Lecture Notes

Artificial Intelligence

Michael Maier

Wintersemester 2015/2016

Lecturer: Dr Piotr Prokopowicz

Faculty of Mathematics, Physics & Technical Sciences

Kazimierz Wielki University

Contents

I Fuzzy Systems 3

1. Fuzzy Sets 3
   - 1.1. Knowledge Representation in a fuzzy set . . . 3
     - 1.1.1. Linguistic Variables and Values . . . 3
   - 1.2. Arithmetic of fuzzy sets . . . 5
   - 1.3. Logical Rules . . . 8
   - 1.4. Fuzzy Numbers . . . 9
     - 1.4.1. α-Cuts . . . 10
     - 1.4.2. Fuzzy Arithmetic . . . 10
     - 1.4.3. Ordering fuzzy numbers . . . 11
   - 1.5. Fuzzy Functions . . . 12
     - 1.5.1. Extension Principle . . . 12
     - 1.5.2. Interval Arithmetic/α-Cuts . . . 12

2. Fuzzy Models 13

II Artificial Neural Networks 16

1. Model of an artificial neuron 16

2. Types of neural networks 17
   - 2.1. Basic feedforward neural networks . . . 17
   - 2.2. Supervised learning . . . 21

III Genetic Algorithms 23

1. Implementation 23
   - 1.1. Encoding and Decoding Schemes . . . 23
   - 1.2. Fitness Evaluation . . . 25
   - 1.3. Selection mechanism . . . 26
     - 1.3.1. Roulette-wheel & Ranking selection . . . 26
     - 1.3.2. Tournament selection . . . 27
   - 1.4. Crossover operation . . . 27
   - 1.5. Mutation operation . . . 28
   - 1.6. Summary . . . 29

2. Analysis of a genetic algorithm 30

I. Fuzzy Systems

In normal Logic we only differentiate if a statement is true or false. But if we want to model the real world, there are some statements which cannot clearly be decided as true or false, e.g. the car is comfortable or the car is old. This can be modelled by fuzzy sets.

1. Fuzzy Sets

In regular crisp set theory an element x of a (crisp) set Ω either belongs to a (crisp) subset A or not. In Fuzzy Logic we define a (membership) value $\overline{A}(x) \in [0,1]$ for an element $x \in \Omega$, which describes the membership of x to a subset A. So in case of a crisp set A we have $\overline{A}(x) = 1$ if $x \in A$ and if $x \notin A$, then $\overline{A}(x) = 0$.

1.1. Knowledge Representation in a fuzzy set

In normal speech the age of a person is not always described with the exact age, mostly some description as *young*, *old*, *around 40* or *between 30 and 40* is used.
This should be modelled by fuzzy set theory too, so a fuzzy set should not only model numerical variables, it should also be able to model linguistic variables like *age*, *speed* or *temperature*, „whose (linguistic) values are words or sentences in a natural or artificial language“[Driankov:1993].
Therefore a fuzzy set can contain different types of elements:

- Real Numbers,
- Fuzzy Numbers (e.g. *approximately 4* or *(approximately) between 3 and 4*; see 1.4.),
- Linguistic values (e.g. *large*, *very large*, *old*, *young*, *pretty* or *nice*; see 1.1.1.).

1.1.1. Linguistic Variables and Values

As mentioned above, linguistic variables are described by linguistic values. But for the calculation with linguistic variables we need some numerical representation or information of the linguistic values.
Therefore a linguistic variable is associated with the following notion:

$$\langle X, \mathcal{L}X, \mathcal{X}, M_X \rangle,$$

where:

X is the *symbolic name* of the linguistic variable (e.g. age, speed or temperature)

$\mathcal{L}X$ is the *set of linguistic values* (also *term-set*, *reference-set*), that X can take (e.g. young, middle, old for age or (very) slow, normal, (very) fast for speed)

$\mathcal{X}$ is the *physical domain* (also *universe of discourse*) over which the (linguistic) variable X takes its quantitative (crisp) values (e.g. the interval [0 km/h, 400 km/h] for speed)

M_X is a *semantic function* that refers to every element (linguistic value) $LX \in \mathcal{L}X$ a physical, so quantitive, meaning or interpretation. This meaning/interpretation is represented by a fuzzy set $\widehat{LX}$, that is defined over $\mathcal{X}$ (e.g. a car is fast if it drives (at least) approximately 100 km/h, see 1.4. for a possible fuzzy number that models this). Ofttimes also the membership function $\mu_{LX} := M_X(LX)$ is used.

As an example, we can define a fuzzy (sub-)set $\overline{A}$ of $\{\textit{very small}, \textit{small}, \textit{medium}, \textit{high}, \textit{very high}\}$ that models e.g. the height of trees:

$$\overline{A}(\textit{small}) = 0.3, \quad \overline{A}(\textit{medium}) = 1.0, \quad \overline{A}(\textit{high}) = 0.7.$$

$\overline{A}$ is a **discrete fuzzy set** and is commonly written as

$$\overline{A} = \left\{ \frac{0.3}{\textit{small}}, \frac{1.0}{\textit{medium}}, \frac{0.7}{\textit{high}} \right\}.$$

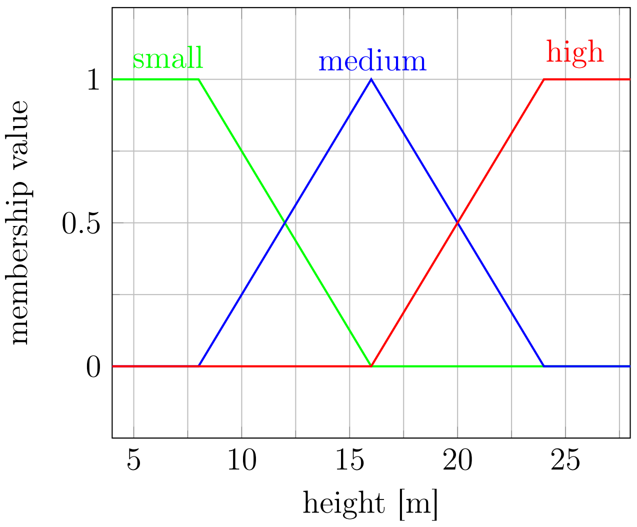
For this example we then can define the linguistic variable $X = H :=$ *height (of trees)*, term-set $\mathcal{L}H = \{\textit{very small}, \textit{small}, \textit{medium}, \textit{high}, \textit{very high}\}$, universe of discourse $\mathcal{H} =$ [0 m, 60 m] and the semantic/membership function M_H, respectively μ_{LH}, defined as following:

