TEKNILLINEN KORKEAKOULU
Teknillisen fysiikan ja matematiikan osasto
Teknillisen fysiikan koulutusohjelma

Ella Bingham

Neurofuzzy Traffic Signal Control

Diplomi-insinöörin tutkintoa varten tarkastettavaksi jätetty diplomityö

Työn valvoja professori Harri Ehtamo
Työn ohjaaja tekn.lis. Jarkko Niittymäki

Espoo 1.9.1998

Tekijä: Ella Bingham

Työn nimi: Neurosumea liikennevalo-ohjaus

English title: Neurofuzzy Traffic Signal Control

Päivämäärä: 1.9.1998 **Sivumäärä:** 107

Osasto: Teknillisen fysiikan ja matematiikan osasto

Professuuri: Mat-2 Sovellettu matematiikka

Työn valvoja: Professori Harri Ehtamo

Työn ohjaaja: TkL Jarkko Niittymäki

Työn tavoitteena oli luoda sumea liikennevalojen ohjausjärjestelmä, jonka parametrit mukautuvat ympäröivään liikennetilanteeseen. Olemassaolevaan sumeaan ohjausjärjestelmään lisättiin oppimisalgoritmi, jonka avulla järjestelmän parametreja hienosäädettiin erilaisiin liikennetilanteisiin sopiviksi. Ohjausjärjestelmän tehokkuuden mittarina olivat ajoneuvojen viiveet.

Sumea liikennevalojen ohjausjärjestelmä käyttää kielellisiä sääntöjä kuten "jos saapuva liikennemäärä on *suuri* ja jonottava liikennemäärä on *pieni*, niin vihreä valo on *pitkä*". Sumeat käsitteet *suuri*, *pieni* ja *pitkä* esitetään jäsenyysfunktioiden avulla.

Neuraaliverkot koostuvat yksinkertaisista laskentayksiköistä, jotka on yhdistetty toisiinsa verkoksi. Neurosumeassa liikennevalojen ohjauksessa jäsenyysfunktioiden parametreja hienosäädetään neuraaliverkon avulla. Tässä työssä käytetty neuraaliverkon opetusalgoritmi on nimeltään vahvistava oppiminen (reinforcement learning). Tarkasteltava neurosumea järjestelmä on sellainen, että tavallisimpia neuraaliverkkojen opetusalgoritmeja ei voida käyttää.

Mukautuvaa liikennevalo-ohjausta tutkittiin liikennesimulaattorissa, johon sisältyy sumea valo-ohjausjärjestelmä. Neuraaliverkon opetusalgoritmi toteutettiin Matlab-ohjelmassa, joka vaihtaa tietoja liikennesimulaattorin kanssa.

Opetusalgoritmi toimii menestyksellisesti tilanteissa, joissa liikennemäärä on vakio. Alkuperäiset jäsenyysfunktiot muuttuvat oppimisen myötä erilaisiksi eri liikennemäärillä. Oppimisen tuloksena saadut jäsenyysfunktiot tuottavat pienempiä viiveitä kuin alkuperäiset jäsenyysfunktiot. Sen sijaan opetusalgoritmi ei anna hyviä tuloksia tilanteissa, joissa liikennemäärä muuttuu nopeasti.

Diplomityössä tehtiin lisäksi pieni muutos sumean liikennevalojen ohjausjärjestelmän sääntökantaan. Muutoksen ansiosta ajoneuvojen viiveet pienenivät merkittävästi pienillä liikennemäärillä.

Avainsanat: sumea logiikka, neuraaliverkot, neurosumea, vahvistava oppiminen, liikennevalojen ohjaus

Ei lainata ennen: **Työn sijaintipaikka:**

Author: Ella Bingham

Title of thesis: Neurofuzzy Traffic Signal Control

Finnish title: Neurosumea liikennevalo-ohjaus

Date: 1st September, 1998 **Pages:** 107

Department: Department of Engineering Physics and Mathematics

Chair: Mat-2 Applied Mathematics

Supervisor: Professor Harri Ehtamo

Instructor: Lic.Tech. Jarkko Niittymäki

The aim of this work was to create an adjustable fuzzy traffic signal controller. An existing fuzzy traffic signal controller was enhanced with a learning algorithm. This adjustable controller can modify its parameters in different traffic situations, and thus reach a better control result. The performance of the traffic signal controller is measured by the delay of vehicles.

A fuzzy traffic signal controller uses linguistic rules such as "if the approaching traffic volume is *large* and the queuing traffic volume is *small*, then the green signal is *long*". The fuzzy concepts *large*, *small* and *long* are presented using membership functions.

Neural networks consists of simple processing elements interconnected as a structured network. In a neurofuzzy controller, the parameters of the fuzzy membership functions are adjusted using a neural network. The neural learning algorithm in this work is reinforcement learning. The neurofuzzy system under consideration is such that the most usual neural learning algorithms cannot be used.

The adjustable traffic signal controller is studied in a traffic simulation system which includes a fuzzy signal controller. The neural learning algorithm is realized in a Matlab program which interacts with the traffic simulation system.

The learning algorithm is found successful at constant traffic volumes. Starting from the initial membership functions, the learning algorithm modifies the parameters of the membership functions in different ways at different but constant traffic volumes. The membership functions after the learning produce smaller delays than the initial membership functions. The learning algorithm is not found successful in situations where the traffic volume changes rapidly.

An additional contribution of this thesis is a small manual modification in the rule base of the fuzzy traffic signal controller. This modification reduces the delays significantly at low traffic volumes.

Keywords: fuzzy logic, neural networks, neurofuzzy systems, reinforcement learning, traffic signal control

Not borrowable till: **Library code:**

Preface

I warmly thank Prof. Harri Ehtamo for his supervision and the encouragement he has given me. My instructor, Lic.Tech. Jarkko Niittymäki and Prof. Matti Pursula have provided me with an interesting topic and a good working environment, for which I am most grateful.

I wish to thank my colleagues at the HUT Transportation Laboratory. I am particularly indebted to Lic.Tech. Iisakki Kosonen and Mr. Mikko Lehmuskoski for helping me with the traffic simulation system, and to M.Sc. Jari Kurri for reviewing and discussing my thesis. I also thank Dr. Esko Turunen for the comments he gave on my manuscript.

My husband Kenrick deserves special mention for supporting me during this process.

Otaniemi, 1st September 1998

Ella Bingham

Contents

Chapter 1

Introduction

1.1 Problem Setting

Fuzzy logic, among other new methods, is becoming popular in traffic signal control. Most of the fuzzy traffic signal controllers are not adjustable, that is, the parameters of the fuzzy controller remain the same in changing traffic situations. In this thesis, an existing fuzzy traffic signal controller is enhanced with a neural learning scheme. The resulting adjustable controller can modify its parameters in different traffic environments. It is examined whether this adjustable controller performs better than the initial non-adjustable controller. The performance measure of the controller is the delay of the vehicles in the system.

A fuzzy control system uses a rule base of simple "if-then" rules to calculate the control action based on the current traffic situation. For example, "if the approaching traffic is *large* and the queuing traffic is *small*, then the green signal is *long*". The concepts *large*, *small* and *long* are fuzzy sets. That is, they are not precise, and elements belonging to one set may partially belong to some other set, too. For example, a measurement of 5 vehicles is *small* to some degree and also *large* to some other degree.

Neural networks form a wide class of nonlinear regression models and dynamical systems. They consist of a large number of simple computing elements, neurons, which are interconnected and organized in layers [61]. The network is able to "learn" or adapt to the data observed by changing the strengths of the network interconnections. The most widely used neural learning algorithms are based on the output error, that is, the difference between the output of the neural network and a "desired" output. In this sense, neural networks are surface-fitting algorithms quite similar to regression models.

Combinations of neural and fuzzy systems have become popular in the last few years. A fuzzy system can be represented in the form of a neural network, resulting in a *neurofuzzy* system. The parameters of the neurofuzzy system can be updated using the same methods as in neural learning.

Applying neural learning to fuzzy traffic signal control involves problems that are not encountered in most neurofuzzy control systems. The output of the signal controller

is the length of the green signal, and the "desired" length of the green signal is not known. Thus the learning cannot be based on the difference between the output of the network and the desired output, and the most usual neural learning algorithms cannot be used. The objective of the signal controller is to minimize the delay and not to reach a "desired" length of the green signal — actually, the desired length would be the one which minimizes the delay! In this thesis, a learning algorithm called *reinforcement learning* is used in neurofuzzy traffic signal control. This type of learning uses a criterion other than the difference of the network output and the desired output; in this thesis, the criterion is the delay of vehicles.

Enhancing traffic signal control has a potential of significant savings in national economy, if the time spent in traffic is reduced. Modern traffic signal controllers base the signal timings on the information about the current traffic situation they receive through traffic detectors. A suitable mathematical representation of traffic signal control is not easy to find, as there are many variables and restrictions. The complexity of the control procedure is sometimes restricted by the limited capacity of the actual signal controller; on the other hand, safety regulations prevent the use of some control solutions. That is why classical mathematical optimization in the actual sense of the word is not always used, but simpler sub-optimal procedures are sought.

Fuzzy control is suitable for traffic signal control because expert knowledge can be easily exploited with linguistic concepts and rules. A fuzzy signal controller is easier to design and maintain than a traditional one. Fuzzy signal control is also simple in the sense that the basic calculations are quite easy and the number of parameters is smaller.

In this thesis, the word "adjustable" is used to refer to a control system whose parameters are modified using a neural or some other learning method. The word "adaptive" is often used in literature in place of "adjustable". The word "adjustable" is chosen here because in traffic control engineering, the word "adaptive" refers to a traffic signal controller which gives a different control output in different situations, but whose parameters are not necessarily modified. The word "adaptive" is used in this thesis only in the literature review where the interpretation of the word is obvious.

1.2 Aims of the Thesis

The main objective of the thesis is to create an adjustable fuzzy control system for traffic signal control. The fuzzy traffic signal controller is already available, and the current objective is to make it adjustable. The adjustable fuzzy controller should perform better than the non-adjustable controller, as it can change its parameters as the surrounding traffic situation changes. In practice, the fuzzy controller is tuned by updating the parameters of the fuzzy membership functions. The objective is to construct a neural learning algorithm which modifies the parameters. It is examined whether the algorithm modifies the parameters in a different way at different (but constant) traffic volumes. It is also tested whether the algorithm can keep up with a changing traffic volume and update its parameters continuously.

If different membership function parameters are found for different traffic volumes, the traffic signal controller can first identify the traffic situation and then choose a proper parameter set for the fuzzy control process. On the other hand, if the algorithm is able to keep up with a changing traffic volume, it is possible to use the algorithm in the field: the signal controller updates its parameters all the time as the traffic volume changes. The practical implementation of the algorithm on a traffic signal controller in the field is not aimed to discuss in this thesis.

Fuzzy traffic signal control has earlier [46] been superior to traditional traffic signal control only at large traffic volumes. It is hoped that fuzzy signal control could achieve better results at low traffic volumes, too.

The location of the traffic detector determines the number of vehicles the fuzzy controller can observe, and therefore affects the output of the controller. It is compared whether the learning algorithm results in different kinds of membership functions at different detector locations.

An additional aim is to study neurofuzzy systems in terms of a literature review and discuss their applicability to traffic signal control.

1.3 Structure of the Thesis

A literature review covering fuzzy systems, neural networks, neurofuzzy systems and reinforcement learning, respectively, is presented in Chapters 2, 3, 4 and 5.

Chapter 6 discusses traffic signal control and the application of the reinforcement learning algorithm (presented in Section 5.3) to traffic signal control.

The results of the learning are presented and discussed in Chapter 7 together with results of some other experiments. Sections 8.1 and 8.2 also discuss possible reasons for the results obtained.

Suggestions for future research and the development of the simulation system are given in Section 8.3.

1.4 Contributions of the Thesis

In the literature review, the nature of different union, intersection and defuzzification operations on fuzzy sets are discussed in Sections 2.3 and 2.5. The choice of these operations affects the learning in the fuzzy traffic signal controller. The drawbacks and advantages of these operations in the context of the reinforcement learning algorithm are considered in detail in Section 5.3.7. In this section, suggestions are given for the choice of the operations, based on theoretical considerations.

An existing fuzzy traffic signal controller is made adjustable by enhancing the system with a learning algorithm. The algorithm found in the literature is interpreted and applied in the context of traffic signal control. The practical limitations in the application of the algorithm are discussed and some of the limitations are overcome, as discussed in Section 6.2.3.

A large series of experiments on the adjustable neurofuzzy traffic signal controller is done. The experiments are realized on the traffic simulation system in which fuzzy control is already implemented. The simulation system interacts with a Matlab [43] program especially designed for this purpose. The membership function parameters of the fuzzy controller are modified using the reinforcement learning algorithm in several different cases, covering different traffic detector locations and both constant and increasing traffic volumes.

The learning algorithm is found successful at constant traffic volumes. New membership function parameters are found in four different cases out of a total of six different cases of constant traffic volumes. These new parameters produce a smaller vehicular delay than the initial parameters. The algorithm modifies the parameters in different ways in different traffic situations. This means that the set of membership functions should be chosen according to the traffic volume and the location of the traffic detectors.

The learning algorithm is not found successful at increasing traffic volumes. The algorithm cannot keep up with a changing traffic, at least if the change in the traffic volume is fast.

In addition, the rule base itself is slightly modified by manually changing some of the conditions in the if-parts in the rules. This results in a substantial decrease in the vehicular delay at low traffic volumes, and the aim of reducing delays especially at low traffic volumes is thus fulfilled.

Chapter 2

Fuzzy Systems

2.1 Introduction

The concepts and terminology of fuzzy logic were brought to public attention by Lotfi A. Zadeh in 1965 [68]. In his article he introduced fuzzy sets and membership functions and extended several definitions on set-theoretic operations on fuzzy sets. Nonetheless, the underlying ideas of imprecise reasoning date back to logicians and philosophers of the late nineteenth and early twentieth century: for example, to Charles Sanders Peirce, Bertrand Russell and Jan Lukasiewicz. For an overview of the history of fuzzy or vague logic, refer to Kosko [33].

In two-valued or bivalent logic, truth value of a statement is either 0 ("false") or 1 ("true"). In multi-valued logic, truth values of statements may assume more values, for example, 0, $\frac{1}{2}$ and 1. In fuzzy logic, a truth value can assume any real value between 0 and 1. In a narrow sense, fuzzy logic is multi-valued logic. In broader sense, fuzzy logic is an extension of multi-valued logic and refers to various methods modelling vague concepts.

Fuzzy sets provide a specific, mathematical interpretation for vague, natural language terms [7]. A fuzzy set is a generalization of a classical or *crisp* set in the sense that a fuzzy set may contain its elements partially, too, whereas an element x of a crisp set either belongs to the set or does not. The *characteristic function*

$$\chi_A : U \to \{0, 1\} \tag{2.1}$$

of a crisp set A maps the elements x of a given *universal set* U to 0 or 1 and is defined as

$$\chi_A(x) = \begin{cases} 0, & x \notin A \\ 1, & x \in A. \end{cases} \tag{2.2}$$

Thus the characteristic function of a crisp set discriminates between members and non-members of the crisp set $A \subset U$ [27]. In a fuzzy set S, defined as

$$S = \{(x, \mu_S(x)) \mid x \in U\}, \tag{2.3}$$

each element $x \in U$ is assigned with a *degree of membership* in S, which is measured by a *membership function* [68]

$$\mu_S : U \to [0, 1]. \tag{2.4}$$

U is always a crisp set, and here the attention is restricted to $U \subset \mathbb{R}$. The membership function $\mu_S(x)$ is zero when x does not belong to S at all, one when x belongs to S totally and $0 < \mu_S(x) < 1$ when x belongs to S partially. Zadeh [68] notes that in a more general setting, the range of the membership function can be a suitable partially ordered set, but it is convenient and sufficient to restrict the range to the unit interval $[0, 1]$.

The degree of vagueness in a fuzzy set S is precisely represented by μ_S. Note the word *precisely*: the membership function actually defines the impreciseness inherent in S. The more the membership function μ_S resembles the characteristic function (2.2), the less fuzzy S is. Similarly, the set S is considered the fuzzier the more $\mu_S(x)$ differs from 0 or 1.

Various forms of membership functions are discussed in Section 2.2.2.

Fuzzy inference deals with propositions "if X is S, then Y is T". Here X and Y are *linguistic variables* and S and T are *linguistic values*. A linguistic variable is a variable which assumes linguistic values. For example, "age" can be a linguistic variable which assumes linguistic values "young", "middle-aged" and "old". Each of these linguistic values can be represented as a fuzzy set which has its own membership function μ. A person may partially belong to one or several of these sets at the same time: a 25-year-old is still young to some degree but also middle-aged to some degree, and so on. Splitting the universe of "age" into non-overlapping sections, that is, into crisp sets, would be both difficult and impractical.

The main idea underlying fuzzy reasoning is that many everyday concepts are vague and imprecise. For example, a phrase such as "The sky is blue" may be true to only some extent. As the sky turns more and more grey, the truth value of this sentence decreases, but it is hard to find a specific instant when the colour ceases to be blue. Also, sometimes it is unnecessary to use precise statements. Bedzek [4] gave a nice example on this: "Suppose, as you approach a red light, you must advise a driving student when to apply the brakes. Would you say 'Begin braking 74 feet from the crosswalk'? Or would your advice be more like 'Apply the brakes pretty soon'? The latter, of course; the former instruction is too precise to be implemented."

The literature on fuzziness is wide. See for example Jurva [21], Klir and Yuan [27], Kosko [33], Lee [39], Mattila [44], Niskanen [48], Orlovski [51], Turunen [65], Yager and Filev [67] or Zimmermann [69]. Unfortunately, many authors are over-optimistic about the possibilities of fuzzy logic. Although the philosophy behind fuzzy systems is "novel" and its application in computation differs from traditional computing systems, it is not self-evident that fuzzy systems yield better results in what they are applied to. For example, the choice of a control system (fuzzy, neural, neurofuzzy or mathematical optimization *etc.*) must be based on the *performance* of the system and not just the philosophy that lies behind it.

2.2 Membership Functions

2.2.1 Definitions

The fuzzy set S (or, equivalently, the membership function μ_S) is *normal* if there exists $x \in U$ such that $\mu_S(x) = 1$. In the opposite case, S (or μ_S) is *subnormal*. In this thesis it is assumed that all fuzzy sets are normal.

The *support* of S (or, of μ_S) is the set of x at which μ_S exceeds zero:

$$\text{supp}(S) = \{x \,|\, \mu_S(x) > 0,\, x \in U\}. \tag{2.5}$$

The *core* of S (or, of μ_S) is the set of x at which μ_S reaches a value of 1:

$$\text{core}(S) = \{x \,|\, \mu_S(x) = 1,\, x \in U\}. \tag{2.6}$$

The fuzzy set S (or, μ_S) is *unimodal* if the core of S consists of one point x only.

2.2.2 Types of Membership Functions

The most frequently used types of membership functions are triangular, trapezoidal, Gaussian, generalized bell and sigmoidal membership functions.

The *triangular* membership function

$$\mu(x;\, p_1, p_2, p_3) = \begin{cases} \frac{x-p_1}{p_2-p_1}, & x \in [p_1, p_2] \\ -\frac{x-p_3}{p_3-p_2}, & x \in (p_2, p_3] \\ 0, & \text{else} \end{cases} \tag{2.7}$$

or, equivalently, [20]

$$\mu(x; p_1, p_2, p_3) = \max\left(\min\left(\frac{x-p_1}{p_1-p_2}, \frac{p_3-x}{p_3-p_2}\right), 0\right), \tag{2.8}$$

is seen in Figure 2.1 a. The parameters $[p_1, p_2, p_3]$ determine the x coordinates of the three corners.

The *trapezoidal* membership function

$$\mu(x;\, p_1, p_2, p_3, p_4) = \begin{cases} \frac{x-p_1}{p_2-p_1}, & x \in [p_1, p_2] \\ 1, & x \in (p_2, p_3] \\ -\frac{x-p_4}{p_4-p_3}, & x \in (p_3, p_4] \\ 0, & \text{else} \end{cases} \tag{2.9}$$

or, equivalently, [20]

$$\mu(x;\, p_1, p_2, p_3, p_4) = \max\left(\min\left(\frac{x-p_1}{p_1-p_2}, 1, \frac{p_4-x}{p_4-p_3}\right), 0\right), \tag{2.10}$$

is seen in Figure 2.1 b. The parameters $[p_1, p_2, p_3, p_4]$ determine the x coordinates of the four corners. A special case of a trapezoidal membership function is a rectangular membership function in which $p_1 = p_2$ and $p_3 = p_4$. It corresponds to the

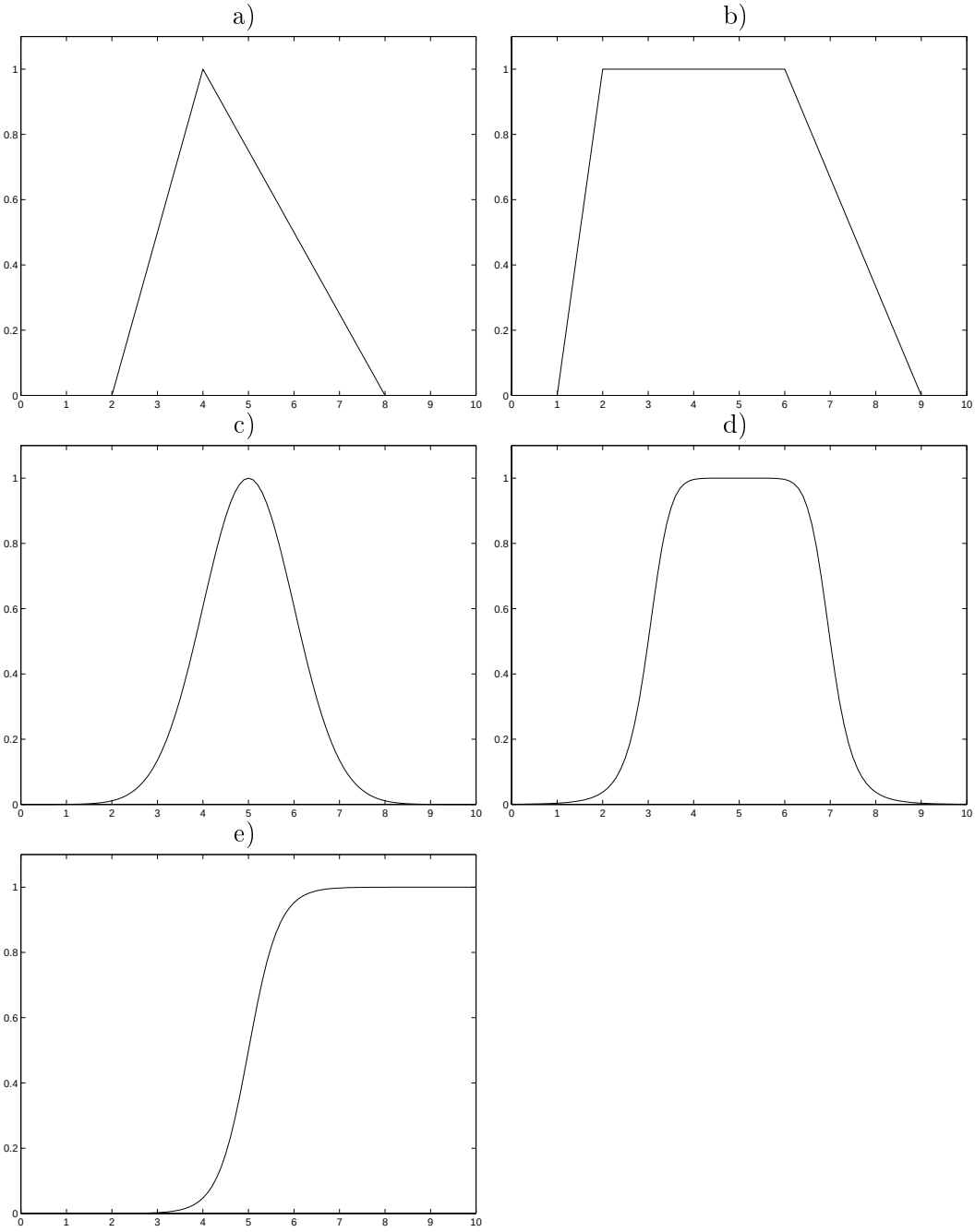


Figure 2.1: Membership functions and their parameters. a) Triangular membership function ($p_1 = 2$, $p_2 = 4$, $p_3 = 8$). b) Trapezoidal membership function ($p_1 = 1$, $p_2 = 2$, $p_3 = 6$, $p_4 = 9$). c) Gaussian membership function ($c = 5$, $\sigma = 1$). d) Generalized bell membership function ($a = 2$, $b = 4$, $c = 5$). e) Sigmoidal membership function ($a = 3$, $c = 5$).

characteristic function χ in Formula (2.2): the value of the membership function is 1 for those elements which belong to the set and 0 for the others. In other words, a rectangular membership function is a membership function of crisp concepts.

Another special case of a trapezoidal membership function is a triangular membership function. A trapezoid is more flexible than a triangle because it has one parameter more. Both membership functions are popular due to their computational simplicity. However, as triangular and trapezoidal membership functions are composed of straight line segments, they are not smooth at the switching points $[p_1, p_2, p_3, p_4]$ [20]. Nonlinear and therefore less simple Gaussian, generalized bell and sigmoidal membership functions are smooth.

The *Gaussian* membership function [7]

$$\mu(x;\, c, \sigma) = e^{-\frac{(x-c)^2}{2\sigma^2}} \tag{2.11}$$

is seen in Figure 2.1 c. The parameters $[c, \sigma]$ determine the center and width of the membership function.

The *generalized bell* membership function [20]

$$\mu(x;\, a, b, c) = \frac{1}{1 + |\frac{x-c}{a}|^{2b}} \tag{2.12}$$

is seen in Figure 2.1 d. The parameters $[a, b, c]$ control the width, slope and center of the membership function. Jang and Sun [20] note that this membership function is a direct generalization of the Cauchy distribution used in probability theory.

The *sigmoidal* or *logistic* membership function [20]

$$\mu(x;\, a, c) = \frac{1}{1 + e^{-a(x-c)}} \tag{2.13}$$

is seen in Figure 2.1 e. Parameters $[a, c]$ control the slope and x coordinate of the cross-over point $x = c$. The sigmoidal membership function differs from above-mentioned membership functions in that it is monotone. Depending on the sign of a, a sigmoidal function is open left or right and is therefore appropriate for representing concepts such as *very large* or *very negative* [20].

2.2.3 Construction of Membership Functions

The membership functions representing linguistic values of a linguistic variable should describe the nature and properties of the linguistic variable. Methods of constructing membership functions can be divided into *direct* and *indirect* methods [27]. They both utilize expert knowledge of the linguistic variables and linguistic values in question.

In direct methods, the expert or experts assign a membership degree $\mu_S(x)$ of a linguistic value S for $x \in U$. The expert may either give a mathematical formula of a suitable membership function or, more often, give examples of the values $\mu_S(x)$ for some x. In this case, the expert answers questions such as "What is the degree of membership of x in S" or "Which elements x have the degree of membership $\mu_S(x)$

in S?" These questions result in a set of pairs $\{x, \mu_S(x)\}$ which are used for constructing the membership function using some curve-fitting method [27]. Sometimes the type — one of those discussed in Section 2.2.2 — of the membership function is decided in advance, and the pairs $\{x, \mu_S(x)\}$ are used to select the parameters of the membership function.

In indirect methods, the expert makes pairwise comparisons between elements x_1, x_2, ... , x_n of the universal set U with respect to how much they belong in S. Sometimes it is easier to compare the degrees to which elements belong to S than give the actual degree of membership for each element as in direct methods. The result of the comparisons is a matrix whose element $p_{ij} = \mu_S(x_i)/\mu_S(x_j)$ specifies how much more x_i belongs to S than x_j. For example, if p_{ij} is 2, then x_i belongs to S twice as much as x_j does. The expert does not have to give actual values of $\mu_S(x_i)$ or $\mu_S(x_j)$. The actual values of the membership function are calculated by finding the eigenvalues and eigenvectors of the matrix [27]. Again we result in pairs $\{x, \mu_S(x)\}$ which are used to construct the membership function as a whole.

In both direct and indirect methods, the answers of a pool of experts must be combined in some way. Klir and Yuan [27] discuss these issues.

2.3 Operations on Fuzzy Sets

Well known operations on fuzzy sets are the complement, intersection and union operations. They all have several mathematical interpretations, some of which are discussed in the following. More operations are presented in Klir and Yuan [27], in Yager and Filev [67] and in Zimmermann [69].

Complement. The complement S^c of a fuzzy set S has a membership function

$$\mu_{S^c}(x) = 1 - \mu_S(x). \tag{2.14}$$

This definition is equivalent to the complement of a classical set. Klir and Yuan [27] note that $\mu_{S^c}(x)$ may be interpreted not only as the degree to which x belongs to S^c, but also as the degree to which x *does not belong to* S. Similarly, $\mu_S(x)$ may also be interpreted as the degree to which x does not belong to S^c.

In Section 2.1 the degree of fuzziness in set S was based on the membership function μ_S. The concept of complement gives us a precise way of measuring fuzziness. Kosko [32] notes that a set S is fuzzy if and only if the intersection of a set S and its complement S^c is nonempty. For crisp sets, the intersection is empty. In terms of membership functions, a set S is fuzzy if the minimum of $\mu_S(x)$ and $\mu_{S^c}(x)$ is nonzero on all x. Using Formula (2.14), the minimum of $\mu_S(x)$ and $1 - \mu_S(x)$ must be nonzero if S is fuzzy.

Above the intersection of fuzzy sets was interpreted as the minimum of membership function values. In the following, the intersection is further discussed.

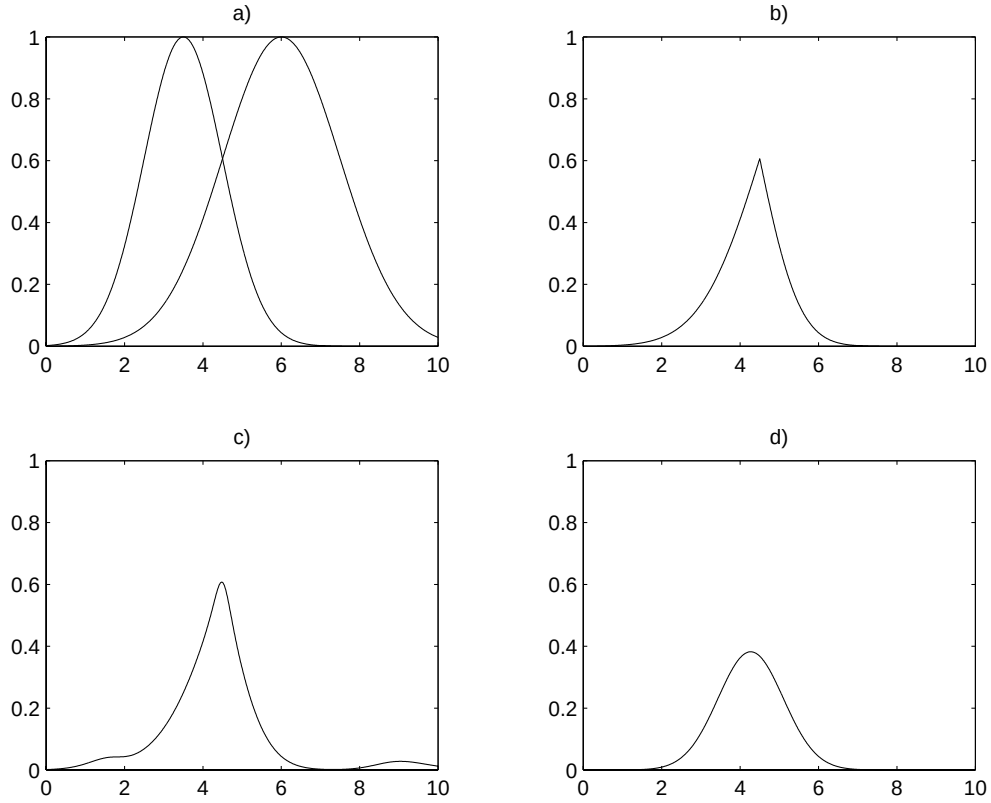


Figure 2.2: Intersection operators. a) Two Gaussian membership functions μ_1 and μ_2 of a linguistic variable S. b) Minimum of μ_1 and μ_2. c) Soft minimum of μ_1 and μ_2 with $k = 10$. d) Product of μ_1 and μ_2.

Intersection Operators. There are various mathematical formulae for the intersection of fuzzy sets, some of which are compared in Figure 2.2 and in the following. (Note that if the membership functions are plotted on the same axis as in Figure 2.2, they all must be functions of the *same* linguistic variable S. The formulae presented here are directly applicable also to membership functions of *different* linguistic variables S_1, S_2, ... , S_n.)

Using the traditional fuzzy intersection operator, *minimum* [68], the intersection $S = \bigcap_{i=1}^{n} S_i$ of fuzzy sets S_i, $i = 1, \ldots, n$, has a membership function

$$\mu_S = \min\{\mu_{S_1}, \ldots, \mu_{S_n}\} \tag{2.15}$$

where μ_{S_i} is the membership function of the set S_i, $i = 1, \ldots, n$. Minimum is seen in Figure 2.2 b. Minimum is also the intersection operator for classical sets. It is not smooth, and it is sometimes replaced by its differentiable counterpart, the *soft minimum*:

$$\mu_S = \text{softmin}\{\mu_{S_1}, \ldots, \mu_{S_n}\} = \frac{\sum_{i=1}^{n} \mu_{S_i} e^{-k\mu_{S_i}}}{\sum_{i=1}^{n} e^{-k\mu_{S_i}}} \tag{2.16}$$

where k is a parameter. The soft minimum approaches the minimum asymptotically as $k \to \infty$. If k is small, the soft minimum gives unexpected results at small membership function values. Figure 2.2 c shows the "tails" of the soft minimum of two Gaussian membership functions. Obviously the value of $k = 10$ is too small.

The *product* combiner

$$\mu_S = \prod_i^n \mu_{S_i} \tag{2.17}$$

is smooth, and it is sometimes preferred in literature because of its conceptual simplicity. The product combiner takes all the membership function values into account, whereas the minimum and soft minimum ignore information. This becomes more important if there are more than just two membership function values. For example, consider two sets of membership function values (μ_1, μ_2, μ_3), namely, $(0.1, 0.9, 0.9)$ and $(0.1, 0.2, 0.2)$. The minimum combiner gives 0.1 for both of these. The soft minimum with $k = 10$ gives 0.1003 for the first one and 0.1269 (!) for the second one and the soft minimum with $k = 100$ gives 0.1000 and 0.1000, respectively. The product combiner gives 0.081 for the first one and 0.04 for the second one. Thus only the product combiner takes into account that the membership function values in the first set are on the average larger than the values in the second set. Soft minimum with $k = 10$ gives the opposite result, and we conclude that a value of $k = 10$ is too small. Depending on the application, the overall minimum might be crucial, or values of the other membership functions should be taken into account, too.

Union Operators. Fuzzy union operations again have several mathematical interpretations. They are compared in Figure 2.3.

Traditionally [68], the union was interpreted as the *maximum*. Using the maximum combiner, the membership function for the union set $S = \bigcup_{i=1}^n S_i$ is

$$\mu_S = \max\{\mu_{S_1}, \ldots, \mu_{S_n}\} \tag{2.18}$$

where μ_{S_i} is the membership function of the set S_i, $i = 1, \ldots, n$. This is also the union operator for classical sets. Another alternative is the *sum* combiner

$$\mu_S = \sum_i^n \mu_{S_i}. \tag{2.19}$$

Again, the sum does not lose any information, whereas the maximum does — remember the example comparing the minimum and the product. As the sum of membership functions may exceed one, the summation operator is usually restricted to lie between zero and one by using the *bounded sum* [67], [69] whose membership function is

$$\mu_S = \min\{1, \sum_{i=1}^n \mu_{S_i}\}, \tag{2.20}$$

or the *probabilistic sum* [20], [69] whose membership function for two sets is

$$\mu_S = \mu_{S_1} + \mu_{S_2} - \mu_{S_1}\mu_{S_2} \tag{2.21}$$

and in general

$$\mu_S = 1 - \prod_{i=1}^n (1 - \mu_{S_i}). \tag{2.22}$$

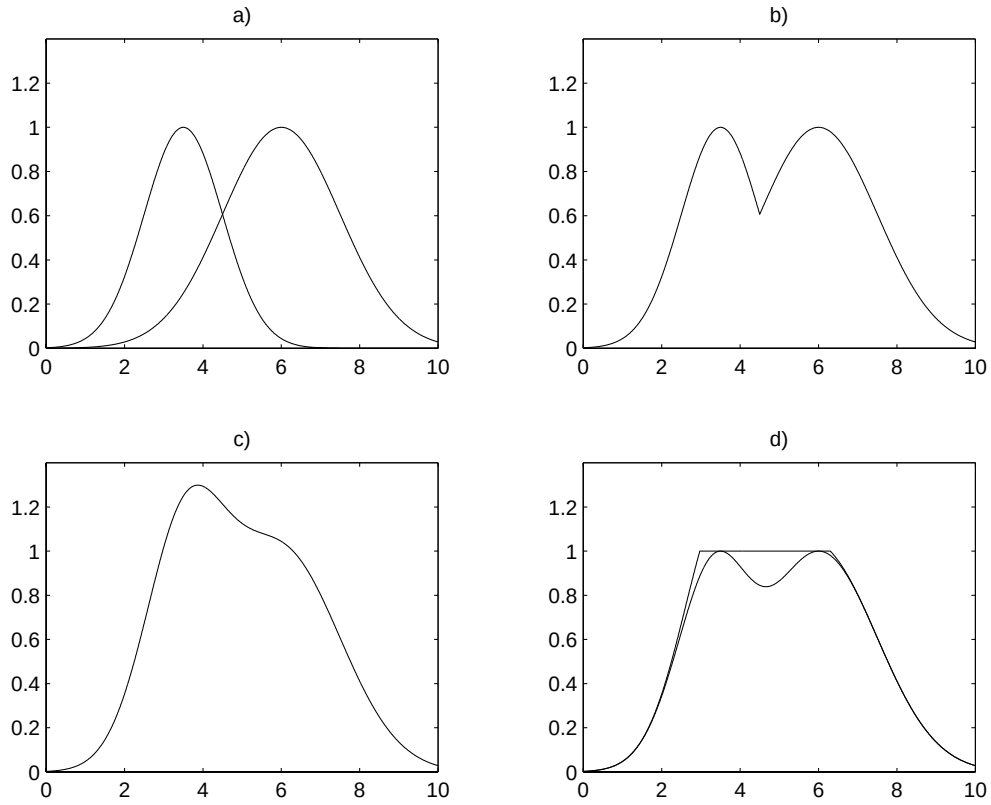


Figure 2.3: Union operators. a) Two Gaussian membership functions μ_1 and μ_2 of a linguistic variable S. b) Maximum of μ_1 and μ_2. c) Sum of μ_1 and μ_2. d) Bounded sum (upper curve) and probabilistic sum (lower curve) of μ_1 and μ_2.

