson, has been recently implemented with a stochastic structure based on a Bayesian approach. Moreover closed formulae have been obtained also for this method with the target to measure the prediction error of reserve estimate. Finally, the International Actuarial Association (IAA) suggested a different way to analyse outstanding claim reserve with the aim to quantify the capital requirement for Reserve Risk.

The new European Regulation Solvency II gives evidence on the importance of Risk evaluation. Claim reserving is the process leading to a best estimate of the outstanding claim reserves

on the basis of deterministic methods. Hoverer, under the new Solvency II regime we are not only interested in the estimated amount of claim reserves, but also in the uncertainty and variability around this estimate. Therefore, the need of variability and probability distribution measurement leads to stochastic reserving. We are living a pre-application period in which Companies are still developing their own Risk evaluation method in compliance with the new regulation. From here the relevance of a model which better estimate the risk arising in the Company's reserving procedure. This gives opportunity to test a feed-forward Artificial Neural Network in understanding the risk intrinsic in the phenomena, which is a step-ahead with respect to the action of understanding the risk intrinsic in the model generating the provision (the Chain Ladder) through the Merz-Wuthrich formula. The idea is the following:

- the Actuarial Office evaluate the provisions through some standard methods;
- then it provides the information to the Risk Management, where the risk evaluation is implemented.

Our work comes at this step. Risk is now the more attractive and interesting item of analysis in the Solvency II framework. For this reason this study is aimed at testing the use of the Artificial Neural Networks in understanding the underlying relation between the ultimate claims and the claims dataset, so to better estimate those risks intrinsic in the phenomena object of the Insurance.

Solvency Capital Requirement

On march 2010 EIOPA published the Level 3 Guidance for Solvency II, which includes the pre-application process for Internal Models. Later in 2011 EIOPA published the final Solvency II guidance. Under this framework of pre-application to Supervisors, Companies are still developing alternative methodologies for Reserving Risk calculation, that is the volatility on how you decompose the reserves under a one-year view. More precisely, Pillar II of Solvency II is on capital and risk management, containing qualitative requirements on undertakings such as risk management as well as supervisory activities.

Solvency II gives chance of modelling its main capital requirement, the Solvency Capital Requirement (SCR), making use of two different models:

- the Standard Formula;
- the Internal Model.

A combination of those two can be implemented. The Standard Formula considers all the main quantifiable risks which insurer is exposed to. It consists of implementing a stress test ($x\%$ worst case shocks) applied to each risk, corresponding to the Value-at-Risk of the basic own funds of an insurance or reinsurance undertaking subject to a confidence level of 99.5% over a one-year period:

$$SCR = BSCR - Adj_{FDB} - Adj_{DT} + SCR_{OP} \tag{9.1}$$

where Basic-SCR includes Life, Non-Life, Health, Market and Default risks

$$BSCR = \sqrt{A^2 + B^2 + C^2 + 2\rho_{A,B,C}ABC} \tag{9.2}$$

thus it profits of any eventual diversification effect. Adj_{FDB} expresses the adjustment for risk of future profit sharing. Adj_{DT} reports the adjustment for deferred taxes. SCR_{OP} is the required capital to cover for operational risk. The Internal Model implementation is conditional to the Supervisory approval, and it better reflects the Company's risk profile. Moving from the Standard Formula towards the Internal Model the Company will achieve more sensitivity, at the cost of a loss in simplicity. Solvency Capital Requirement is the target capital the Company would need to ensure solvency with $99,5\%$ probability over one year horizon, i.e. limit the ruin probability to $0,5\%$, i.e. one every two hundred years.

Notice that Solvency II requires a consistency between the Technical Provisions evaluation method and the Risk evaluation method.

Reserving risk

When projecting the balance sheets for solvency we have an opening balance sheet with expected outstanding liabilities. We then project one year forwards, simulating the payments that will emerge in the year. Finally, we require a closing balance sheet with simulated expected outstanding liabilities, conditional on the payments in the year. Under Solvency II a one-year perspective is taken, requiring a distribution of the expected value of the liabilities after one year, for the one-year forthcoming balance sheet in internal capital models. If the standard formula is used, a one year-ahead reserve risk standard deviation % is required. This could be:

- The standard parameter for the line-of-business;
- An undertaking specific parameter.

The one year-ahead Reserve Risk standard deviation is the standard deviation of the distribution of profit or loss on reserves after one year.