$$\mu_{\textit{small}}(x) = M_H(\textit{small})(x) := \begin{cases} 1, & x \leq 8 \\ \frac{16-x}{8}, & 8 < x < 16 \\ 0, & x \geq 16 \end{cases}$$
$$\widehat{=} \quad \text{„height is less than 16m.“}$$

$$\mu_{\textit{medium}}(x) = M_H(\textit{medium})(x) := \begin{cases} 0, & x \leq 8 \\ \frac{x-8}{8}, & 8 < x < 16 \\ 1, & x = 16 \\ \frac{24-x}{8}, & 16 < x < 24 \\ 0, & x \geq 24 \end{cases}$$
$$\widehat{=} \quad \text{„height is about 16m.“}$$

$$\mu_{\textit{high}}(x) = M_H(\textit{high})(x) := \begin{cases} 0, & x \leq 16 \\ \frac{x-16}{8}, & 16 < x < 24 \\ 1, & x \geq 24 \end{cases}$$
$$\widehat{=} \quad \text{„height is more than 16m.“}$$

μ_{small}, μ_{medium} and μ_{high} are socalled *fuzzy numbers* (1.4.) and can be illustrated as following:



Therefore the linguistic variable *height* is in our case associated with this notion:

$\langle H, \mathcal{L}H, \mathcal{H}, M_H \rangle = \langle height, \{very\,small, small, medium, high, very\,high\}, [0, 60], M_H \rangle.$

1.2. Arithmetic of fuzzy sets

As in the regular sets theory we want to define the intersection $\overline{A} \cap \overline{B}$, the union $\overline{A} \cup \overline{B}$ and the complement $\overline{A}^c$ of $\overline{A}$:
The *complement* is defined via the membership-function

$$\overline{A}^c(x) = 1 - \overline{A}(x).$$

The *intersection* for fuzzy sets $\overline{A}, \overline{B}$ defined on the same support $X := \operatorname{supp}(\overline{A}) = \operatorname{supp}(\overline{B})$ can be defined by $(x \in X)$

$$(\overline{A} \cap \overline{B})(x) = T(\overline{A}(x), \overline{B}(x)),$$

for a so-called *t-norm* T, that has the following specifications $(x \in X)$:

1) $T(\overline{A}(x), 1) = \overline{A}(x)$

2) $T(\overline{A}(x), \overline{B}(x)) = T(\overline{B}(x), \overline{A}(x))$ (Symmetry)

3) if $\overline{B}_1(x) \leq \overline{B}_2(x)$, then $T(\overline{A}(x), \overline{B}_1(x)) \leq T(\overline{A}(x), \overline{B}_2(x))$ (Monotony)

4) $T(\overline{A}(x), T(\overline{B}(x), \overline{C}(x))) = T(T(\overline{A}(x), \overline{B}(x)), \overline{C}(x))$ (Associative)

Note that $T(1,1) = 1, T(0,1) = T(1,0) = T(0,0) = 0$, so if it is reduced to crisp sets we obtain exactly the intersection $A \cap B$ for crisp sets.
Similar as above, the *union* for fuzzy sets $\overline{A}, \overline{B}$ defined on the same support $X := \text{supp}(\overline{A}) = \text{supp}(\overline{B})$ can be defined by $(x \in X)$

$$(\overline{A} \cup \overline{B})(x) = C(\overline{A}(x), \overline{B}(x)),$$

for a so-called *t-conorm* C, that has the following specifications: $(x \in X)$

1) $C(\overline{A}(x), 0) = \overline{A}(x)$

2) $C(\overline{A}(x), \overline{B}(x)) = C(\overline{B}(x), \overline{A}(x))$ (Symmetry)

3) if $\overline{B}_1(x) \leq \overline{B}_2(x)$, then $C(\overline{A}(x), \overline{B}_1(x)) \leq C(\overline{A}(x), \overline{B}_2(x))$ (Montony)

4) $C(\overline{A}(x), C(\overline{B}(x), \overline{C}(x))) = C(C(\overline{A}(x), \overline{B}(x)), \overline{C}(x))$ (Associative)

Note that $C(1,1) = C(0,1) = C(1,0) = 1, C(0,0) = 0$, so if it is reduced to crisp sets we obtain exactly the union $A \cup B$ for crisp sets.

Cylindrical Extension

If the support of $\overline{A}$ and $\overline{B}$ is not the same (e.g. $\overline{A}$ and $\overline{B}$ are describing different aspects like age and speed of cars), the *cylindrical extension* is a possibility to obtain the intersection/union:
The idea of the cylindrial extension is to define a fuzzy relation between the fuzzy sets $\overline{A}$ and $\overline{B}$. Therefore we define extensions $\overline{A}^*$ and $\overline{B}^*$ that are defined on $\text{supp}(\overline{A}) \times \text{supp}(\overline{B})$:

$$A^*(x_1, x_2) = A(x_1), \quad \forall x_2 \in \text{supp}(\overline{B}), x_1 \in \text{supp}(\overline{A})$$

and

$$B^*(x_1, x_2) = B(x_2), \quad \forall x_1 \in \text{supp}(\overline{A}), x_2 \in \text{supp}(\overline{B})$$

Now we can calculate the resulting $\overline{C}_{inter} = \overline{A}^* \cap \overline{B}^*$ and $\overline{C}_{union} = \overline{A}^* \cup \overline{B}^*$ on $\text{supp}(\overline{A}) \times \text{supp}(\overline{B})$ with an intersection- and union-operator.

With these definitions of a t-norm and t-conorm we can obtain (at least most of) the rules of regular sets theory for fuzzy sets (without proof):

- Idempotency,
- Commutativity (respect to $\cup, \cap$),
- Associativity (respect to $\cup, \cap$),
- Absorption,
- Distributivity,

- Identity,
- Law of Contradiction,
- Law of Excluded Middle,
- Involution,
- De Morgan laws.