2.4 Different Types of Uncertainty

2.4.1 Fuzziness Versus Probability

There has been a lot of discussion about differences between fuzziness and probability. First of all, a distinction should be made between different types of uncertainty. It may be of probabilistic, fuzzy or some other nature. The terminology varies in the literature, and in some texts, the word *uncertain* refers only to probabilistic uncertainty; the words *imprecise* or *vague* usually refer to fuzziness. Niskanen [49], [50] gives one explanation of the terminology, and many authors prefer to explain the words in their own way.

Consider a proposition "X *is* S". If there is probabilistic uncertainty inherent, X may assume other values than S, too, but our definition of S is clear. If we have evidence that X is actually S, we then know that X cannot be S' at the same time, if S and S' are mutually exclusive values of X. On the other hand, in fuzzy uncertainty, we cannot define S clearly and X may be S' at the same time, too. For example, a proposition "It will rain tomorrow" may be uncertain in both senses of the word. In the probabilistic sense, we do not know yet whether it will rain or not, but we know the changes are "rain" and "no rain" and the right answer will be

revealed the following day. The situation is uncertain only because our perception of the following day's weather is restricted. In fuzzy uncertainty, we cannot clearly define the concept of rain — if it drizzles slightly from time to time, we would say it both is raining and is not. The uncertainty is still present the following day, because the uncertainty is associated with the concept.

Zadeh [68] already pointed out that "the notion of a fuzzy set is completely non-statistical in nature". Since then, a lively and often disagreeing discussion of the differences and similarities of fuzziness and probability has been going on. Bedzek [4] has collected a series of papers where various points of view are discussed [10], [16], [25], [31], [38], [66].

Bedzek [3], [5] summarizes the difference between fuzziness and probability in an example. Let L be a linguistic value *potable*. Suppose that you have been in the desert for a week without drink and you are given two bottles, A and B. For bottle A, the degree of membership in L is $\mu_L(A) = 0.91$, whereas for bottle B, the probability of belonging to L is $p_L(B) = 0.91$. Which bottle would you choose? The membership function value of 0.91 means that the contents of A are fairly similar to potable liquids (for example, water). The probability of 0.91 means that over a long run of experiments, the contents of B are perfectly potable in 91 per cent of the cases, but in the other 9 per cent, the contents of B will not be potable (for example, B may contain hydrochloride acid). Thus to be on the safe side, you should choose A, because it may contain swamp water but it will not contain hydrochloride acid.

The example of Bedzek [3], [5] continues. Suppose that we examine the contents of A and B and discover that A contains beer and B contains hydrochloride acid. Now the membership function value of A is still 0.91, but the probability value of B drops from 0.91 to 0 after this observation.

Bedzek summarizes his example with the philosophical difference between fuzziness and probability [5]: A membership function represents the similarity of an object (liquid) to an imprecisely defined property (potable). A probability density function contains information about relative frequencies. Stating this, Bedzek obviously supports a frequentistic view of probability. With a subjective view of probability the difference would be summarized in another way, but we do not discuss this further here.

2.4.2 Ambiguity

One aspect of uncertainty is *ambiguity*, which is presented in Klir and Folger [26] and in Klir and Yuan [27] and further discussed in Kikuchi and Pursula [24]. Ambiguity of the proposition "X *is* S" means that our information about X is incomplete, but S itself is clearly defined. The same definition applies to probabilistic uncertainty, as discussed in Section 2.4.1, and according to [24], probabilistic uncertainty is one type of ambiguity. Other types of ambiguity are *possibilistic* uncertainty and a combination of probabilistic and possibilistic uncertainty.

A mathematical framework for ambiguity is *evidence theory*. The information about X is the evidence. In an uncertain situation, the evidence partially supports the proposition, and three situations can arise [24]:

1. Given many pieces of evidence (*e.g.*, experimental results), each piece of evidence points to only one subset. This kind of uncertainty is measured by a *probability measure*.

2. The body of evidence points to nested sets, so that one piece of evidence supports all the subsets of the proposition. This uncertainty is measured by a *possibility measure*. The possibility distribution assumes the same numerical value as the membership function of fuzzy set S.

3. A combination of 1 and 2, in which the uncertainty is measured by a *belief measure*. The measures are found in Dempster-Schafer theory [26]. This theory subsumes probability and possibility theories.

These situations are compared in Figure 2.4.

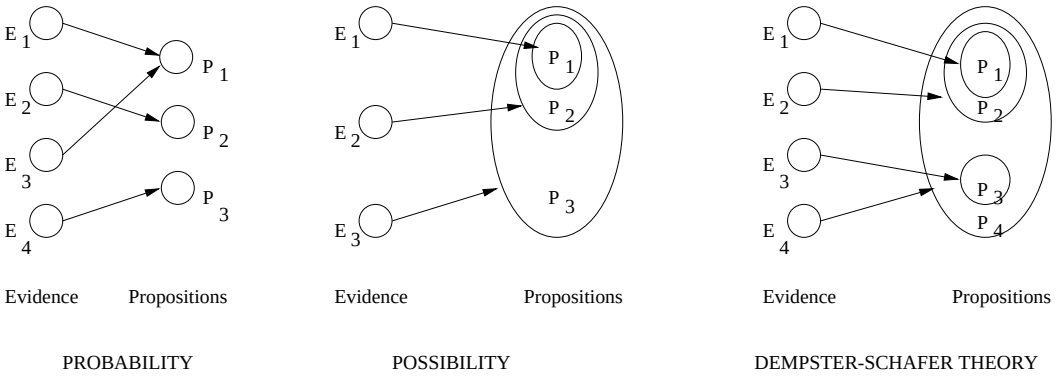


Figure 2.4: Relationships between evidence and propositions in an uncertain situation [24].

Kikuchi and Pursula [24] give examples of uncertainty measures in transportation problems, where uncertainty is associated with human judgement and behaviour. In real-world situations, the data observed may have a probabilistic character, but people often process them as a possibility distribution. For example, consider a phrase "Travel time is less than T". The evidence consists of observations of travel time, which are of the form "travel time is in the range $[T_1, T_2]$". In possibility theory, a piece of evidence that supports an interval also supports all intervals that are inside it. If it is possible that the travel time is in the range $[T_1, T_2]$, then it is also possible that the travel time is in a wider range $[T_1 - \delta_1, T_2 + \delta_2]$ where $\delta_1 > 0$ and $\delta_2 > 0$.

2.5 Fuzzy Control

2.5.1 Introduction

The most practical and successful application of fuzzy systems is fuzzy control. A clear advantage of fuzzy control systems over traditional ones is their ability to use

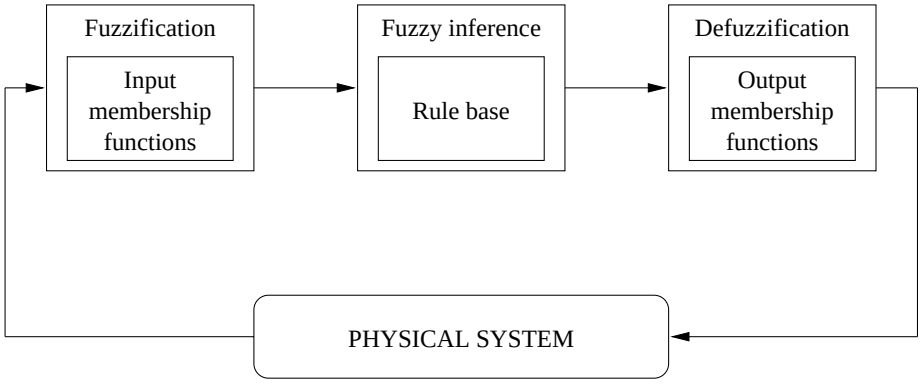


Figure 2.5: A fuzzy controller.

expert knowledge as such. The knowledge can be expressed as a *rule base* where the rules are propositions of the form "if X is S, then Y is T"; here S and T are fuzzy sets.

A fuzzy controller consists of *fuzzification*, *fuzzy inference* and *defuzzification* modules, as seen in Figure 2.5. The *fuzzification* module obtains measurements of controller input variables from the system under control. The measurements are converted to values of input membership functions. Depending on the context, the word "fuzzification" may include the construction of the membership functions used in the fuzzy control process as discussed in Section 2.2.3. In an ongoing control process, the membership functions are already constructed, and at most some fine-tuning of the functions is done during the control process. The *fuzzy inference* engine evaluates the control rules stored in the fuzzy rule base. Fuzzy inference includes rule firing strength calculation, fuzzy implication, and rule aggregation, which are discussed in the following sections. The result of the fuzzy inference is one or several output fuzzy sets, whose membership functions are *defuzzified* for obtaining the control action.

The rule base consists of N rules:

> if X_1 is S_{11} and/or ... and/or X_n is S_{n1}, then Y_1 is T_{11} and ... and Y_p is T_{p1}
> $\vdots$
> if X_1 is S_{1N} and/or ... and/or X_n is S_{nN}, then Y_1 is T_{1N} and ... and Y_p is T_{pN}

Each input variable X_j, $j = 1, \ldots, n$, assumes linguistic values or fuzzy sets S_{ji}, $i = 1, \ldots, N$, and each output variable Y_k assumes linguistic values T_{ki}, $k = 1, \ldots, p$. The membership functions for these fuzzy sets are $\mu_{S_{ji}}(x_j)$ and $\mu_{T_{ki}}(y_k)$, respectively. Note that the same linguistic values S_{ji} of the variable X_j may appear in several rules, making $S_{ji} = S_{ji'}$ for rules i and i'. The actual number of linguistic values may thus be smaller than N. The same applies for the linguistic values T_{ki} of Y_k. Also note that each input variable X_j need not have the same number of linguistic values.

Each rule in the rule base consists of an "if" part, called *antecedent*, and a "then" part, called *consequent*. One can also attach a rule importance weight m_i to each rule i. In this way, some rules can be considered more important than others.

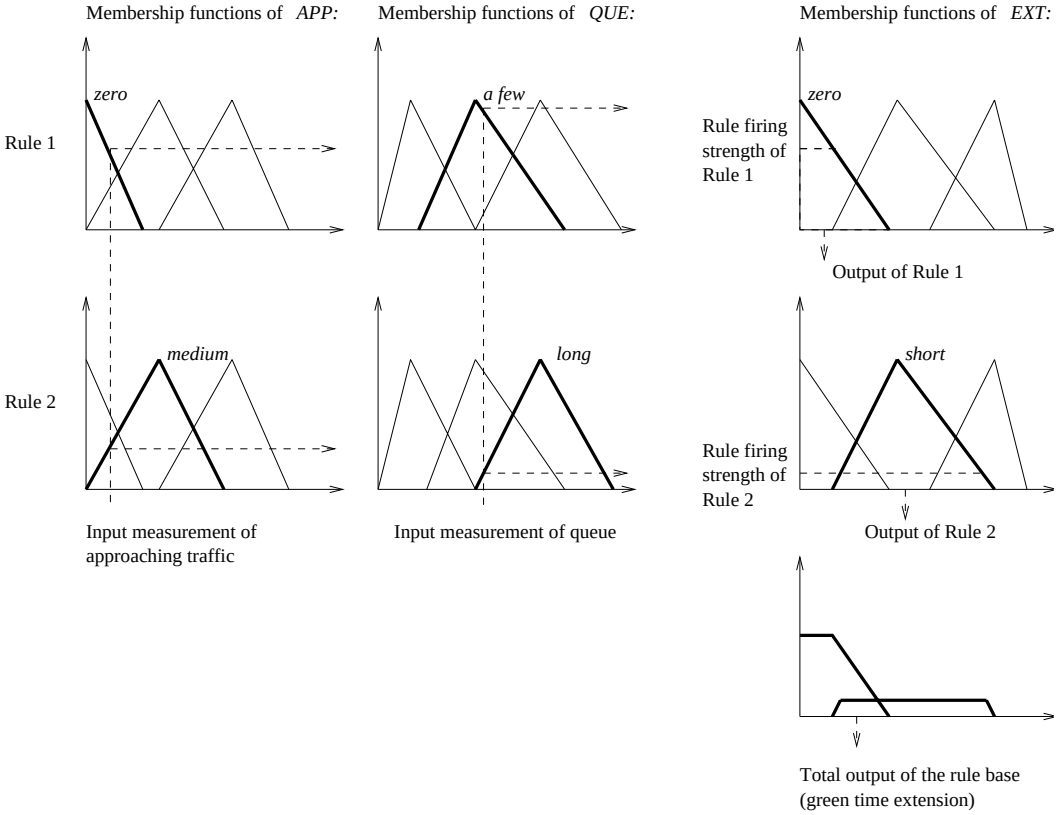


Figure 2.6: Fuzzy inference in a rule base of two rules. Rule 1: "if APP is *zero* and QUE is a few, then EXT is *zero*". Rule 2: "if APP is *medium* and QUE is *long*, then EXT is *short*".

Both in the antecedent and the consequent part of a rule, several fuzzy sets may be combined using "and" (intersection) or "or" (union) operators. In this work, an "and" operator is used in the antecedent, and there is only one output variable Y. The extension to more output variables is straightforward.

Figure 2.6 gives an example of the steps involved in a fuzzy control system of two input variables, one output variable and two rules. Each of the steps may be realized in several different ways, as discussed in the following sections. The system under consideration is a fuzzy traffic signal controller. The input variables are APP, the approaching traffic from the green direction, and QUE, the queue in the red direction. The output variable is EXT, the green signal extension. In this example, the following two rules are used:

if APP is *zero* and QUE is a few, then EXT is *zero*
if APP is *medium* and QUE is *long*, then EXT is *short*.

In Figure 2.6, membership functions for linguistic values used in each rule are drawn with a thick line. An input measurement pair (x_1, x_2) of the approaching traffic and the queue is obtained. Dotted horizontal lines show the membership function values: $\mu_{zero}(x_1)$ and $\mu_{afew}(x_2)$ in rule 1, and $\mu_{medium}(x_1)$ and $\mu_{long}(x_2)$ in rule 2. The rule firing strengths of each rule depend on these membership function values. Outputs

of two rules are combined and defuzzified to produce a numerical output value of the system. The details of these steps are discussed in Sections 2.5.2 and 2.5.3.

It should be pointed out that fuzzy control is completely non-statistical in nature. A given input always results in the same output if the control system is kept constant. An exception are those adjustable controllers whose learning algorithm employs some kind of stochastic exploration. In these algorithms, the output of the controller is deviated by a random amount. This stochastic exploration may lead to a better control performance. One of these learning algorithms is reinforcement learning, which is used in this thesis and will be discussed in detail.

2.5.2 Steps Involved in Fuzzy Inference

Rule Firing Strength. The *rule firing strength* w indicates how well the conditions in the antecedent part of the rule are satisfied: an input x "fires" a rule to a degree $w \in [0, 1]$. It determines the degree to which conclusions drawn in the consequent part apply. Rule firing strength is calculated using the membership function values for fuzzy sets used in the rule antecedent, and the formula depends on whether an "and" or an "or" combiner was used.

Modern fuzzy literature chooses the product combiner (2.17) for "and" (intersection), because it is smooth and avoids wasting information. Hence, the rule firing strength w is the product of the membership function values in the rule antecedent. The minimum combiner (2.15) is a traditional choice.

An "or" (union) combiner is not so often used in the rule antecedent of a control process, but when it is, one of the sum variants (2.19), (2.20) and (2.21) is preferred over the maximum operator, and w is a sum of the membership function values in the rule antecedent.

As an example, consider Figure 2.6. Rules 1 and 2 use the "and" combiner in combining the conditions in the rule antecedents. The rule firing strength w_1 of Rule 1 is the minimum of $\mu_{zero}(x_1)$ and $\mu_{afew}(x_2)$ and the rule firing strength w_2 of Rule 2 is the minimum of $\mu_{medium}(x_1)$ and $\mu_{long}(x_2)$. If the product combiner is used, the rule firing strengths are the products of the membership function values, respectively.

Fuzzy Implication. After the rule firing strength w is calculated, the fuzzy set T in the rule consequent part is now either clipped or scaled at a level specified by the rule firing strength. Comparing Figures 2.7 a and b shows the difference between clipping and scaling. Clipping or *correlation-minimum inference* means that the membership function $\mu_{T'}$ of the resulting set T' is a minimum of μ_T and w:

$$\mu_{T'} = \min\{w, \mu_T\}. \tag{2.23}$$

Clipping does not preserve the original shape of the fuzzy set, and therefore it is usually not preferred in modern literature. It is also troublesome in calculations, as the minimum function is not differentiable, and the derivative of μ is needed later.

Scaling or *correlation-product inference* means that the membership function μ_T is multiplied by the rule firing strength w:

$$\mu_{T'} = w\,\mu_T. \tag{2.24}$$

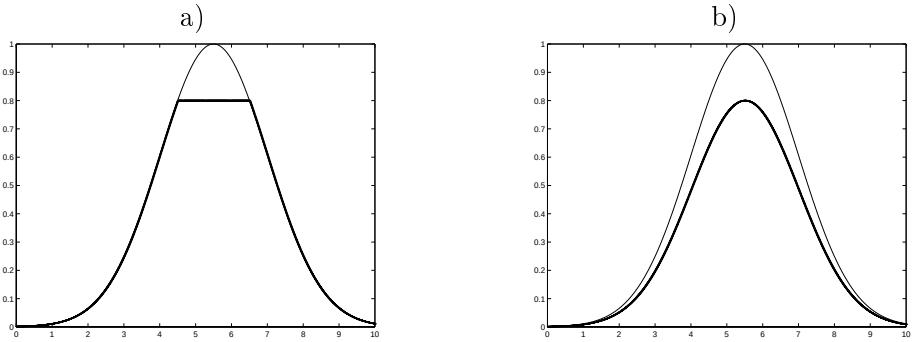


Figure 2.7: A membership function μ and its a) clipped version b) scaled version ($w = 0.8$).

Scaling produces a set T' which is of the same shape as the original set T. Scaling is functionally more feasible, as product is a differentiable function.

Consider again the example in Figure 2.6. In this case, correlation-minimum inference is used. In Rule 1, the membership function μ_{zero} of EXT is cut at level w_1, and the membership function of the remaining set is $\mu_{zero'} = \min\{w_1, \mu_{zero}\}$. Correspondingly, the remaining set in Rule 2 has membership function $\mu_{short'} = \min\{w_2, \mu_{short}\}$.

Rule Aggregation. A collection of N scaled or clipped fuzzy sets T_i', $i = 1, \ldots, N$, obtained through one of the procedures described above now represents the total output of the rule base. Each T_i' has its own membership function $\mu_{T_i'}$. This section describes how the T_i' are combined together to form a union $T' = \cup_{i=1}^{N} T_i'$ which is later used in the defuzzification phase. However, some defuzzification procedures use the sets T_i' as such, and the union is not needed, depending on the algorithm.

As discussed earlier, union aggregation can be realized in several different ways. The maximum combiner (2.18) does not take into account the overlapping of the sets and therefore ignores information. If several output sets overlap in some region, this region obviously describes the rule set output best, and overlapping values should be given more emphasis. The sum combiner (2.19) simply adds up all membership functions. The resulting set's membership function may assume values greater than 1, as seen in Figure 2.3. This is not a problem in practical applications, although it is usually assumed that membership function values are restricted between 0 and 1. The bounded sum (2.20) or the probabilistic sum (2.21) are restricted to be less than or equal to 1, but they fail to reveal output values concentrated on the same area. In this respect, the sum combiner (2.19) maintains the most information.

Various ways of rule aggregation are presented in Figure 2.8. In Figure 2.8 a, three Gaussian membership functions μ_1, μ_2 and μ_3 for an output linguistic variable Y are shown, together with rule firing strengths w_1, w_2 and w_3. In Figure 2.8 b, these membership functions are first cut at levels w_i, resulting in a minimum (2.15) of μ_i and w_i, $i = 1, 2, 3$, and the union is then formed by taking a maximum (2.18). This *min-max decomposition* is the traditional choice in fuzzy literature. Figure 2.8 c shows the *prod-sum decomposition*, where membership functions μ_i are first multiplied (see Formula (2.24)) with w_i and the remaining sets are summed (see Formula (2.19)) together. In Figure 2.8 d, the same is done using the bounded sum and the probabilistic

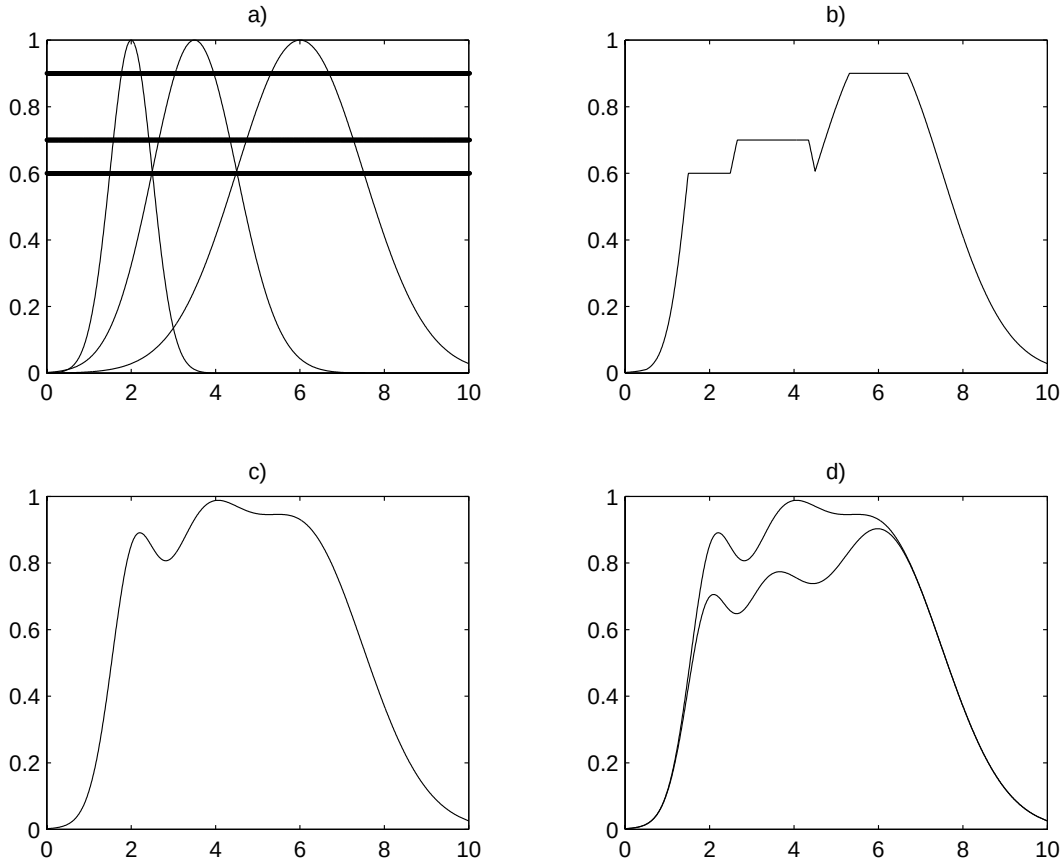


Figure 2.8: a) Membership functions μ_{T_1}, μ_{T_2} and μ_{T_3} and rule firing strengths $w_1 = 0.6$, $w_2 = 0.7$ and $w_3 = 0.9$. b) The *min-max* decomposition. c) The *prod-sum* decomposition. d) Upper curve: the *prod-sum* decomposition using the bounded sum (Formula (2.20)). Lower curve: the *prod-sum* decomposition using the probabilistic sum (Formula (2.21)).

sum, whose formulae for scaled membership functions are

$$\mu_{T'} = \min\{1, \sum_{i=1}^{N} w_i \mu_{T_i}\} \quad \text{(bounded sum)} \tag{2.25}$$

$$\mu_{T'} = 1 - \prod_{i=1}^{N} (1 - w_i \mu_{T_i}) \quad \text{(probabilistic sum)} \tag{2.26}$$

Note that in this case, the bounded sum in Figure 2.8 d is the same as the sum in Figure 2.8 c, as the sum of the membership functions does not exceed 1.

If rule importance weights m_i are used, each membership function $\mu_{T_i'}$ must first be scaled by m_i and the result normalized:

$$\mu_{T_i' new} = \frac{m_i \mu_{T_i'}}{\sum_{i=1}^{N} m_i}. \tag{2.27}$$

This scaling can also take place in the defuzzification phase, the weights m_i being used in the process of combining the output sets T_i or T_i', depending on the algorithm.

Going back to our example in Figure 2.6, the scaled fuzzy sets *zero'* and *short'* of EXT are not combined but are presented separately, so that the sets can be used as such in the defuzzification phase.

2.5.3 Defuzzification

After the union T' has been defined using one of the procedures described above, it must be defuzzified to obtain a numerical value for the output control variable Y. Methods for the defuzzification of a union T' of several membership functions are the *center of area* (COA) and the *mean of maxima* (MOM) methods and their variations, which are explained in the sequel. The variations of COA and MOM discussed here are *local* applications in the sense that instead of forming a union T', each output fuzzy set T_i is defuzzified separately and the output crisp values are averaged in some way. Different defuzzification methods are compared in Figure 2.9.

One should keep in mind that defuzzification methods usually lack logical justification; mostly they are technical in nature. However, the methods differ in such ways that the choice between them can be made based on practical considerations, as is discussed in the sequel.

Center of Area (COA) Defuzzification. The *center of area* defuzzification method calculates the centroid of the fuzzy set T':

$$y^* = \frac{\int_U y\,\mu(y)\,dy}{\int_U \mu(y)\,dy}. \tag{2.28}$$

Here $\mu(y)$ is used instead of $\mu_{T'}(y)$, for simplicity. An example of COA applied to a union of fuzzy sets is seen in Figure 2.9 a. In the discrete case, in which μ is defined on a finite universal set $\{y_1, \ldots, y_k\}$, the formula is

$$y^* = \frac{\sum_{i=1}^{k} y_i\,\mu(y_i)}{\sum_{i=1}^{k} \mu(y_i)}. \tag{2.29}$$

Note that the fractions

$$\frac{\mu(y)}{\int_U \mu(y)\,dy} \tag{2.30}$$

or

$$\frac{\mu(y_i)}{\sum_{i=1}^{k} \mu(y_i)}, \quad i = 1, \ldots, k \tag{2.31}$$

form a probability distribution, as they are in the interval $[0, 1]$ and they sum to unity. Therefore the defuzzified value obtained by COA can be interpreted as the expected value of the output variable Y [27].

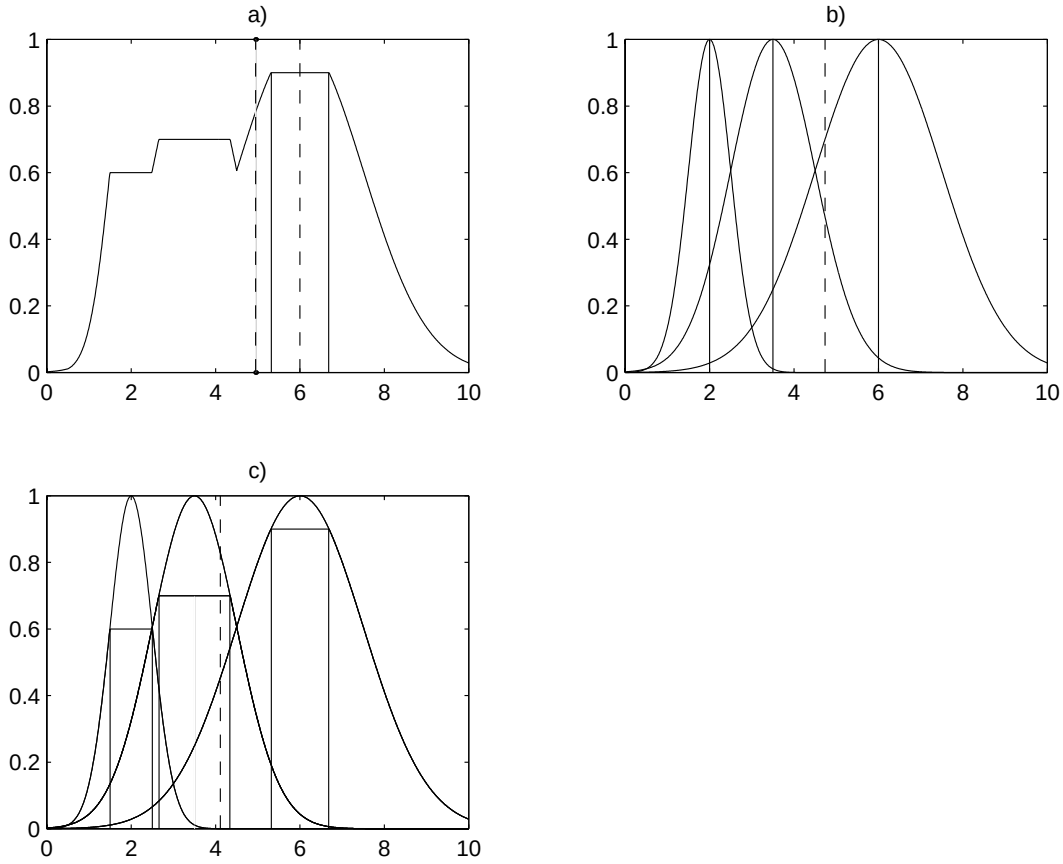


Figure 2.9: Defuzzification of membership functions μ_1, μ_2 and μ_3 (cf. Figure 2.8). Rule firing strengths are $w_1 = 0.6$, $w_2 = 0.7$ and $w_3 = 0.9$. a) The COA ($y^* = 4.95$, dashed line on the left) and MOM ($y^* = 6.00$, dashed line on the right) methods applied to a union of membership functions. The solid vertical lines show the infimum and supremum of M (Formula (2.35)). b) The LCOA method in which w_i and V_i are used as weights. The dashed line shows $y^* = 4.73$. c) The LMOM method in which w_i are used as weights. The solid vertical lines show the infima and suprema of the M_{w_i} (Formula (2.39)). The dashed line shows $y^* = 4.11$.

Local Center of Area (LCOA) Defuzzification. A "local" and additive approach of the COA method is proposed by Kosko [33] and also mentioned in Jang and Sun [20] and Zimmermann [69]. In this approach, the centroid of T' is expressed as a convex sum of the centroids of the consequent sets. The total output fuzzy set T' is not actually needed — instead, each rule is defuzzified separately. The defuzzified output is

$$y^* = \frac{\sum_{i=1}^{N} m_i \, w_i \, V_i \, y_i}{\sum_{i=1}^{N} m_i \, w_i \, V_i} \tag{2.32}$$

where N is the number of rules in the rule base, m_i is the weight of rule i (if rule importance weights are used), w_i is the firing strength of rule i, V_i is the volume of the (unscaled and unclipped) consequent set T_i and y_i is the centroid of T_i, calculated using the common formula (2.28) of a centroid. Hence a weighted average of the then-part set centroids is computed, each weight being the product of the rule importance, the rule firing strength and the then-part set volume.

This is equivalent to taking the centroid of T', where T' was obtained using the *prod-sum* decomposition (see section 2.5.2), but this way of calculating the centroid is faster to implement: output set volumes and set centroids can be calculated in advance and stored in a look-up table.

Note that the LCOA formula (2.32) is equivalent to the COA formula (2.28) if and only if in (2.28) the area of the output fuzzy set is formed using the sum combiner (2.19). The centroid formula (2.28) could of course be used for areas formed using the maximum (2.18), the bounded sum (2.20) or the probabilistic sum (2.21), too, but another problem would then be faced: the overlapping of sets would be partly or totally ignored. In (2.32), this problem is avoided, as overlapping areas are added as many times as they overlap.

A drawback of the LCOA defuzzification is that then-part set *volumes* are used as averaging weights in (2.32). Large sets are thus given more emphasis than small sets. The wider the fuzzy set is, the more uncertainty it often possesses. Thin sets represent more certain information and they should be given more weight than wide sets. For example, *inverses* of set volumes could be used as weights. In the LCOA, the situation is the other way round.

The objective would thus be to avoid giving emphasis to large and uncertain rules and, on the other hand, to take into account overlapping areas. One solution for this is to defuzzify each output set without any scaling or clipping and then calculate a weighted average of these defuzzified values y_i, using rule firing strengths w_i as weights:

$$y^* = \frac{\sum_{i=1}^{N} m_i \, w_i \, y_i}{\sum_{i=1}^{N} m_i \, w_i}. \tag{2.33}$$

This leads to simple calculations, as the set volumes V_i need not be calculated. The rule importance weights m_i are optional. Note that this is not anymore equivalent to centroid calculation (2.32), as the set volumes V_i are not used.

In Figure 2.9 b, three membership functions and their centroids together with the weighted average (2.32) are shown. The rule importances m_i were not used in this example. The membership functions are the same as in Figure 2.9 a, but now they

are not clipped at levels w_i, $i = 1, 2, 3$. Instead, the w_i are used as weights together with the set volumes V_i.

Mean of Maxima (MOM) Defuzzification In the *mean of maxima* defuzzification method, also called the *center of maxima* method, the defuzzified value is defined as the average of the smallest value and the largest value of y for which $\mu(y)$ reaches its maximum. That is,

$$y^* = \frac{\inf M + \sup M}{2} \tag{2.34}$$

for continuous μ, where

$$M = \{y \in U \mid \mu(y) = \max_y\{\mu(y)\}\}. \tag{2.35}$$

An example of MOM applied to a union of fuzzy sets is presented in Figure 2.9 a. In the discrete case, y^* is the mean of all values of the universe of discourse U having maximal membership grades [67]:

$$y^* = \frac{1}{|M|} \sum_{y_i \in M} y_i \tag{2.36}$$

where $|M|$ is the number of elements in M.

The MOM method is suitable for output fuzzy sets T_i clipped at level w_i or at level 1, because they have a plateau M of maximal values as in Figure 2.8 b. Instead, if the fuzzy set T' is unimodal as in Figure 2.8 c, the maximum is unique and M consists of one point only. In this case, the MOM method is also called the *mode* defuzzification method, as the defuzzified value y^* is the one at which $\mu(y)$ reaches its maximum:

$$y^* = \arg\max \mu(y). \tag{2.37}$$

Klir and Yuan [27] note that if M is not connected, (that is, M consists of several intervals $[y_1, y_2]$), y^* may be defined as the arithmetic average of mean values of all intervals contained in M, including intervals of length zero. Alternatively, y^* may be defined as a weighted average of mean values of the intervals, in which the weights are interpreted as the relative lengths of the intervals.

The mean of maxima method has a drawback of ignoring the shape of $\mu(y)$ outside M. Only the output values with maximal degree of membership are taken into account.

Local Mean of Maxima (LMOM) Defuzzification Berenji and Khedkar [6] propose a *local mean of maxima* (LMOM) defuzzification method, where each output fuzzy set is defuzzified using an adaptation of the MOM method instead of applying MOM to the union of the output sets. Let w be the rule firing strength. If the membership function μ for the output fuzzy set is strictly monotone, the defuzzified output y^* is simply $\mu^{-1}(w)$, the inverse of μ. As membership functions are seldom

monotone, a mathematical inverse may not exist. In this case, y^* is defined as the centroid of the set of those y for which $\mu(y)$ exceeds w as in Figure 2.10:

$$y_i^* = \frac{\inf M_w + \sup M_w}{2} \tag{2.38}$$

where

$$M_w = \{y \in U \,|\, \mu(y) \geq w\}. \tag{2.39}$$

Note that μ does not have to be symmetric. If it is, the centroid of the set M_w is the same as the centroid (2.28) of the original set.