The Claims Development Result

The Claim Development Result (CDR) simply is the view of profit or loss on reserves after one year, i.e. an analysis of the run-off result (undiscounted)

$$CDR_1 = R_0 - C_1 - R_1 = U_0 - U_1 \tag{9.3}$$

where R_0 corresponds to the opening reserve estimate, R_1 represents the reserve estimate after one year, C_1 constitutes the payments in the year. It indeed equals the difference between the opening estimate of ultimate claims U_0 and the estimate of the ultimate after one year U_1.

The Merz-Wuthrich method

The attention is paid on the CDR volatility calculation through the Merz-Wuthrich formula. In 2008 they derived analytic formula for the standard deviation of the claims development result after one year assuming:

- the opening reserves were set using the pure Chain Ladder model (no tail);
- claims develop in the year according to the assumptions underlying Mack's model;
- reserves are set after one year using the pure Chain Ladder model (no tail).

Merz-Wuthrich method limitations The Merz-Wuthrich method has huge popularity. However, as from its assumptions, it has some limitations arising whenever:

- we need a tail factor to extrapolate into the future;
- Mack model is not used as underlying assumption;
- a different risk measure is desired other than the standard deviation, for instance the VaR 99,5%;
- the distribution of the CDR is required.

The Bootstrapping technique

Alternatively to such deterministic method the bootstrapping technique can be used. This consists in bootstrapping the initial triangle subsequently calculating a series of stochastic ultimate costs aimed at build a sort of probability distribution of the reserves at time 0 and time 1 in order to calculate the CDR.

We will implement the bootstrapping technique as a better-comprehension intermediate step between the deterministic Merz-Wuthrich and the Artificial Neural Networks.

Artificial Neural Networks

The prospect of the test is that Artificial Neural Networks would provide us the relationship describing a behaviour in the claims development and Loss Reserve setup. This work want to gain through the Neural Network regression the volatility intrinsic in the claim development.

The test procedure

The test procedure consists in:

- consider a dataset of claims paid in triangular form;
- implement the Merz-Wuthrich method, in order for solvency purposes to earn the prediction uncertainty associated with the development of claims reserves;

- implement the Bootstrapping technique, generating n stochastic scenarios to modelling the loss development;
- apply the Neural Networks in order to understand the underlying relation in the dataset;
- make comparisons and conclusions.

Further applications could prove useful. The availability and the relevance of Big data is increasing. Even more complex systems require Agent-based modelling to play a central role for the analysis and comprehension of their dynamics. My intention is to deal with the above features in next future, accepting any suggestions and constructive criticism.

Bibliography

ACI-ISTAT (2012). *Incidenti stradali-comunicato stampa*.

Akinyokun (2002). *Neuro fuzzy expert system for evaluation of human Resource performance*.

Anderson, D. and McNeill, G. (1992). *Artificial neural networks technology*. In «Kaman Sciences Corporation», vol. 258, pp. 13502–462.

Anderson, J. R. (1996). *ACT: A simple theory of complex cognition.*. In «American Psychologist», vol. 51(4), p. 355.

Axelrod, R. and Tesfatsion, L. (2006). *Appendix AA Guide for Newcomers to Agent-Based Modeling in the Social Sciences*. In «Handbook of computational economics», vol. 2, pp. 1647–1659.

Bandini, s., Manzoni, S. and Vizzari, G. (2009). *Agent Based Modeling and Simulation*. In .

Battiti, R. (1992). *First-and second-order methods for learning: between steepest descent and Newton's method*. In «Neural computation», vol. 4(2), pp. 141–166.

Bechara, A., Damasio, A. R., Damasio, H. and Anderson, S. W. (1994). *Insensitivity to future consequences following damage to human prefrontal cortex*. In «Cognition», vol. 50(1), pp. 7–15.

Beltratti, A., Margarita, S. and Terna, P. (1996). *Neural networks for economic and financial modelling*. International Thomson Computer Press London, UK.

Bezdek, J. (1993). *Fuzzy Models - What Are They, and Why?*. In «IEEE Transactions on Fuzzy Systems».

Brackstone, M. and McDonald, M. (1999). *Car-following: a historical review*. In «Transportation Research Part F: Traffic Psychology and Behaviour», vol. 2(4), pp. 181–196.

Broekens, J., Kosters, W. A. and Verbeek, F. J. (2007). *Affect, anticipation, and adaptation: affect-controlled selection of anticipatory simulation in artificial adaptive agents*. In «Adaptive behavior», vol. 15(4), pp. 397–422.

Caballero, A. and Botia, J. (2012). *Hybrid Multi-agent system simulations: cognitive and social agents*. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*. IEEE, pp. 1231–1238.

Castiglione, F. (2009). *Agent Based Modeling and Simulation, Introduction to*. In .