Examples of t-norms are (on a set $X = X_1 \times X_2$, where A is defined on X_1, B on X_2 and $(x, y) \in X_1 \times X_2$):

- Standard intersection:
$$T_m(\overline{A}(x), \overline{B}(y)) = \min(\overline{A}(x), \overline{B}(y))$$
- Bounded sum:
$$T_b(\overline{A}(x), \overline{B}(y)) = \max(0, \overline{A}(x) + \overline{B}(y) - 1)$$
- Algebraic product:
$$T_p(\overline{A}(x), \overline{B}(y)) = \overline{A}(x) \cdot \overline{B}(y)$$
- Drastic intersection:
$$T^*(\overline{A}(x), \overline{B}(y)) = \begin{cases} \overline{A}(x), & \text{if } \overline{B}(y) = 1, \\ \overline{B}(y), & \text{if } \overline{A}(x) = 1, \\ 0, \text{otherwise} \end{cases}$$

The advantage of T_m is the fast and easy calculation, but there's no difference in the intersection if $\overline{A}_1(x) > \overline{A}_2(x) > \overline{B}(x)$, so we lose information about $\overline{A}$ (resp. $\overline{A}_1, \overline{A}_2$), since only the value $\overline{B}(x)$ will be used. Therefore T_p is sometimes better since the information from both fuzzy sets will be used. But the choice of the intersection-operator is dependent on the application.
It states (for better readability without arguments):

$$T^* \leq T_b \leq T_p \leq T_m, \text{ in fact: } T^* \leq T \leq T_m, \quad \forall\, t\text{-norms } T.$$

Examples of t-conorms are (on a set $X = X_1 \times X_2$, where A is defined on X_1, B on X_2 and $(x, y) \in X_1 \times X_2$):

- Standard union:
$$C_m(\overline{A}(x), \overline{B}(y)) = \max(\overline{A}(x), \overline{B}(y))$$
- Bounded sum:
$$C_b(\overline{A}(x), \overline{B}(y)) = \min(1, \overline{A}(x) + \overline{B}(y))$$

- Algebraic sum:

$$C_p(\overline{A}(x), \overline{B}(y)) = \overline{A}(x) + \overline{B}(y) - \overline{A}(x) \cdot \overline{B}(y)$$

- Drastic union:

$$C^*(\overline{A}(x), \overline{B}(y)) = \begin{cases} \overline{A}(x), & \text{if } \overline{B}(y) = 0, \\ \overline{B}(y), & \text{if } \overline{A}(x) = 0, \\ 1, \text{otherwise} \end{cases}$$

As with t-norms the choice of the union-operator is dependent on the application. It states (for better readability without arguments):

$$C_m \leq C_p \leq C_b \leq C^* \text{ in fact: } C_m \leq C \leq C^*, \quad \forall\, t\text{-conorms } C.$$

$$T \text{ and } C \text{ are called } \textit{dual} :\Leftrightarrow \begin{cases} T(\overline{A}(x), \overline{B}(y)) = 1 - C(1 - \overline{A}(x), 1 - \overline{B}(y)) \\ C(\overline{A}(x), \overline{B}(y)) = 1 - T(1 - \overline{A}(x), 1 - \overline{B}(y)) \end{cases}.$$

1.3. Logical Rules

In this part we want to calculate logical connections between fuzzy statements like

IF ((a car is older than 30 years) AND (a car is fast)) THEN (a car is expensive).

or

IF ((a car is older than 30 years) OR (a car is fast)) THEN (a car is expensive).

In part 1.2. we learned amongst building the complement how to intersect and unite fuzzy sets. Therefore by identifying the complement with the negation $\neg$, union with OR, intersection with AND, $\emptyset$ with 0 and the *support* X with 1 we get a connection between calculations of fuzzy sets and fuzzy logic. For a complete logical model we also need the implication, which we will derive in this chapter. To clarify we show the connection between fuzzy sets and fuzzy logic with the examplary statements above:
If we have the fuzzy set $\overline{A}_1$ of cars older than 30 years and the fuzzy set $\overline{A}_2$ of fast cars and want to calculate the set $\overline{B}$ of fast cars that are older than 30 years, we obtain this via

$$\overline{B} = \overline{A}_1 \cap \overline{A}_2.$$

Respectively, if we want to compute the (fuzzy) set $\overline{C}$ of fast cars or cars older than 30 years, we get

$$\overline{C} = \overline{A}_1 \cup \overline{A}_2.$$

The resulting set is hereby dependent on the choice of the intersection- resp. the union-operator (compare 1.2.).
For the implication we can make use of the following fact (let p, q be statements):

$$\text{IF } p \text{ THEN } q \Leftrightarrow \neg p \text{ OR } q$$

Then we can calculate the *fuzzy implication* $\overline{A} \to \overline{B}$ with support $X_1 \times X_2$ as following:

$$(\overline{A} \to \overline{B})(x, y) = C(1 - \overline{A}(x), \overline{B}(y)), \quad (x, y) \in X_1 \times X_2$$

with a t-norm C, e.g. C_m:

$$(\overline{A} \to \overline{B})(x, y) = max(1 - \overline{A}(x), \overline{B}(y)),$$

Another common implication operator is the *Mamdani implication operator*:

$$(\overline{A} \to \overline{B})(x, y) = \min(\overline{A}(x), \overline{B}(y)).$$

It is based on the assumption that the truth-value of the conclusion $\overline{B}(y)$ cannot be higher than the truth-value of the premise $\overline{A}(x)$.

1.4. Fuzzy Numbers

Fuzzy numbers are special fuzzy sets. They express e.g. approximately a (crisp) number, an interval of the real numbers or just values „(much) bigger/smaller than“ a real value (compare 1.1.1.).
From [Buckley:2002] we obtain the following definition of fuzzy numbers:
A fuzzy number $\overline{N}$ is a fuzzy subset of $\mathbb{R}$ with following restrictions:

(i) The core of $\overline{N}$ is non-empty ($\Leftrightarrow \exists x : \overline{N}(x) = 1$).

(ii) α-Cuts of $\overline{N}$ (explained in the following section) are all closed and bounded intervals.

(iii) The support $\{x | \overline{N}(x) > 0\}$ of $\overline{N}$ is bounded.

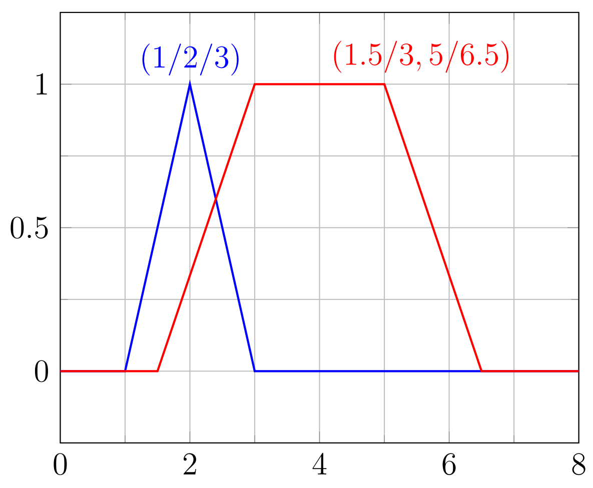
Since the core of $\overline{N}$ should be non-empty, $\overline{N}$ must be a normal fuzzy set, that means there is at least one x, such that $\overline{N}(x) = 1$.