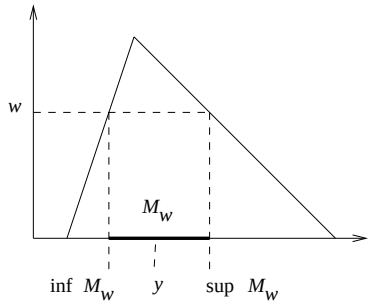


Figure 2.10: The LMOM method.

The output of the fuzzy system is a weighted average of the individual rule outputs, the weights being the rule firing strengths w_i and rule importance weights m_i just as in (2.33):

$$y^* = \frac{\sum_{i=1}^{N} m_i \, w_i \, y_i^*}{\sum_{i=1}^{N} m_i \, w_i}. \tag{2.40}$$

Again, the rule importance weights m_i are optional. The LMOM defuzzification is presented in Figure 2.9 c, which shows the infima and suprema of M_w (Formula (2.39) and the weighted average y^*.

The LMOM defuzzification method ignores the shape of the membership function outside M_w, just as MOM does, but as LMOM is applied to each rule, all rules are taken into account. MOM notices only those rules whose w is maximal. LMOM has other advantages, too. As mentioned earlier, overlapping output fuzzy sets should not be ignored. In LMOM, each rule consequent set is defuzzified separately, and centroids concentrated near each other are taken into account, too. Also, the defuzzified values are averaged using the rule firings w, not the rule consequent set volumes V, as weights.

Chapter 3

Neural Networks

3.1 Introduction

An artificial neural network, henceforth a neural network (NN), consists of simple processing elements called *neurons* or *cells* interconnected as a structured network. Each connection has a strength that is expressed as a network parameter, a *weight*. Neural networks are a wide class of nonlinear regression models and dynamical systems [61]. They adapt themselves to observed data by changing the strengths of the interconnections of the network. In this sense, neural networks are surface-fitting algorithms quite similar to regression models. As the structure of the network is complex, it is often difficult to interpret the knowledge it has acquired. For example, the significances and contributions of different input variables cannot be compared in the same way as in regression models. The name "neural network" stems from NNs' parallel data processing similar to biological neural networks, although the research on NN systems has later clearly departed from neurobiology. For an overview of the history of neural networks, see [15] and [57].

Neural networks are, at their best, able to model a system without a rule base or analytical knowledge about the system. Using a large data set, a network which can predict the behaviour of the system can be constructed. The success of the modelling depends largely on the quality of the data — whether it represents the true behaviour of the system — and the structure of the network.

The learning in a neural network may be *supervised* or *unsupervised*. In supervised learning, the network is presented both an input value and a desired output value. A learning algorithm compares the NN's output with this target output and alters the network weight parameters in a direction which decreases the output error.

A special type of supervised learning is *reinforcement learning*, which is an important learning scheme in this work. Reinforcement learning is treated in Chapter 5.

In unsupervised learning, no target output is used. Instead, the neural network classifies inputs according to some common features which they share. The network's output signal should correspond to the category of the input. In other words, the network should display some *self-organization*. An important NN architecture of this kind is Kohonen's self-organizing map. This thesis will not deal with unsupervised

learning, and the reader is referred to [28], [29] or [30] or to literature listed at the end of this section.

Neural networks have become widespread as the performance of computers has increased. Traditional statistical methods do not utilize the full abilities of modern computing anymore. NN science did not originate from statistics and these two branches of science have not developed hand in hand. Anyway, most of the NN methods have already been used in statistics to one degree or another, and equivalent statistical methods can often be found. For example, the most widespread NN system, the multilayer perceptron, is nothing more than a general nonlinear regressor. Non-parametric regression, discriminant analysis and principal component analysis also find their equivalents in NN literature [61]. Unfortunately, the terminologies in these two schools are different and may cause misunderstandings. Sarle [60], [61] has listed several terms used in neural network science and gives their equivalents in statistical terminology. For example,

- independent variable is *input* in the NN terminology
- predicted value is *output*
- dependent value is *target value* or *training value*
- residual is *error*
- estimation is *training, learning, adaptation* or *self-organization*
- estimation criterion is *error function* or *cost function*
- observation is *pattern* or *training pair*
- parameter estimate is *(synaptic) weight*
- regression and discriminant analysis are *supervised learning* or *heteroassociation*
- data reduction is *unsupervised learning* or *autoassociation*
- interpolation and extrapolation are *generalization.*

In this thesis, mainly neural network terminology is used.

There have been various over-optimistic claims about the intelligence and abilities of neural networks. NNs' brain-like structure does not make them intelligent, and if learning and generalization are symptoms of intelligence, then many statistical methods are intelligent, too. If the training data do not include enough information of the system, you cannot assume the network to figure out the information you desire. The same problem goes with statistical learning methods.

The literature on neural networks is wide. Refer to *e.g.* Cichocki and Unbehauen [9], Hecht-Nielsen [14], Hertz *et al.* [15], Holmström and Kohonen [17], Klir and Yuan [27] and Rumelhart and McClelland [57]. Comparisons between statistical and neural methods are treated in Sarle [61].

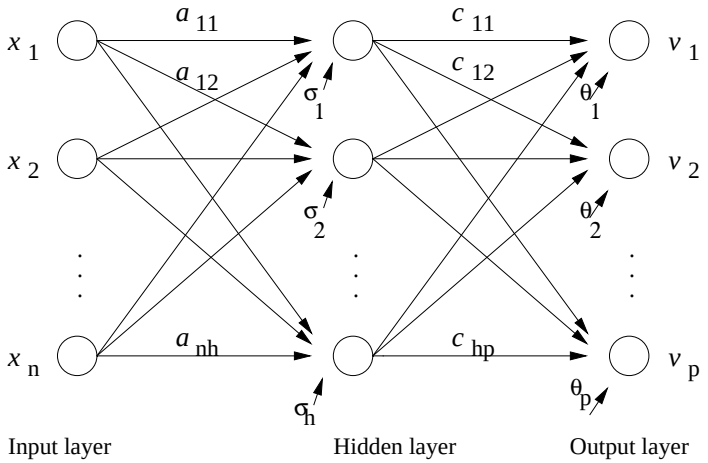


Figure 3.1: A feedforward neural network with one hidden layer. a_{ij} and c_{jk} are the network weight parameters.

3.2 Feedforward Neural Networks

The most often used neural networks are feedforward networks (Figure 3.1). A feedforward neural network consists of an input layer, zero or more hidden layers and an output layer. Types of feedforward networks are a *multilayer perceptron* (MLP) network and a *radial basis function* (RBF) network, of which a MLP network is used in this work.

The input layer contains the input variables $x_1, \ldots, x_n$ of the system. This layer does not compute anything — it is just a habit in the NN literature to draw networks with this first layer.

In the hidden layer, each cell j receives weighted input variables $(a_{1j}x_1, \ldots, a_{nj}x_n)$. In addition, each cell j may receive a constant input σ_j. If this constant input is used, it is usually included as an additional input $x_{n+1} = -1$ with a weight $a_{n+1,j} = \sigma_j$ for the ease of notation. The hidden layer output $\mathbf{z} = (z_1, \ldots, z_h)$ is computed using the hidden layer *activation function*, which is by default the same in every cell of the layer. In an MLP, the activation function f may be, for example [61]

- identity: $f(x) = x$
- hyperbolic: $f(x) = \tanh(x)$
- sigmoidal or logistic: $f(x) = (1 + e^{-x})^{-1}$
- threshold: $f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
- Gaussian: $f(x) = e^{-x^2/2}$.

Usually the activation function must be differentiable and it must saturate at both extremes. Of the above functions only the hyperbolic and sigmoidal (also called logistic) functions fulfill the requirements of differentiability and saturation at both

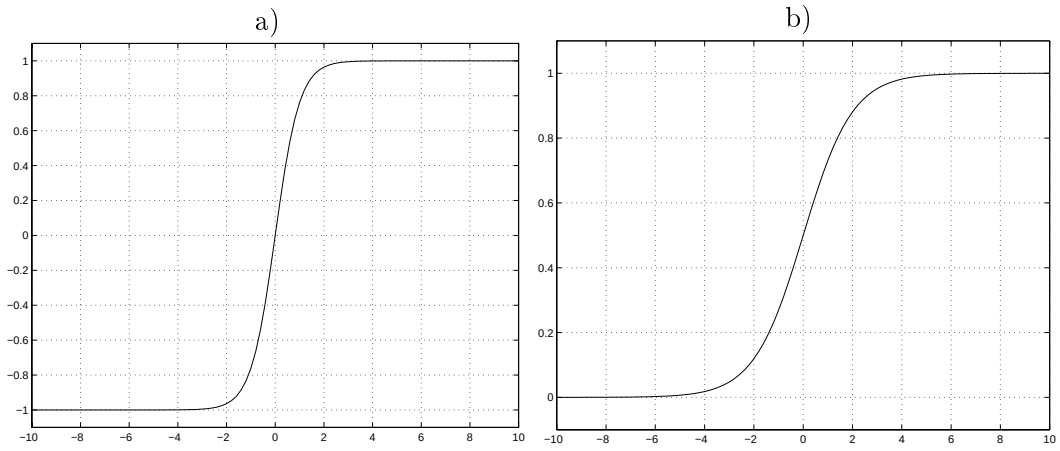


Figure 3.2: a) Hyperbolic activation function squashes from -1 to 1. b) Sigmoidal (or logistic) activation function squashes from 0 to 1.

extremes, and that is why they are the most usual activation functions. Hyperbolic and sigmoidal activation functions are seen in Figure 3.2.

In an MLP network, the activation function operates on a linear combination of inputs:

$$z_j = f\left(\sigma_j + \sum_{i=1}^{n} a_{ij} x_i\right). \tag{3.1}$$

In a RBF network, the hidden layer activation function operates on the distance between the input vector and the weight vector:

$$z_j = f\left(\left[\sum_{i=1}^{n} \frac{(a_{ij} - x_i)^2}{2\sigma_j}\right]^{1/2}\right). \tag{3.2}$$

Here the weights a_{ij} are often considered as centers of Gaussian curves, and there is a width σ_j associated with each hidden node. The activation function in a RBF network may be any function $f(x)$ which operates on $x \geq 0, x \in \mathbb{R}$ and which reaches a maximum at zero and approaches zero at infinity, such as the Gaussian function

$$f(x) = e^{-x^2/2}. \tag{3.3}$$

In both MLP and RBF networks, the size h of the hidden layer may vary, and there are no precise rules for determining how many cells it should contain. The more cells there are, the more complex systems the network can model, but the computation may become burdensome. Too complex models result in overparametrization: random perturbations in the data set are also modelled even if they should not be.

There may be several hidden layers, although one is usually enough. Each layer can have a different activation function. Adding more hidden layers increases the network's ability to model complex systems. One can never be sure if the model is complex enough, and trial and error are needed in the network construction phase.

Input layer cells i may also be connected straight to output layer cells k with weights b_{ik}.

Each hidden layer cell j (in the *last* hidden layer, if there are several) is connected to each output layer cell k with a weight c_{jk}. In addition, output layer cell k may receive a constant input $-\theta_k$, which is again often included as $z_{h+1} = -1$ with a weight $c_{h+1,k} = \theta_k$ for ease of notation. The size p of the output layer is the same as the number of output variables in the system under consideration. The output layer activation function is often linear both in an MLP and in an RBF:

$$v_k = \theta_k + \sum_{j=1}^{h} c_{jk} z_j. \tag{3.4}$$

3.3 Surface Fitting in Neural Networks

Neural networks are surface fitting algorithms. That is, they construct a surface which best fits in the observations. The surface consists of small "bumps". The form of the bumps depends on the choice of the activation function. Hyperbolic and sigmoidal activation functions produce smooth "thresholds", as seen in Figure 3.2; Gaussian activation function produces bell-shaped hills and threshold activation function builds surfaces using rectangular "bricks".

The number of connections in the network determines the complexity of the surface. Each weight parameter produces one "bump", as the shape of the activation function is different at different weight parameters. In the learning phase, the parameters are altered so that the surface would better fit in the observations. Modifying the parameters in turn modifies the shape and location of the "bump". The more bumps there are, the more complex surface the network can construct.

Roughly speaking, the "bumps" are centered in the region of the observations. As seen in Figure 3.2, the activation function gives non-constant outputs on a restricted area, and outside that, the output of the activation function is nearly constant. In the surface-fitting process, the areas where there are no or only little observations, the surface remains flat and obviously the network cannot model the system in these areas. That is why extrapolation of the data is not possible using neural networks.

3.4 Backpropagation Algorithm

An MLP is often trained using the so-called *backpropagation algorithm* or the *generalized delta rule* which was presented by Rumelhart *et al.* [56]. The weights of the network are altered in such a way that the error function is minimized. Minimization is done using the gradient descent technique. The error function E measures the difference between the network output $\mathbf{v}$ and the desired value $\mathbf{d}$ at each observation as a Euclidean distance:

$$E = \frac{1}{2}\|\mathbf{v} - \mathbf{d}\|^2 \tag{3.5}$$

where $\mathbf{v} = (v_1, \ldots, v_p)$ and $\mathbf{d} = (d_1, \ldots, d_p)$. In gradient descent, each weight parameter ω is updated as

$$\omega(t+1) = \omega(t) - \eta \frac{\partial E}{\partial \omega} \tag{3.6}$$

where η is a learning parameter, and the partial derivative is computed using the chain rule.

The rules for updating the weight parameters c_{jk} for connections between the hidden layer and the output layer can be derived as follows. The error function E is

$$E = \frac{1}{2}\|\mathbf{v} - \mathbf{d}\|^2 \tag{3.7}$$

$$= \frac{1}{2}\sum_{k=1}^{p}(v_k - d_k)^2 \tag{3.8}$$

$$= \frac{1}{2}\sum_{k=1}^{p}(\sum_{j=1}^{h} c_{jk}\, z_j - d_k)^2. \tag{3.9}$$

The gradient descent algorithm (3.6) gives now

$$c_{jk}(t+1) = c_{jk}(t) - \eta\,\frac{\partial E(t)}{\partial c_{jk}} \tag{3.10}$$

$$= c_{jk}(t) - \eta\,\frac{\partial E(t)}{\partial v_k}\frac{\partial v_k}{\partial c_{jk}} \tag{3.11}$$

$$= c_{jk}(t) - \eta\,(v_k(t) - d_k)\, z_j(t). \tag{3.12}$$

When updating a_{ij}, the error function is

$$E = \frac{1}{2}\|\mathbf{v} - \mathbf{d}\|^2 \tag{3.13}$$

$$= \frac{1}{2}\sum_{k=1}^{p}(v_k - d_k)^2 \tag{3.14}$$

$$= \frac{1}{2}\sum_{k=1}^{p}(\sum_{j=1}^{h} c_{jk}\, z_j - d_k)^2 \tag{3.15}$$

$$= \frac{1}{2}\sum_{k=1}^{p}(\sum_{j=1}^{h} c_{jk}\, f(\sum_{i=1}^{n} a_{ij}x_i) - d_k)^2 \tag{3.16}$$

where f is the activation function. The derivative of the activation function used in this work, the sigmoidal function $f(x) = (1+e^{-x})^{-1}$, is $f'(x) = f(x)[1-f(x)]$. In the gradient descent formula (3.6), a few more passes of the derivative chain rule for E are needed, and the update formula gives

$$a_{ij}(t+1) = a_{ij}(t) - \eta\,\frac{\partial E(t)}{\partial a_{ij}} \tag{3.17}$$

$$= a_{ij}(t) - \eta\left(\sum_{k=1}^{p}\frac{\partial E(t)}{\partial v_k(t)}\frac{\partial v_k(t)}{\partial z_j(t)}\right)\frac{\partial z_j(t)}{\partial a_{ij}} \tag{3.18}$$

$$= a_{ij}(t) - \eta\left(\sum_{k=1}^{p}(v_k(t) - d_k)c_{jk}(t)\right) z_j(t)\,(1 - z_j(t))\, x_i(t). \tag{3.19}$$

To avoid getting stuck in a local minimum in the error surface, the learning rate η must be sufficiently large. On the other hand, too large η may cause oscillations when the minimum lies in a narrow ravine and the algorithm jumps from one side

of the valley to another. One simple way of enhancing the algorithm is to include a *momentum* term in the update formula (3.6):

$$\omega(t+1) = \omega(t) - \eta \frac{\partial E}{\partial \omega} + \alpha(\omega(t) - \omega(t-1)) \tag{3.20}$$

where the momentum parameter α lies between 0 and 1 and a value of $\alpha = 0.9$ is often chosen [15]. The momentum term gives a contribution from the previous time step $t-1$, as the value of $\omega(t-1)$ is included. Using the momentum, the learning rate η can be increased without resulting in oscillations.

Chapter 4

Neurofuzzy Systems

4.1 Introduction

In this chapter, combinations of fuzzy and neural approaches are presented. Some texts make a difference between *fuzzy neural* and *neural fuzzy*. A fuzzy neural network refers to a neural network in which some of the properties have been "fuzzified". In adjacent layers, every cell of one layer is connected to every cell of the other layer, just as in Figure 3.1. A neural fuzzy system is a fuzzy system presented as a neural net, and not all network connections are present. Neural fuzzy systems are discussed more in the following. In this text, the word "neurofuzzy" is used for both fuzzy neural and neural fuzzy, as it is often difficult to judge whether the system is initially a neural network or a fuzzy inference engine. Besides, the terminology used in literature is somewhat ambiguous.

A neurofuzzy network, be it fuzzy neural or neural fuzzy, differs from a traditional neural network in one or several of the following respects [27]:

1. Inputs are fuzzy numbers
2. Outputs are fuzzy numbers
3. Weights are fuzzy numbers
4. Activation functions do not use a weighted sum of inputs; instead, fuzzy logical operations are used to combine the inputs

Here a fuzzy number is a fuzzy set which presents a classical number. For example, "fuzzy 5" can be presented with a fuzzy set whose membership function reaches a maximum at 5. For an extensive treatment of calculus on fuzzy numbers, see for example Kaufmann and Gupta [22], Klir and Yuan [27], or Turunen [65].

In neural fuzzy systems, the parameters of the fuzzy reasoning are expressed as the network connection weights. These systems are surface fitting algorithms, just as neural networks are, but their structure can be interpreted as a fuzzy algorithm. In a typical neural network, the amount of connections between the cells is vast, and in each layer, every input cell is linked to every output cell. This means that the

connections cannot often be given any logical interpretation. On the other hand, in a neural fuzzy network, the number of connections is smaller, and one can find out the rule base by following the connections. Only those cells that form a rule are connected to each other.

Brown and Harris [7] note that a neurofuzzy system should only be used if its modelling and learning abilities are superior to a more conventional nonlinear model. The linguistic interface is useful for incorporating expert knowledge or for validating system behaviour, but a neurofuzzy system is better than a conventional model only if its numerical input-output behaviour is more appropriate.

4.2 Advantages of Neurofuzzy Systems

Combining fuzzy and neural systems may give several advantages over traditional systems especially for control purposes. Fuzzy systems are transparent and lend themselves easily to human-like reasoning. On the other hand, few modelling and learning theories exist for fuzzy systems. A neural network is able to learn from data and to perform massive parallel processing, but as a black-box approach a NN system is not easy to interpret. A combination of these systems may have both a qualitative and a quantitative interpretation and may avoid drawbacks of a solely fuzzy or neural approach.

For example, fuzzy controllers are sometimes required to process a large number of rules simultaneously. Enhancing the fuzzy rule processing with the parallel processing abilities of neural networks gives high computational efficiency.

A neural network can be used to find a feasible structure for a fuzzy controller. The number of rules, the formulation of the rules and the shapes of the membership functions affect the control performance of the system — all of these can be tuned using a neural learning scheme. Learning in a neural net means updating the network weight parameters. In a neurofuzzy system, these weights are just the parameters of the fuzzy system.

Although the fuzzy linguistic representation adds nothing to the modelling abilities of the network, it is a significant contribution because it allows the designer to initialize the system (that is, include expert knowledge) or validate it (check if an automatically generated system structure is feasible) [7]. Thus expert knowledge can be easily incorporated in a fuzzy system. In a neural network, expert knowledge can usually be used only in choosing the initial values of the network parameters.

4.3 Structure of a Fuzzy System in Neural Network Form

The structure of a fuzzy control system used in this thesis is presented in Figure 4.1 in neural network form. This is a common architecture for a neurofuzzy (more precisely, neural fuzzy) system, although other structures are also used. The architecture was presented by Berenji and Khedkar [6].

The first layer of a neurofuzzy network consists of measurements x_j of input variables

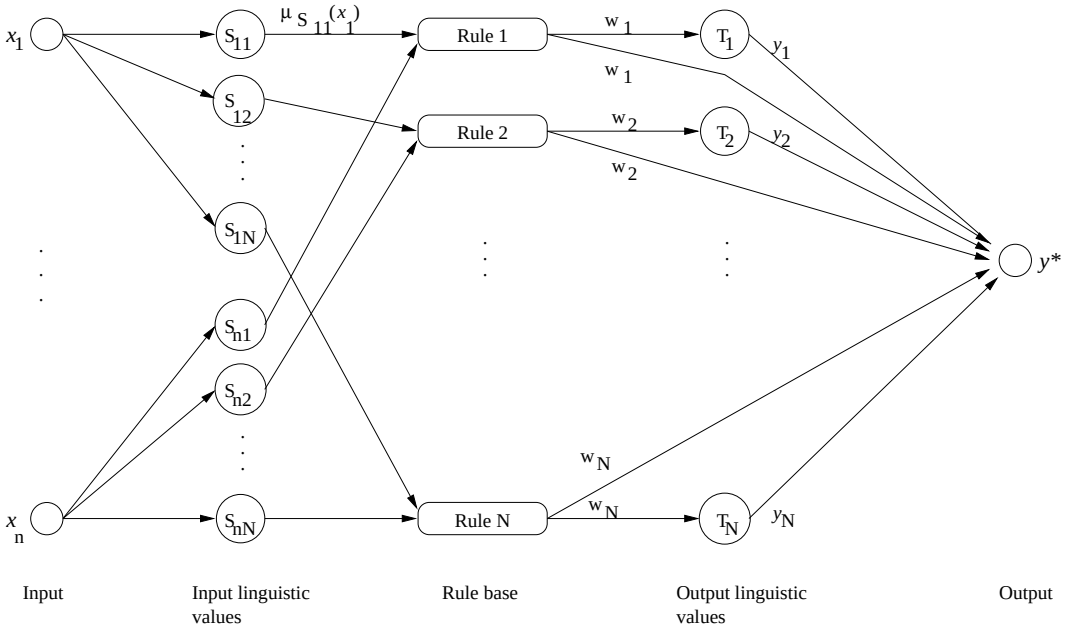


Figure 4.1: Structure of a fuzzy system in neural network form. The output of the second layer is the membership function value $\mu_{S_{ji}}(x_j)$, the output of the third layer is the rule firing strength w_i and the output of the fourth layer is the output y_i of Rule i. The weight parameters of this neural network are the shape parameters of the membership functions in layers 2 and 4. The weight parameters are not shown in the figure.

X_j, $j = 1, \ldots, n$. The second layer computes membership function values $\mu_{S_{ji}}$ for input linguistic values S_{ji}; for example, $\mu_{S_{1i}}(x_1)$ and $\mu_{S_{2i}}(x_2)$. Each first layer cell is connected only to those second layer cells which represent linguistic values for this particular input variable: X_1 is connected to S_{1i}, X_2 is connected to S_{2i}, and so on. Thus each second layer cell receives input only from one first layer cell, and the network weight parameters between the first and the second layer are not of the form a_{ij}, as in a typical neural network, but rather $\mathbf{p}_{ji}$, where $\mathbf{p}_{ji}$ is a parameter vector for the i:th membership function of the j:th linguistic variable, as membership functions usually require more parameters than just one. For example, a triangular membership function requires three parameters which correspond to the x coordinates of the corners of the triangle. Also note that the weight vector now affects the *form* of the membership function μ — in a neural network, the form of the activation function f (see Formula (3.1)) was fixed and the parameters a_{ij} were parameters of the *argument* of f.

Another thing to point out is that as mentioned in Section 2.5.1, the same linguistic values S_{ji} of a linguistic variable X_j may appear in several rules, making $S_{ji} = S_{ji'}$ for rules i and i'. An example of this is seen in Figure 4.2. For notational purposes rule i, $i = 1, \ldots, N$, seems to use linguistic values $S_{1i}, S_{2i}, \ldots, S_{ni}$ in Figure 4.1, although the same linguistic values may be used in another rule.

The third layer corresponds to a fuzzy rule base. A cell i in the third layer combines all the conditions "if X_1 is S_{1i} and X_2 is S_{2i} and ... " in the antecedent of rule i by using "and" and "or" operators (see section 2.3) on corresponding membership

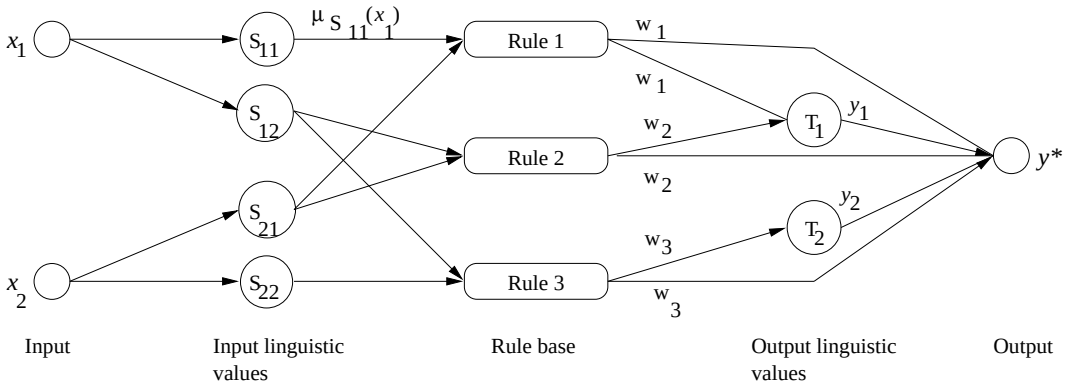


Figure 4.2: Example of a fuzzy system in neural network form. The same linguistic values appear in several rules.

function values. The output of cell i is w_i, the firing strength of rule i. No network weight parameters are used in third layer.

The fourth layer corresponds to rule consequents "then Y is T_i". Each cell represents a linguistic value for the output variable. As several rules may use the same linguistic values in their consequents, cells in layer 4 can have more than one input. In other words, the same linguistic value T_i may appear in several rules as in Figure 4.2. Again for notational purposes in Figure 4.1 each rule i has its own consequent linguistic value T_i. The fourth layer computes a defuzzified output of each rule. (Defuzzification is treated in Section 2.5.3.) The network parameters in this layer are membership function parameters $\mathbf{p}$ for output fuzzy sets.

The fifth layer combines the outputs of each rule and gives the system output value. Rule outputs are usually combined using some averaging method. No modifiable weights are used in the fifth layer.

4.4 Examples of Neurofuzzy Systems

In this section, examples of neurofuzzy systems are presented. The learning in these systems is supervised in the sense that a training data set of known input-output observations is always needed.

Lin and Lee [40] propose a general neurofuzzy system which combines unsupervised and supervised learning. An unsupervised learning algorithm is used to find clusters of data indicating the presence of fuzzy rules, as seen in Figure 4.3 a. Supervised learning is then used for the fine-tuning of the membership functions. As the training data is *a priori* classified, supervised learning performs better than without classification. Kosko [33] presents a *product space clustering* method with vector quantization, in which the input-output space is partitioned with a grid as in Figure 4.3 b. If several observations fall in a cell, a rule is placed in that cell. The grid can also be modified. The situation is quite similar to Lin and Lee's method in Figure 4.3 a, except that in Figure 4.3 a, a grid is not used.

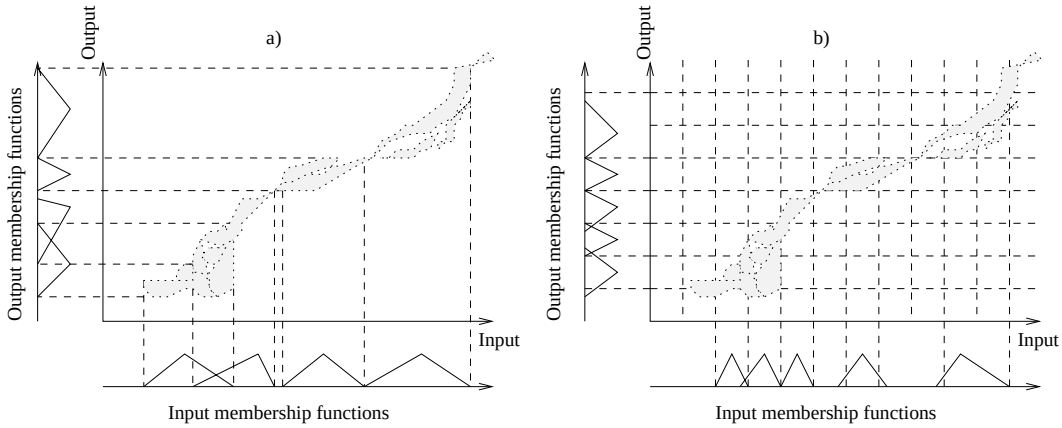


Figure 4.3: a) Clusters of data in the system input-output space indicate the presence of fuzzy rules. b) Product space clustering method.

Keller and Hunt [23] describe how a fuzzy neural network using fuzzy membership functions can distinguish linearly separable data sets. Data sets are linearly separable if they can be separated with a straight line such as in Figure 4.4, or, more generally, if points in n-dimensional space can be separated by a $(n-1)$-dimensional hyperplane.

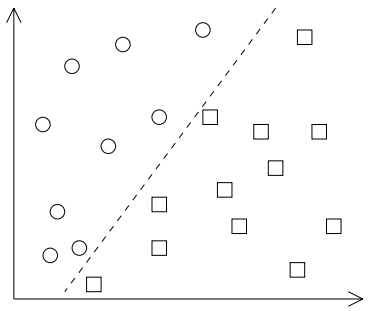


Figure 4.4: Linearly separable data sets can be separated by a straight line.

Hayashi *et al.* [12] present a fuzzy neural network where a classical feedforward neural network is equipped with fuzzy features. All real numbers that characterize a classical neural network (the inputs and outputs of each cell, and the weights in each layer) become fuzzy numbers in its fuzzified counterpart. The activation functions of each layer are the same as the ones used in the original neural network, but the weighted sum of inputs in Formula 3.1 must be calculated using fuzzy arithmetic. The error function and the backpropagation learning algorithm (Section 3.4) must be fuzzified, too. One way of fuzzifying the backpropagation algorithm is to replace real numbers with fuzzy numbers and apply fuzzy arithmetic.

Horikawa *et al.* [18] present different neurofuzzy network structures especially for modelling purposes. The network structures are categorized with respect to the type of the consequent of the fuzzy inference system. The consequent may either be a non-fuzzy constant: "if X_1 is S_1 and ... and X_n is S_n, then Y is c", a first order

linear equation: "if X_1 is S_1 and ... and X_n is S_n, then $Y = a_0 + a_1 X_1 + \ldots + a_n X_n$", or a fuzzy variable: "if X_1 is S_1 and ... and X_n is S_n, then Y is T". The authors propose a method for determining the network structure, that is, the combination of input variables and the number of membership functions in the antecedents and consequents. This is done using a backpropagation learning algorithm on the connection weights. After the learning, only the relevant connections between input and output variables are present. The network structure determines the rules of the fuzzy inference system. After the network structure has been selected, the membership function parameters can be modified using the backpropagation algorithm.

Jang [19] proposes an ANFIS (Adaptive-Network-based Fuzzy Inference System) architecture which uses both expert knowledge and input-output data pairs to fine-tune the membership functions. In this system, a hybrid learning scheme is used that combines the gradient descent technique (Section 3.4) and the least squares estimate to identify the parameters. The result is an input-output mapping which models the behaviour of the system. The ANFIS is more "fuzzy neural" than "neural fuzzy" as all possible connections in the network are used, and the network structure cannot be interpreted as a fuzzy rule base.

Klir and Yuan [27], Kosko [33] and Yager and Filev [67] also present methods for the construction and adaptation of the membership functions. All these methods require a training data set of known input-output observations. The situation is more cumbersome if observations of input-output pairs are not available. In this case, the learning must be based on some other criteria. This situation is treated in Chapter 5.

Extensive discussion about neurofuzzy systems can be found *e.g.* in Brown and Harris [7], Jang [19], Jang and Sun [20], Kosko [33] and Lin and Lee [40], [41].

Chapter 5

Reinforcement Learning

5.1 Introduction

Most neurofuzzy and other control systems require training data in the learning phase. The system learns to give a desired output if it has been presented with a lot of known input-output pairs. In this case, the control process is based on *tracking*: the objective is to follow a desired trajectory in the input-output space. When input-output data are not available and only some performance index is at hand, the control system must be based on other methods than tracking. Sutton *et al.* [63] divide control problems into two classes: 1) tracking problems and 2) optimal control problems. In the latter, the objective is to minimize or maximize a functional of system behaviour, and the functional is not defined in terms of a reference trajectory.

Sutton *et al.* [63] note that tracking problems assume prior knowledge of a reference trajectory, but in many problems, the determination of a reference trajectory is an important part — if not the most important part — of the overall problem. Sutton *et al.* also distinguish between *indirect* and *direct* adaptive control methods both for tracking and optimal control problems. An indirect method estimates an explicit model of the system at each step and determines the control rule from the model. Direct methods determine the control action without system modelling.

In *reinforcement learning*, the controller receives a signal of whether the previous control action was good or not. This signal may be available right after the action, or several time steps later, which makes the learning more challenging. Reinforcement learning methods are suitable for optimal control problems where input-output data is not available. They are *direct* methods: they do not identify a model of the system.

Reinforcement learning is based on the idea that if an action has good consequences, then the tendency to produce that action is strengthened, *i.e.*, reinforced. If the consequences are not revealed until several time steps later, the system must "predict" the goodness of the consequences. The theory of reinforcement learning originates from the studies on animal learning but it also has strong connections to adaptive optimal control theory. Lin and Lee [41] divide the history of reinforcement learning into two stages. In the first stage in the 1950's, computational models of human and animal learning were developed. The learning was seen as a kind of a stochastic pro-

cess. The second stage in the 1980's brought the concept of associative reinforcement learning, where an input pattern was associated with output patterns according to a reinforcement signal.

There are two types of reinforcement learning systems [63]: *actor-critic learning* and *Q-learning*. An actor-critic system contains two subsystems, one for choosing the optimal control action at each state (an actor) and another for estimating the long-term utility of each state (a critic). If a neurofuzzy system uses reinforcement learning, actor-critic learning is often chosen. A fuzzy control system is the actor who chooses the control action, and another system (usually a neural network) is the critic who evaluates the success of the control action chosen by the fuzzy system; this critic is used in the learning. A Q-learning system estimates utilities for all state-action pairs in the current state x. The evaluation of the utilities is based on the assumption that the system performs optimally in the future: "What is the utility of choosing action a in state x and performing optimally thereafter?" The system has to make some assumption about its future performance, and the optimality assumption is the most natural one.

Barto and Sutton have written many good introductions to reinforcement learning. See [2], [62] and [63].

5.2 Reinforcement Learning Versus Supervised and Unsupervised Learning

At first sight, reinforcement learning may sound similar to supervised or unsupervised learning algorithms (see Chapter 3) used in neural networks. Reinforcement learning can be categorized as one type of supervised learning. Nevertheless, there are substantial differences between reinforcement learning and both supervised and unsupervised learning.

In supervised learning, a "teacher" is able to provide the system with a desired output at each input pattern in the training sequence. After the system has given its output, the output and this desired response are compared and the difference is used when updating the parameters of the system. The contribution of individual elements to the output of the system is known, and parameters of each element are updated according to how much the element contributed to the output. On the contrast, reinforcement learning is used in problems where certain consequences of the control action are known, but no teacher can provide the desired responses of the individual elements.

Unsupervised learning systems do not need a "teacher" to tell the right output. They cluster information by giving a certain output at inputs belonging to a certain subset in the input space. Reinforcement learning algorithms are used in problems where the system must not just cluster information but must find the right actions at inputs belonging to different subsets in the input space.

An additional difference between reinforcement learning and both supervised and unsupervised learning is the stochastic search for the best output. As the environment is unable to provide desired outputs, the reinforcement learning system must

discover by itself which responses lead to improvements in performance [2]. The system calculates its output as a function of network weight parameters and input measurements, just as in a typical neural network, but this output is later deviated or "shaken" by a random amount. In this way, the system is able to try different kinds of outputs to see which ones are successful at which inputs. To summarize, the reinforcement learning system learns in a way human beings and animals do: by trial and error.

5.3 A Reinforcement Learning Algorithm for a Neurofuzzy Controller: GARIC

5.3.1 Introduction

In the following, a reinforcement learning algorithm for a neurofuzzy control system is presented. This algorithm, proposed by Berenji and Khedkar [6], is capable of minimizing an objective function using the reinforcement learning technique.