Chen, H., Lee, Y., Sun, G., Lee, H., Maxwell, T. and Giles, C. L. (1986). *High order correlation model for associative memory*. In *Neural networks for computing*, vol. 151. AIP Publishing, pp. 86–99.

Clark, A. and Grush, R. (1999). *Towards a cognitive robotics*. In «Adaptive Behavior», vol. 7(1), pp. 5–16.

Diale, G. (2013). *Lecture Notes*. In .

Dyer, M. G. (1991). *A society of ideas on cognition: Review of Marvin Minsky's The Society of Mind*. In «Artificial Intelligence», vol. 48(3), pp. 321–334.

Eluyode, O. and Akomolafe, D. T. (2013). *Comparative study of biological and artificial neural networks*. In .

Froese, T. (2012). *From adaptive behavior to human cognition: a review of Enaction*. In «Adaptive Behavior», vol. 20(3), pp. 209–221.

Gershenson, C. (2004). *Cognitive paradigms: Which one is the best?*. In «Cognitive Systems Research», vol. 5(2), pp. 135–156.

Gesmann, M., Murphy, D. and Zhang, W. (2014). *Claims reserving with R*.

Ginsberg, M. (1991). *The society of mind: Marvin Minsky*. In «Artificial Intelligence», vol. 48(3), pp. 335–339.

Goldberg, L. R. (1990). *An alternative" description of personality": the big-five factor structure.*. In «Journal of personality and social psychology», vol. 59(6), p. 1216.

Gonzalez, S., Economic, C. and Branch, F. P. (2000). *Neural networks for macroeconomic forecasting: a complementary approach to linear regression models*. Department of Finance Canada.

Grimm, M., Kroschel, K., Harris, H., Nass, C., Schuller, B., Rigoll, G. and Moosmayr, T. (2007). *On the necessity and feasibility of detecting a driver's emotional state while driving*. In *Affective computing and intelligent interaction*. Springer, pp. 126–138.

Hornik, K., Stinchcombe, M. and White, H. (1989). *Multilayer feedforward networks are universal approximators*. In «Neural networks», vol. 2(5), pp. 359–366.

Ingrand, F. F., Georgeff, M. P. and Rao, A. S. (1992). *An architecture for real-time reasoning and system control*. In «IEEE expert», vol. 7(6), pp. 34–44.

Kitchens, F. L. (2009). *Financial implications of artificial Neural Networks in automobile insurance underwriting*. In «International Journal of Electronic Finance», vol. 3(3), pp. 311–319.

Kometani, E. and Sasaki, T. (1959). *A safety index for traffic with linear spacing*. In «Operations Research», vol. 7(6), pp. 704–720.

Laird, J. (2012). *The Soar cognitive architecture*. MIT Press.

Larsen, C. R. (2007). *An individual claims reserving model*. In «Astin Bulletin».

Lee, T.-H., White, H. and Granger, C. W. (1993). *Testing for neglected nonlinearity in time series models: A comparison of neural network methods and alternative tests*. In «Journal of Econometrics», vol. 56(3), pp. 269–290.

Leitner, S. and Wall, F. (2009). *Artificial Economics and Self Organization*. In .

Leu, G., Curtis, N. J. and Abbass, H. (2013). *Society of Mind cognitive agent architecture applied to drivers adapting in a traffic context*. In «Adaptive Behavior», p. 1059712313509652.

McClelland, J. L., Rumelhart, D. E., Group, P. R. *et al.* (1986). *Parallel distributed processing*. In «Explorations in the microstructure of cognition», vol. 2.

Mills, A. (2010). *Complexity Science: An introduction (and invitation) for actuaries*. In «Society Of Actuaries Health Section (Version 1)».

Minsky, M. (1988). *Society of mind*. Simon and Schuster.

Müller, J. P., Pischel, M. and Thiel, M. (1995). *Modeling reactive behaviour in vertically layered agent architectures*. In *Intelligent Agents*. Springer, pp. 261–276.

Mulquiney, P. and Actuaries, T. F. C. (2004). *Application of Soft-Computing Techniques in Accident Compensation*. In *IAA Accident Compensation Seminar*.

Newell, A. (1990). *Unified theories of cognition*. In «Cambridge, MA: Harvard University».

Nicholas Refenes, A., Zapranis, A. and Francis, G. (1994). *Stock performance modeling using neural networks: a comparative study with regression models*. In «Neural Networks», vol. 7(2), pp. 375–388.

Pieragnoli, G. (2013). *Lecture Notes*. In .