Special Case

A special case of fuzzy numbers are *triangular* or *triangular shaped* fuzzy numbers. They are defined by three numbers $a < b < c$, where the positive values of the fuzzy number are in the interval (a, c) and the maximum value 1 is at point b.
A *triangular* fuzzy number has linear connection lines between $\overline{N}(a)$ to $\overline{N}(b)(= 1)$ and from $\overline{N}(b)$ to $\overline{N}(c)$. For such a fuzzy number we write $\overline{N} = (a/b/c)$.
A *triangular shaped* fuzzy number has to be continuous (or rather the graph of $\overline{N}$), monotonically increasing on $[a, b]$ and monotonically decreasing on $[b, c]$. We denote such a fuzzy number as $\overline{N} \approx (a/b/c)$.

Besides the triangular (shaped) fuzzy numbers, there is another big set of common fuzzy numbers: *trapezoidal (shaped)*. They are described via $\overline{T} = (a/b, c/d)$ and it states: $\overline{T}([b, c]) = \{1\}$. As the triangular (shaped) fuzzy numbers they have the same monotoneous requirements.

As an example, a triangular shaped fuzzy number $(1/2/3)$ is expressing *approximately* 2, a fuzzy interval $(1.5/3, 5/6.5)$ is expressing a number *approximately between* 3 *and* 5:



1.4.1. α-Cuts

An α-Cut $\overline{A}[\alpha]$ of a fuzzy number $\overline{A}$ with $\overline{A} > 0$ on $[a, c]$ is defined by:

$$\overline{A}[\alpha] := \{x \in \Omega | \overline{A}(x) \geq \alpha\} \subset [a, c] \quad \forall \alpha \in (0, 1]$$

$$\overline{A}[0] := [a, c].$$

$\overline{A}[0]$ is called the *support or base of* $\overline{A}$.
$\overline{A}[1]$ is called the *core of* $\overline{A}$.
So for a triangular or triangular shaped fuzzy number $\overline{N} = (a/b/c)$ we obtain $\overline{N}[1] = \{b\}$ and for a trapezoidal (shaped) fuzzy number $\overline{N} = (a/b, c/d)$ we get $\overline{N}[1] = [b, c]$.

We also know (without proof) that an α-Cut $\overline{Q}[\alpha]$ for a fuzzy number $\overline{Q}$ is a closed and bounded interval $(0 \leq \alpha \leq 1)$:

$$\overline{Q}[\alpha] = [q_1(\alpha), q_2(\alpha)],$$

where $q_1(\alpha)$ is an increasing, $q_2(\alpha)$ a decreasing function of α and $q_1(1) = q_2(1)$.

1.4.2. Fuzzy Arithmetic

There are two (equivalent; without proof) ways of adding, subtracting, multiplying and dividing two fuzzy numbers:

- Extension principle: For $\overline{C} := \overline{A} * \overline{B}, \quad * \in \{+, -, \cdot, /\}$ we define:

$$\overline{C} := \sup_{x,y}\{\min(\overline{A}(x), \overline{B}(y)) \,|\, x * y = z\}$$

- Interval Arithmetic/α-Cuts: For $\overline{C} := \overline{A} * \overline{B} \quad * \in \{+, -, \cdot, /\}$ we define:

$$\overline{C}[\alpha] := \overline{A}[\alpha] * \overline{B}[\alpha]$$

1.4.3. Ordering fuzzy numbers

Since we have a total ordering $(<, \leq, =)$ on the real numbers, we would like to have such a total ordering on the fuzzy numbers (as an extension of the real numbers) too, or at least a partial ordering (meaning: $\overline{A} \geq \overline{B}$ or $\overline{A} \leq \overline{B}$ not for all fuzzy numbers $\overline{A}, \overline{B}$). For $\delta \in \mathbb{R}$ and a fuzzy number $\overline{N} = (a/b/c)$ or $\overline{N} \approx (a/b/c)$ we write $\overline{N} \geq (>)\, \delta$ if $a \geq (>)\, \delta$ and $\overline{N} \leq (<)\, \delta$ if $c \leq (<)\, \delta$.

There are a few definitions of $(<, \leq, \approx\ /\ =)$ between fuzzy numbers, but not all fulfilling the axioms of a total ordering (partial ordering: without point 4):

1. reflexive
2. transitive
3. $\overline{M} \leq \overline{N}$ and $\overline{M} \leq \overline{N} \Rightarrow \overline{M} \approx \overline{N}$
4. $\overline{M} \leq \overline{N}$ or $\overline{M} \leq \overline{N} \quad \forall \overline{M}, \overline{N}$

In this section we want to present one possible definition for the meaning of the symbols $\leq, <$ and $\approx$ for fuzzy numbers. In [Buckley:2005] the following definition is introduced:

$$v(\overline{M} \leq \overline{N}) = \max\{\min(\overline{M}(x), \overline{N}(y)) \,|\, x \leq y\}$$

and

$$\begin{aligned} \overline{N} < \overline{M} &:\Leftrightarrow v(\overline{N} \leq \overline{M}) = 1 \text{ and } v(\overline{M} \leq \overline{N}) < \eta \in (0,1] \text{ fixed} \\ \overline{N} \approx \overline{M} &:\Leftrightarrow \text{neither } \overline{N} < \overline{M} \text{ nor } \overline{M} < \overline{N} \text{ is correct} \\ \overline{N} \leq \overline{M} &:\Leftrightarrow \overline{N} < \overline{M} \text{ or } \overline{N} \approx \overline{M}. \end{aligned}$$

Because of being the highest value two fuzzy numbers M and N have in common (in respect to $x \leq y, x \in$ support M, $y \in$ support N), η is a regulating value for M and N being equal or M smaller (bigger) than N.