Berenji and Khedkar's GARIC (Generalized Approximate Reasoning -based Intelligent Control) system consists of two neural networks. One is a fuzzy controller and the other a neural predictor. Thus the reinforcement learning technique is of actor-critic type. Figure 5.1 describes the structure of the system.

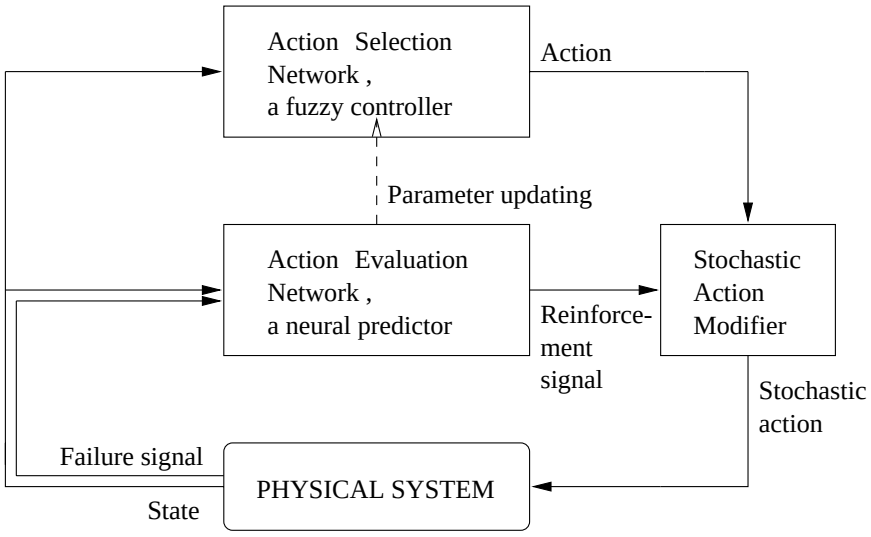


Figure 5.1: The architecture of GARIC. The Action Selection Network includes all the steps of fuzzy control discussed in Section 2.5.2 and in Figure 2.5.

The fuzzy controller or *actor* (*Action Selection Network*) chooses a suitable action based on the state of the system. The neural predictor or *critic* (*Action Evaluation Network*) produces an evaluation of the state and a prediction of a future reinforcement. In the learning phase, parameters of both networks are updated using the

information provided by the neural predictor.

Berenji and Khedkar implement their algorithm on a cart-pole balancing problem, also called an inverted pendulum problem. In this problem, an upright pole must be balanced by moving the cart on which the bottom of the pole is hinged. The cart may move along a finite-length track to its right or to its left. Both the pole and the cart may move only in the vertical plane [41]. The situation is depicted in Figure 5.2 [42]. This problem is used as an example of the application of various control algorithms, especially reinforcement learning algorithms [2], [6], [11], [41], [42]. The cart-pole balancing problem is a good example of a problem where reinforcement learning might be successful. The failure (the falling of the pole) occurs only after a long sequence of individual control decisions, and it is difficult to determine which decisions were responsible for the failure [2]. This problem of choosing which part of the system (for example, which parameters) should be updated in the learning phase is called a *credit assignment problem* [2]. It is a major problem in reinforcement learning.

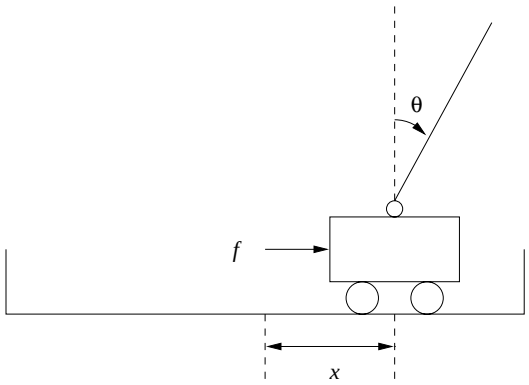


Figure 5.2: The cart-pole balancing problem [42]. The system tries to minimize the angle θ by applying a force f to the cart, whose displacement from the center of the track is x.

5.3.2 Action Evaluation Network, a Learning Network

The Action Evaluation Network (AEN) in Figure 5.3 is a neural predictor. It receives the state $\mathbf{x} = (x_1, \ldots, x_n)$ of the physical system as an input. The AEN produces a prediction of the state "goodness" v which is then coupled with the external error signal r to produce an *internal reinforcement* $\hat{r}$. The AEN is needed in order that the fuzzy controller can learn.

The Action Evaluation Network is a feedforward, multilayer perceptron -type network whose input layer and hidden layer are as presented in Section 3.2. The hidden layer activation function is chosen in [6] to be a sigmoidal function:

$$z_j(t, t+1) = \frac{1}{1 + \exp(\sum_{i=1}^{n} a_{ij}(t) x_i(t+1))}, \quad j = 1, \ldots, h. \tag{5.1}$$

The output layer receives input values both from the input layer (x_i, $i = 1, \ldots, n$)

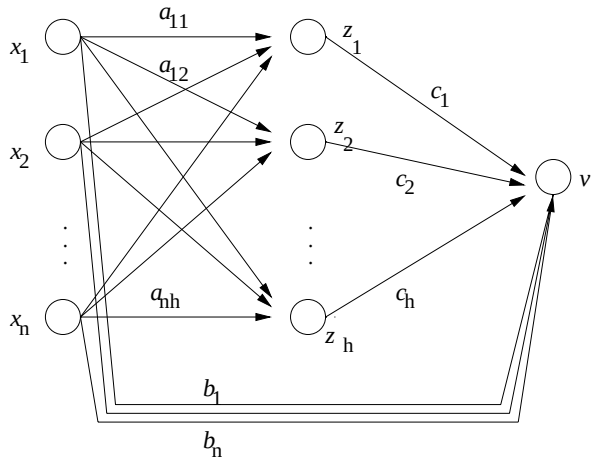


Figure 5.3: The Action Evaluation Network, a neural predictor [6].

and the hidden layer (z_j, $j = 1, \ldots, h$). The network output v is a measure of state goodness, a *prediction of future reinforcement*:

$$v(t-1,t) = \sum_{i=1}^{n} b_i(t-1)x_i(t) + \sum_{j=1}^{h} c_j(t-1)z_j(t-1,t). \tag{5.2}$$

The prediction of future reinforcement is utilized in the learning phase of the membership functions. This learning cannot be based on the difference between the actual and desired outputs as a desired output is not available, and therefore the learning is based on the control performance of the system. Again, the success of a control action is not revealed at the moment of action but one step later, and that is why a *prediction* of future reinforcement or a prediction of state goodness is needed.

The predicting capability of this neural network is based on adjusting the parameters a_{ij}, b_i and c_j, $i = 1, \ldots, n$, $j = 1, \ldots, h$. These are network weight parameters which do not have any physical interpretation.

Note that v depends on both t and $t-1$. The first time index refers to the parameters a_{ij}, b_i and c_j and the second to the state x of the system. When computing v, new inputs $x_i(t)$ have been obtained, but the network parameters $a_{ij}(t-1)$, $b_j(t-1)$ and $c_i(t-1)$ have not yet been updated. The updating may only take place after an evaluation of the network performance is obtained. A change in v may result either from a change in the network parameters or from a change in the state of the system, and thus both time indexes are needed. Writing v with double time indices as in (5.2) allows us to compare different v's over time and notice whether the system has moved to a better state or to a worse state [6]. Similarly to Formula (5.2), $v(t,t)$ is computed using parameter values a_{ij}, b_i and c_j from time step t and also the state x of the system at time step t.

The *internal reinforcement* $\hat{r}$ rewards the system of successful behaviour. Let the system move from a state with a low v (prediction of low reinforcement) to a state with a higher v (prediction of higher reinforcement). In other words, the state of the system improves. This positive change or internal reinforcement is used to reinforce the selection of the action which caused this move. The formula for internal

reinforcement is

$$\hat{r}(t) = \begin{cases} 0, & \text{start state} \\ r(t) - v(t-1, t-1), & \text{failure state} \\ r(t) + \gamma v(t-1, t) - v(t-1, t-1), & \text{else} \end{cases} \tag{5.3}$$

where $r(t)$ is an external performance measure caused by the parameters and input measurements at time $t-1$ but revealed at time t. Note that the internal reinforcement $\hat{r}$ is the larger the better the performance of the system. If $r(t)$ measures error (for example, delay) instead of performance, it must be included with a *negative* sign, whence a large error r causes a small value of $\hat{r}$.

In (5.3), future values of v are given less emphasis than the current v by using a discount rate $0 \leq \gamma \leq 1$. A failure state is such a state that the system needs to be rebooted after facing this state. In Berenji and Khedkar's example [6] — cart-pole balancing — the failure state is a state in which the pole falls down.

5.3.3 Action Selection Network, a Fuzzy Controller

The Action Selection Network is a fuzzy controller. Its structure is similar to a neurofuzzy network presented in Section 4.3 and in Figure 4.1. The steps involved in fuzzy inference were discussed in Section 2.5.2. Many of the steps may be realized in several different ways. In their article [6], Berenji and Khedkar use triangular membership functions (2.7). The rule firing strength w is the soft minimum 2.16 of membership function values, and the LMOM defuzzification (Formula (2.38)) is used. Thus the output $y^*(t)$ of the Action Selection Network is a weighted average of individual rule outputs y_i^*:

$$y^* = \frac{\sum_{i=1}^{N} w_i \, y_i^*}{\sum_{i=1}^{N} w_i}. \tag{5.4}$$

5.3.4 Stochastic Action Modifier

Berenji and Khedkar [6] propose a stochastic deviation to the output of the neurofuzzy controller Action Selection Network. This deviation is said to lead to a better exploration of the state space and a better generalization ability. Instead of the actual output $y^*(t)$, the control action applied to the system is $y^{*\prime}(t)$, a Gaussian random variable with mean $y^*(t)$ and standard deviation $e^{-\hat{r}(t-1)}$. The difference $|y^{*\prime}(t) - y^*(t)|$ is large when the internal reinforcement $\hat{r}(t)$ (5.3) is low, and small when $\hat{r}(t)$ is high. It means that when the last action taken by the controller was bad, the control action is deviated more, and when the previous action was good, only a small deviation is given. The numerical amount of deviation is

$$s(t) = \frac{y^{*\prime}(t) - y^*(t)}{e^{-\hat{r}(t-1)}} \tag{5.5}$$

and it is used as a learning factor when updating the parameters of the Action Selection Network.

5.3.5 Learning in the Action Evaluation Network

Learning in the Action Evaluation Network utilizes the value of the internal reinforcement $\hat{r}$. If a *positive* internal reinforcement signal is received, the network weights are rewarded by being changed in the direction which *increases* their contribution to the total sum. If a *negative* signal is received, the weights are punished by being changed in the direction which *decreases* their contribution [6].

The backpropagation algorithm (Section 3.4) is used in the learning. The intent is to maximize v, so the change in each parameter is proportional to the derivative of v with respect to this parameter. Internal reinforcement $\hat{r}$ (Formula (5.3)) is used as a learning factor.

The change in the weight parameters b_i between the input layer and the output layer is proportional to $\frac{\partial v}{\partial b_i} = x_i$. Thus the b_i are updated as

$$b_i(t) = b_i(t-1) + \beta \hat{r}(t) x_i(t-1); \quad i = 1, \ldots, n \tag{5.6}$$

where $\beta > 0$ is a constant.

The change in the parameters c_j between the hidden layer and the output layer is proportional to $\frac{\partial v}{\partial c_j} = z_j$. The update formula is

$$c_j(t) = c_j(t-1) + \beta \hat{r}(t) z_j(t-1, t-1); \quad j = 1, \ldots, h. \tag{5.7}$$

The weight parameters a_{ij} between the input layer and the hidden layer are updated using $\frac{\partial v}{\partial a_{ij}} = c_j z_j (1 - z_j) x_i$, resulting in

$$\begin{aligned} a_{ij}(t) = {} & a_{ij}(t-1) \\ & + \beta' \, \hat{r}(t) \, z_j(t-1, t-1) \, [1 - z_j(t-1, t-1)] \, \mathrm{sgn}(c_j(t-1)) \, x_i(t-1); \\ & i = 1, \ldots, n, \quad j = 1, \ldots, h \end{aligned} \tag{5.8}$$

where $\beta' > 0$ is a constant, and the *signum* function is defined as

$$\mathrm{sgn}(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0. \end{cases} \tag{5.9}$$

In [6], the hidden layer activation function z is a sigmoidal function (5.1) and its derivative is $z\,(1 - z)$ as seen in (5.8). The sign of c_j rather than its value is used, as the algorithm is thus more robust [6].

5.3.6 Learning in the Action Selection Network

The Action Selection Network produces a control action y^*. The objective of this action is to maximize v (the prediction of future reinforcement, Formula (5.2)) as a function of the parameters of the ASN. The gradient descent algorithm (see Section 3.4) is used in the learning.

The ASN has modifiable weights in the second layer (in the membership functions of the antecedent linguistic values) and in the fourth layer (in the membership functions of the consequent linguistic values). Let p_V be one of the parameters of the linguistic

value V. For example, if triangular membership functions are used, the parameters are p_{V1}, p_{V2} and p_{V3} which are the x coordinates of the corners of the triangle. For linguistic values in the antecedent of rule i, a notation $V \in \mathrm{Ant}(R_i)$ is used, where $\mathrm{Ant}(R_i)$ is the set of all linguistic values used in the antecedent of rule i. Similarly, for linguistic values in the consequent of rule i, $V \in \mathrm{Con}(R_i)$.

Learning in the Consequent Linguistic Values. Because we are maximizing v, the change in parameter p_V is proportional to $\frac{\partial v}{\partial p_V}$ [6]:

$$\Delta p_V \propto \frac{\partial v}{\partial p_V} = \frac{\partial v}{\partial y^*} \frac{\partial y^*}{\partial p_V}; \quad V \in \mathrm{Con}(R_i). \tag{5.10}$$

Here the consequent linguistic value V appears in rule i. The derivative $\frac{\partial v}{\partial y^*}$ is approximated as

$$\frac{\partial v}{\partial y^*} \approx \mathrm{sgn}\left(\frac{v(t,t) - v(t,t-1)}{y^*(t) - y^*(t-1)}\right) \tag{5.11}$$

and if $y^*(t) = y^*(t-1)$, $\frac{\partial v}{\partial y^*} = 0$.

Note that $y^*(t)$ must be different from $y^*(t-1)$ in order that any learning could take place. If the input x is the same at time instants t and $t-1$, the controller output y is the same, too, and no learning occurs, unless the Stochastic Action Modifier is used. As discussed in Section 5.3.4, the Stochastic Action Modifier helps learning by "shaking" the controller output y a bit, so that similar inputs x yield different outputs y.

Going back to Formula (5.10), in $\frac{\partial y^*}{\partial p_V}$, all rules i which use V in their consequent part have to be taken into account, so a summation over $V \in \mathrm{Con}(R_i)$ is done. (As discussed in Sections 2.5.1 and 4.3 and in Figure 4.2, several rules may use the same linguistic value in their consequents.) Furthermore, y^* is a weighted average of individual rule outputs y_i as in Formula (5.4), so we get

$$\frac{\partial y^*}{\partial p_V} = \sum_{V \in \mathrm{Con}(R_i)} \frac{\partial y^*}{\partial y_i} \frac{\partial y_i}{\partial p_V} \tag{5.12}$$

$$= \frac{1}{\sum_{j=1}^{N} w_j} \sum_{V \in \mathrm{Con}(R_i)} w_i \frac{\partial y_i}{\partial p_V} \tag{5.13}$$

where the summation of w_j in the denominator goes through all rules in the rule base, and $\frac{\partial y_i}{\partial p_V}$ depends on the choice of the membership function and the defuzzification method. These issues are discussed in Section 5.3.7.

Learning in the Antecedent Linguistic Values. The change in the parameter p_V in antecedent V in rule i requires a few more passes of the derivative chain rule:

$$\Delta p_V \propto \frac{\partial v}{\partial p_V} \tag{5.14}$$

$$= \frac{\partial v}{\partial y^*} \frac{\partial y^*}{\partial \mu_V} \frac{\partial \mu_V}{\partial p_V} \tag{5.15}$$

$$= \frac{\partial v}{\partial y^*} \left(\sum_{V \in \mathrm{Ant}(R_i)} \frac{\partial y^*}{\partial w_i} \frac{\partial w_i}{\partial \mu_V} \right) \frac{\partial \mu_V}{\partial p_V}; \quad V \in \mathrm{Ant}(R_i). \tag{5.16}$$

Here $\frac{\partial v}{\partial y^*}$ is again approximated as in (5.11). In $\frac{\partial y^*}{\partial \mu_V}$ on line (5.15), all rules i in which V appears must again be summed. Remember again that the same linguistic values may appear in several rule antecedents, as discussed in Sections 2.5.1 and 4.3 and in Figure 4.2. In the summation on line (5.16),

$$\frac{\partial y^*}{\partial w_i} = \frac{y_i + w_i \frac{\partial y_i}{\partial w_i} - y^*}{\sum_{j=1}^{N} w_j} \tag{5.17}$$

where the summation of w_j in the denominator again goes through all rules in the rule base. (In this formula there is an error in the article of Berenji and Khedkar [6], and this is the correct form.) The term $\frac{\partial y_i}{\partial \mu_i}$ depends on the defuzzification formula and is discussed in Section 5.3.7.

Going back to (5.16), the term $\frac{\partial w_i}{\partial \mu_i}$ depends on how the membership function values in rule i are combined, and the last term $\frac{\partial \mu_V}{\partial p_V}$ again depends on the choice of the membership function.

The learning rate depends on $\hat{r}(t)$ and $s(t)$ in addition to a small constant $\eta > 0$. A multiplicative learning factor $\hat{r}(t)s(t)$ is interpreted as follows: [6] If a large perturbation $s(t)$ results in a good action (that is, in a large internal reinforcement $\hat{r}(t)$), then the weights should receive a large reward, since the probabilistic search has really helped the system in this case. Conversely, if a large random deviation $s(t)$ is not beneficial (a small or negative $\hat{r}(t)$ is observed), it should have minimal effect on the weights [6]. As v is maximized, the total formula for parameter updates for both consequent and antecedent linguistic values is

$$\Delta p = \eta\, \hat{r}(t) s(t) \frac{\partial v}{\partial p} \tag{5.18}$$

where $\frac{\partial v}{\partial p}$ is computed using Formulae (5.11)–(5.17).

Note that if the input is outside the support (Formula (2.5)) of the membership function μ_V, no learning will occur for the parameters p_V of this μ_V. That is reasonable, since this V played no role in determining the control action for this particular input. In other words, the system fails to learn in those parts of the input space where there are not enough data available — a problem common in all neural and statistical learning algorithms. The algorithm proposed by Berenji and Khedkar tries to avoid this problem partially by using the stochastic action modifier, which randomly deviates the action chosen [6].

The gradient descent learning algorithm described here is often slow, but it can be enhanced by using a momentum term [15] or a line-search technique for determining the optimal step size at each point.

5.3.7 Notes on the Learning Algorithm

There are a few things worth mentioning concerning the learning formulae in Section 5.3.6. The form of the membership functions and the realization of the steps in fuzzy inference discussed in Sections 2.5.2 and 2.5.3 affect the formulae. For example, the formulae become different if one uses the minimum combiner instead of the product combiner in calculating the rule firing strength. In some cases no learning or only

very simple learning occurs. Attention should therefore be paid on the choice of the functions and methods used in the fuzzy inference.

We consider here triangular and trapezoidal membership functions. Although a triangle is simply a special case of a trapezoid, it is used in fuzzy control so often that it deserves attention of its own.

Of the steps involved in fuzzy inference, the rule firing strength is calculated here either using the soft minimum combiner or the product combiner. The minimum combiner is not discussed, because it is not differentiable. The defuzzification methods considered here are the LMOM, LCOA and a "mixture" of the LMOM and LCOA methods. These defuzzification methods include the steps of fuzzy implication and rule aggregation, so each step of the fuzzy inference presented in Sections 2.5.2 and 2.5.3 is discussed.

In this section we also discuss some details of the GARIC algorithm whose interpretation on the basis of Berenji and Khedkar's article [6] is not obvious.

Learning in the Consequent Linguistic Values. In the original article of Berenji and Khedkar [6], Formula (5.11) was presented in the form

$$\frac{\partial v}{\partial y^*} \approx \operatorname{sgn}\left(\frac{v(t)-v(t-1)}{y^*(t)-y^*(t-1)}\right). \tag{5.19}$$

where v has single time indices. As discussed in Section 5.3.2, double time indices are needed to distinguish the effect of state and the effect of network parameters. In $v(t,t)$, the first t refers to the parameters a_{ij}, b_i and c_j at time t and the second t refers to the state of the system at time t. We choose to use the time indices shown in Formula (5.11) for the following reasons: As the time indices of $y^*(t)$ and $y^*(t-1)$ refer only to the change in the state of the system, the time indices referring to the state of the system in v must also be t and $t-1$, whence the numerator in Formula (5.19) must be $v(?,t)-v(?,t-1)$. Now we must choose the time indices of the parameters a_{ij}, b_i and c_j. Remember that we use the reinforcement $\hat{r}$ in the learning in Formula (5.18). The reinforcement was defined in Formula (5.3) which shows that the reinforcement measures the change in v between successive states of the system (the second time index) but the parameters (the first time index) are kept fixed. We choose to follow this notation, and write the numerator as $v(t,t)-v(t,t-1)$.

Now we turn our attention to the other learning formulae, and see how the choice of the membership function and the defuzzification method affect the formulae. When updating the parameters in the membership functions of the consequent linguistic values, the derivatives $\frac{\partial y}{\partial p}$ are needed in (5.13). For triangular membership functions (2.7), the LMOM (2.38) defuzzification method gives

$$y = \frac{1}{2}(1-w)(p_1+p_3)+wp_2 \tag{5.20}$$

and the derivatives $\frac{\partial y_i}{\partial p_V}$ for consequent linguistic values V in Formula (5.13) are

$$\frac{\partial y}{\partial p_1} = \frac{1}{2}(1-w) \tag{5.21}$$

$$\frac{\partial y}{\partial p_3} = \frac{1}{2}(1-w) \tag{5.22}$$

$$\frac{\partial y}{\partial p_2} = w. \tag{5.23}$$

If the triangle is symmetric, its peak parameter p_2 can be expressed using p_1 and p_3 as $p_2 = (p_1 + p_3)/2$, and Formula (5.20) reduces to

$$y = \frac{p_1 + p_3}{2} \tag{5.24}$$

and the derivatives in Formula (5.13) reduce to

$$\frac{\partial y}{\partial p_1} = \frac{\partial y}{\partial p_2} = \frac{\partial y}{\partial p_3} = \frac{1}{2}. \tag{5.25}$$

If trapezoidal membership functions (2.9) are used, the LMOM method gives

$$y = \frac{1}{2}(1-w)(p_1 + p_4) + \frac{1}{2}w(p_2 + p_3) \tag{5.26}$$

and the derivatives are

$$\frac{\partial y}{\partial p_1} = \frac{\partial y}{\partial p_4} = \frac{1}{2}(1-w) \tag{5.27}$$

$$\frac{\partial y}{\partial p_2} = \frac{\partial y}{\partial p_3} = w. \tag{5.28}$$

Again, if the trapezoid is symmetric, then $p_1 = p_2 + p_3 - p_4$ and Formula (5.26) reduces to

$$y = \frac{p_2 + p_3}{2} = \frac{p_1 + p_4}{2} \tag{5.29}$$

and the derivatives $\frac{\partial y_i}{\partial p_V}$ in Formula (5.13) reduce to

$$\frac{\partial y}{\partial p_1} = \frac{\partial y}{\partial p_2} = \frac{\partial y}{\partial p_3} = \frac{\partial y}{\partial p_4} = \frac{1}{2}. \tag{5.30}$$

Thus if the LMOM defuzzification is used on symmetric triangles or trapezoids, the x coordinates of the corners change in the same way, either both to the left or both to the right by the same amount. A triangle or a trapezoid may move horizontally but its size and shape remain constant. Hence symmetric triangular and trapezoidal membership functions cannot utilize the full abilities of the learning algorithm.

If the LCOA (2.32) defuzzification is used with triangular membership functions, the crisp value of a triangle is its centroid (2.28), which is simply the average of the parameters p_1, p_2 and p_3:

$$y = \frac{1}{3}(p_1 + p_2 + p_3) \tag{5.31}$$

and the derivative formulae $\frac{\partial y_i}{\partial p_V}$ are

$$\frac{\partial y}{\partial p_1} = \frac{\partial y}{\partial p_2} = \frac{\partial y}{\partial p_3} = \frac{1}{3}. \tag{5.32}$$

From (5.32) we observe that the LCOA defuzzification leads to a very simple — if not too simple — learning rule on triangular membership functions in the rule consequent. Even if the triangle was asymmetric, all of its parameters change by the same amount, and the triangle cannot change in size or shape.

Moreover, if the triangle is symmetric, its centroid lies at the peak p_2, and the crisp value in (5.31) is always p_2. Thus one did not achieve anything on introducing symmetric triangular membership functions on output linguistic values — crisp values could have been used as well.

Using the LCOA method on trapezoids, the centroid (2.28) of a trapezoid is

$$y = \frac{1}{3} \frac{p_1^2 + p_2^2 + p_1 p_2 - p_3^2 - p_4^2 - p_3 p_4}{p_1 + p_2 - p_3 - p_4} \tag{5.33}$$

and the derivative formulae $\frac{\partial y_i}{\partial p_V}$ are

$$\frac{\partial y}{\partial p_1} = \frac{1}{3} \frac{2p_1 + p_2 - 3y}{p_1 + p_2 - p_3 - p_4} \tag{5.34}$$

$$\frac{\partial y}{\partial p_2} = \frac{1}{3} \frac{p_1 + 2p_2 - 3y}{p_1 + p_2 - p_3 - p_4} \tag{5.35}$$

$$\frac{\partial y}{\partial p_3} = -\frac{1}{3} \frac{2p_3 + p_4 - 3y}{p_1 + p_2 - p_3 - p_4} \tag{5.36}$$

$$\frac{\partial y}{\partial p_4} = -\frac{1}{3} \frac{p_3 + 2p_4 - 3y}{p_1 + p_2 - p_3 - p_4}. \tag{5.37}$$

Thus the LCOA defuzzification does not lead to overly simple learning rules on trapezoidal membership functions. An exception is a symmetric trapezoid, in which $p_1 = p_2 + p_3 - p_4$. Its centroid is always as in Formula (5.29), and the derivative formulae are as in Formula (5.30). We face the same problem as when using LMOM on symmetric trapezoids. Thus *asymmetric* trapezoidal membership functions should be preferred over any triangular membership functions, if the LCOA defuzzification is used.

A third defuzzification approach is a mixture of LMOM and LCOA. This method is discussed here because the fuzzy inference engine in our application uses it. The membership function of the consequent linguistic value in rule i is cut at level w_i, as in LMOM, and a centroid (Formula (2.28)) of the remaining set is calculated. The centroids are averaged using w_i as weights as in Formula (5.4) to get the final output y^* of the rule base.

As a membership function is cut at level w, its parameters change. Both a triangle and a trapezoid become trapezoids. Let $[p_1, p_2, p_3, p_4]$ denote the old parameters of a triangle or a trapezoid. Naturally, $p_2 = p_3$ for a triangle. The new parameters of the trapezoid are

$$p_1' = p_1 \tag{5.38}$$

$$p_2' = (1 - w)p_1 + wp_2 \tag{5.39}$$

$$p_3' = (1 - w)p_4 + wp_3 \tag{5.40}$$

$$p_4' = p_4. \tag{5.41}$$

Now the centroid of the trapezoid must be calculated using these new parameters, and the result is slightly different from (5.33):

$$y = \frac{1}{3}\frac{{p'_1}^2 + {p'_2}^2 + p'_1\,p'_2 - {p'_3}^2 - {p'_4}^2 - p'_3\,p'_4}{p'_1 + p'_2 - p'_3 - p'_4} \tag{5.42}$$
$$= \frac{1}{3}\frac{(w^2 - 3w + 3)\,(p_1^2 - p_4^2) + w^2\,(p_2^2 - p_3^2) + (3w - 2w^2)\,(p_1\,p_2 - p_3\,p_4)}{(2 - w)\,(p_1 - p_4) + w\,(p_2 - p_3)}.$$

The derivatives (5.34) to (5.37) change, too:

$$\frac{\partial y}{\partial p_1} = \frac{1}{3}\frac{(w^2 - 3w + 3)2p_1 + (3w - 2w^2)p_2 - 3y(2 - w)}{(2 - w)(p_1 - p_4) + w(p_2 - p_3)} \tag{5.43}$$
$$\frac{\partial y}{\partial p_2} = \frac{1}{3}\frac{(3w - 2w^2)p_1 + 2w^2 p_2 - 3yw}{(2 - w)(p_1 - p_4) + w(p_2 - p_3)} \tag{5.44}$$
$$\frac{\partial y}{\partial p_3} = -\frac{1}{3}\frac{2w^2 p_3 + (3w - 2w^2)p_4 - 3yw}{(2 - w)(p_1 - p_4) + w(p_2 - p_3)} \tag{5.45}$$
$$\frac{\partial y}{\partial p_4} = -\frac{1}{3}\frac{(3w - 2w^2)p_3 + (w^2 - 3w + 3)2p_4 - 3y(2 - w)}{(2 - w)(p_1 - p_4) + w(p_2 - p_3)}. \tag{5.46}$$

Again a symmetric triangle or trapezoid results in problems. The centroid is as in Formula (5.29) and the derivative formulae $\frac{\partial y}{\partial p_V}$ are as in Formula (5.30), and no changes in the size or shape of the membership function can occur. Again we have the same problem as we had with LMOM and LCOA using symmetric trapezoids or triangles.

As a summary, symmetric triangular or trapezoidal membership functions should not be used with the LMOM, LCOA or "LMOM plus LCOA" defuzzification. (Other defuzzification methods have problems discussed in Section 2.5.3, and the methods discussed here are considered as the best approaches.) Triangular membership functions should not be used at all with the LCOA defuzzification.

Learning in the Antecedent Linguistic Values. For antecedent linguistic values V, the derivative $\frac{\partial y_i}{\partial w_i}$ is needed in (5.17). It depends on the defuzzification formula. The LMOM defuzzification formula for triangular membership functions (5.20) gives

$$\frac{\partial y_i}{\partial w_i} = \frac{1}{2}(-p_1 + 2p_2 - p_3) \tag{5.47}$$

and the LMOM defuzzification formula for trapezoidal membership functions (5.26) gives

$$\frac{\partial y_i}{\partial w_i} = \frac{1}{2}(-p_1 + p_2 + p_3 - p_4). \tag{5.48}$$

Note that in a symmetric triangle, $p_2 - p_1 = p_3 - p_2$ and Formula (5.47) gives 0. Also in a symmetric trapezoid, $p_2 - p_1 = p_4 - p_3$, and again Formula (5.48) gives 0. So for both symmetric triangles and symmetric trapezoids, $\frac{\partial y_i}{\partial w_i} = 0$ if the LMOM defuzzification is used, and Formula (5.17) reduces to

$$\frac{\partial y^*}{\partial w_i} = \frac{y_i - y^*}{\sum_{j=1}^{N} w_j}. \tag{5.49}$$

If the LCOA defuzzification is used, then $\frac{\partial y_i}{\partial w_i} = 0$, too, on both triangular and trapezoidal membership functions (symmetric or not), as the centroids (5.31) and (5.33) do not depend on w_i. Again Formula (5.17) reduces to (5.49).

If a combination of LMOM and LCOA is used, the derivative $\frac{\partial y_i}{\partial w_i}$ in (5.17) is

$$\begin{aligned}\frac{\partial y_i}{\partial w_i} = &\frac{1}{3}\,\frac{(2w-3)(p_1^2-p_4^2)+2w(p_2^2-p_3^2)}{(2-w)(p_1-p_4)+w(p_2-p_3)} \\ &+\frac{1}{3}\,\frac{(3-4w)(p_1p_2-p_3p_4)+3y(p_1-p_2+p_3-p_4)}{(2-w)(p_1-p_4)+w(p_2-p_3)}\end{aligned} \tag{5.50}$$

where y is as in Formula (5.43). If the membership function is symmetric, the above formula reduces to zero, and we result in Formula (5.49).

Now we have two cases in which $\frac{\partial y_i}{\partial w_i} = 0$ and in which Formula (5.17) reduces to (5.49):

1. The membership functions are *symmetric* triangles or trapezoids, and the LMOM or "LMOM plus LCOA" defuzzification is used.
2. The membership functions are *any* triangles or trapezoids, and the LCOA defuzzification is used.

Furthermore, if only one rule i fires at a time, the control action y^* is just the output y_i of this particular rule i. The reason is that only one output set T_i is present in Formula (2.32) in Case 2 and in Formula (2.38) in Case 1, and the formulae give $y^* = y_i$. This results in $\frac{\partial y^*}{\partial w_i} = 0$. In this case, no learning occurs, as Δp_V in (5.16) reduces to 0. The problem can be avoided using overlapping input fuzzy sets and overlapping rules, as then several rules fire at the same time. Of course, during the learning the membership functions may again slide farther away from each other until they do not overlap any more.

In the learning formula (5.16) for antecedent linguistic values, other observations are in order, too. The derivative $\frac{\partial w_i}{\partial \mu_V}$ depends on how the membership function values in rule i are combined. If the "and" operation is interpreted as the soft minimum (2.16), then

$$\frac{\partial w_i}{\partial \mu_V} = \frac{e^{-k\mu_V}\,(1 - k\,\mu_V + k\,w_i)}{\sum_j e^{-k\mu_j}} \tag{5.51}$$

where the summation in the denominator goes through all membership function values in rule i as in (2.16), and k is the steepness factor of the soft minimum. If the "and" operation is calculated as the product (2.17) of the membership function values, then

$$\frac{\partial w_i}{\partial \mu_V} = \prod_{j \neq V} \mu_j \tag{5.52}$$

where j runs through all membership function values in rule i.

The last factor in (5.16), $\frac{\partial \mu_V}{\partial p_V}$, is for a triangular membership function (2.7)

$$\frac{\partial \mu_V(x)}{\partial p_1} = \begin{cases} \frac{x-p_2}{(p_2-p_1)^2}, & x \in [p_1, p_2] \\ 0, & \text{else} \end{cases} \tag{5.53}$$

$$\frac{\partial \mu_V(x)}{\partial p_2} = \begin{cases} -\frac{x-p_1}{(p_2-p_1)^2}, & x \in [p_1, p_2] \\ -\frac{x-p_3}{(p_3-p_2)^2}, & x \in (p_2, p_3] \end{cases} \tag{5.54}$$

$$\frac{\partial \mu_V(x)}{\partial p_3} = \begin{cases} \frac{x-p_2}{(p_3-p_2)^2}, & x \in [p_2, p_3] \\ 0, & \text{else} \end{cases} \tag{5.55}$$

and for a trapezoidal membership function (2.9)

$$\frac{\partial \mu_V(x)}{\partial p_1} = \begin{cases} \frac{x-p_2}{(p_2-p_1)^2}, & x \in [p_1, p_2] \\ 0, & \text{else} \end{cases} \tag{5.56}$$

$$\frac{\partial \mu_V(x)}{\partial p_2} = \begin{cases} -\frac{x-p_1}{(p_2-p_1)^2}, & x \in [p_1, p_2] \\ 0, & \text{else} \end{cases} \tag{5.57}$$

$$\frac{\partial \mu_V(x)}{\partial p_3} = \begin{cases} -\frac{x-p_4}{(p_4-p_3)^2}, & x \in [p_3, p_4] \\ 0, & \text{else} \end{cases} \tag{5.58}$$

$$\frac{\partial \mu_V(x)}{\partial p_4} = \begin{cases} \frac{x-p_3}{(p_4-p_3)^2}, & x \in [p_3, p_4] \\ 0, & \text{else.} \end{cases} \tag{5.59}$$

Triangular or trapezoidal membership functions $\mu(x)$ are not continuously differentiable with respect to x, and the derivatives do not exist at p_1, p_2, p_3 and p_4, since the two limiting values of μ are not equal at these points. Berenji and Khedkar [6] suggest a heuristic approach to solve the problem: an average of the two limiting values for the derivative are used at the singular points.

5.4 Other Actor-Critic Type Reinforcement Learning Algorithms

Barto *et al.* [2] introduced one of the first actual reinforcement learning algorithms. Learning in the algorithm is actor-critic type learning. The controller is not a fuzzy controller but a kind of a neural network. The article summarizes the properties of reinforcement learning in comparison with other learning algorithms.

Lin and Lee [41] present a reinforcement learning algorithm which can learn both the structure (that is, the rules) and the parameters (that is, the shape of the membership functions) of the fuzzy controller simultaneously. The structure learning starts with a network in which all possible connections between input and output linguistic values are present. During the learning, some of the connections are deleted, and the remaining ones constitute the rule base. The parameter learning is mainly the same as in the GARIC algorithm.

Lin and Lee's reinforcement learning algorithm is of actor-critic type, too. The structure of the control system is quite similar to GARIC: it consists of a fuzzy

controller (actor) and a fuzzy predictor (critic). One difference is that the predictor is now fuzzy in contrast to the neural predictor AEN in GARIC. Lin and Lee's algorithm can be extended to problems where the result of choosing an action is not revealed at the following time step but several time steps later, whereas in GARIC it is assumed that the result is revealed at the following time step.