Ranjitkar, P., Nakatsuji, T. and Kawamura, A. (2005). *Car-following models: an experiment based benchmarking*. In «Journal of the Eastern Asia Society for Transportation Studies», vol. 6, pp. 1582–1596.

Reeke Jr, G. N. (1991). *The society of mind: Marvin Minsky.* In «Artificial Intelligence», vol. 48(3), pp. 341–348.

Refaeilzadeh, P., Tang, L. and Liu, H. (2009). *Cross-validation.* In , pp. 532–538.

Richiardi, M. (2014). *The Missing Link: AB Models and Dynamic Microsimulation.* In *Artificial Economics and Self Organization*. Springer, pp. 3–15.

Richiardi, M. G. (2012). *Agent-based computational economics: a short introduction.* In «The Knowledge Engineering Review», vol. 27(02), pp. 137–149.

Ripley, B. (2011). *nnet: Feed-forward neural networks and multinomial log-linear models.* In «R package version», vol. 7(5).

Russell, S. and Norvig, P. (1995). *Intelligent agents.* In «Artificial intelligence: A modern approach», pp. 46–47.

Russello, G., Mostarda, L. and Dulay, N. (2011). *A policy-based publish/subscribe middleware for sense-and-react applications.* In «Journal of Systems and Software», vol. 84(4), pp. 638–654.

Sarle, W. S. (1995). *Stopped training and other remedies for overfitting.* In *Proceedings of the 27th Symposium on the Interface of Computing Science and Statisfi (.'. _\'. pp. 352-360. Interface Foundation of North America, Fairfax Station. VA, USA.*

Scheutz, M. and Logan, B. (2001). *Affective vs. deliberative agent control.* In *Proceedings Symposium on Emotion, cognition and affective computing AISB01 Convention, York*. Citeseer.

Sekhar Reddy, M. C. (2009). *BLACK BOX FOR VEHICLES.* In .

Sher, G. I. (2012). *Handbook of neuroevolution through erlang.* Springer.

Sloman, A. (2003). *The cognition and affect project: Architectures, architecture-schemas, and the new science of mind.* Tech. rep., Tech. rep., School of Computer Science, University of Birmingham.

— (2009). *What cognitive scientists need to know about virtual machines.* In *Proceedings of the 31st Annual Conference of the Cognitive Science Society*. pp. 1210–1215.

Smith, K. and Hancock, P. (1995). *Situation awareness is adaptive, externally directed consciousness.* In «Human Factors: The Journal of the Human Factors and Ergonomics Society», vol. 37(1), pp. 137–148.

Smoliar, S. W. (1991). *The society of mind: Marvin Minsky.* In «Artificial Intelligence», vol. 48(3), pp. 349–370.

Stefik, M. J. and Smoliar, S. (1991). *Four reviews of< i> The Society of Mind</i> and a response.* In «Artificial Intelligence», vol. 48(3), pp. 319–320.

Sun, R. (2006). *The CLARION cognitive architecture: Extending cognitive modeling to social simulation.* In «Cognition and multi-agent interaction», pp. 79–99.

Taylor, G. and McGuire, G. (2004). *Loss reserving with GLMs: a case study.*

Tesfatsion, L. and Judd, K. L. (2006). *Agent-based computational economics.* Elsevier.

Thagard, P. (1993). *Societies of minds: Science as distributed computing.* In «Studies in History and Philosophy of Science Part A», vol. 24(1), pp. 49–67.

Ulbinaite, A. and Le Moullec, Y. (2010). *Towards an ABM-based framework for investigating consumer behaviour in the insurance industry.* In «Economics», vol. 89(2), pp. 95–110.

Verrall, R. (1994). *Statistical Methods for the Chain Ladder Technique.* In *Casualty Actuarial Society Forum*, vol. 1. pp. 393–446.

White, H. (1989). *Learning in artificial neural networks: A statistical perspective.* In «Neural computation», vol. 1(4), pp. 425–464.

— (1990). *Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings.* In «Neural networks», vol. 3(5), pp. 535–549.

Wolfram, S. (2002). *A new kind of science*, vol. 5. Wolfram media Champaign.

Wooldridge, M. (2002). *Intelligent agents: The key concepts.* In *Multi-Agent Systems and Applications II*. Springer, pp. 3–43.

Zadeh, S. H., Shouraki, S. B. and Halavati, R. (2006). *Emotional behavior: A resource management approach.* In «Adaptive Behavior», vol. 14(4), pp. 357–380.

Zehnwirth, B. (1994). *Probabilistic development factor models with applications to loss reserve variability, prediction intervals and risk based capital.* In *Casualty Actuarial Society Forum*, vol. 2. pp. 447–606.