$$\begin{aligned} v(\overline{N} \leq \overline{M}) = 1 &\Leftrightarrow \max\{\min(\overline{N}(x), \overline{M}(y)) \,|\, x \leq y\} = 1 \\ &\Leftrightarrow \exists x \in \operatorname{supp}(\overline{N}), y \in \operatorname{supp}(\overline{M}) \text{ with } x \leq y : \min(\overline{N}(x), \overline{M}(y)) = 1 \\ &\Leftrightarrow \exists x \in \operatorname{supp}(\overline{M}), N \in \operatorname{supp}(\overline{M}) \text{ with } x \leq y : \overline{N}(x) = 1 = \overline{M}(y) \\ &\Leftrightarrow \text{The left border of the core of } \overline{N} \text{ lies left of one element of the core of } \overline{M}. \\ v(\overline{M} \leq \overline{N}) < \eta &\Leftrightarrow \max\{\min(\overline{M}(x), \overline{N}(y)) \,|\, x \leq y\} < \eta \\ &\Leftrightarrow \exists x \in \operatorname{supp}(\overline{M}), y \in \operatorname{supp}(\overline{N}) \text{ with } x \leq y : \min(\overline{M}(x), \overline{N}(y)) < \eta \\ &\Leftrightarrow \exists x \in \operatorname{supp}(\overline{M}), y \in \operatorname{supp}(\overline{N}) \text{ with } x \leq y : \overline{M}(x), \overline{N}(y) \geq \eta \\ &\Leftrightarrow \eta \text{ is the highest number } \overline{M} \text{ and } \overline{N} \text{ have in common in respect to } x \leq y. \end{aligned}$$

With this ordering we obtain some clustering depending on $\approx$ and $\leq$ / $<$.
With such an relation/ordering on the fuzzy numbers we can calculate the min. and max. value of a set of fuzzy numbers.

1.5. Fuzzy Functions

As for functions $h\colon [a,b] \to \mathbb{R}$ in the real numbers we can also obtain a (fuzzy) function $H(\overline{X}) = \overline{Z}$ on fuzzy numbers. This can be done by the *extension principle* (1.5.1.) or *interval arithmetic/α-Cuts* (1.5.2.).

1.5.1. Extension Principle

We extend a real number function $h\colon [a,b] \to \mathbb{R}$ to $\overline{Z} = H(\overline{X})$ by the following definition:

$$\overline{Z}(z) := \sup_x \{\overline{X}(x) | h(x) = z,\, a \leq x \leq b\}.$$

For the α-Cuts $\overline{Z}[\alpha] = [z_1(\alpha), z_2(\alpha)]$ of a fuzzy number $\overline{Z}$ it lasts, if h is continuous (without proof):

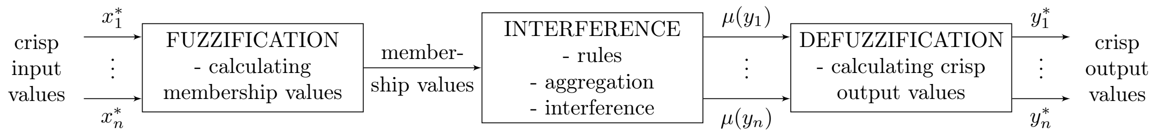
$$\begin{aligned} z_1(\alpha) &= \min\{h(x) | x \in \overline{X}[\alpha]\} \\ z_2(\alpha) &= \max\{h(x) | x \in \overline{X}[\alpha]\} \end{aligned}$$

1.5.2. Interval Arithmetic/α-Cuts

To extend a real number function h (see above) with interval arithmetic and α-Cuts, we calculate $H(\overline{X}[\alpha]) = \overline{Z}[\alpha]$. If there are other fixed fuzzy numbers used in $H(\overline{X})$, then we use their α-Cuts too for the computation.

2. Fuzzy Models

The basic principle of a fuzzy model is illustrated and described in this chapter.


crisp input values
x_1^*
x_n^*
FUZZIFICATION - calculating membership values
member-ship values
INTERFERENCE - rules - aggregation - interference
$\mu(y_1)$
$\mu(y_n)$
DEFUZZIFICATION - calculating crisp output values
y_1^*
y_n^*
crisp output values

In a fuzzy model we get n crisp input and m crisp output values. Therefore, to work with fuzzy logic/variables, we first have to calculate the membership values in particular fuzzy sets (called *Fuzzification*).
The next block, which gets the membership values as an input, is called *Interference*. The rules of the fuzzy system will be applied here following this mechanism:

- rule base: consists of logicaI rules determining causal relationships existing in the system between fuzzy sets of its inputs and outputs, e.g. one examplary rule for a 2-input, 1-ouput model:

$$\text{IF } (x_1 = A_1) \text{ AND } (x_2 = B_1) \text{ THEN } (y = C_1),$$

  where x_1, x_2 and y are the fuzzy input values and A_1, B_1 and C are the fuzzy reference values.

- interference mechanism:
  - calculating the results of the rules: Usage of the fuzzy logic (see 1.3.) and (mostly) via *Generalized Modus Ponens*, which states, that if we have the following rule:

$$\text{IF } (x = \textit{small}) \text{ THEN } (y = \textit{big})$$

    and x^* is *very* small, then it follows that y^* is *very* big.
  - *aggregation* of the results of the rules and calculating the resulting membership-function. As an illustration: If we have two rules

$$\begin{aligned} &(\text{R1}) \qquad \text{IF } (x_1 = A_1) \text{ THEN } (y = C_1) \\ &(\text{R2}) \qquad \text{IF } (x_1 = A_2) \text{ THEN } (y = C_2), \end{aligned}$$

    then we get a membership value of y for the fuzzy set C_1 and one for C_2.

The last block *Defuzzification* gets the calculated membership function values and calculates crisp, numeric output values $y_1^*, \ldots, y_m^*$, being an effect of the numerical input values $x_1^*, \ldots, x_n^*$. This operation is accomplished by a defuzzification mechanism determining the calculation method.