Esogbue and Murrell [11] present a fuzzy adaptive controller which uses reinforcement learning neural networks. This algorithm partitions the state space into fuzzy sets using a generalization of Kohonen's self-organizing map [28], [29], [30]. In addition, the algorithm finds the structure of the fuzzy controller, but it does not actually tune the membership functions.

Other reinforcement learning algorithms of actor-critic type are found in [8] and in [42].

Chapter 6

Neurofuzzy Traffic Signal Control

6.1 Traffic Signal Control

6.1.1 Objectives of Traffic Signal Control

The main reason for introducing traffic signals in an intersection is traffic safety. Traffic signalization increases safety by keeping conflicting traffic streams apart. After the decision of the introduction of traffic signalization in an intersection is made, several goals can be set: minimization of vehicular and/or pedestrian delay and stops, minimization of queue lengths, maximization of safety and/or comfort for both vehicles and pedestrians, minimization of environmental impacts, and so on. Traditionally, the main performance measure of signal control systems has been the reduction of vehicular delay and stops [1], as the other objectives have been difficult to measure.

Concentrating on the delay minimization (which in our approach includes stop minimization) may in the long run impair safety and harm the environment. Each of these three objectives reaches its optimum state on a different signal cycle length, and optimizing one of them may lead further away from the optima of the remaining two. On the other hand, the delays and stops affect both safety and environmental aspects: braking and stopping increase the risk of rear-end collisions; the delay increases the time spent in traffic and thereby increases pollution; braking and accelerating associated with stopping increase the fuel consumption and pollution.

A special performance index PI is used to measure the effectiveness of the traffic signal control. Delays and stops are combined as

$$PI = d + \alpha p \tag{6.1}$$

where d is the average delay of vehicles, p is the probability (or frequency) of stops and α is a scaling factor. Using different α, different goals can be set, as either delays or stops can be emphasized. Using a large α the performance index emphasizes safety and environmental aspects, as these are mainly associated with vehicular stopping. Using a small α, the interest is placed on delay minimization.

In this thesis, the aim of the control system is delay minimization. We define delay

as the difference between a vehicle's real travel time and its desired travel time [34]:

$$d = t_{tr} - \frac{s_{tr}}{v_{des}} \tag{6.2}$$

where d is the vehicular delay in seconds, t_{tr} is the actual travel time of the vehicle, s_{tr} is the travel distance and v_{des} is the desired speed level. The desired travel time is the travel distance divided by the desired speed. The desired travel time assumes that other vehicles or signal controllers do not disturb the vehicle's travel. Thus the delay consists partly of congestion and partly of extra waiting time at a signalized intersection, but these are not separated.

6.1.2 Control Procedures

The traffic environment used in this work is seen in Figure 6.1. It is an intersection of two one-way streets. One of the oldest applications of fuzzy control was Pappis and Mamdani's simulation [52], [53] in which the intersection configuration was the same as here.

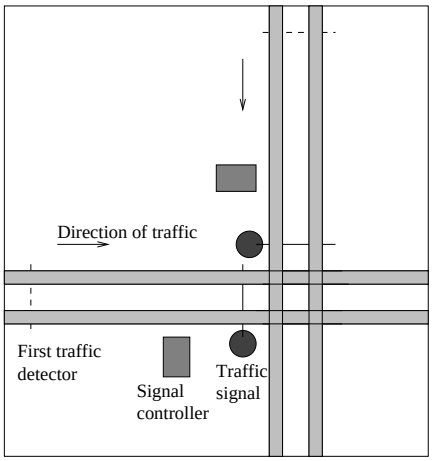


Figure 6.1: Intersection configuration

In Figure 6.1, both streets have two lanes of equal traffic volume. For example, a traffic volume of 300 vehicles per hour means that in both approaching directions, in both lines approximately 150 vehicles are generated per hour, making a total of 300 vehicles per hour in each direction. In each approaching lane there are two traffic detectors, the first one is usually 100 metres before the stop line and the other is always at the stop line. These detectors send measurements of traffic to the signal controller.

As only one intersection is studied, the traffic control process is *isolated signal control*. In isolated signal control, the signal timing is independent and does not co-operate with any other intersection. A definition given by Traffic Control Systems Handbook [64] for isolated signal control is a "*form of signal control for a single signalized intersection through which the flow of traffic is controlled without giving any consideration to the operation of adjacent signalized intersections*".

The traffic at a given moment is either in longitudinal or in crosswise direction. Right- or left-turning vehicles are not treated separately in signalization, nor is there any pedestrian signal control. This means that the intersection is a *two-phase* controlled intersection. Over 40 per cent of all traffic signal controllers worldwide are two-phase controllers.

As traffic detectors are used to measure the traffic in both directions, the control process is called *vehicle-actuated*. This control system is feedback control, as real-time information about the physical situation is used. The controller can use any predetermined strategy based upon the traffic situation when choosing the signal timings.

Before traffic detectors were used, the traffic signal control systems have been *pre-timed*, open-loop controllers. In those systems, the length of each phase is fixed: each approaching direction receives its own, pre-specified length of green, no matter how many vehicles there are. Even if the approaching direction was empty, the control system gives it the usual amount of green, and the vehicles in the queuing direction have to wait. Fixed-time control performs optimally only if the traffic volumes in each direction are constant and do not display any random variation, which is seldom the case.

Vehicle-actuated control is nowadays the most usual traffic signal control procedure. A type of vehicle-actuated control used in Finland is the *extension principle*, in which a minimum green time is given first and the current green time is extended if needed. The philosophy is to discharge the queue, to let the approaching vehicles pass the intersection and avoid unnecessary stopping. An upper limit of maximum green time ensures that vehicles queuing in the other direction do not have to wait intolerably long. The extension principle is not mathematically the most optimal control strategy. Optimization becomes tedious as several signal controllers are linked with each other and the number of variables increases. There are also many practical restrictions on signal timings. Due to safety reasons, nearby intersections must have the same signal control strategies, cycle lengths and phase ordering. As a result, signal control planning is usually based on expert knowledge and tailor-made solutions.

Fuzzy traffic signal control is one type of vehicle-actuated, extension-based control, but it differs from the usual vehicle-actuated, extension-based control used in Finland in a few major points. First, the number of parameters is much smaller in fuzzy signal control than in traditional vehicle-actuated control. In a traditional controller the number of parameters may be as much as 600, many of which are on/off-parameters. A traffic control engineer can effectively handle only a portion of them. In a fuzzy signal controller, the parameter set consists mainly of membership function parameters. The number of parameters is smaller and they are easier to comprehend, making the design process more suitable for human-like reasoning.

The second difference between fuzzy and traditional vehicle-actuated, extension-based traffic signal controllers is that a fuzzy controller uses the measurements of incoming traffic in both the green and the red direction, whereas a traditional controller uses only the information about vehicles in the green direction when deciding the green time extension. A problem with the extension principle is that when the vehicles approach the intersection separately but within a few seconds from each other, every vehicle is given a green light extension, and the total extension grows

very large. Fuzzy control takes into account the length of the queue behind the red signal, too, and if the queue is too long compared to the amount of vehicles approaching from the green direction, no green extension is given anymore.

Fuzzy control is suitable for traffic signal control systems, as the basic control rules are quite intuitive and can be easily extracted from expert knowledge. In many other control problems, for example in process industry, the control rules are more difficult to find as the physical process is not as intuitive.

6.1.3 FUSICO — a Fuzzy Traffic Signal Controller

A traffic simulation system HUTSIM [34], [36], [59] has been developed at Helsinki University of Technology, Laboratory of Transportation Engineering. The HUTSIM simulation system consists of a simulation software which runs on a PC and which interacts with either a real traffic signal control device or an internal signal control program. The simulation program creates traffic, propagates it through the intersection or road network in the simulation environment and generates detector inputs for the signal controller. The controller reacts to the detector inputs according to a user-designed control scheme and sends the traffic signals to the simulator. The vehicles react to these signals. Several measures of signal control effectiveness are computed and stored in external files.

In the HUTSIM system, various different traffic conditions and traffic control strategies can be simulated. A fuzzy controller FUSICO [46] is also available. It contains the fuzzy rules and membership functions, and it evaluates the rules using fuzzy set operations. In the FUSICO system, the traffic detectors give the following input variables to the fuzzy controller: APP, number of approaching vehicles (vehicles between detectors of green phase) and QUE, number of queuing vehicles (vehicles between detectors of red phase). Both input variables assume only nonnegative integer values. Depending on the traffic situation, the green phase can be extended with one or several seconds, and the output of the fuzzy controller is EXT, green time extension (in seconds). The rule base of FUSICO is as follows [46]:

after minimum green (5 s)

if APP is *zero*, then EXT is *zero*
if APP is *a few* and if QUE is *a few*, then EXT is *short*
if APP is *more than a few*, then EXT is *medium*
if APP is *medium*, then EXT is *long*

after the first extension

if APP is *zero*, then EXT is *zero*
if APP is *a few* and if QUE is *a few*, then EXT is *short*
if APP is *medium*, then EXT is *medium*
if APP is *many*, then EXT is *long*

after the second extension

if APP is *zero*, then EXT is *zero*
if APP is *a few* and if QUE is *a few*, then EXT is *short*
if APP is *medium* and if QUE is *less than medium*, then EXT is *medium*
if APP is *many* and if QUE is *less than medium*, then EXT is *long*

after the third extension

if APP is *zero*, then EXT is *zero*
if QUE is *too long*, then EXT is *zero*
if APP is *more than a few* and if QUE is *a few*, then EXT is *short*
if APP is *medium* and if QUE is *less than medium*, then EXT is *medium*
if APP is *many* and if QUE is *less than a few*, then EXT is *long*

after the fourth extension

if APP is *zero*, then EXT is *zero*
if QUE is *too long*, then EXT is *zero*
if APP is *more than a few* and if QUE is *a few*, then EXT is *short*
if APP is *medium* and if QUE is *less than a few*, then EXT is *medium*
if APP is *many* and if QUE is *less than a few*, then EXT is *long*

In this thesis, the rule base is used "as is" and the intent is not to modify it systematically. The rule base consists of five rule sets. The choice of the rule set depends on how many green extensions have already been given. Some of these rule sets do not cover the whole input space, and for those inputs which do not fire any rule, the output of the controller is 0.

6.2 Neural Learning in Fuzzy Traffic Signal Control

6.2.1 Structure of the Neurofuzzy Control System

The structure of the neurofuzzy traffic signal control system used in this work is presented in Figure 6.2. This structure is quite similar to the fuzzy controller in Figure 2.5 but it has a few additions: A *traffic situation model* based on information about the physical system is created in the simulation program, and the variables of

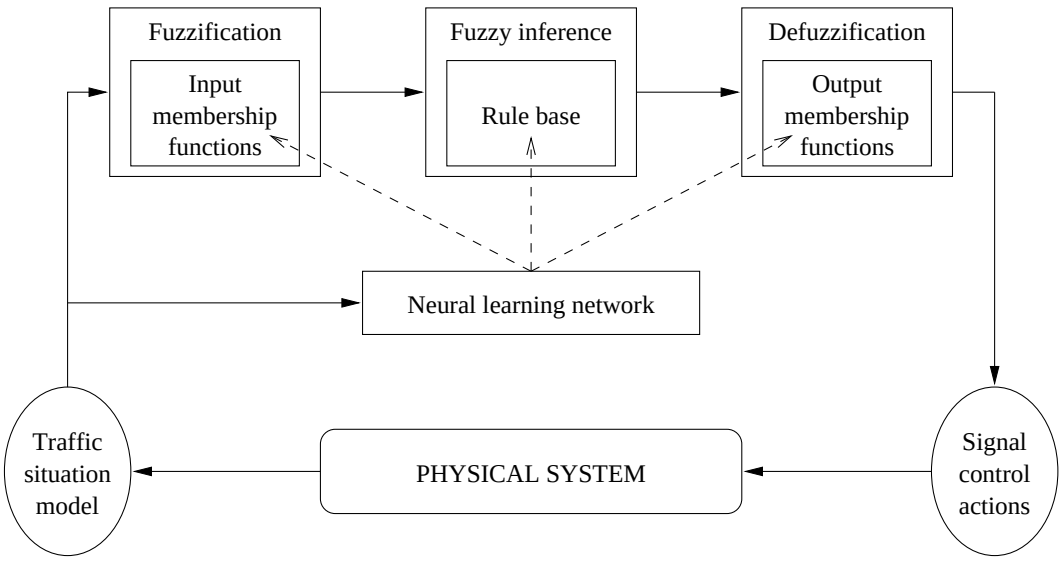


Figure 6.2: Fuzzy traffic signal control system FUSICO enhanced with a neural learning network

this model are fed to the fuzzy controller. After the fuzzy inference process, the crisp output of the fuzzy controller is converted to a "real" *signal control action* which is applied to the physical system. A *neural learning network* uses the variables of the traffic situation model, and it suggests parameter updates in the input and output membership functions of the fuzzy controller.

The original fuzzy controller is presented as a neural network, too, so that its parameters can be updated. The general structure of a fuzzy controller in neural network form was seen in Figure 4.1 in Section 4.3. The network includes the fuzzification, fuzzy inference and defuzzification parts of the fuzzy controller. It is now important to note that the neurofuzzy traffic signal control system used in this thesis contains *two* neural networks: the controller network and the learning network. The reason why a separate learning network is needed is that the most usual supervised neural learning algorithms cannot be used in the current control problem. If they could be used, it would suffice to update the parameters of the controller network by itself. In this thesis a more complex algorithm is needed, and the GARIC reinforcement learning algorithm presented in Section 5.3 is used.

The FUSICO fuzzy traffic signal controller has a rule base which contains five separate rule sets, and the choice of the rule set depends on the situation. The structure of the fuzzy controller in neural network form thus also varies, and there are five alternatives, seen in Figures 6.3 and 6.4. The alternatives are quite similar to each other. The weight parameters of the network are the shape parameters of the membership functions in layers 2 and 4, and these are not shown in Figures 6.3 and 6.4.

The linguistic variables in the neurofuzzy controller are the input variables APP and QUE and the output variable EXT. The linguistic values or fuzzy sets for linguistic variable APP are noted $Azero$, $Aafew$, $Amedium$ and $Amany$; the corresponding membership functions are noted μ_{Azero}, μ_{Aafew}, $\mu_{Amedium}$ and μ_{Amany}. The linguistic values for QUE are noted $Qafew$, $Qmedium$ and $Qtoolong$; the corresponding membership functions are μ_{Qafew}, $\mu_{Qmedium}$ and $\mu_{Qtoolong}$. The linguistic values for EXT are noted $Ezero$, $Eshort$, $Emedium$ and $Elong$ and their membership functions μ_{Ezero}, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong}. Observe that linguistic values *zero*, *a few* and *medium* are used with more than one variable, but they are not the same linguistic values. That is why the notation $Azero$, $Ezero$, and so on, is used.

Linguistic values of the form "*more than V*" and "*less than V*", where V is a linguistic value, are also used in the rule base. These are interpreted as functions of V, not as independent values. The membership functions of *more than V* and *less than V* are

$$\mu_{morethan(V)}(x) = \begin{cases} 1 - \mu_V(x), & x \geq \sup M \\ 0, & \text{else} \end{cases} \tag{6.3}$$

$$\mu_{lessthan(V)}(x) = \begin{cases} 1 - \mu_V(x), & x \leq \inf M \\ 0, & \text{else} \end{cases} \tag{6.4}$$

where

$$M = \{x \in X \,|\, \mu_V(x) = \max_x\{\mu_V(x)\}\}. \tag{6.5}$$

Figure 6.5 shows a trapezoidal membership function μ_V together with M and the membership functions for *less than V* and *more than V*.

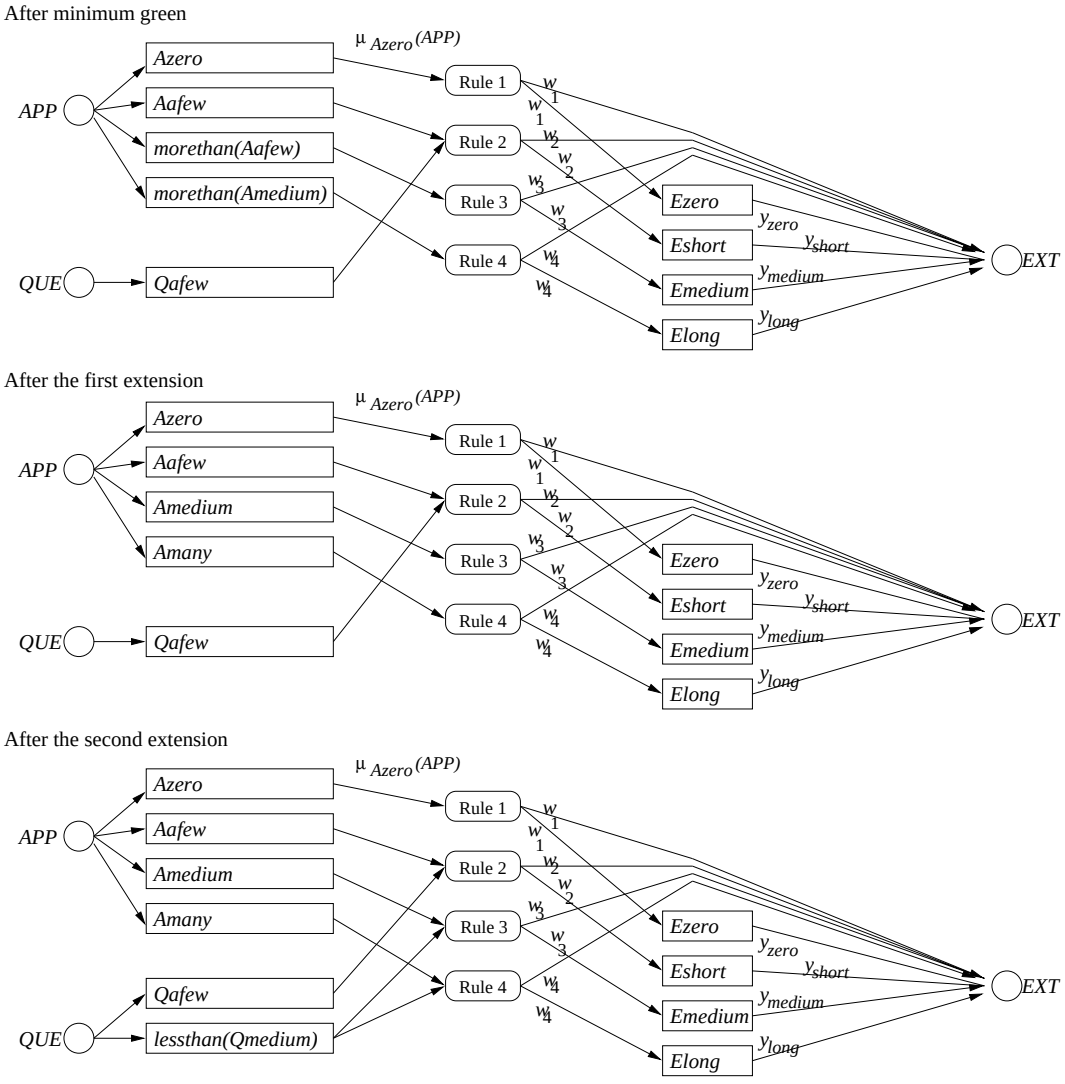


Figure 6.3: Fuzzy traffic signal controller presented as a neural network. The first three rule sets.

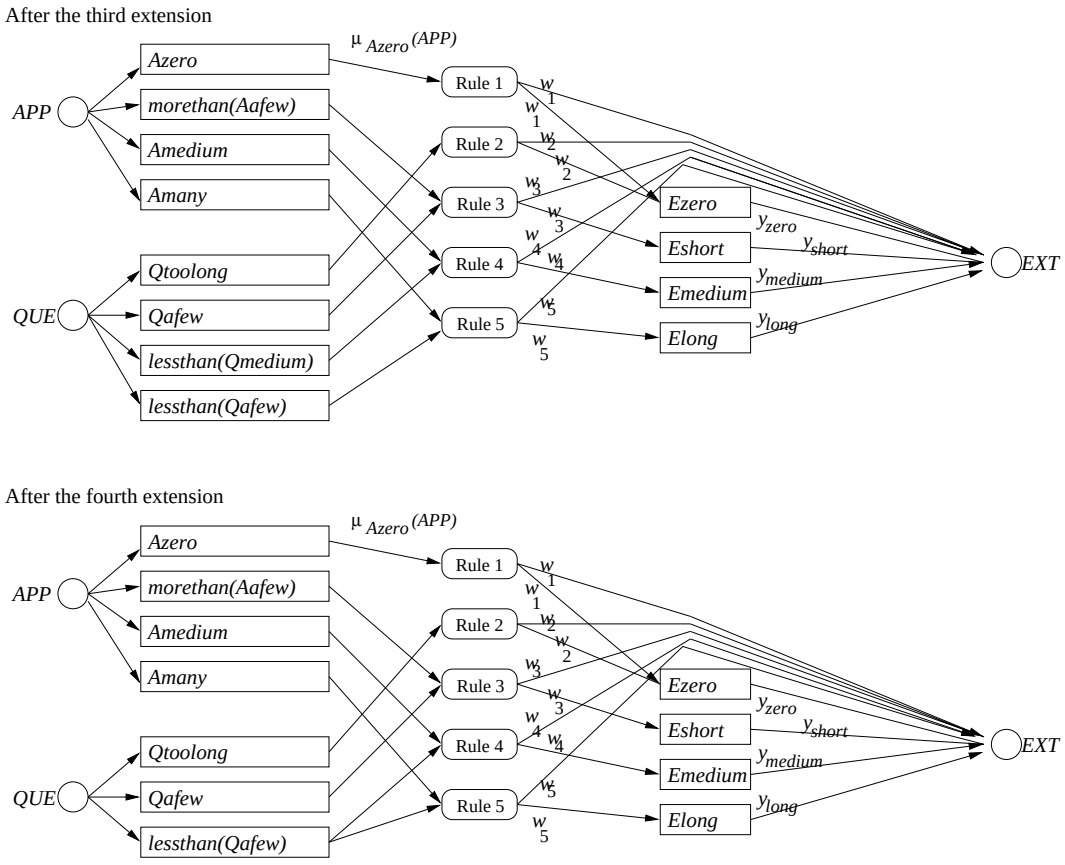


Figure 6.4: Fuzzy traffic signal controller presented as a neural network. The last two rule sets.

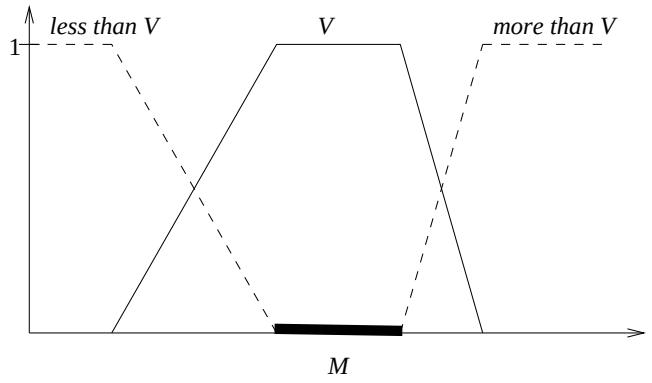


Figure 6.5: Membership functions for linguistic values V, *less than* V and *more than* V. The thick line shows M (Formula (6.5)).

6.2.2 Why Use the GARIC Algorithm?

In this work, a fuzzy traffic signal controller is equipped with neural learning capabilities. The aim is to minimize delays, and a way to achieve this is to find a proper control action *EXT*. The control action is given as an output of the fuzzy control network whose structure and parameters affect the output. Thus a way to minimize the delays is to adjust the structure and/or membership function parameters of the fuzzy control network. In this thesis, the structure of the fuzzy control network is kept constant and only membership function parameters are adjusted.

The parameter updating or learning in neural networks is usually based on observing the difference between the network output and the desired output. In this application, no training data of proper control actions is available. That is, given an input (*APP, QUE*) of the fuzzy controller, one does not have a target *EXT*, a "best" possible control action for this particular input; the best one would be the one which minimizes the delay, and that is what we are searching for. The control system is not trying to follow any desired trajectory in the input-output space. Therefore the learning must be based on some other criteria than an error in *EXT*, and standard neural network supervised learning algorithms cannot be used.

A simple measure of error in the traffic control system is delay. How could it be used in the parameter updating? We do not construct a neural network whose output is the delay, because we do not know which input variables contribute to it — there must be many more than just *APP* and *QUE*. Also, we do not have any desired delay, as a zero delay is often physically impossible. Instead, we construct a neural network which uses the delay in addition to *APP* and *QUE* and produces information on how the parameters of the controller network should be modified. This is what Berenji and Khedkar's [6] GARIC algorithm in Section 5.3 and other reinforcement learning algorithms do. They combine two neural networks, a fuzzy control network and a learning network. This learning network evaluates the state of the system and uses delay as a performance measure. It produces a measure of how the parameters of the control network, that is, the membership functions of the fuzzy controllers, should be modified.

The reason for using the GARIC algorithm is thus the nature of the information available in the control problem. Other reinforcement learning algorithms could be used, too, but GARIC was one of the first of its kind. Some algorithms [41] present methods for constructing the rule base, but in our approach the rule base is already available and the learning algorithm does not modify it. Some reinforcement algorithms [41] are capable of constructing a *multi-step* prediction of the future reinforcement v which is used in the situations when the success of a control action is not revealed until several time steps later. In the traffic control problem, the success of a control action (the delay) is revealed right after the action, so a multi-step prediction is not needed, and a single-step prediction v (Formula (5.2) is enough.

6.2.3 Realization of the GARIC Algorithm

Berenji and Khedkar's reinforcement learning algorithm GARIC (see Section 5.3) is used to adjust the membership functions of the fuzzy traffic signal controller. The architecture of GARIC was shown in Figure 5.1. As the GARIC algorithm is applied to

traffic signal control, the Action Selection network is the fuzzy traffic signal controller and the Action Evaluation Network is the neural learning network. The Stochastic Action Modifier is not used due to reasons explained later. The architecture corresponds to the architecture of a neurofuzzy traffic signal controller in Figure 6.2 because the Action Selection Network comprises all the steps of fuzzification, fuzzy inference and defuzzification.

The GARIC algorithm is implemented as a Matlab program which interacts with the HUTSIM simulation system. The HUTSIM system simulates the traffic and fuzzy signal control for a period of time, using membership functions stored in an input file presented in Appendix A.1. During the simulation, the fuzzy controller is used continuously, so each simulation run contains several control actions. The results of the simulation are written to an output file that can be seen in Appendix A.2. The Matlab program reads the output file and performs one step of neural network training using the GARIC reinforcement learning algorithm. New membership function parameters are computed and values of membership functions are stored in the input file seen in Appendix A.1. HUTSIM uses this file during a new simulation period. This two-stage loop is repeated a few hundred times. The experimental setting is further discussed in Section 7.5. Figure 6.6 describes the interaction between HUTSIM and Matlab.

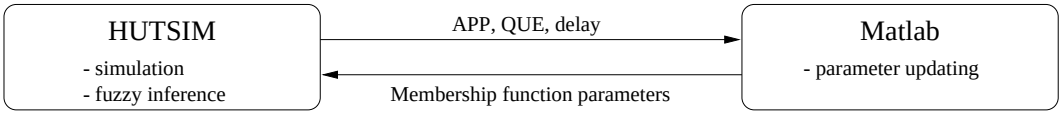


Figure 6.6: Interaction between HUTSIM and Matlab.

In practice, the fuzzy inference process of HUTSIM must be imitated in Matlab in order to be able to update the parameters according to the GARIC algorithm. The output file of the simulation run (see Appendix A.2) contains observations of the input values APP and QUE used in each signal control decision, the number the extension (this determines the rule set used), the actual value of extension EXT given and the amount of delay for each particular vehicle. The Matlab program goes through every observation of signal control action and runs the reinforcement learning algorithm at each observation. The amount of update of the parameter set is calculated for each observation. After computing the amount of update for all observations, the actual update is the average of the individual updates. Note that the parameters are updated only after all observations of the simulation run are processed — we must use the same parameters that were used in HUTSIM, and in HUTSIM the parameters were fixed during the whole simulation run.

In the following, the realization of the subsystems of the GARIC algorithm is discussed.

Action Evaluation Network. The Action Evaluation Network, the learning network, is used as was presented in Section 5.3.2. The input of the network is the difference of the state (APP, QUE) of the system and a reference state. The reference value of APP is the average of APP values observed in the first simulation run, and correspondingly for QUE. This kind of input assumes both negative and

positive values.

The values of v and $\hat{r}$ in Formulae (5.2) and (5.3) are computed at each observation. The state evaluation v in Formula (5.2) corresponds to how well the system has avoided delays. If the delays are avoided successfully, v is large. The external performance measure r in the formula for internal reinforcement (5.3) is the change in the delay between successive simulation runs. Since the internal reinforcement increases as r increases, r must measure the "success" and not the "error"; therefore, r is computed as the difference of delay between the previous run and the current run. If the delay is decreased, r is positive and if it is increased, r is negative. A failure state in (5.3) is not encountered in traffic signal control.

It is important to note that delays cannot be assigned to individual signal control actions, as the delay is measured after the vehicle exits the model and not at the moment of the control action. The input variables *APP* and *QUE* of the control action are measured at that moment, but we do not know *which* particular vehicles they consist of, and we cannot assign the delay of these vehicles to the control action. Appendix A.2 shows an output file of the HUTSIM simulation system. This file lists the time instants when vehicles enter and exit the model, the delays of these vehicles and the times of signal control actions together with the input and output variables *APP*, *QUE* and *EXT* of each signal control decision. We can only assign a "pooled" delay to each action: the delay caused by a control action is the average of the delays of those vehicles which, according to their enter and exit times, have any possibility of belonging to the *APP* or *QUE* of this control action.

In the learning phase of the Action Evaluation Network, formulae presented in Section 5.3.5 are used. The momentum method presented in Section 3.4 in Formula (3.20) is used in the learning formula of the parameters b_i between the input layer and the output layer. Thus Formula (5.6) changes to

$$b_i(t) = b_i(t-1) + \beta\hat{r}(t)x_i(t-1) - \alpha(b_i(t-1) - b_i(t-2)); \quad i = 1, \ldots, n \tag{6.6}$$

and the momentum parameter $\alpha = 0.9$ is used. The parameters β and β' have values $\beta = 0.1$ and $\beta' = 0.3$.

Action Selection Network. The Action Selection Network (Section 5.3.3) is the fuzzy traffic signal controller. In practice, the fuzzy inference is done in the HUTSIM simulation system and our algorithm just reproduces it for learning purposes. The controller processes rules of the form "if *APP* is *a few* and *QUE* is *a few*, then *EXT* is *short*". Berenji and Khedkar [6] proposed the soft minimum (2.16) combiner with the steepness parameter $k = 10$ for the "and" operator. Thus they computed the rule firing strength w as the soft minimum of the membership function values in each rule. Although the product operator (2.24) would be better than the soft minimum because it does not lose any information, as discussed in Section 2.3, the soft minimum combiner is used in this work, too. The reason is that the HUTSIM simulation system uses the minimum operator (2.15), and the fuzzy inference used in HUTSIM cannot be changed. As the minimum function is not continuously differentiable and we need differentiability in the learning phase in Formula (5.16), we approximate minimum by the soft minimum using $k = 100$. (This value of k is better than Berenji and Khedkar's [6] $k = 10$, as was discussed in Section 2.3.)

In each rule, the membership function of the output variable EXT is cut at level w as in Formula (2.23) and in Figure 2.7 a, and the center of area of the remaining set is calculated using Formula (2.28). A weighted average of the centers of area of each rule is calculated, the weights being the rule firing strengths w. Thus the defuzzification is a "mixture" of the LCOA and LMOM defuzzification methods presented in Section 2.5.3. Again, this defuzzification procedure is implemented in the HUTSIM simulation system and our algorithm must follow the steps involved in HUTSIM.

In the learning phase of the Action Selection Network in Formula (5.11), the values of $v(t-1)$ and $y^*(t-1)$ from the previous time step are needed. The values of $v(t)$ and $y^*(t)$ are computed for each observation in the current simulation run, but $v(t-1)$ and $y^*(t-1)$ are the average values from the previous simulation run. Actually we would need values of v and y^* of the previous control action, computed using the previous values of network parameters, but in our application, the parameters are updated only after the whole simulation run and not between each control action as in [6]. That is why we must resort to average values of v and y^* from the previous simulation run.

In our rule base, a linguistic value V may also appear in the form "*more than V*" or "*less than V*". This must be taken into account in the learning phase of the Action Selection Network, especially in Formula (5.16). Let $f(\mu_V)$ denote a function of μ_V, where f is either "*more than*" (Formula (6.4)) or "*less than*" (Formula (6.4)). Formula (5.16) is now

$$\Delta p_V = \eta \frac{\partial v}{\partial y^*} \left(\sum_{V \in \mathrm{Ant}(R_i)} \frac{\partial y^*}{\partial w_i} \frac{\partial w_i}{\partial f(\mu_V)} \frac{\partial f(\mu_V)}{\partial \mu_V} \right) \frac{\partial \mu_V}{\partial p_V} \tag{6.7}$$

where $\frac{\partial w_i}{\partial f(\mu_V)}$ is the same as $\frac{\partial w_i}{\partial \mu_V}$ earlier in Formula (5.16), and $\frac{\partial f(\mu_V)}{\partial \mu_V}$ is

$$\frac{\partial f(\mu_V)}{\partial \mu_V} = \begin{cases} -1, & x \geq \sup M \\ 0, & \text{else} \end{cases} \qquad (\textit{"more than V"}) \tag{6.8}$$

$$\frac{\partial f(\mu_V)}{\partial \mu_V} = \begin{cases} -1, & x \leq \inf M \\ 0, & \text{else} \end{cases} \qquad (\textit{"less than V"}) \tag{6.9}$$

where M is as in Formula (6.5).

Formula (5.18) reduces to

$$\Delta p = \eta \, \hat{r}(t) \frac{\partial v}{\partial p} \tag{6.10}$$

because the stochastic deviation is not used, as discussed in the following.

Stochastic Action Modifier. Berenji and Khedkar also proposed a Stochastic Action Modifier (see Section 5.3.4) which deviates the control signal randomly. The stochastic action modifier cannot be used in the traffic signal control problem at hand. The reason is that the fuzzy inference takes place in the HUTSIM system, and this refinement cannot be incorporated in HUTSIM. Thus the controller output is not deviated, and $s(t)$ in Formula 5.5 is not used as a learning factor. Note that HUTSIM uses integer outputs for the control action EXT, so a small perturbation $s(t)$ would not make any change even if a stochastic modification could be done.

Some kind of stochastic search is implemented in our algorithm anyway. During the learning, the internal reinforcement $\hat{r}$ in Formula (5.3) sometimes falls near zero. According to Formula (5.18), the magnitude of $\hat{r}$ affects the change in the parameters of the membership functions. If the absolute value of $\hat{r}$ is very small at each control action in the simulation run, the change in the parameters becomes so small that it does not affect the output of the fuzzy controller, and the learning stops. In this case, we deviate the parameters of the membership functions by a small random amount. This deviation is done after each observation of signal control action is processed and the actual amount of update is calculated. Only those parameters that contribute to the output of the controller are deviated. In this way, the learning starts again, as the parameters result in a different control output than before the deviation, and $\hat{r}$ starts to increase again. Thus in our application, the *output* of the controller is not randomly deviated, but the *parameters* of the membership functions are deviated if the learning slows down too much.

Chapter 7

Experimental Results

7.1 Earlier Results

Niittymäki [46], [47] has compared fuzzy traffic signal control with other traffic control methods. Figure 7.1 shows the vehicular delays using a traditional vehicle-actuated control with the extension principle and the FUSICO fuzzy traffic control. Both control systems were simulated in the HUTSIM simulation environment using a simulation period of two hours. The vehicle-actuated controller was the best of those which are actually used in the field. The delays produced by these two control procedures can be reliably compared because the vehicle sequences in the simulation runs for both methods were identical.

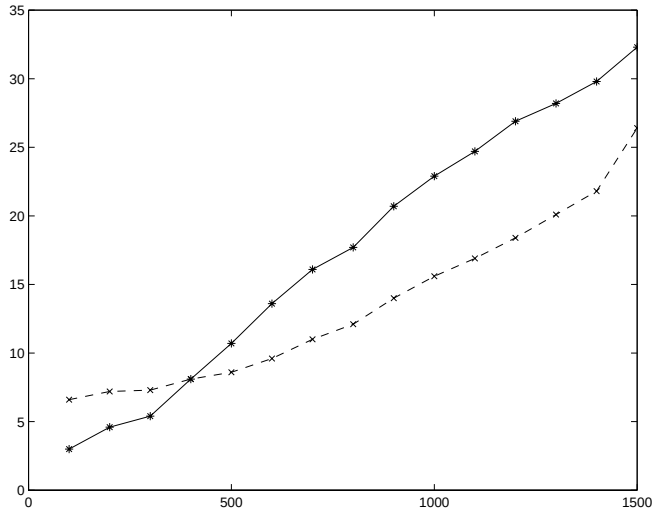


Figure 7.1: Vehicular delays using a traditional vehicle-actuated traffic signal control (solid line) and the FUSICO fuzzy control (dashed line) [47]. The x axis is the traffic volume (vehicles per hour) and the y axis is the vehicular delay (seconds).