Example

As an illustration we consider a 2-input, 1-output model. We want to calculate $y = x_1 + x_2$, where x_1 and x_2 are input values in the universe of discourse $X = [0, 10]$ and y an output value in $Y = [0, 20]$. In our fuzzy model we use the input fuzzy numbers „small“, represented by $S_X = (0/0/10)$ and „large“, represented by $L_X = (0/10/10)$. For the output we use crisp values „small“ ($S_Y = (0/0/0)$), „medium“ ($M_Y = (10/10/10)$) and „large“ ($L_Y = (20/20/20)$).
The next step in setting up our fuzzy model is to define the rules to obtain our rule base:

(R1)	$x_1 = \mathit{small}$ AND $x_2 = \mathit{small}$	THEN $y = \mathit{small}$
(R2)	$(x_1 = \mathit{small}$ AND $x_2 = \mathit{large})$ OR $(x_1 = \mathit{large}$ AND $x_2 = \mathit{small})$	THEN $y = \mathit{medium}$
(R3)	$x_1 = \mathit{large}$ AND $x_2 = \mathit{large}$	THEN $y = \mathit{large}.$

Written with fuzzy numbers we get:

(R1)	$x_1 = S_X$ AND $x_2 = S_X$	THEN $y = S_Y$
(R2)	$(x_1 = S_X$ AND $x_2 = L_X)$ OR $(x_1 = L_X$ AND $x_2 = S_X)$	THEN $y = M_Y$
(R3)	$x_1 = L_X$ AND $x_2 = L_X$	THEN $y = L_Y.$

For example, if we have two input values $x_1^* = 2.5$ and $x_2^* = 7.5$, we calculate their membership values of the fuzzy sets S_X and L_X. These memberships values describe now, how big (respectively small) the crisp input values are (or rather to which amount they are a high (low) number).

Fuzzification

Calculating the membership values for x_1 and x_2:

$$\begin{aligned}
\mu_{S_X}(x_1) &= 0.75 \\
\mu_{L_X}(x_1) &= 0.25 \\
\mu_{S_X}(x_2) &= 0.25 \\
\mu_{L_X}(x_2) &= 0.75.
\end{aligned}$$

Interference

Calculating the results of the rules (*AND* is modelled with the „min-“, *OR* with the „max-Operator“ and the *implication* is just the identity):

(R1) $x_1 = S_X$ AND $x_2 = S_X$ THEN $y = S_Y$:

$$\mu_{(R1)}(y) = \mu_{S_Y}(y) = \mathrm{id}\left(\min(\mu_{S_X}(x_1), \mu_{S_X}(x_2))\right) = \min(0.75, 0.25) = 0.25$$

(R2) $(x_1 = S_X$ AND $x_2 = L_X)$ OR $(x_1 = L_X$ AND $x_2 = S_X)$ THEN $y = M_Y$:

$$\begin{aligned}
\mu_{(R2)}(y) = \mu_{M_Y}(y) &= \mathrm{id}\left(\max\left(\min(\mu_{S_X}(x_1), \mu_{L_X}(x_2)), \min(\mu_{L_X}(x_1), \mu_{S_X}(x_2))\right)\right) \\
&= \max\left(\min(0.75, 0.75), \min(0.25, 0.25)\right) = 0.75
\end{aligned}$$

(R3) $x_1 = L_X$ AND $x_2 = L_X$ THEN $y = L_Y$:

$$\mu_{(R3)}(y) = \mu_{L_Y}(y) = \mathrm{id}\left(\min(\mu_{L_X}(x_1), \mu_{L_X}(x_2))\right) = \min(0.25, 0.75) = 0.25$$

Aggregation

In this example we chose the „max"-mechanism for the aggregation of the resulting membership-values for y:

$$\mu(y) = \max\big(\mu_{(R1)}(y), \mu_{(R2)}(y), \mu_{(R3)}(y)\big) = \max(0.25, 0.75, 0.25) = 0.75 = \mu_{M_Y}(y)$$

Defuzzification

Since we have only crisp values as possible output values for y, there is no defuzzification mechanism needed and we get (since the aggregation mechanism tells us that we should look up the value in the fuzzy number *medium*):

$$y^* = 10.$$

Other examples have been done with the Matlab Toolbox *Fuzzy Logic Design*.

II. Artificial Neural Networks

Artificial neural networks consist of artificial neurons, compared to our brain. The advantage of these networks is their massively parallel structure. Instead of programming, a learning process is applied to an artificial network. Therefore the network has the „feature of generalization“[Waszczyszyn:1999, p.7].
First we have to obtain a model of an artificial neuron before turning to an artificial neuronal network.

1. Model of an artificial neuron

An artificial neuron is a model of a biological neuron which contains synapses, dendrices, a cell body and an axon. At the synapses the neuron gets chemical signals that are converted in electrical signals and sent over the dendrices to the cell body. In this part of the neuron, all signals will be accumulated and if they exceed a limit an electrical signal is being processed down the axon to the cells connected to the axon.
Therefore an artificial neuron contains of these main parts:

- N inputs (from the synapses/dendrices),
- transformator (cell body) with summing junction Σ and activation (threshold) unit F,
- 1 output (axon).

It is described by the following variables and parameters:

$x = \{x_1, \dots, x_N\}$	input vector
$w = \{w_1, \dots, w_N\}$	vector of weights (for the summing junction)
$b = -\theta = w_0$	constant component (bias)
θ	threshold
$v = \sum_{j=0}^{N} w_j x_j$	Artificial Neuron potential, $x_0 = 1$
$F(v)$	activation function

There exist different activation functions F, dependent on the application. We will be working with the *binary step function*

$$F(v) = F(u - \theta) = \begin{cases} 1, & \text{for} \quad u \geq \theta \\ 0, & \text{for} \quad u < \theta \end{cases},$$

where $u = \sum_{j=1}^{N} w_j x_j$.

2. Types of neural networks

After modelling the basic part of the network, one neuron, the more complex part, the connections between the neurons, called *neural network (NN)* is being studied in this part.
„The arrangement of nodes and pattern of connection links between them is called the *architecture/type/structure* of a neural network“[Waszczyszyn:1999, p.6]. There exist three main architectures:

- Feedforward NN: the nodes are arranged in layers, where the output from the lower layer is the input for the next layer. It can be written as

  $$N - H1 - H2 - \ldots - M,$$

  where N is the number of inputs, $H1, H2, \ldots$ are the numbers of neurons in the so called hidden layers and M is the number of outputs.

- Recurrence NN: the input is processed by a neuron more than once. Therefore we obtain a feedback here.

- Cellular NN: The nodes and connections are described by a specified topology, e.g. a full mesh.

We will concentrate on feedforward neural networks.