Figure 7.1 reveals that the FUSICO fuzzy traffic signal control was superior to traditional vehicle-actuated control in large traffic volumes. At small traffic volumes, FUSICO produced larger delays than vehicle-actuated control.

The membership functions used in the FUSICO fuzzy traffic signal controller are presented in Figure 7.2. The functions were piecewise linear and they were adjusted manually. The output variable EXT was not actually fuzzy in [46] and [47], and the output of the fuzzy controller was a weighted average of the values of EXT in different rules. So no actual defuzzification was done.

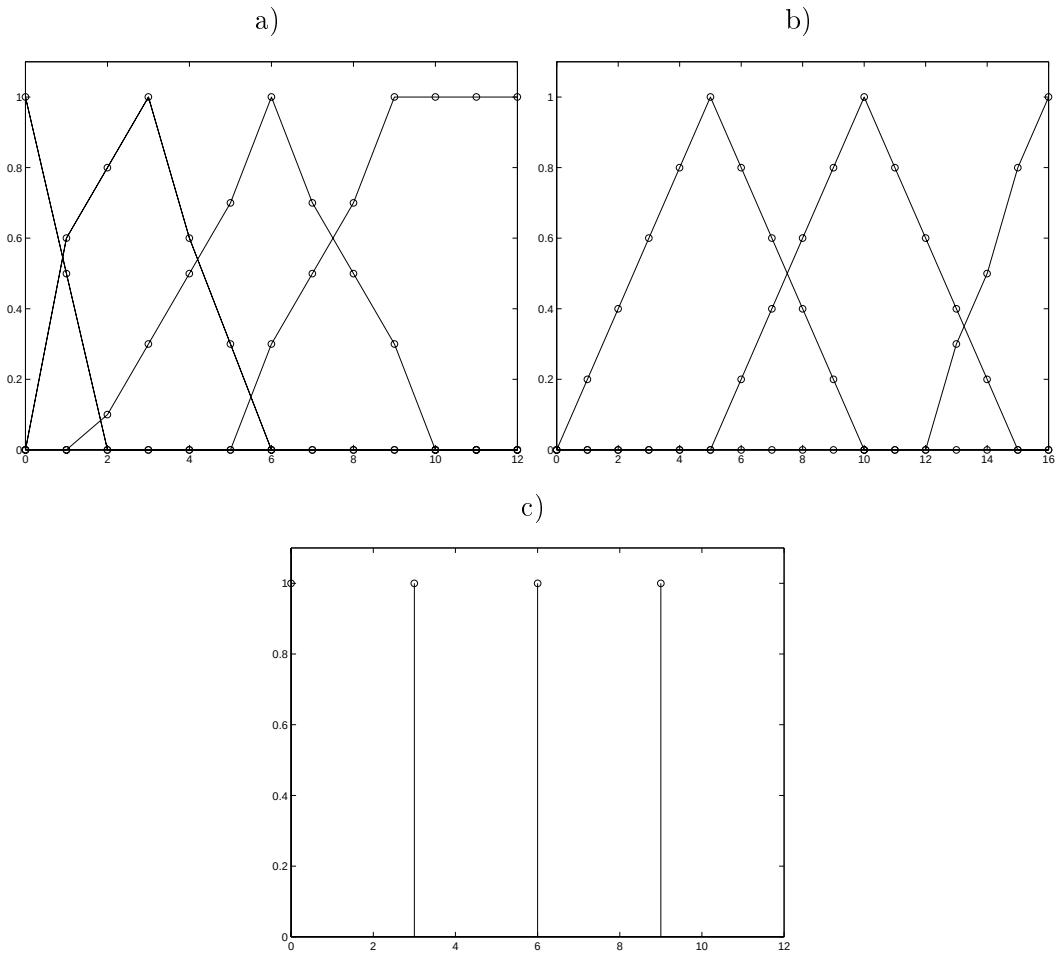


Figure 7.2: Membership functions used in [46] and [47] to produce the fuzzy control result of Figure 7.1. a) μ_{Azero}, μ_{Aafew}, $\mu_{Amedium}$ and μ_{Amany}. b) μ_{Qafew}, $\mu_{Qmedium}$ and $\mu_{Qtoolong}$. c) μ_{Ezero}, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong}.

The initial membership functions used in this work are shown in Figure 7.3. They differ from the membership functions in Figure 7.2 in a few respects. The most important difference is that the output variable EXT has now truly fuzzy membership functions. The membership functions are now trapezoids (although these initial functions are reduced to triangles) and not piecewise linear functions as in Figure 7.2. The initial membership functions used in this work produce delays different from those in Figure 7.1 which were produced by the FUSICO controller,

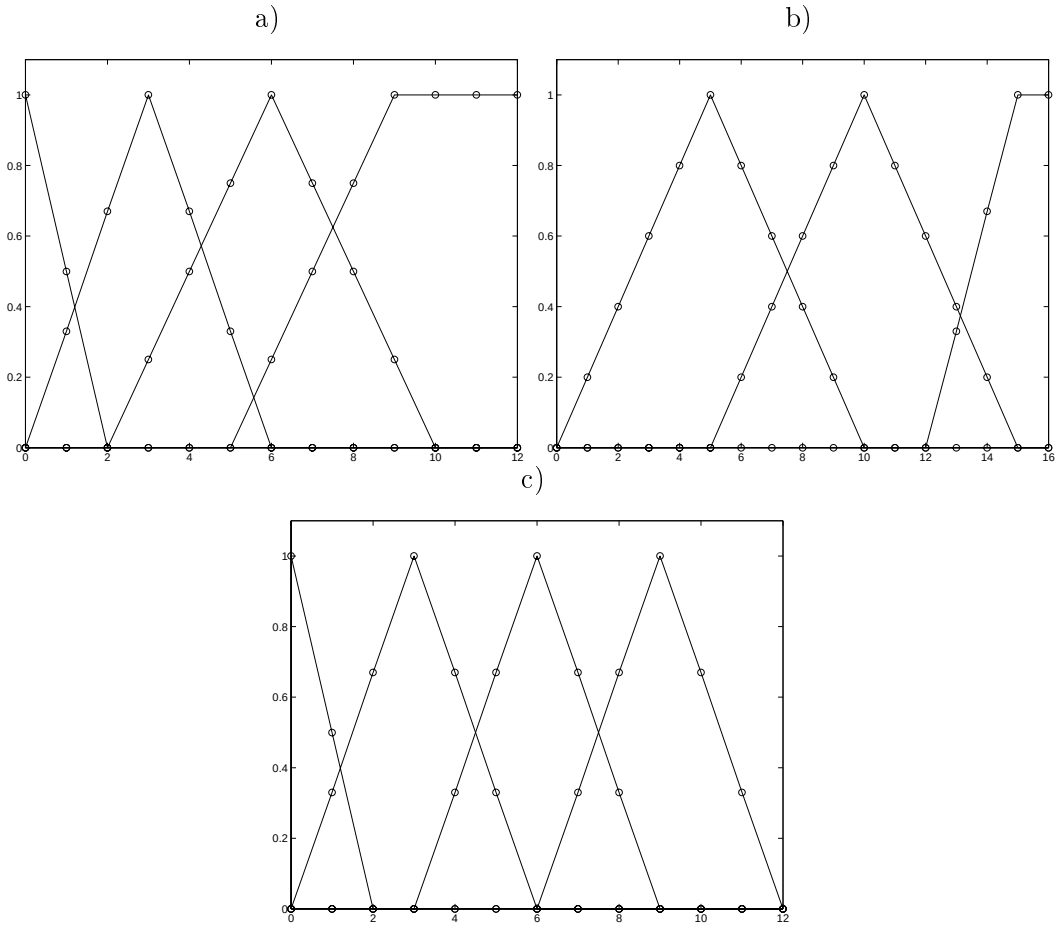


Figure 7.3: The initial form of membership functions before learning. a) μ_{Azero}, μ_{Aafew}, $\mu_{Amedium}$ and μ_{Amany}. b) μ_{Qafew}, $\mu_{Qmedium}$ and $\mu_{Qtoolong}$. c) μ_{Ezero}, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong}.

even though the delays are produced by exactly the same vehicle sequences. One reason for this is the difference in membership functions but the major reason must be that the HUTSIM simulation system has also gone through some other changes since the results [46], [47] in Figure 7.1 were obtained. The defuzzification procedure is now different, as the output variables are presented as fuzzy sets. Some other changes have taken place, too. Due to this, the results obtained in this thesis cannot be directly compared to results in [46] and in [47]. Instead, the results of the FUSICO fuzzy controller are computed again using the initial membership functions in Figure 7.3 and these results are used as reference values.

7.2 Statistical Significance of the Results

The statistical significance of the results obtained in this thesis was tested using several two-hour simulation periods. In simulation tests, the traffic situation in one particular simulation run may — by coincidence — be extremely suitable for the controller, and the control performance seems better than it actually is. To avoid such coincidences several two-hour simulations were run.

To compare two fuzzy controllers, identical simulations were run on both controllers. The vehicles were generated at the same time instants at both simulation runs, and any change in their behaviour resulted only from a change in the fuzzy controller. Thus the observations of delay in these two simulation runs were paired, and a t test on paired observations [37], [45], [55] was used to compare the means of the delays in the simulation runs.

In a t test on paired observations the difference $\mu_D = \mu_1 - \mu_2$ is tested, where μ_1 and μ_2 are the expected values of observations in sets 1 and 2. In this thesis, set 1 is the set of delays d_{ini} produced by the initial controller and set 2 is the set of delays d_{new} produced by the "new" controller (for example, after the learning or after some other modification). The null hypothesis H_0 states that $\mu_D = 0$, no difference between the sets exist. The research hypothesis H_1 states that $\mu_D > 0$, the initial delay is larger than the new delay. As the aim is to decrease the delay, it is desirable to be able to reject H_0 in favor of H_1.

The t test on paired observations assumes that the random variable $D = d_{ini} - d_{new}$ is at least approximately normally distributed. As the number of observations in this work is always several thousands, one may use the central limit theorem [37], [45], [55] and conclude that the use of the t test on paired observations is appropriate.

The test statistics for a t test on paired observations is

$$t_0 = \frac{\bar{D}}{S_D/\sqrt{n}}; \quad df = n - 1 \tag{7.1}$$

where $\bar{D}$ is the sample mean, S_D is the sample standard deviation and n is the number of observations of D. The statistics follows a T distribution with $n - 1$ degrees of freedom if H_0 is true [45]. The P value of the test is the probability that a value T drawn from a T_{n-1} distribution is larger than the test statistics t_0. If the P value is smaller than some pre-specified confidence level, one can reject H_0 and conclude that the delay is decreased.

7.3 Modification of the Rule Base

By examining the output file listings of the HUTSIM simulation system (see Appendix A.1) it was observed that there are many combinations of input variables which do not fire any rule. In these cases, the output of the fuzzy controller is zero. A frequently occurring example is a situation where the number of approaching vehicles is one, two or three and the number of queuing vehicles is zero. An obvious choice would be to give a green extension and let the approaching vehicles pass the intersection, as the queue is empty. Anyway, the rule "if *APP* is *a few* and *QUE* is *a few*, then *EXT* is *short*" or any other rule is not fired as the number of queuing vehicles is zero.

Instead of introducing a new linguistic value *zero* for *QUE*, the linguistic value *a few* of *QUE* can be replaced with *less than medium* which covers the support (Formula (2.5)) of *a few*, as seen in Figure 7.4. The construction of linguistic values *less than V* was discussed in Section 6.2.1 and in Formula (6.4). The function *less than medium* gives a membership function value of 1 for those small input measurements for which *a few* gives 0.

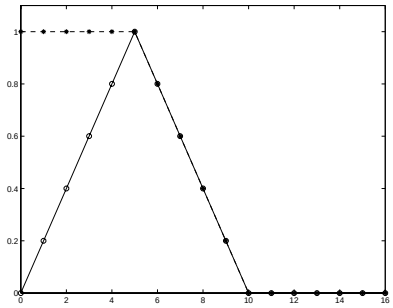


Figure 7.4: The membership function of *less than medium* (dotted line) covers the support of *a few* (solid line) of *QUE*.

In the first four rule sets, the antecedent constraint "*QUE* is *a few*" is changed to "*QUE* is *less than medium*". This modification of the rule base is simple in that the number of rules remains the same, and the linguistic value *less than medium* is already defined. At this stage, the parameters of the membership functions are not modified and the initial membership functions introduced in Figure 7.3 are used. The rule base is now

after minimum green (5 s)

if *APP* is *zero*, then *EXT* is *zero*
if *APP* is *a few* and if *QUE* is *less than medium*, then *EXT* is *short*
if *APP* is *more than a few*, then *EXT* is *medium*
if *APP* is *medium*, then *EXT* is *long*

after the first extension

if *APP* is *zero*, then *EXT* is *zero*
if *APP* is *a few* and if *QUE* is *less than medium*, then *EXT* is *short*
if *APP* is *medium*, then *EXT* is *medium*
if *APP* is *many*, then *EXT* is *long*

after the second extension

if *APP* is *zero*, then *EXT* is *zero*
if *APP* is *a few* and if *QUE* is *less than medium*, then *EXT* is *short*
if *APP* is *medium* and if *QUE* is *less than medium*, then *EXT* is *medium*
if *APP* is *many* and if *QUE* is *less than medium*, then *EXT* is *long*

after the third extension

if *APP* is *zero*, then *EXT* is *zero*
if *QUE* is *too long*, then *EXT* is *zero*
if *APP* is *more than a few* and if *QUE* is *less than medium*, then *EXT* is *short*
if *APP* is *medium* and if *QUE* is *less than medium*, then *EXT* is *medium*
if *APP* is *many* and if *QUE* is *less than a few*, then *EXT* is *long*

after the fourth extension

if *APP* is *zero*, then *EXT* is *zero*
if *QUE* is *too long*, then *EXT* is *zero*
if *APP* is *more than a few* and if *QUE* is *a few*, then *EXT* is *short*
if *APP* is *medium* and if *QUE* is *less than a few*, then *EXT* is *medium*
if *APP* is *many* and if *QUE* is *less than a few*, then *EXT* is *long*

The modified rule base leads to smaller delays than the original rule base, as seen in Figure 7.5. The delays in this figure result from five two-hour simulation periods. The vehicle sequences in both cases (before and after the modification) are identical, and any difference between the behaviours of the vehicles results from the differences between the fuzzy controllers. The differences in the delay are noteworthy especially at low traffic volumes, where the FUSICO fuzzy control has earlier performed worse than traditional vehicle-actuated control (Figure 7.1). One explanation for the poor earlier performance could be just this lack of rules in situations where the length of the queue is zero, as is often the case at low traffic volumes. At 100 vehicles per hour, the delay decreases now 30 per cent, from 7.65 s per vehicle to 5.38 s per vehicle.

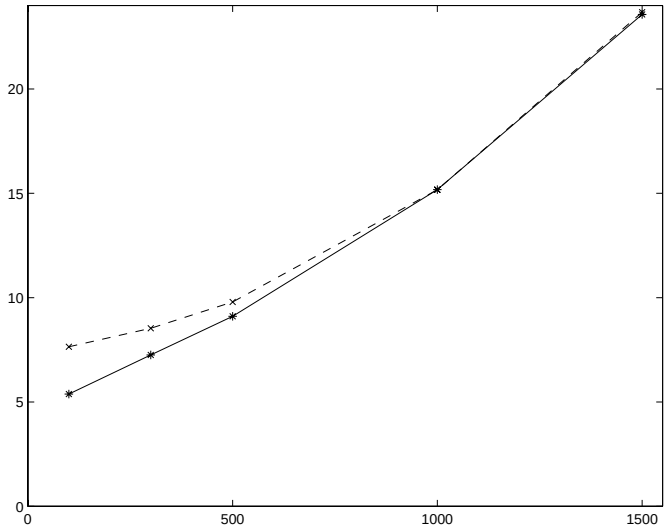


Figure 7.5: Delays at different traffic volumes before (dashed line) and after (solid line) the modification of the rule base. In the first four rule sets, the antecedent constraint "*QUE* is *a few*" is changed to "*QUE* is *less than medium*".

Volume	$\bar{d}_{ini}$	$\bar{d}_{new}$	$\bar{D}$	S_D	n	P value	Conclusion
100	7.65	5.38	2.27	7.66	2 260	0	Reject H_0
300	8.53	7.26	1.28	8.86	6 075	0	Reject H_0
500	9.79	9.11	0.69	10.64	10 136	$4.5 \cdot 10^{-11}$	Reject H_0
1000	15.17	15.18	-0.01	13.93	20 016	0.53	Accept H_0
1500	23.68	23.57	0.11	17.61	30 205	0.14	Accept H_0

Table 7.1: Statistical significance of the decrease in delay due to the modification of the rule base. Hypotheses $H_0 : \mu_D = 0$ (delay is not changed) and $H_1 : \mu_D > 0$ (delay is decreased) are tested using a t test on paired observations.

The statistical significance of the results is seen in Table 7.1. A t test on paired observations (Formula (7.1)) is conducted, and we test the hypotheses $H_0 : \mu_D = 0$

(delay is not changed) and $H_1 : \mu_D > 0$ (delay is decreased) where $D = d_{ini} - d_{new}$ is the change in the delay of one vehicle due to the modification of the rule base.

In Table 7.1, $\bar{d}_{ini}$ is the average delay during five two-hour simulation runs using the initial membership function parameters and $\bar{d}_{new}$ is the average delay using the parameters after the learning. $\bar{D}$ is the average of D and S_D is the sample standard deviation of D. The number of observations n is larger at high traffic volumes than at low traffic volumes, because at high volumes the number of vehicles is naturally larger. If the P value is very small, we reject the null hypothesis H_0. Here we could have chosen any of the usual confidence levels α (0.1, 0.05, 0.01, 0.005 or 0.0001) and the conclusions would always be the same, as the P values are either very small or very large.

It is observed that at volumes of 100, 300 and 500 vehicles per hour the delay is decreased significantly. At 1000 and 1500 vehicles per hour the modification of the rule base does not cause a significant difference in delay, because there are very seldom situations where the number of queuing vehicles in the red direction is 0.

From now on, in the tests and during the learning, the modified rule base is always used. One might ask whether the reinforcement learning algorithm could have fixed this problem by modifying the membership functions suitably. The answer is no. If the input is such that no rules fire, then no learning occurs, as the rule firing strengths of each rule are 0 and the output fuzzy sets are empty. Thus the algorithm is not used at all in such a situation.

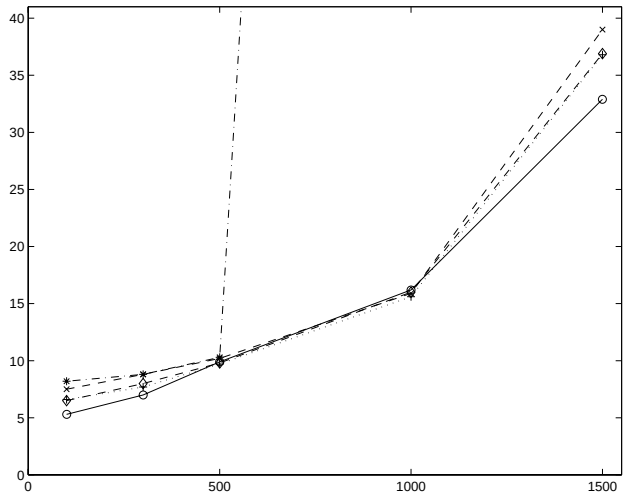
7.4 Different Detector Locations

In the basic simulation setting, the traffic detectors are located 100 metres before the stop line and at the stop line. The location of the first detector affects the number of vehicles the signal controller perceives at each signal control action. In practice, the first detector cannot often be placed as far as 100 metres away from the stop line — instead, distances of about 50 metres are usual.

It was now examined how the location of the first detector affects the control performance. The delays of the vehicles were compared at five traffic volumes: 100, 300, 500, 1000 and 1500 vehicles per hour. The location of the first detector was 50, 75, 100 or 150 metres from the stop line in both approaching directions. In one experiment, the first detector was placed at 50 metres in one approaching direction and at 100 metres in another approaching direction. The membership functions were the initial ones as in Figure 7.3. The results are seen in Figure 7.6.

Figure 7.6 reveals that a distance of 100 metres from the stop line is the best in delay minimization. If the vehicle is observed this early, its delay is minimized if the signal controller takes it into account. In the optimal case the vehicle does not have to slow down at all if the traffic volume is very low and other vehicles are not present at the moment. As the distance decreases to 50 or 75 metres, the vehicular delay increases as the vehicles need to slow down when approaching the intersection.

a)



b)

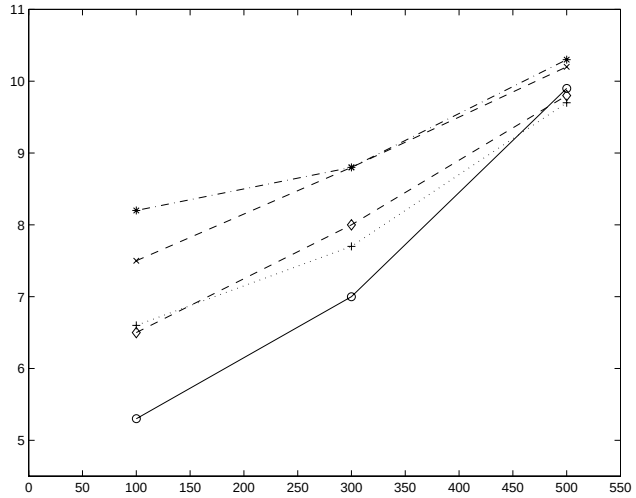


Figure 7.6: a) Vehicular delays when the first traffic detector is in both approaching directions at 100 m ($\circ$), 50 m ($\times$), 75 m ($+$) and 150 m ($*$, the last ones are too large to fit in this figure). In the last case ($\diamond$), the detector is in one approaching direction at 50 m and in the other direction at 100 m. b) Magnification at lower traffic volumes.

The delay increases also as the detector is moved further away to 150 metres. The reason for this is that at low traffic volumes (100 to 300 vehicles per hour) the traffic controller takes into account also those vehicles that are too far away from the intersection to reach it during the green extension, and a green extension is given unnecessarily. At a very large traffic volume (1500 vehicles per hour) the delays become unbearable, because now the observed queues in both the red direction and

the green direction are outside the ranges of the fuzzy controller (maximum values of APP and QUE are 12 and 16, respectively), and a zero extension is always given, because no rules fire. At large traffic volumes the best approach would be to have as many green extensions as possible, as then the time needed for amber signal is minimized. Now the situation is the opposite.

The results in this section were produced by one two-hour simulation run in each case. The statistical significance of the results is not tested, and the results are not used in further experiments in this work.

7.5 Experimental Setting in Learning

In the learning phase of a neural network using supervised learning, it is usually desirable to have variations in the input observations. If the observations are concentrated on a small area in the input-output space, the system will not learn outside this area. Variations in the observations lead to a better exploration of the state space. In this work it was anyway decided to teach the network first in a constant simulation environment where the vehicles enter the model always at the same time instants. Random variation in vehicle generation causes substantial differences in the delays. The effect of the modification of the membership function parameters can not be observed if the effect of random variation in delays is larger than the effect of parameter modification. On the other hand, teaching in a constant environment results in a network which is suitable for this particular environment only.

The HUTSIM simulation system has a pseudo-random generator which produces vehicles at pseudo-random instants. The seed of the generator must be changed between successive simulation runs if random variation between the runs is desired. It was now decided not to change the generator seed between successive runs during the network learning phase. The seed was changed only after every 30 simulation runs. During each 30 simulation runs, the vehicle generation sequences were identical and the changes in the vehicular delays were due to changes in the signal controller only. In this way, quite a short simulation run was sufficient during the learning phase of the neural network. The total time needed for the learning was not unbearable, even though neural networks usually require a very large set of observations.

In this work, a simulation period of ten minutes was used during the learning, in addition to margins of two minutes in both ends to ensure "normal" traffic — in the beginning of the simulation, the system was empty, and in the end it was again emptied, and the vehicles entering the model near the beginning or the end did not suffer from delays caused by other vehicles. These margins were excluded from the observation set. The simulation was done using a 266 MHz Pentium with which 10 minutes of simulation time took approximately a minute of real time.

The reinforcement learning algorithm was used at three different constant traffic volumes: 300, 500 and 1000 vehicles per hour. A traffic volume of 300 vehicles per hour was chosen as the smallest volume, because at volumes smaller than 300 vehicles per hour there was not much to learn. The controller used the fuzzy rule base only after a minimum green of 5 seconds. At low traffic volumes the vehicles approached the intersection one at a time and this minimum green time often sufficed

to discharge the queue in the green direction.

Each of these constant traffic volumes was used in two different situations: the location of the first traffic detector was either 100 or 50 metres from the stop line. So the learning algorithm was tested in a total of six different situations of constant traffic volumes. It was expected that each situation has its own membership function parameters, as the physical conditions are very different, too. In practice, if different membership function parameters are suitable for different traffic situations, the traffic signal controller could first identify the traffic situation and then choose a proper parameter set for the fuzzy control process.

In addition to constant traffic volumes, the reinforcement learning algorithm was also used in a changing traffic situation, where the volume of the traffic was increased step by step. In practice, if the algorithm is able to follow and keep up with a changing traffic situation, the algorithm can be installed in the field. A traffic signal controller using the reinforcement learning algorithm could update its parameters all the time as the traffic volume changes. Of course, there are many practical limitations which hinder this kind of on-line learning in the field: for example, the computations needed by the learning algorithm can be burdensome.

The size of the hidden layer in the neural network AEN (Sections 5.3.2 and 6.2.3) was in the first experiments 4 or 6 but finally a size of 10 hidden layer cells was selected. A larger hidden layer contributed to a better learning performance, but also the number of observations needed for learning was larger, because there were more adjustable parameters.

7.6 Learning with First Traffic Detector at 100 Metres

In the HUTSIM simulation environment the location of the first traffic detector is usually 100 metres before the stop line, and the reinforcement learning algorithm was first used in this kind of situation. The traffic volumes were 300, 500 and 1000 vehicles per hour, and during the learning the volume was kept constant. The initial membership functions are shown in Figure 7.3. Figure 7.7 compares the delays before and after learning at traffic volumes of 300, 500 and 1000 vehicles per hour. The delays in this figure are average delays of five two-hour simulation runs. The location of the first traffic detector is 100 metres before the stop line.

The statistical significance of the results in Figure 7.7 is determined by a t test on paired observations (Formula (7.1)). Hypotheses $H_0 : \mu_D = 0$ (delay is not changed) and $H_1 : \mu_D > 0$ (delay is decreased) are tested, where $D = d_{ini} - d_{new}$ is the change in the delay of one vehicle due to the modification of the rule base. The results of the t tests on paired observations are seen in Table 7.2. In this table, $\bar{d}_{ini}$ is the average delay during five two-hour simulation runs using the initial membership function parameters and $\bar{d}_{new}$ is the average delay using the parameters after the learning. $\bar{D}$ is the average of D and S_D is the sample standard deviation of D. The number of observations is n. If the P value is very small, the null hypothesis H_0 is rejected. It is seen that the decrease in the delay is statistically significant at traffic volumes of 500 and 1000 vehicles per hour but not at 300 vehicles per hour. The confidence level α may be any of 0.1, 0.05, 0.01 or 0.005.

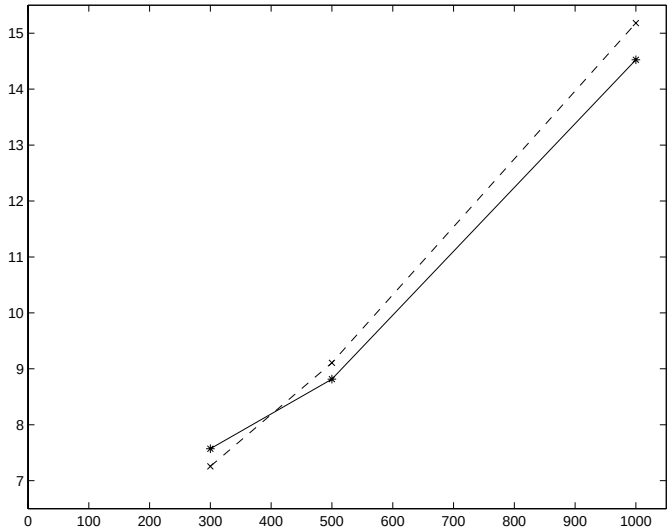


Figure 7.7: Average vehicular delays (in seconds) before (dashed line) and after (solid line) the learning at traffic volumes of 300, 500 and 1000 vehicles per hour. The location of the first traffic detector is 100 metres before the stop line.

Volume	$\bar{d}_{ini}$	$\bar{d}_{new}$	$\bar{D}$	S_D	n	P value	Conclusion
300	7.26	7.57	-0.32	8.33	6 075	1.00	Accept H_0
500	9.11	8.82	0.29	9.84	10 136	$1.4 \cdot 10^{-3}$	Reject H_0
1000	15.18	14.52	0.66	14.35	20 016	$5.1 \cdot 10^{-11}$	Reject H_0

Table 7.2: Statistical significance of the decrease in delay due to the learning. The location of the first traffic detector is 100 metres before the stop line. Hypotheses $H_0 : \mu_D = 0$ (delay is not changed) and $H_1 : \mu_D > 0$ (delay is decreased) are tested using a t test on paired observations.

In the following, the learning at each traffic volume is discussed separately.

Traffic volume 300 vehicles per hour. Figure 7.8 shows the delay during the learning phase. The delay does not decrease during the learning. The figure also shows the average results of five two-hour test simulation runs during the learning, marked with small circles.

According to Figure 7.8, the best membership function parameter set during the learning is the one obtained after 150 simulation periods, because the delay in the test simulations is then smallest. This parameter set was used to compute the values in Figure 7.7 and Table 7.2. Even this parameter set is inferior to the initial parameter set.

The membership functions after the learning are not presented here, because the learning was not successful and one cannot draw any conclusions about the shapes of the membership functions.

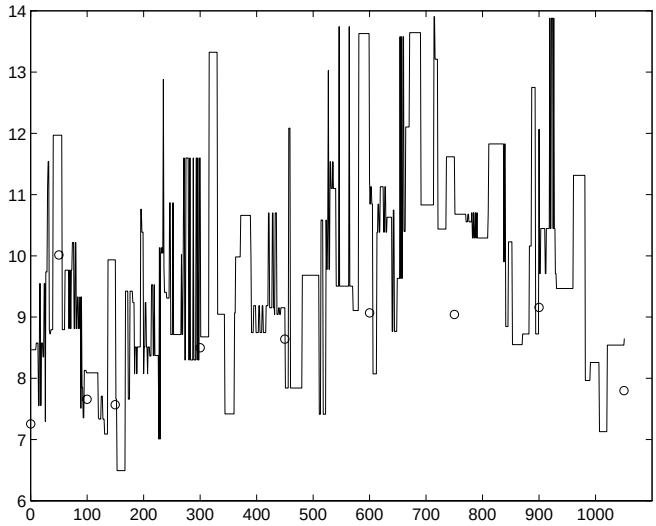


Figure 7.8: Delay during the learning at 300 vehicles per hour. Small circles show the results of test simulation runs before the learning and after 50, 100, 150, 300, 450, 600, 750, 900 and 1050 learning loops. The location of the first traffic detector is 100 metres before the stop line.

Traffic volume 500 vehicles per hour. The learning at a traffic volume of 500 vehicles per hour differs from all other learning situations in Sections 7.6 and 7.7 in the respect that the results were obtained with a neural network whose hidden layer size is 6 instead of 10. The network used in all other situations, with 10 hidden layer cells, gave a worse result in this case. Also, the seed of the pseudo-random generator of the HUTSIM simulation system was changed at 40, 70, 130 and 150 simulation runs instead of every 30 simulation runs. As seen in Figure 7.7 and Table 7.2, the new membership functions produce smaller delays at 500 vehicles per hour than the initial membership functions. The learning was continued for 200 simulation periods.

The membership functions before and after the learning are presented in Figure 7.9. It is observed that the membership functions μ_{Amany} and $\mu_{Qtoolong}$ were not updated, because at 500 vehicles per hour there were seldom observations in the support of these functions. The other membership functions were modified, and especially the membership functions of EXT were changed substantially.

The function μ_{Azero} shrank a little. An explanation for this is that at reasonably small traffic volumes it makes a difference if the number of vehicles is 0 or 1. Only a pure 0 is interpreted as a fuzzy *zero*.

Unfortunately, there is now a gap between μ_{Azero} and μ_{Aafew} so that an input measurement of one approaching vehicle does not fire any membership function. The delay using these membership functions is still smaller than the initial delay. The result might be even better if the gap was filled for example by extending μ_{Aafew} to smaller values.

The membership functions μ_{Qafew} and $\mu_{Qmedium}$ were updated only by a small amount. As the rule base was now modified as explained in Section 7.3, the membership function μ_{Qafew} was seldom used and therefore its parameters were seldom modified

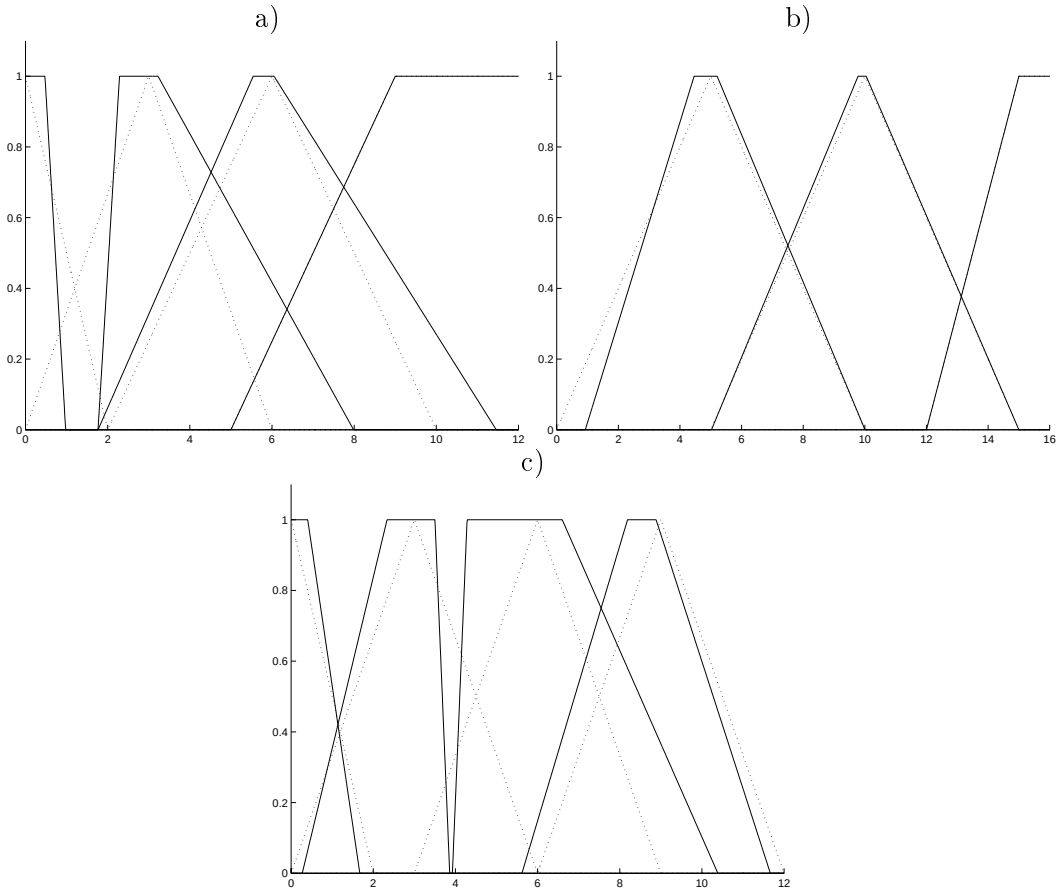


Figure 7.9: Membership functions before (dotted line) and after (solid line) the learning at 500 vehicles per hour. The location of the first traffic detector is 100 metres before the stop line. a) μ_{Azero}, μ_{Aafew}, $\mu_{Amedium}$ and μ_{Amany}. b) μ_{Qafew}, $\mu_{Qmedium}$ and $\mu_{Qtoolong}$. c) μ_{Ezero}, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong}.

during the learning. The membership function $\mu_{Qmedium}$ was also seldom modified even though it was often used in the form "*less than medium*", because in the update formula (5.16) the derivative term $\frac{\partial \mu_V}{\partial p_V}$ was zero. The derivatives of μ with respect to the parameters p_1, p_2, p_3, p_4 in Formulae (5.56)–(5.59) were zero because small input values of QUE were outside the support of $\mu_{Qmedium}$, and the derivative was nonzero only if the input value was within $[p_1, p_2]$ or $[p_3, p_4]$.