2.1. Basic feedforward neural networks

Let us consider a two-input,one-output feedforward neural network:

$$\begin{tikzcd} x_1 \arrow[r] & \boxed{\Sigma} \arrow[r] & \boxed{F} \arrow[r] & \text{y} \\ x_2 \arrow[ur] & & & \end{tikzcd}$$

Furthermore we calculate y via the binary step function:

$$y = F(v) = F(u - \theta) = \begin{cases} 1, & \text{for} \quad u \geq \theta \\ 0, & \text{for} \quad u < \theta \end{cases},$$

where $u = \sum_{j=1}^{N} w_j x_j$, so we use an abbreviated notation:

$$\begin{tikzcd} x_1 \arrow[r, "w_1"] & \boxed{\theta} \arrow[r] & \text{y} \\ x_2 \arrow[ur, "w_2"'] & & \end{tikzcd}$$

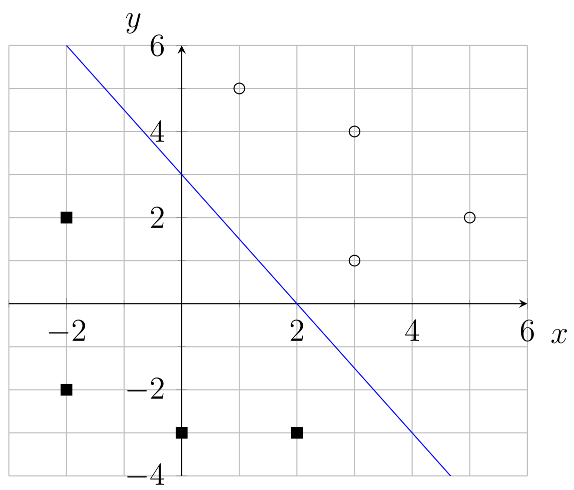
In summary we get:

$$y = \begin{cases} 1, & \text{for} \quad w_1x_1 + w_2x_2 - \theta \geq 0 \\ 0, & \text{for} \quad w_1x_1 + w_2x_2 - \theta < 0 \end{cases},$$

As an example let us consider the *separation problem*, where we want to separate points of different shape.
Assuming the points are linearly separable, we can devide the given points with the line $l = w_1x_1 + w_2x_2 - \theta = 0$, for some parameters w_1, w_2 and θ.

Now we want to present two small examples how to obtain parameters (weights, biases and number of needed layers) for a feedforward neural network for the separation problem.
Therefore let us consider the following problem:
Find a neural network that can divide the following given groups (squares and circles) of points.

Points: $\{(-2,-2), (-2,2), (0,-3), (1,5), (2,-3), (3,1), (3,4), (5,2)\}$



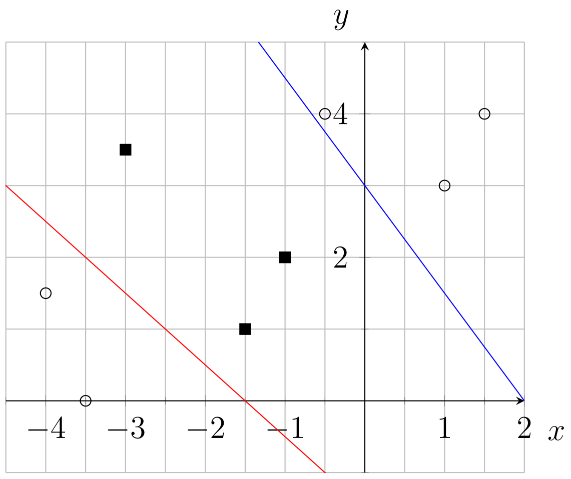
Our goal is to find a line $0 = l = w_1x_1 + w_2x_2 - \theta$, which is equivalent to $x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_1}$, that separates the circles and the squares. From the graph we see that the blue line fulfills this.
It's equation is given by:

$$x_2 = -\frac{3}{2}x_1 + 3 \quad \widehat{=} \quad x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_1}$$

Therefore we can choose $w_1 = 3$, $w_2 = 2$ and obtain $\theta = 6$. So the network

$$\begin{tikzcd} x_1 \arrow[r, "w_1 = 3"] & \boxed{\theta = 6} \arrow[r] & \text{y} \\ x_2 \arrow[ur, "w_2 = 2"'] & & \end{tikzcd}$$

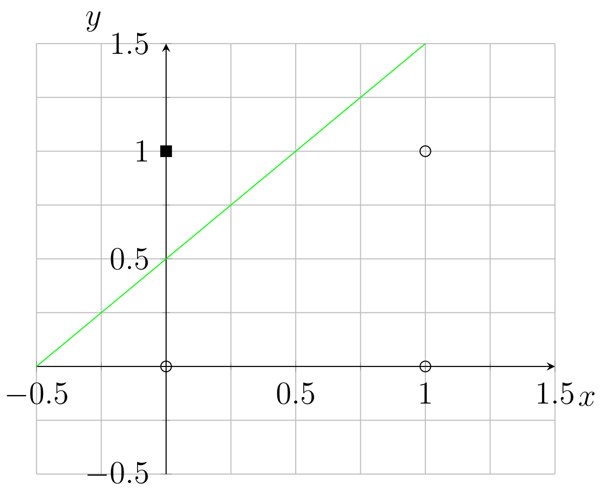
is able to separate our given points and can decide if another given point is in the set of squares or belongs to the set of circles.
If we consider another set of points, displayed in the following plot:



we see that we cannot separate them with just one line. Therefore we need at least two layers and three neurons, where the first layer neurons model the red (let's call it l_1) and the blue line (l_2). If we denote the results of the respective neurons with y_1 and y_2 we obtain the following table where $y = 1$ means: „The given point belongs to the set of squares".
For $i = 1, 2$ we know: $y_i = 1$ is equivalent to $l_i \geq 0$, so the point lies right of or on the line, on case of $y_i = 0$ we have the point lying left of the line.

y_1	y_2	y
0	0	0
0	1	1
1	0	0
1	1	0

This table can be represented (and solved) by the following system:



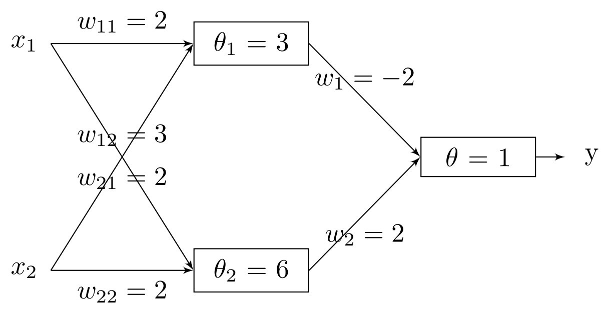
Our next step is to calculate again the parameters for a feedforward artificial network that can divide two groups of points where the groups are given by the exemplary points above.
The equation for the lines are:

$$\begin{aligned}
l_1\,(red):&\qquad x_2 = -x_1 - \frac{3}{2}\\
&\qquad\;\widehat{=} -\frac{w_{11}}{w_{21}}x_1 + \frac{\theta_1}{w_{21}}\\
l_2\,(blue):&\qquad x_2 = -\frac{3}{2}x_1 + 3\\
&\qquad\;\widehat{=} -\frac{w_{12}}{w_{22}}x_1 + \frac{\theta_2}{w_{22}}\\
l\,(green):&\qquad x_2 = x_1 + \frac{1}{2}\\
&\qquad\;\widehat{=} -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}.
\end{aligned}$$

We obtain for example:

$$\begin{aligned}
w_{11} = 2, w_{21} = 2, \theta_1 = 3\\
w_{12} = 3, w_{22} = 2, \theta_2 = 6\\
w_1 = -2, w_2 = 2, \theta = 1
\end{aligned}$$

Therefore our feedforward artificial network which separates these two groups of points is given by:



Now we presented two examples how to obtain the parameters for different separation problems.
In the next part we present the basic principle of obtaining the parameters with an algorithm via supervised learning.