In Section 5.3.7 it was concluded that if the membership function is originally a symmetric trapezoid or a symmetric triangle, all its parameters change in the same way and the shape and size of the membership function remain constant. Yet in Figure 7.9 it is seen that initially symmetric triangles μ_{Aafew}, $\mu_{Amedium}$, μ_{Qafew}, $\mu_{Qmedium}$, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong} have transformed to trapezoids or in some other way lost their original shape. The reason is the stochastic deviation of the parameters discussed in Section 6.2.3. Once the parameters receive a stochastic deviation — no matter how small — the triangle becomes asymmetric. The same effect is caused by small rounding errors in Matlab, which may also be present.

Traffic volume 1000 vehicles per hour. Learning at 1000 vehicles per hour was successful, as was seen in Figure 7.7. The learning was continued for 600 loops. Figure 7.10 shows the delay during the learning together with the average delays of five test simulation runs after 0, 150, 300, 450 and 600 learning loops. It is interesting to see that the delay in the test cases behaves in the same manner as the delay during the learning: the delay is smallest after 300 learning loops and larger both before and after that.

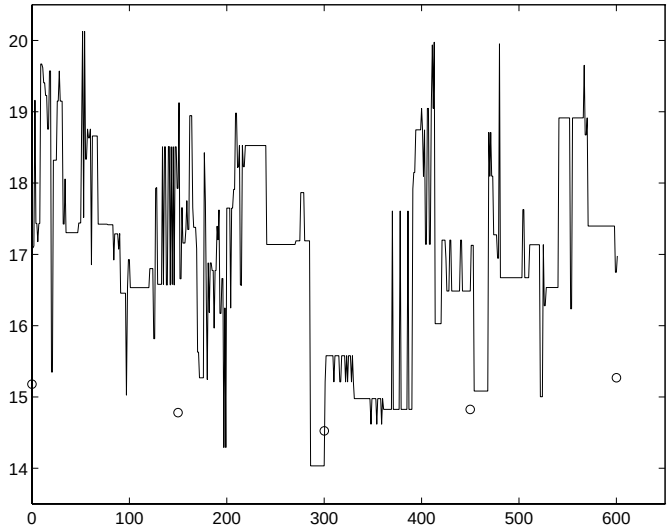


Figure 7.10: Delay in the learning phase of a traffic situation of 1000 vehicles per hour. The location of the first traffic detector is 100 metres before the stop line. Small circles show the results of test simulation runs before the learning and after 150, 300, 450 and 600 learning loops.

The membership functions before and after the learning are presented in Figure 7.11. The function μ_{Azero} has grown wider than in the beginning. This reflects the fact that at large traffic volumes, input measurements of 0, 1 or 2 vehicles are all fairly equal to zero. The membership functions of QUE have changed only a little, the reasons for which were discussed earlier in this section. The membership function μ_{Ezero} has shrunk, meaning that the rule consequent "EXT is *zero*" is really defuzzified to 0. The functions $\mu_{Emedium}$ and μ_{Elong} have shifted to larger values, meaning that extensions at large traffic volumes should be quite long — a fact acknowledged by traffic engineers.

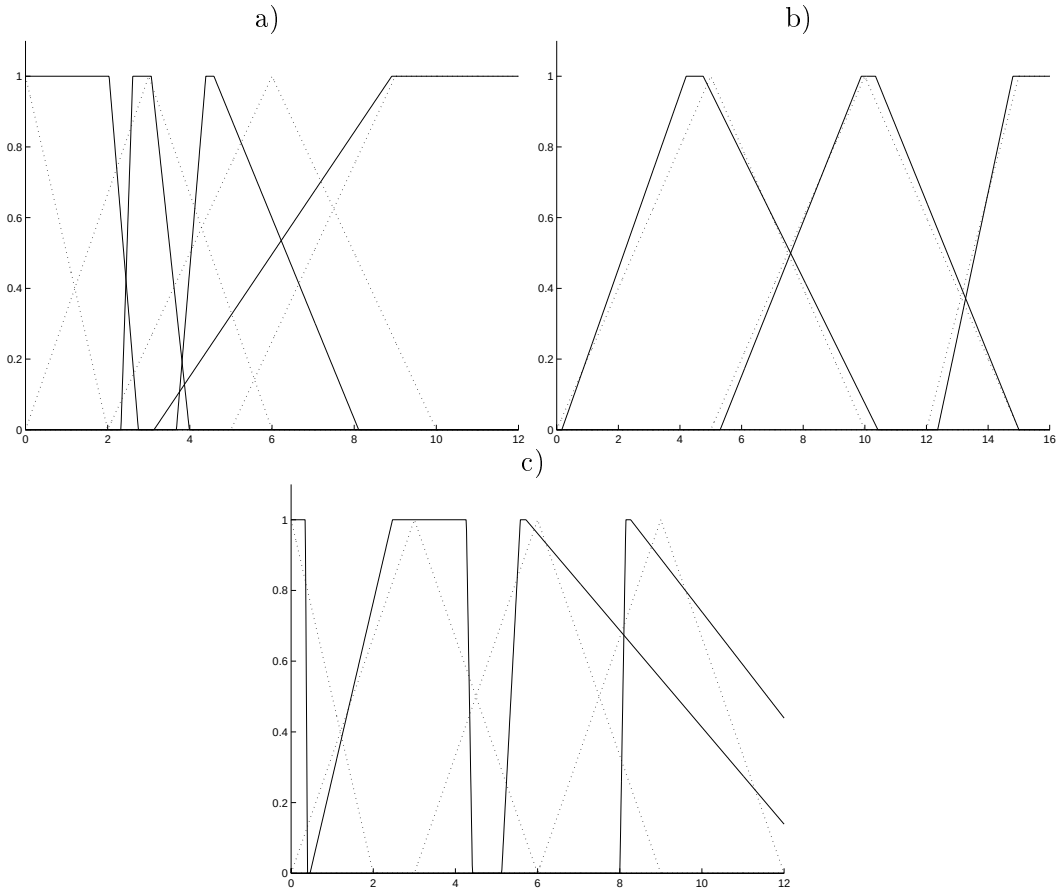


Figure 7.11: Membership functions before (dotted line) and after (solid line) the learning at 1000 vehicles per hour. The location of the first traffic detector is 100 metres before the stop line. a) μ_{Azero}, μ_{Aafew}, $\mu_{Amedium}$ and μ_{Amany}. b) μ_{Qafew}, $\mu_{Qmedium}$ and $\mu_{Qtoolong}$. c) μ_{Ezero}, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong}.

7.7 Learning with First Traffic Detector at 50 Metres

The membership functions of the fuzzy controller were also taught at a situation where the first traffic detector in both approaching directions was located 50 metres before the stop line. The location of the first detector affects the number of vehicles the controller observes at each signal control action. If the first detector is located nearer the intersection than at 100 metres (which is the basic case), the input measurements of APP and QUE are smaller. In practice the location of the first detector is more often near 50 metres than 100 metres before the stop line. It is assumed that the location affects the membership function parameters, too.

The results of the learning at constant traffic volumes of 300, 500 and 1000 vehicles per hour are seen in Figure 7.12. The delays in this figure are average delays of five two-hour simulation runs before and after the learning. The statistical significance of the results in Figure 7.12 is determined by a t test on paired observations (Formula (7.1)). Hypotheses $H_0 : \mu_D = 0$ (the delay is not changed) and $H_1 : \mu_D > 0$ (the delay is decreased) are tested, where $D = d_{ini} - d_{new}$ is the change in the delay

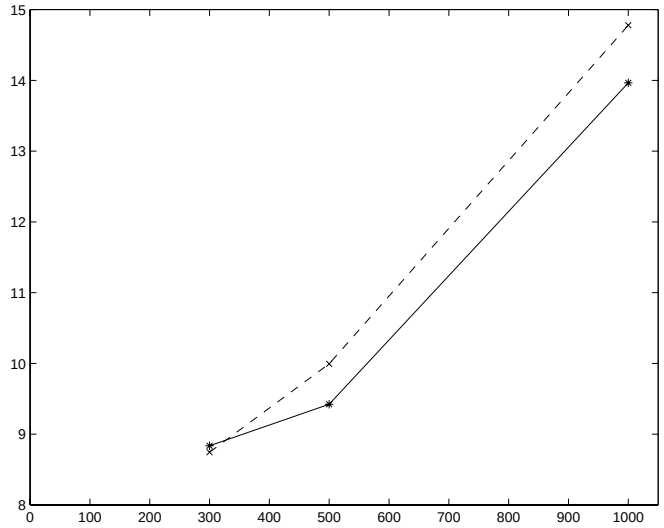


Figure 7.12: Average vehicular delays (in seconds) before (dashed line) and after (solid line) learning at traffic volumes of 300, 500 and 1000 vehicles per hour. The location of the first traffic detector is 50 metres before the stop line.

Volume	$\bar{d}_{ini}$	$\bar{d}_{new}$	$\bar{D}$	S_D	n	P value	Conclusion
300	8.75	8.84	-0.09	8.33	6 075	0.80	Accept H_0
500	9.99	9.42	0.57	9.01	10 136	$9.1 \cdot 10^{-11}$	Reject H_0
1000	14.78	13.96	0.81	12.32	20 016	0	Reject H_0

Table 7.3: Statistical significance of the decrease in delay due to the learning. The location of the first traffic detector is 50 metres before the stop line. Hypotheses $H_0 : \mu_D = 0$ (the delay is not changed) and $H_1 : \mu_D > 0$ (the delay is decreased) are tested using a t test on paired observations.

of one vehicle due to the modification of the rule base. The results of the t tests on paired observations are seen in Table 7.3.

In Table 7.3, $\bar{d}_{ini}$ is the average delay during five two-hour simulation runs using the initial membership function parameters and $\bar{d}_{new}$ is the average delay using the parameters after the learning. $\bar{D}$ is the average of D and S_D is the sample standard deviation of D. The number of observations is n. If the P value is very small, the null hypothesis H_0 is rejected. Here the choice of the confidence level α (0.1, 0.05, 0.01, 0.005 or 0.0001) does not make any difference in the conclusion, as the P values are either very small or very large. It is seen that the decrease in the delay is statistically significant at traffic volumes of 500 and 1000 vehicles per hour.

In the following, the learning at each traffic volume is discussed separately.

Traffic volume 300 vehicles per hour. At 300 vehicles per hour, the delay after the learning was a little larger than before the learning, but the difference is not statistically significant, as seen in Table 7.3. Testing a hypothesis $H_1 : \mu_D < 0$ (the

delay has increased during the learning) would give a P value of $1 - 0.80 = 0.20$ and again one must accept H_0. So it cannot be concluded that the delay after the learning is either smaller or larger than before the learning at 300 vehicles per hour.

The learning was stopped after 500 loops because the parameters at this stage gave a smaller delay in a two-hour test simulation than the initial parameters. Unfortunately the average delay in five two-hour test simulation runs was not smaller than before the learning, as was seen in Table 7.3. Thus the first test simulation was by coincidence more suitable for the new parameters than what the simulation runs were on average.

Traffic volume 500 vehicles per hour. The learning at 500 vehicles per hour was successful, as seen in Figure 7.12 and in Table 7.3. The delay after the learning was smaller than before the learning. Figure 7.13 shows the delay during the learning together with the average delays of five test simulation runs after 0, 300, 450, 600 and 750 learning loops. The test result was the best with the parameter set obtained after 450 learning loops, so this parameter set was chosen.

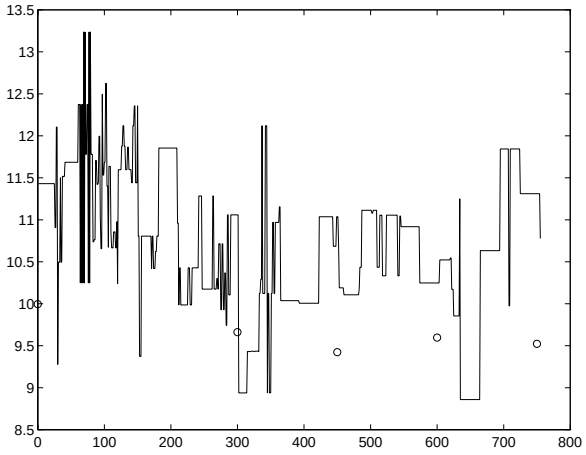


Figure 7.13: Delay in the learning phase of a traffic situation of 500 vehicles per hour. The location of the first traffic detector is 50 metres before the stop line. Small circles show the results of test simulation runs before the learning and after 300, 450, 600 and 750 learning loops.

The membership functions before and after the learning are presented in Figure 7.14. It is observed that the membership functions μ_{Amany}, $\mu_{Qmedium}$, $\mu_{Qtoolong}$ and μ_{Elong} were not updated. At a traffic volume of 500 vehicles per hour, there were seldom observations for which these membership functions were needed, especially now when the first traffic detector was closer to the stop line and there were fewer vehicles between the detectors.

The other membership functions were modified, for example μ_{Azero} grew wider and μ_{Aafew} turned to the right. There is actually a small gap between μ_{Azero} and μ_{Aafew} now. The function μ_{Azero} ends at 1.870 and μ_{Aafew} begins at 1.999 but as the input

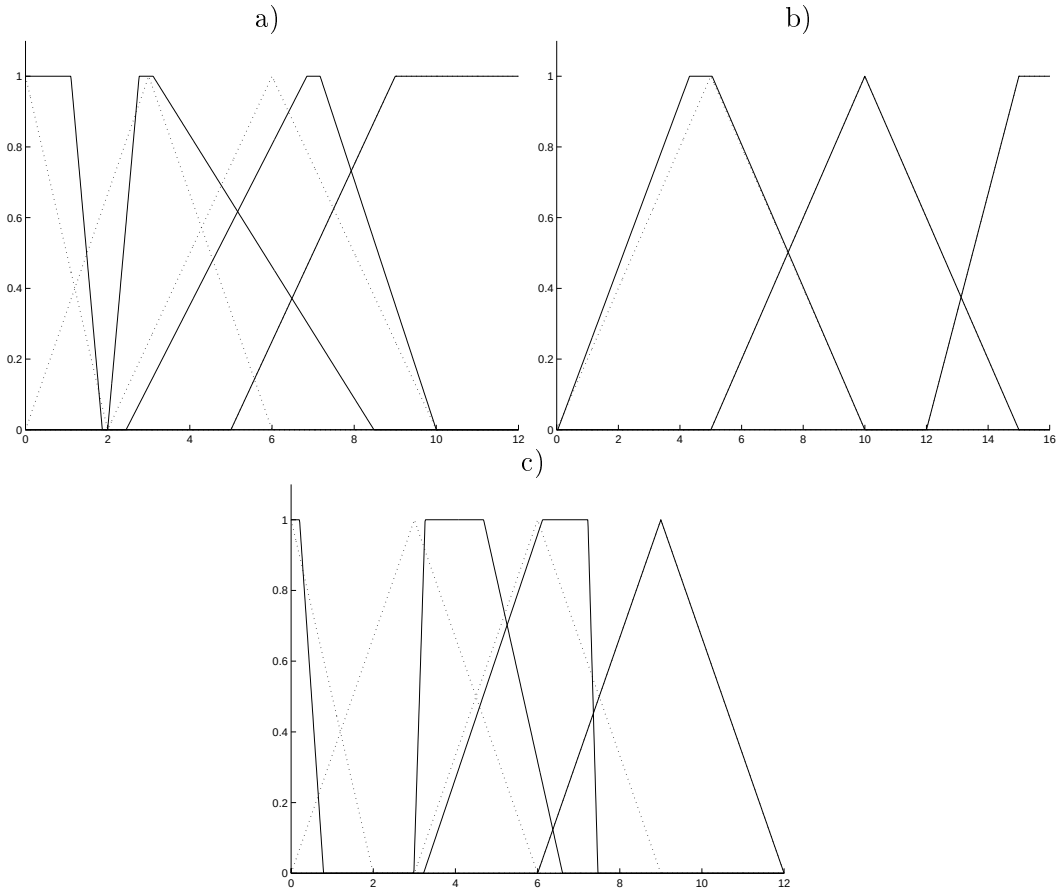


Figure 7.14: Membership functions before (dotted line) and after (solid line) the learning at 500 vehicles per hour. The location of the first traffic detector is 50 metres before the stop line. a) μ_{Azero}, μ_{Aafew}, $\mu_{Amedium}$ and μ_{Amany}. b) μ_{Qafew}, $\mu_{Qmedium}$ and $\mu_{Qtoolong}$. c) μ_{Ezero}, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong}.

measurements are always integers, this is not a problem. An input measurement of 2 approaching vehicles fires the membership function μ_{Aafew} to a non-zero degree.

Function μ_{Ezero} shrank and so did μ_{Eshort} and $\mu_{Emedium}$, but they also came closer to each other. The gap between μ_{Eshort} and $\mu_{Emedium}$ indicates that extensions of two seconds are seldom given. As the first traffic detector is located only 50 metres before the stop line now, the traffic signal controller cannot know if there are vehicles behind the 50 m point. With a speed of 40 km/h a distance of 50 m takes 4.5 seconds, so it is wise to give an extension of 4 to 5 seconds so that the vehicles between the detectors can pass the stopline. A longer extension is unnecessary. Both μ_{Eshort} and $\mu_{Emedium}$ are now concentrated around 5 seconds.

Traffic volume 1000 vehicles per hour. Figure 7.15 shows the decrease in the delay during the learning phase. The learning was continued for over 1050 loops, and a decision to stop the learning was made based on two-hour simulation runs during the learning. The best parameter set was the one obtained after 1050 learning loops. After that, the delay increased again. The results of the learning were successful, as

seen in Figure 7.12 and in Table 7.3.

Figure 7.16 shows the prediction of future reinforcement v (Formula (5.2)) and the internal reinforcement $\hat{r}$ (Formula (5.3)). It is seen that v increases during the learning, as the delay decreases. This is in agreement with the nature of v. The internal reinforcement $\hat{r}$ does not have an increasing trend — instead, it oscillates between -2 and 2. The reason for this is the definition of $\hat{r}$ in Formula (5.3) where it is seen that $\hat{r}$ reflects both the external error term (the delay) and the difference of v between successive time steps.

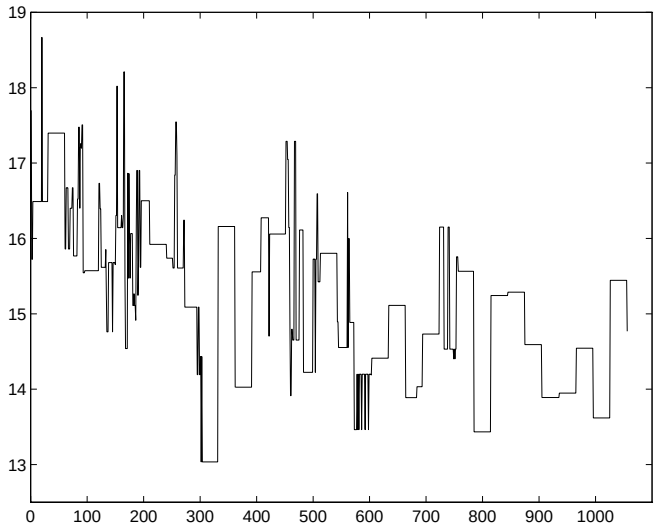


Figure 7.15: Delay in the learning phase of a traffic situation of 1000 vehicles per hour. The location of the first traffic detector is 50 metres before the stop line.

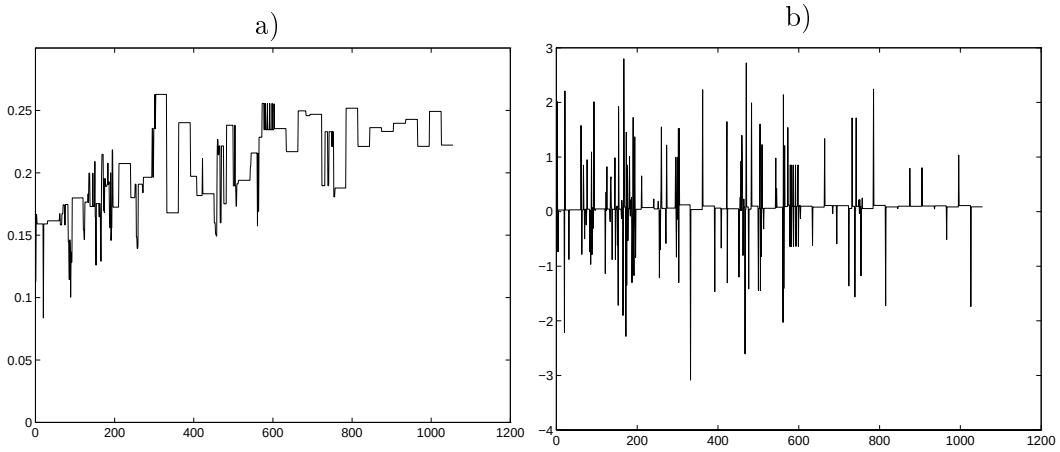


Figure 7.16: a) The value of v, the prediction of future reinforcement during the learning at a traffic volume of 1000 vehicles per hour. The location of the first traffic detector is 50 metres before the stop line. b) The value of $\hat{r}$, the internal reinforcement during the learning.

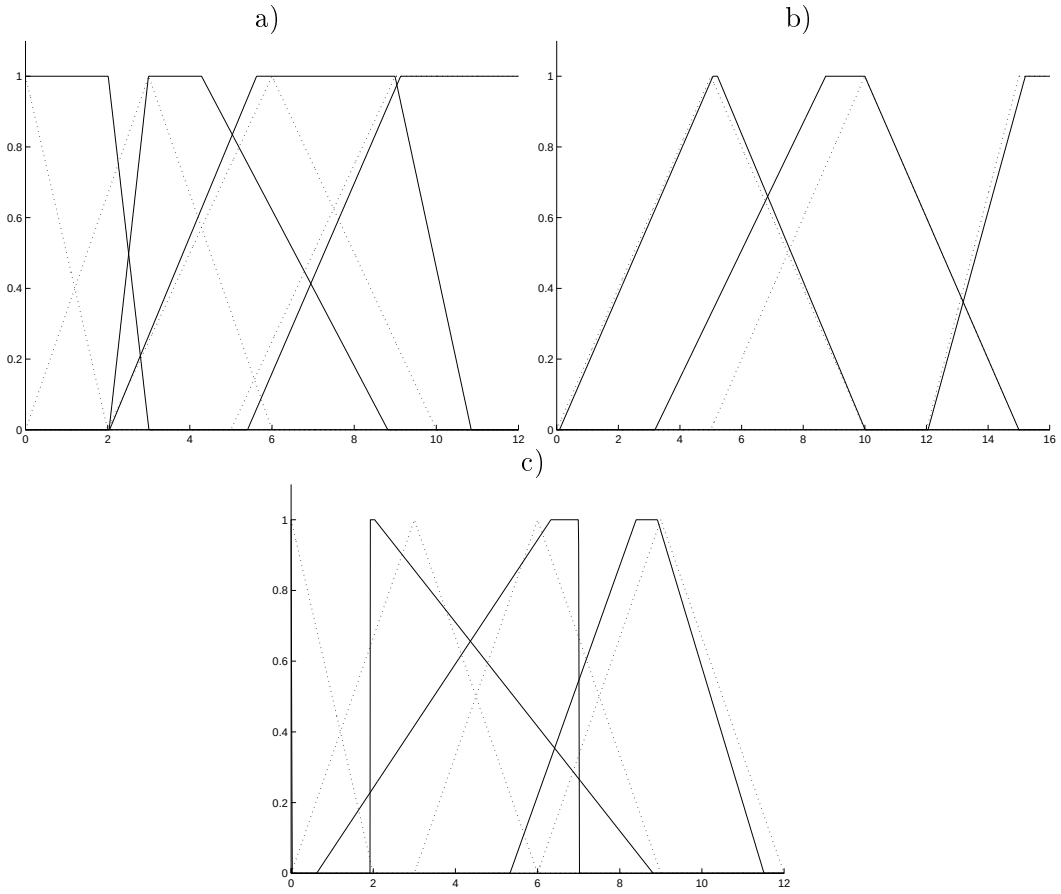


Figure 7.17: Membership functions before (dotted line) and after (solid line) the learning at 1000 vehicles per hour. The location of the first traffic detector is 50 metres before the stop line. a) μ_{Azero}, μ_{Aafew}, $\mu_{Amedium}$ and μ_{Amany}. b) μ_{Qafew}, $\mu_{Qmedium}$ and $\mu_{Qtoolong}$. c) μ_{Ezero}, μ_{Eshort}, $\mu_{Emedium}$ and μ_{Elong}.

The membership functions before and after the learning are presented in Figure 7.17. Most of them changed significantly during the learning, partly because the learning was very long. Functions μ_{Azero}, μ_{Aafew} and $\mu_{Amedium}$ all grew wider and moved rightward. This means that at large traffic volumes the "mean" values of *zero*, *a few* and *medium* are larger.

The growth of μ_{Azero} means that input measurements of 0, 1 or 2 approaching vehicles are all interpreted as a fuzzy *zero* at a large traffic volume. The same phenomenon was seen in Figure 7.11 where the traffic volume was also 1000 vehicles per hour and the location of the first traffic detector was 100 metres from the stop line.

The function μ_{Ezero} shrank almost to zero, and μ_{Eshort} and $\mu_{Emedium}$ turned to face each other. These observations were already seen in Figure 7.14 and here they are even more significant.

Figure 7.18 shows how some of the membership function parameters changed during the learning. Figure 7.18 a shows all the parameters of $\mu_{Amedium}$. The other parameters except p_3 changed very modestly, as was already seen in Figure 7.17 a. In

Figure 7.18 b, the parameters p_1 and p_4 of both μ_{Eshort} and $\mu_{Emedium}$ are shown. The parameters of μ_{Eshort} increased and the parameters of $\mu_{Emedium}$ decreased and they actually crossed each other so that p_1 of μ_{Eshort} passed over p_1 of $\mu_{Emedium}$ and p_4 of μ_{Eshort} passed over p_4 of $\mu_{Emedium}$. The same phenomenon was seen in Figure 7.17 c.

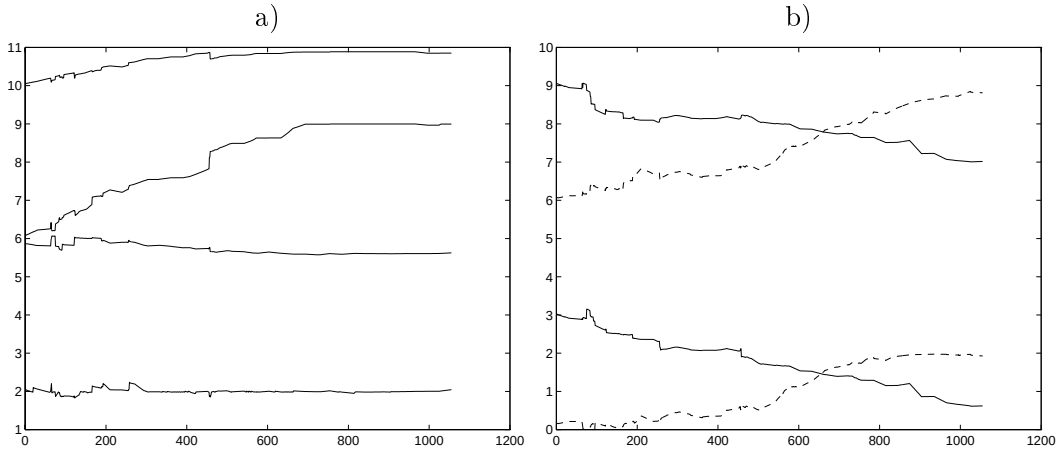


Figure 7.18: The change in some of the membership function parameters during the learning at a traffic volume of 1000 vehicles per hour. The first traffic detector is located 50 metres before the stop line. a) The parameters p_1, p_2, p_3 and p_4 (from bottom to top) of $\mu_{Amedium}$. b) The parameters p_1 and p_4 of μ_{Eshort} (dashed lines) and $\mu_{Emedium}$ (solid lines).

7.8 Learning at an Increasing Traffic Volume

The reinforcement learning algorithm was tested in a changing environment where the traffic volume was increased step by step during the learning. In the beginning, the traffic volume was 300 vehicles per hour. This situation was taught during 100 simulation runs. After this, the volume increased to 350 vehicles per hour for another 100 simulation runs, and then to 400 vehicles per hour and so on. Finally, the traffic volume was 850 vehicles per hour.

The size of the hidden layer in the neural network AEN was 6 in this experiment. The seed of the pseudo-random generator of HUTSIM was changed between every 20 simulation runs, unlike in the experiments in Sections 7.6 and 7.7 where the seed was changed after every 30 simulation runs.

Figure 7.19 reveals that the learning was not very successful. The delay after the learning drifts further and further away from the delay before the learning, as the traffic volume increases. The network had only learned 100 simulation runs per each traffic volume, which was quite a short learning period. It is possible that each traffic volume would have needed more simulation runs, say at least 200 or 300 runs for each volume. In that case another problem might have been faced: continuing the learning too long results in an increase in the delay, as was the case in Figure 7.10.

Increasing the learning time means that the algorithm adjusts the membership func-

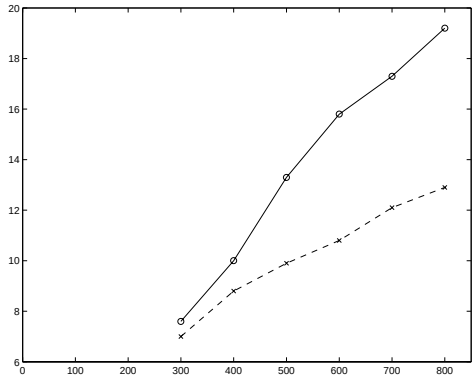


Figure 7.19: Delays before (dashed line) and after (solid line) the learning at an increasing traffic volume. The x axis is the traffic volume.

tion parameters for each situation separately before moving to a new situation — the algorithm does not actually keep up with its environment any more. One of the aims in this thesis was to find out whether the GARIC algorithm can be implemented in the field so that the parameters of the membership functions are adjusted all the time in changing traffic situations. The conclusion after this experiment is that this is not possible.

In this experiment, the traffic volume increased steadily. A more realistic situation would have been the one in which the volume first increases and then decreases, for example. In any case the result would have been the same: the reinforcement learning algorithm cannot follow a changing traffic situation.

Chapter 8

Discussion

8.1 Conclusions

The reinforcement learning algorithm adjusted the membership functions successfully in some cases and unsuccessfully in other cases. The algorithm was used in six different cases of constant traffic volume: the traffic volumes were 300, 500 and 1000 vehicles per hour and the location of the first traffic detector was either 50 or 100 metres before the stop line in all cases. In addition, the algorithm was used with an increasing traffic volume. Of these cases, new membership functions were found for constant traffic volumes of 500 and 1000 vehicles per hour and for both traffic detector locations, making a total of four different successful cases. The case of an increasing traffic volume was not successful.

The initial membership functions were the best for a constant traffic volume of 300 vehicles per hour. Comparing Figures 7.7 and 7.12 shows that at 500 and 1000 vehicles per hour, with both traffic detector locations, the new membership functions decrease the vehicular delay about 0.5 seconds. The new membership functions for these four different cases, shown in Figures 7.9, 7.11, 7.14 and 7.17, share only one common property in which they differ from the initial membership functions. In all four cases, the membership function μ_{Aafew} now begins at 2 vehicles instead of 0. There are other similarities, too, but they do not apply to all four cases, and they were discussed in Sections 7.6 and 7.7.

The reinforcement learning algorithm faced many difficulties which may have affected the performance of the algorithm. These difficulties are discussed in Section 8.2. In the light of these problems and other observations made in this thesis it seems that other reinforcement learning algorithms would not have performed significantly better than the GARIC algorithm used in this thesis.

The conclusion of the practical applicability of the reinforcement learning algorithm is that different membership functions are optimal at different traffic situations. The fuzzy traffic signal controller must identify the traffic volume and choose the proper membership functions accordingly. The algorithm in its current form cannot be used in the field, because the time needed to adjust the membership functions is too long. Also, one cannot know how long the learning should go on, as the learning times at constant traffic volumes were quite different. The applicability of the algorithm

remains a topic for further studies, especially if some of the problems presented in Section 8.2 can be overcome.

The new membership functions do not always look very nice, and it is possible that hand-tuning after the learning would give still better results. It was difficult to find membership function parameters which are definitely better than the initial ones. Obviously the initial membership functions were already fairly good. The modification of the rule base discussed in Section 7.3 caused a larger decrease in the delay than any modifications in the membership function parameters. The rules are actually more important than the membership functions, and as long as the membership functions do not produce any gaps in the rule base, the fuzzy traffic signal controller works fairly well. It is very important that there are no gaps in the rule base, that is, every input must fire at least one rule. Moreover, the learning is more effective if more than one rule is fired.

The membership function parameters or the delay did not converge to any certain level. Usually in supervised learning in neural networks, the error during the learning decreases to almost zero; that is, the output of the network becomes almost the same as the desired output. The learning is continued until some pre-specified error level is reached. In this work one could not measure such an error in the output of the controller network. Naturally, the delay cannot decrease to zero or to any other level, as it is not known which level of delay is realistic. So one had to keep on teaching the membership functions, and examine how they behaved in separate test cases during the learning. If the delay in the test cases (one or several simulation runs of two hours) was larger than in the beginning, the learning was continued. After the first 100 or 200 learning loops the delay was usually large, then it started decreasing, and if the learning was continued for a very long time, say 1000 loops, the delay increased again. The parameters of the membership functions were often either decreasing or increasing slowly in time, and obviously after a very long learning time they had moved too far away from the initial values, as the delay increased again. One had to check the parameters from time to time and in the end choose those which gave the smallest delay in the test case.

For practical reasons, the modification of the membership function parameters during the learning was restricted in a few ways. The parameters were restricted to nonnegative values. Without this restriction, some of the parameters actually fell below zero. The input measurements of the controller were always nonnegative, so the membership functions were not needed on the negative side. Also among the parameters of each membership function a restriction $p_1 \leq p_2 \leq p_3 \leq p_4$ was used, so that the membership functions remained trapezoids. This restriction did not hinder the membership functions sliding over each other, however. This is seen for example in Figure 7.17 c, where p_1 of μ_{Eshort} has moved to the right and p_1 of $\mu_{Emedium}$ has moved to the left and they have actually passed over each other. Also, now it was possible that the membership functions slided away from each other so much that at some input, the values of all membership functions were zero. This means that no rule was fired, and there was a gap in the rule base. This was a possible problem with the membership functions of APP and QUE, but not with EXT, as the input measurements were not fed into the membership functions of EXT. It would not be simple to construct a restriction for this kind of gaps in the rule base, as the parameters changed with respect to each other in many different ways during the learning.

8.2 Reasons for the Poor Performance of the Reinforcement Learning Algorithm

8.2.1 Robustness of the Fuzzy Controller

The fuzzy traffic signal controller in the HUTSIM simulation system is quite insensitive. The output of the controller (the extension of green time) is rounded to the nearest integer. Also, input values of the controller are always integer-valued, as the number of vehicles is naturally an integer. Due to these reasons, minor changes in the membership function parameters have no effect on the output of the controller, and thus on the delay in the system. It cannot be seen whether a minor adjustment is a change for the better or for the worse.

The simulation system presents each membership function as a vector of values of the function at integer points 0, 1, ... , 12 or 0, 1, ... , 16. Appendix A.1 shows an input file in which these function values are listed. The values are now presented using two decimals but the user is free to choose the precision. Only the membership function values at these integer points matter in the fuzzy inference. The form of the membership function (triangular, trapezoidal, gaussian *etc.*) does not make a substantial difference in the output of the fuzzy controller.

On the whole, fuzzy control is quite insensitive, too. Other control systems are often more detailed. If the input fires only one rule in the fuzzy control system, the output of the rule base is just the output of this particular rule. The degree to which the rule was fired — the rule firing strength w — does not matter. In our rule base, the membership functions do not overlap largely, so the input is often in the range of only one membership function, and only one rule is fired. The output of the controller remains the same even if the inputs differ slightly.

8.2.2 Lack of Stochastic Exploration

One reason why effective neural learning was not possible was the lack of stochastic exploration. The Stochastic Action Modifier (see Section 5.3.4) of the GARIC algorithm could not be realized because the actual control signal was calculated in and given by the HUTSIM simulation system.

It has been proposed [41] that if the control action is deviated by a random amount, the system can find a better action and also discover which actions are not suitable. In other words, the system "tests" different control actions to see which lead to a desired result.

The only part of our algorithm that resembles stochastic search in any way is the deviation of the membership functions by a small random amount in the case when the learning slows down too much. This was discussed in Section 6.2.3. Note that the stochastic exploration in the Stochastic Action Modifier deviates the *output* of the fuzzy controller, whereas in our application the *parameters* of the fuzzy controller are deviated.

8.2.3 Credit Assignment Problem

The problem of credit assignment was briefly discussed in Section 5.3.1. This is the problem of determining which parts of the system (which actions, parameters or decisions *etc.*) deserve credit for improvements in the overall performance of the system [2]. The problem is more challenging if information on the performance of the system is received a long time after the action. Reinforcement learning tries to tackle this problem which many other learning algorithms leave without attention.

In the cart-pole balancing problem (see Section 5.3.1) the credit assignment problem is present, as the pole falls down after a long sequence of cart movements and the system does not know which movement caused the falling. A similar problem occurs in traffic signal control: the vehicular delay in the system is revealed after the simulation run is completed, and we do not know which decisions of green time extension were successful and which were not.

The reason why reinforcement learning could tackle the credit assignment problem successfully in the cart-pole balancing system is that in cart-pole balancing, the neural networks were updated after each action, even if information of the success of the action was not available. In traffic signal control the networks are updated only after the simulation run is completed. The simulation cannot be "frozen" for parameter updating in the middle of the simulation run. In other words, the reinforcement learning in cart-pole balancing is on-line learning whereas in traffic signal control it is off-line learning.

To ease the problem of credit assignment in traffic signal control, the learning system would need information about the delays caused by individual signal control decisions. In the current HUTSIM simulation system this information is not available. Instead, a "pooled" delay is used in the learning, as discussed in Section 6.2.3. It would be beneficial to know exactly which vehicles belong to the approaching (APP) and queuing (QUE) vehicles each time the fuzzy controller makes the decision of green time extension; the delays of these vehicles would then be assigned to this particular control decision and parameters contributing to this decision could be updated.

8.3 Suggestions for Future Work

8.3.1 Implementation in the Field

In this work, the neurofuzzy traffic signal controller has only been used in a simulation environment. Its applicability in the field was not tested. There are a few alternatives for the implementation of the reinforcement learning algorithm in the field.

First, different membership functions can be found for different traffic situations using a simulation system. In this thesis, a few different situations were examined, but many others still remain. It is an unsolved question how many different membership function sets should be used: in this work, even quite similar traffic volumes of 300 and 500 vehicles per hour had different membership functions. Another aspect is whether different directions should have different membership functions, if the traffic

volumes differ. The number of different traffic situations soon becomes burdensome.

A second alternative is to apply the algorithm on-line in the field so that the parameter set is modified all the time. The reinforcement learning algorithm is then used to modify the parameters at every signal control action. In the experiments this was not possible, because the parameters could only be modified after the simulation run was completed and not during the simulation — the learning was off-line instead of on-line. The problem with off-line learning was discussed in Section 8.2.3.

In the light of the experiments made in this thesis it is not known whether the on-line learning is successful or not, as it could not be tested. Another problem is the capacity of the signal controller. Computing one loop of the algorithm at each signal control action requires an effective computer, for example a 266 MHz Pentium.

Observations of the behaviour of the traffic from the previous hour or the previous days *etc.* cannot be used in the learning, because the idea in the learning is just to modify the parameters a little and see how it affects the delays, and then modify again, and so on. During the learning the delays may first increase, and it is not beneficial in a real-world application to use membership functions whose performance is inferior. The only way to utilize old observations is to compare the current traffic situation with the old situation and if they are similar, choose the same parameter set as the one used in the old situation.

A third alternative of the implementation of the reinforcement learning algorithm is to use the simulation system in the field, too. In this case, the real traffic situation is fed into the simulator from time to time, and the algorithm modifies the parameters in the simulator. In this way, the trial and error during the learning need not be tested with real traffic, and the new membership functions can be introduced after they have proved successful in the simulation environment. The real traffic volumes must be quite stable so that the learning algorithm in the simulator can keep up with the changes — in the experiments in this work, learning of the membership functions for one traffic volume took several hundred minutes using a 266 MHz Pentium. The time needed to learn is impractical at the moment. Also, it is not known how long the learning should go on — in this work, the user must decide when to stop the learning.

8.3.2 Hand-Tuning the Membership Functions

The membership functions produced by the reinforcement learning algorithm do not always look very nice. The algorithm cannot consider all membership functions at the same time in the way a human can. A change in one function sometimes needs a change in another function, too, to avoid gaps between the functions. For example in Figures 7.9 and 7.14 there are gaps between the membership functions, and hand-tuning them might result in a better control performance. Of course, such modifications require testing before they can be accepted.

8.3.3 Changes in the Fuzzy Inference System

The forms of the membership functions, the forms of the union and intersection operators and the defuzzification methods all affect the control performance of the system. In this work, the membership functions were triangles or trapezoids. Other membership functions remain to be investigated. Gaussian bell curves are considered more "fuzzy" than triangles and trapezoids. However, the form of the membership function may not affect the controller extensively, as was discussed in Section 8.2.1.

The intersection operator was interpreted as the soft minimum, although the product combiner could have been used, too. The defuzzification method used was a combination of the center of area and mean of maximum methods applied locally to each rule. Other defuzzification methods might have yielded different results. The union and intersection operators and the defuzzification method used in this work were mainly dictated by the use of the HUTSIM simulation program — the fuzzy inference in HUTSIM uses these operators and methods.

The rule base can be modified, too, and different rules can have different importance weight factors. Using rule importance weights the controller can be used in multi-objective control. Different — and often conflicting — objectives of signal control include *e.g.* the minimization of delay, fuel consumption, pollution, noise, pedestrian waiting time or delays of public transport, and maximization of safety. Different rules can emphasize different objectives. Another aspect of multi-objectivity is how different antecedent constraints in one rule are combined. In the current fuzzy traffic signal controller, the minimum combiner is used, and the constraint which is fulfilled to the lowest degree (the constraint whose membership function value at the current input is the lowest) automatically determines the degree to which this rule is fired. Using the product combiner would take into account other antecedent constraints, too, as discussed in Section 2.3, and the controller would be more "multi-objective" than it is at the moment.

It was found out that in the rule base used by HUTSIM, some input values do not fire any rule, and the output of the fuzzy controller in this situation is zero. A majority of these situations were now handled by making a small change in the rule base as discussed in Section 7.3, but no systematic studies on "missing" rules were done. This systematic study remains a topic for further work.

A simple extension to the fuzzy inference process would be the inclusion of fuzzy complements or *not V* linguistic variables, whose membership functions are calculated using Formula 2.14. Another extension is the ability to use the "or" combiner in addition to the "and" combiner in the rule antecedents. After these modifications one could use rules such as "if *APP* is *not zero* and *QUE* is *not too long*, then *EXT* is *not zero*" or "if *APP* is *zero* or *QUE* is *too long*, then *EXT* is *zero*".

Some algorithms [41] update the fuzzy rule base in addition to membership function parameters using a reinforcement learning technique. This could be applied to the rule base of the fuzzy traffic signal controller. Moreover, different rule sets could be developed for different traffic situations.

8.3.4 Additional Input Variables

The GARIC algorithm could be expanded for example by including the time of day as an additional input. The system might be able to recognize the dependence of traffic situation on the time. A neural network could even be able to track a time series in the traffic situation. Time series data are difficult to test in a simulation environment, because the time series must first be created in the simulation system itself. The data are no longer "real world" data but behave in a nice way which is of course easy to detect. The identification of this simulator-created time series data gives no information on how well "real world" data would be identified.

Some algorithms [8], [13] use the error signal and its derivative as inputs to a fuzzy controller. An example of this kind of rule in traffic signal control is "if *delay* is *large* and *change in delay* is *positive large*, then *change in parameter* is *large*".

8.3.5 Modelling

The current traffic signal control system is not based on system modelling. As discussed in Section 5.1, reinforcement learning algorithms do not estimate a model of the system at each time step and use this model in choosing the control action.

A different approach would be to build a model of the delay. Independent variables of the model would be traffic volume, *APP, QUE*, *EXT* and many other values describing the traffic situation. In this approach, delay minimization would be based on the model of the traffic situation.

Appendix A

HUTSIM files

A.1 HUTSIM Input File `twostg.fuz`

The input file `twostg.fuz` contains the values of the membership functions and the rule base used in the HUTSIM simulation system. During the learning, parameters of the membership functions are modified, and the membership function values in this file are updated accordingly. The following file contains the initial membership function values before the learning.

The rule base was modified from that used in [46], [47] as discussed in Section 7.3, and the rule base in this file is thus different from the rule base introduced in Section 6.1.3: the second rule in the first, second, third and fourth rule sets is changed from "if *APP* is *a few* and if *QUE* is *a few*, then *EXT* is *short*" to "if *APP* is *a few* and if *QUE* is *less than medium*, then *EXT* is *short*".

```
2101  FUZZY INFERENCE INITIALIZATION FILE;
;
BLOCK 1, MEMBERSHIP FUNCTIONS;
;
Any Value:       11  0  16  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
  1.0  1.0  1.0  1.0  1.0  1.0  1.0
;
App.veh.Memb.Func.   A =    0    1    2    3    4    5    6    7    8    9
  10   11   12;
A(zero):         11  1  12 1.00 0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
  0.00 0.00 0.00
A(a few):        11  2  12 0.00 0.33 0.67 1.00 0.67 0.33 -0.00 0.00 0.00 0.00
  0.00 0.00 0.00
A(medium):       11  3  12 0.00 0.00 0.00 0.25 0.50 0.75 1.00 0.75 0.50 0.25
  0.00 0.00 0.00
A(many):         11  4  12 0.00 0.00 0.00 0.00 0.00 0.00 0.25 0.50 0.75 1.00
  1.00 1.00 1.00
;
Q length. Memb.Func. Q =    0    1    2    3    4    5    6    7    8    9
  10   11   12   13   14   15   16;
Q(a few):        11  5  16 0.00 0.20 0.40 0.60 0.80 1.00 0.80 0.60 0.40 0.20
  -0.00 0.00 0.00 0.00 0.00 0.00 0.00
Q(medium):       11  6  16 0.00 0.00 0.00 0.00 0.00 0.00 0.20 0.40 0.60 0.80
  1.00 0.80 0.60 0.40 0.20 -0.00 0.00
```

```
Q(too long):     11  7  16 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
  0.00 0.00 0.00 0.33 0.67 1.00 1.00
;
mt(A) Memb.Func. mt(A) =    0    1    2    3    4    5    6    7    8    9
  10   11   12;
mt(A)(a few):    11  8  12 0.00 0.00 0.00 0.00 0.33 0.67 1.00 1.00 1.00 1.00
  1.00 1.00 1.00
mt(A)(medium):   11  9  12 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.25 0.50 0.75
  1.00 1.00 1.00
;
lt(Q) Memb.Func. lt(Q) =    0    1    2    3    4    5    6    7    8    9
  10   11   12   13   14   15   16;
lt(Q)(a few):    11 10  16 1.00 0.80 0.60 0.40 0.20 0.00 0.00 0.00 0.00 0.00
  0.00 0.00 0.00 0.00 0.00 0.00 0.00
lt(Q)(medium):   11 11  16 1.00 1.00 1.00 1.00 1.00 1.00 1.00 0.80 0.60 0.40 0.20
  0.00 0.00 0.00 0.00 0.00 0.00 0.00
;
No Value:        11 12  16  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
  0.0  0.0  0.0  0.0  0.0  0.0  0.0
;
BLOCK 2, FUZZY RULE SETS;
;
Extension Lengths           0    1    2    3    4    5    6    7    8    9
  10   11   12;
Zero:            11 13  12 1.00 0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
  0.00 0.00 0.00
Short:           11 14  12 0.00 0.33 0.67 1.00 0.67 0.33 -0.00 0.00 0.00 0.00
  0.00 0.00 0.00
Medium:          11 15  12 0.00 0.00 0.00 0.00 0.33 0.67 1.00 0.67 0.33 -0.00
  0.00 0.00 0.00
Long:            11 16  12 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.33 0.67 1.00
  0.67 0.33 -0.00
;
RULE SET 1, after min green, 5th second;
;
RULE1; IF       A = zero                        THEN E1=zero;
RULE1: 21 1  3  13 1                0                0
RULE2; IF       A =a few     AND Q = lt(medium)    THEN E1=short;
RULE2: 21 2  3  14 2                11                3
RULE3; IF       A =mt(a few) AND Q = any      THEN E1=medium;
RULE3: 21 3  3  15 8                0                6
RULE4; IF       A =mt(medium)AND Q = any      THEN E1=long;
RULE4: 21 4  3  16 9                0                9
RULE5; IF       A =None      AND Q = None      THEN E1=None;
RULE5: 21 5  3  0  12               12               12
;
RULE SET 2, after the first extension, 5th + E1 seconds
;
RULE6; IF       A = zero                        THEN E2=zero;
RULE6: 21 6  3  13 1                 0                0
RULE7; IF       A = a few     AND Q = lt(medium)    THEN E2=short;
RULE7: 21 7  3  14 2                 11                3
RULE8; IF       A = medium    AND Q = any      THEN E2=medium;
RULE8: 21 8  3  15 3                 0                6
RULE9; IF       A = many      AND Q = any      THEN E2=long;
RULE9: 21 9  3  16 4                 0                9
RULE10; IF      A =None      AND Q = None      THEN E1=None;
RULE10: 21 10 3 0  12               12               12
;
RULE SET 3, after the second extension, 5th + E1 + E2
;
RULE11;IF       A = zero                         THEN  E3=zero;
```

```
RULE11: 21 11 3 13 1                0                    0
RULE12;IF       A = a few    AND Q = lt(medium)      THEN  E3=short;
RULE12: 21 12 3 14 2                11                    3
RULE13;IF       A = medium   AND Q = lt(medium) THEN  E3=medium;
RULE13: 21 13 3 15 3                11                    6
RULE14;IF       A = many     AND Q = lt(medium) THEN  E3=long;
RULE14: 21 14 3 16 4                11                    9
RULE15;IF       A =None      AND Q = None      THEN E1=None;
RULE15: 21 15 3 0 12                12               12
;
RULE SET 4, after the third extension, 5th + E1 + E2 + E3
;
RULE16;IF       A = zero                        THEN   E4=zero;
RULE16: 21 16 3 13 1                0                    0
RULE17;IF       A =mt(a few) AND Q = lt(medium)      THEN   E4=short;
RULE17: 21 17 3 14 8                11                    3
RULE18;IF       A = medium   AND Q = lt(medium) THEN   E4=medium;
RULE18: 21 18 3 15 3                11                    6
RULE19;IF       A = many     AND Q = lt(a few)  THEN   E4=long;
RULE19: 21 19 3 16 4                10                    9
RULE20;IF                       Q = too long          E4=zero;
RULE20: 21 20 3 0 0                 7                     0
;
RULE SET 5, after the fourth extension, 5th + E1 + E2 + E3 + E4
;
RULE21;IF       A = zero                        THEN    E5=zero;
RULE21: 21 21 3 13 1                 0                    0
RULE21;IF       A = mt(a few) AND Q = a few     THEN    E5=short;
RULE21 :21 22 3 14 8                 5                    3
RULE22;IF       A = medium    AND Q = lt(a few) THEN    E5=medium;
RULE22: 21 23 3 15 3                10                    6
RULE23;IF       A = many      AND Q = lt(a few) THEN    E5=long;
RULE23: 21 24 3 16 4                10                    9
RULE24;IF                      Q = too long          E4=zero;
RULE24: 21 25 3 0 0                 7                     0
;
GENERAL PARAMETERS
;
NUMBER OF RULE SETS: 101 5
OUTPUT RANGE: 110  0   12
OUTPUT SCALING: 111  1   0
DEFUZZIFICATION: 112 2
;
END OF FILE: 1000
```

A.2 HUTSIM Output File `hutsim.out`

The following is an example of the output file `hutsim.out` of the HUTSIM simulation system. This file lists the signal control actions and the time instants of vehicles entering or exiting the model. The file is read as follows. The first column lists the time of the observation in seconds. The second column contains the row indicator: 1 corresponds to vehicle information, 2 to signal information and 102 to signal control actions. In this work, the most important rows are vehicle information and signal control action rows.

In vehicle information rows, the time in the first column is the time when the vehicle exits the model. The vehicle enters the model at time shown in the eighth column on the same row, and the delay of this vehicle is shown in the ninth column.

In signal control action rows, the third column shows the number of the signal controller in question (in our example, this is always 53 or 54). The length of the green extension *EXT* is shown in the fourth column and the control input variables *APP* and *QUE* in the fifth and sixth columns. The eighth column shows the number of this extension (first, second and so on), which determines the rule set used.

```
1344  HUTSIM  Delay-File
   17   102    54     0     1     0     0     1     0     0     0     0
   28   102    53     0     1     1     0     1     0     0     0     0
   40   102    54     0     1     0     0     1     0     0     0     0
   44     1     0     0     1    21     1     4     5     0   497     0
   45     1     0     2     1    21     1     6     5     0   496     0
   51     1     0     3     1    21     1    13    -0     0   496     0
   52   102    53     0     0     0     0     1     0     0     0     0
   58     1     0     1     4    11     2     6    19     1   502     1
   60     1     0     7     1    12     2    26    -0     0   502     0
   63   102    54     0     1     0     0     1     0     0     0     0
   69     1     0     5     1    21     1    22    11     1   497     0
   70     1     0     4     1    22     1    21     9     1   495     0
   70     1     0     6     1    21     1    24    15     1   497     1
   75   102    53     0     1     0     0     1     0     0     0     0
   85     1     0    11     1    12     2    47    -0     0   501     0
   86   102    54     3     2     0     0     1     0     0     0     0
   89     1     0     9     1    21     1    45    16     1   496     0
   89   102    54     4     3     1     0     2     0     0     0     0
   93   102    54     4     3     2     0     3     0     0     0     0
   93     1     0     8     1    22     1    42    12     1   497     0
   96     1     0    10     1    22     1    46     8     1   496     1
   97   102    54     0     1     2     0     4     0     0     0     0
  102     1     0    12     1    22     1    50     1     0   496     0
  107     1     0    13     1    12     2    52    10     1   502     0
  109     1     0    15     1    12     2    70     5     0   501     1
  109   102    53     6     6     0     0     1     0     0     0     0
  111     1     0    17     4    12     2    73     0     0   501     0
  113     1     0    18     1    12     2    75     2     0   501     1
  115   102    53     3     2     0     0     2     0     0     0     0
  115     1     0    21     1    12     2    79     3     0   503     0
  117     1     0    22     1    11     2    82    -0     0   502     0
  118   102    53     0     1     1     0     3     0     0     0     0
  128     1     0    19     4    21     1    77    14     1   497     0
  130   102    54     3     2     1     0     1     0     0     0     0
  131     1     0    23     1    21     1    86     7     0   495     1
  132     1     0    14     1    22     1    60    21     1   495     0
```

132	1	0	25	1	21	1	89	6	0	497	1
133	102	54	0	1	2	0	2	0	0	0	0
133	2	1	0	9	35	3	1	16	0	0	0
134	1	0	16	1	22	1	70	19	1	496	1
136	1	0	20	1	22	1	78	16	1	497	1
137	1	0	28	1	21	1	95	-0	0	496	0
137	1	0	26	1	22	1	89	4	0	496	1
139	1	0	31	4	21	1	103	0	0	495	0
139	1	0	27	1	22	1	93	2	0	495	0
144	102	53	0	1	0	0	1	0	0	0	0
145	2	2	0	6	26	3	1	18	0	0	0
149	1	0	29	2	12	2	97	13	1	502	0
152	1	0	24	1	11	2	88	15	1	501	0
152	1	0	36	4	12	2	116	1	0	503	0
153	1	0	30	1	11	2	100	21	1	500	1
155	1	0	35	1	11	2	110	16	0	500	1
156	102	54	0	1	2	0	1	0	0	0	0
156	2	1	0	6	23	1	0	0	0	0	0
160	1	0	33	1	21	1	108	18	1	496	0
164	1	0	32	1	22	1	107	15	1	496	0
167	102	53	3	2	0	0	1	0	0	0	0
168	1	0	34	1	22	1	109	12	1	497	0
170	102	53	0	1	1	0	2	0	0	0	0
171	2	2	0	9	26	3	0	22	0	0	0
171	1	0	37	1	12	2	125	14	1	501	0
174	1	0	40	4	12	2	136	3	0	501	1
182	102	54	3	3	0	0	1	0	0	0	0
185	102	54	4	3	0	0	2	0	0	0	0
185	1	0	38	1	21	1	129	18	1	498	1
187	1	0	42	1	22	1	140	2	0	495	0
188	1	0	39	1	21	1	133	15	1	496	1
189	1	0	44	1	22	1	145	4	0	497	1
189	102	54	3	2	0	0	3	0	0	0	0
189	1	0	41	1	21	1	139	10	1	495	1
191	1	0	45	1	22	1	151	-0	0	496	0
192	102	54	7	2	2	0	4	0	0	0	0
199	102	54	7	2	4	0	5	0	0	0	0
201	1	0	43	1	12	2	145	16	1	502	0
202	1	0	47	2	12	2	166	3	0	503	1
204	1	0	49	1	12	2	169	1	0	501	0
205	1	0	50	1	11	2	170	-0	0	502	0
206	102	54	6	1	4	0	6	0	0	0	0
207	2	1	0	30	50	1	1	0	0	0	0
211	1	0	51	1	12	2	173	-0	0	502	0
212	1	0	53	3	12	2	176	0	0	503	0
218	102	53	3	3	0	0	1	0	0	0	0
220	1	0	55	3	12	2	179	0	0	503	0
221	102	53	0	1	0	0	2	0	0	0	0
221	2	2	0	9	50	5	1	26	0	0	0
222	1	0	56	2	12	2	184	3	0	500	0
224	1	0	57	1	12	2	190	-0	0	502	0
232	102	54	3	3	0	0	1	0	0	0	0
235	102	54	0	1	0	0	2	0	0	0	0
236	2	1	0	9	29	2	1	31	0	0	0
236	1	0	48	1	21	1	168	29	1	497	0
238	1	0	52	1	21	1	174	30	1	495	1
239	1	0	54	1	21	1	179	22	1	496	1
240	1	0	46	1	22	1	165	27	1	496	0
241	1	0	59	1	21	1	192	17	1	496	1
241	1	0	58	1	22	1	191	2	0	495	1
244	1	0	60	1	21	1	200	1	0	496	0
247	102	53	3	2	0	0	1	0	0	0	0

```
   248     1     0    62     1    11     2   206     6     0   501     0
   250   102    53     3     2     0     0     2     0     0     0     0
   250     1     0    61     4    12     2   203    13     1   501     0
   251     1     0    64     1    11     2   209     2     0   502     0
   253   102    53     3     2     0     0     3     0     0     0     0
   253     1     0    63     1    12     2   207     9     0   501     1
   254     1     0    67     1    11     2   221     0     0   502     0
   255     1     0    65     1    12     2   212     5     0   501     1
   256   102    53     0     1     1     0     4     0     0     0     0
   256     2     2     0    15    35     1     1     0     0     0     0
   267   102    54     3     2     1     0     1     0     0     0     0
   268     1     0    66     1    22     1   212    11     1   497     0
   269     1     0    68     1    22     1   222     8     0   496     1
   270   102    54     0     1     1     0     2     0     0     0     0
   271     2     1     0     9    35     3     1    16     0     0     0
   275     1     0    69     1    22     1   228     0     0   496     0
   279     1     0    70     1    22     1   232    -0     0   496     0
   282   102    53     3     2     1     0     1     0     0     0     0
   283     1     0    71     1    11     2   233    18     1   503     0
   283     1     0    72     1    12     2   236    15     1   503     0
   285   102    53     0     1     1     0     2     0     0     0     0
   285     1     0    74     1    12     2   241     8     0   500     0
   286     2     2     0     9    29     2     1    18     0     0     0
   289     1     0    75     1    12     2   246     2     0   502     1
   296     1     0    80     1    21     1   261     4     0   497     0
   297   102    54     3     2     2     0     1     0     0     0     0
   299     1     0    81     1    21     1   263     1     0   496     1
   300   102    54     3     2     2     0     2     0     0     0     0
   302     1     0    73     1    22     1   239    21     1   496     1
   303   102    54     0     1     2     0     3     0     0     0     0
   303     2     1     0    12    32     3     1    16     0     0     0
   306     1     0    76     1    22     1   247    11     1   497     1
   307     1     0    79     1    22     1   258     5     0   495     1
   313     1     0    77     1    12     2   257    22     1   501     1
   314   102    53     4     4     0     0     1     0     0     0     0
   318   102    53     4     3     0     0     2     0     0     0     0
   320     1     0    78     1    11     2   257    12     1   501     0
   322     1     0    82     1    11     2   268    18     1   501     1
   322   102    53     0     1     0     0     3     0     0     0     0
   323     2     2     0    14    37     2     1    20     0     0     0
   323     1     0    85     1    11     2   277    16     0   502     1
   325     1     0    86     1    11     2   280     5     0   502     0
   326     1     0    87     1    11     2   288     7     0   502     1
100    0    End of HUTSIM Output File
HUTSIM 4.2 Delay File Format:
Ti  Ri  Di  Vi  Ty  Ap  De  At  Dl  St  Tr
Ti=Exit time or detector passing time (seconds)
Ri=Row type identifier, vehicle=1
Di=Detector identifier number, 0=no detector=exit
Vi=Vehicle identifier number
Ty=Vehicle type 1..5
Ap=Approach number
De=Destination number
At=Arrival time (seconds)
Di=Delay (seconds)
St=Stop count
Tr=Travel (meters)
```

Bibliography

[1] Jessica Anderson, Tessa Sayers, Michael Bell: *The Objectives of Traffic Signal Control.* Traffic Engineering + Control, Vol. 39, No. 3, March 1998

[2] Andrew G. Barto, Richard S. Sutton, Charles W. Anderson: *Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems.* IEEE Transactions on Systems, Man and Cybernetics, Vol. 13, No. 5, September/October 1983

[3] James C. Bedzek: *Fuzzy Models — What Are They and Why?* IEEE Transactions on Fuzzy Systems, Vol. 1, February 1993

[4] James C. Bedzek: *Fuzziness vs. Probability — Again (!?).* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[5] James C. Bedzek: *The Thirsty Traveler Visits Gamont: A Rejoinder to "Comments on Fuzzy Sets — What Are They and Why?".* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[6] Hamid R. Berenji, Pratap Khedkar: *Learning and Tuning Fuzzy Logic Controllers Through Reinforcements.* IEEE Transactions on Neural Networks, Vol. 3, No. 5, September 1992

[7] M. Brown, C. J. Harris: *A Perspective and Critique of Adaptive Neurofuzzy Systems Used for Modelling and Control Applications.* International Journal of Neural Systems, Vol. 6, No. 2, June 1995

[8] Yung-Yaw Chen, Kao-Zong Lin, Shung-Tang Hsu: *A Self-Learning Fuzzy Controller.* IEEE International Conference on Fuzzy Systems 1992, San Diego, CA, U.S.A, March 1992

[9] A. Cichocki, R. Unbehauen: *Neural Networks for Optimization and Signal Processing.* John Wiley & Sons, 1993

[10] Didier Dubois, Henri Prade: *Fuzzy Sets — A Convenient Fiction for Modeling Vagueness and Possibility.* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[11] Augustine O. Esogbue, James A. Murrell: *A Fuzzy Adaptive Controller Using Reinforcement Learning Neural Networks.* IEEE International Conference on Fuzzy Systems 1993, San Francisco, CA, U.S.A., March 1993

[12] Yoichi Hayashi, James J. Buckley, Ernest Czogala: *Fuzzy Neural Network with Fuzzy Signals and Weights.* International Journal of Intelligent Systems, Vol. 8, No. 4, 1993

[13] Yoichi Hayashi, Ernest Czogala, James J. Buckley: *Fuzzy Neural Controller.* IEEE International Conference on Fuzzy Systems 1992, San Diego, CA, U.S.A., March 1992

[14] Robert Hecht-Nielsen: *Neurocomputing.* Addison-Wesley, 1990

[15] John Hertz, Anders Krogh, Richard G. Palmer: *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991

[16] Ellen Hisdal: *Interpretative Versus Prescriptive Fuzzy Set Theory.* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[17] Lasse Holmström, Teuvo Kohonen: *Neuraaliverkot.* In: Eero Hyvönen, Ilkka Karanta, Markku Syrjänen (eds.): *Tekoälyn ensyklopedia.* Gaudeamus, 1993 (in Finnish)

[18] Shin-ichi Horikawa, Takeshi Furuhashi, Yoshiki Uchikawa: *On Fuzzy Modelling Using Fuzzy Neural Networks with the Back-Propagation Algorithm.* IEEE Transactions on Neural Networks, Vol. 3, No. 5, September 1992

[19] Jyh-Shing Roger Jang: *ANFIS: Adaptive-Network-Based Fuzzy Inference System.* IEEE Transactions on Systems, Man and Cybernetics, Vol. 23, No. 3, May/June 1993

[20] Jyh-Shig Roger Jang, Chuen-Tsai Sun: *Neuro-Fuzzy Modelling and Control.* Proceedings of the IEEE, Vol. 83, No. 3, March 1995

[21] Erkki Jurva (ed.): *Sumean logiikan mahdollisuudet.* Tekes, 1998 `<URL:http://www.kareltek.fi/opp/sumea/index.html>` (in Finnish)

[22] Arnold Kaufmann, Madan M. Gupta: *Fuzzy Mathematical Models in Engineering and Management Science.* Elsevier Science Publishers, 1988

[23] James M. Keller, Douglas J. Hunt: *Incorporating Fuzzy Membership Functions into the Perceptron Algorithm.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 7, No. 6, November 1985

[24] Shinya Kikuchi, Matti Pursula: *Treatment of Uncertainty in Study of Transportation: Fuzzy Set Theory and Evidence Theory.* Journal of Transportation Engineering, Vol. 124, No. 1, January/February 1998

[25] George J. Klir: *On the Alleged Superiority of Probabilistic Representation of Uncertainty.* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[26] George J. Klir, Tina A. Folger: *Fuzzy Sets, Uncertainty, and Information.* Prentice Hall, 1988

[27] George J. Klir, Bo Yuan: *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prentice Hall, 1995

[28] Teuvo Kohonen: *Self-Organization and Associative Memory.* Springer-Verlag, 3rd edition, 1989

[29] Teuvo Kohonen: *The Self-Organizing Map.* Proceedings of the IEEE 78, pp. 1464–1480, 1990

[30] Teuvo Kohonen: *Self-Organizing Maps. Optimization Approaches.* In: T. Kohonen, K. Mäkisara, O. Simula, J. Kangas (eds.): *Artificial Neural Networks.* Proceedings of the 1991 International Conference on Artificial Neural Networks (ICANN91), Espoo, Finland, June 24-28, Vol. 2, 1991

[31] Bart Kosko: *The Probability Monopoly.* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[32] Bart Kosko: *Fuzzy Systems as Universal Approximators.* IEEE Transactions on Computers, Vol. 43, No. 11, November 1994

[33] Bart Kosko: *Fuzzy Engineering.* Prentice Hall, 1997

[34] Iisakki Kosonen: *HUTSIM — Simulation Tool for Traffic Signal Control Planning.* Helsinki University of Technology, Transportation Engineering, Publication 89, 1996

[35] *A Traffic Simulation Tool — HUTSIM*
<URL:http://www.hut.fi/Units/Transportation/Research/hutsim.html>

[36] Iisakki Kosonen, Matti Pursula: *Object-oriented and Rule-base Modelling — Experiences from Traffic Signal Simulation.* In: Peter Jan Pahl, Heinrich Werner (eds.): *Computing in Civil and Building Engineering.* Proceedings of the Sixth International Conference on Computing in Civil and Building Engineering, Berlin, Germany, July 12–15, 1995

[37] Pertti Laininen: *Sovellettu todennäköisyyslasku.* Helsinki University of Technology, Systems Analysis Laboratory, Research Report B20, 1996 (in Finnish)

[38] Michael Laviolette, John W. Seaman, Jr.:*The Efficacy of Fuzzy Representations of Uncertainty.* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[39] Chuen Chien Lee: *Fuzzy Logic in Control Systems: Fuzzy Logic Controller — Part 1, Part 2.* IEEE Transactions on Systems, Man and Cybernetics, Vol. 20, No. 2, March 1990

[40] Chin-Teng Lin, C. S. George Lee: *Neural-Network-Based Fuzzy Logic Control and Decision System.* IEEE Transactions on Computers, Vol. 40, No. 12, December 1991

[41] Chin-Teng Lin, C. S. George Lee: *Reinforcement Structure/Parameter Learning for Neural-Network-Based Fuzzy Logic Control Systems.* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[42] Cheng-Jian Lin, Chin-Teng Lin: *Reinforcement Learning for an ART-Based Fuzzy Adaptive Learning Control Network.* IEEE Transactions on Neural Networks, Vol. 7, No. 3, May 1996

[43] *MATLAB® 5.* The Math Works Inc. <URL:http://www.mathworks.com/>

[44] Jorma K. Mattila: *Sumean logiikan oppikirja. Johdatus sumeaan matematiikkaan.* Art House, 1997 (in Finnish)

[45] J. S. Milton, Jesse C. Arnold: *Introduction to Probability and Statistics. Principles and Applications for Engineering and the Computing Sciences.* McGraw-Hill, 1990

[46] Jarkko Niittymäki: *Isolated Traffic Signals — Vehicle Dynamics and Fuzzy Control.* Helsinki University of Technology, Transportation Engineering, Publication 94, 1998

[47] Jarkko Niittymäki, Matti Pursula: *Signal-Group Control Using Fuzzy Logic.* Paper presented at the 9th Mini-EURO Conference on Fuzzy Sets in Traffic and Transport Systems, EURO Working Group on Transportation, Budva, Yugoslavia, September 15-19, 1997. Paper submitted for review in Fuzzy Sets and Systems, Focused Issue on Transportation.

[48] Vesa A. Niskanen: *Sumeat järjestelmät.* In: Eero Hyvönen, Ilkka Karanta, Markku Syrjänen (eds.): *Tekoälyn ensyklopedia.* Gaudeamus, 1993 (in Finnish)

[49] Vesa A. Niskanen: *Inexactness, Vagueness, Impreciseness.* <URL:http://www.helsinki.fi/~niskanen/vague.html>

[50] Vesa A. Niskanen: *Uncertainty, Probability.* <URL:http://www.helsinki.fi/~niskanen/probab.html>

[51] Sergei A. Orlovski: *Calculus of Decomposable Properties, Fuzzy Sets, and Decisions.* Allerton Press, Inc., 1994

[52] C. Pappis, E. Mamdani: *A Fuzzy Logic Controller for a Traffic Junction.* IEEE Transactions on Systems, Man and Cybernetics, Vol. 7, No. 10, 1977

[53] Matti Pursula, Jarkko Niittymäki: *Evaluation of Traffic Signal Control with Simulation. A Comparison of the Pappis-Mamdani Fuzzy Control vs. Vehicle Actuation with the Extension Principle.* Extended Abstract for the 4th Meeting of the EURO Working Group on Transportation, University of Newcastle, Newcastle upon Tyne, UK, September 9-11, 1996

[54] Witold Pedrycz, Armando F. Rocha: *Fuzzy-Set Based Models of Neurons and Knowledge-Based Networks.* IEEE Transactions on Fuzzy Systems, Vol. 1, No. 4, November 1993

[55] Esa Ranta, Hannu Rita, Jari Kouki: *Biometria. Tilastotiedettä ekologeille.* Yliopistopaino, 1994 (in Finnish)

[56] David E. Rumelhart, G. E. Hinton, R. J. Williams: *Learning Internal Representations by Error Propagation.* In: David E. Rumelhart, James L. McClelland (eds.): *Parallel Distributed Processing. Explorations in the Microstructure of Cognition I.* MIT Press, 1986

[57] David E. Rumelhart, James L. McClelland: *Parallel Distributed Processing. Explorations in the Microstructure of Cognition I & II.* MIT Press, 1986

[58] Philip N. Sabes, Michael I. Jordan: *Reinforcement Learning by Probability Matching.* Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, Department of Brain and Cognitive Sciences, A.I. Memo No. 1568, C.B.C.L. Paper No. 134. <URL:ftp://publications.ai.mit.edu/ai-publications/1500-1999/AIM-1568.ps.Z>

[59] Kari Juhani Sane, Iisakki Kosonen: *HUTSIM 4.2 Reference Manual.* Helsinki University of Technology, Laboratory of Transportation Engineering, Publication 90, 1996

[60] Warren S. Sarle: *Neural Network and Statistical Jargon.*
<URL:ftp://ftp.sas.com/pub/neural/jargon.txt>

[61] Warren S. Sarle: *Neural Networks and Statistical Models.* Proceedings of the 19th Annual SAS® Users Group International Conference, April 1994
<URL:ftp://ftp.sas.com/pub/neural/neural1.ps>

[62] Richard S. Sutton, Andrew G. Barto: *Reinforcement Learning. Course Notes.*
<URL:ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/IntroRL/RL1.ps>
<URL:ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/IntroRL/RL2.ps>
<URL:ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/IntroRL/RL3.ps>

[63] Richard S. Sutton, Andrew G. Barto, Ronald J. Williams: *Reinforcement Learning is Adaptive Optimal Control.* IEEE Control Systems, Vol. 12, No. 2, April 1992

[64] *Traffic Control Systems Handbook.* U.S. Department of Transportation, Federal Highway Administration FHWA-IP-85-11, U.S.A., April 1985

[65] Esko Turunen: *Sumean logiikan matematiikka.* Otatieto, 1997 (in Finnish)

[66] Nic Wilson: *Vagueness and Bayesian Probability.* IEEE Transactions on Fuzzy Systems, Vol. 2, No. 1, February 1994

[67] Ronald R. Yager, Dimitar P. Filev: *Essentials of Fuzzy Modelling and Control.* John Wiley & Sons, 1994

[68] Lotfi A. Zadeh: *Fuzzy sets.* Information and Control, Vol. 8, No. 3, 1965

[69] Hans-Jürgen Zimmermann: *Fuzzy Set Theory — and its Applications.* Kluwer Academic Publishers, 1996