2.2. Supervised learning

In artificial intelligence we speak from supervised learning if the network gets some positive and negative examples and calculates the parameters of the network via updating by itself. Afterwards the quality of the network is being tested by some exemplary test-sets.
For example in the part above we got some point sets where we knew that the circles were „negative“ examples and the squares „positive“ examples.

Update

In the case of an n-input, one-output neuron we can adapt the following learning rule: Beginning from default (e.g. 0) or other specified starting values we adjust the respective weights w_j, $j = 1, \dots, n$ and θ with an update Δw_j, $j = 1, \dots, n$ and $\Delta\theta$.
Let $j \in \{1, \dots, n\}$ and $s \in \mathbb{N}$ be the number of the sample we are processing. Our starting values are subscribed with $w_j(0)$ and $\theta(0)$.
The learning rule is calcutated as following:

$$\begin{aligned} w_j(s+1) &= w_j(s) + \Delta w_j(s), & \theta(s+1) &= \theta(s) + \Delta\theta(s), \\ \Delta w_j(s) &= x_j \delta(s), & \Delta\theta(s) &= \delta(s), \\ \delta(s) &= t - y(s), \end{aligned}$$

where t is the output of the given sample and $y(s)$ the computed output for the sample. We know that $\delta(s) \in \{-1, 0, 1\}$ because $t, y(s) \in \{0, 1\}$. Therefore we obtain for the updates

$$\Delta w_j(s) = \begin{cases} x_j, & \delta(s) = 1 \\ 0, & \delta(s) = 0 \\ -x_j, & \delta(s) = -1 \end{cases} \quad \text{and} \quad \Delta\theta(s) = \begin{cases} 1, & \delta(s) = 1 \\ 0, & \delta(s) = 0 \\ -1, & \delta(s) = -1 \end{cases}$$

Another approach for not linearly separable sets is the gradient of the steepest descend. Herefore we mostly have a smooth activation function F and calculate the least mean squared error, which we use with a specified learning rate in the update rule.
We didn't treat this approach in this lecture.

III. Genetic Algorithms

Genetic Algorithms have the aim of finding the optimal (global) solution in a search space (given by parameter restrictions).
To achieve this, they operate on genetic structures called *chromosomes*. One chromosome represents a point in the search space and is associated with a so called „fitness“ value, which measures the performance of the point for the given problem.
By using genetic operations like crossover and mutation, a genetic algorithm builds new generations and thus improves the performance of the whole population (set of chromosomes/points) instead of only the performance of an individual point. Analogous to the Darwinian model of „survival of the fittest“, the individuals with a higher fitness value are more likely to reproduce than the ones with a lower fitness value. In summary, a genetic algorithm represents a model of evolution by random mutation and natural selection.
By improving the whole set of chromosomes this methods are more likely to find the global optimum.

1. Implementation

As described above, there are a few main features of a genetic algorithm that should be implemented properly:

- encoding and decoding schemes: for the representation of the parameters in the chromosomes,
- fitness evaluations: fitness function that applies a fitness value to every point/chromosom in the set/population,
- selection mechanism: selects the chromosomes which will build the parental set of chromosomes for the next population,
- crossover operators: simulate a random crossover mechanism for the chromosomes,
- mutation operators: simulate a random mutation in the chromosomes.

1.1. Encoding and Decoding Schemes

The aim of the encoding technique is to translate the original problem into a format that is fitting to genetic computations by transforming the parameters of an optimization problem into finite-length string representations. Therefore by developing an en- and decoding schema, you should also consider the later executed genetic operations

(crossover and mutation).
In this part we only present the *binary encoding*, since it has been shown optimal.

Concatenated, multiparameter, mapped, fixed-point en- and decoding

Let us assume we have n parameter.

Encoding:
For every parameter x_i, $i \in \{1, \dots, n\}$, we know the range $[x_{i,min}, x_{i,max}] \subset \mathbb{R}$ and precision $prec_i \in \mathbb{N}$ (the number of digits after the decimal point). Now we can calculate the needed length of the string representation by the following formula:

$$l_i = \min\{l : (x_{i,max} - x_{i,min}) \cdot 10^{prec_i} \leq 2^l - 1\} \tag{1.1}$$

To obtain the binary string representation $\mathbf{b}$ of a value $x_i^0 \in [x_{i,min}, x_{i,max}]$ we first calculate the decimal value $decimal(\mathbf{b})$ of the bitstring with the given precision $prec_i$/length l_i:

$$decimal(\mathbf{b}) = (x_i^0 - x_{i,min}) \frac{2^{l_i} - 1}{x_{i,max} - x_{i,min}}$$

By transforming this decimal value $decimal(\mathbf{b})$ in binary representation we obtain our bitstring representation $\mathbf{b} = enc(x_i^0)$:

$$\begin{aligned} \mathbf{b} = enc(x_i^0) &= (\, \text{round}\,(decimal(\mathbf{b}))\,)_2 \\ &= \left(\text{round} \left((x_i^0 - x_{i,min}) \frac{2^{l_i} - 1}{x_{i,max} - x_{i,min}} \right) \right)_2 \end{aligned}$$

For example we have a parameter $x_1 \in [-5, 7]$ with precision $prec_1 = 2$ (means 2 digits after the decimal point). Now we want to encode the value $x_1^0 = -4.38$ as a bit string. We calculate the length and the decimal value of the bit string:

$$l_1 = \min\{l : (7 + 5) \cdot 10^2 \leq 2^l - 1\} = \min\{l : 12 \cdot 100 \leq 2^l - 1\} = 11$$

$$decimal(enc(x_1^0)) = (-4.38 + 5) \cdot \frac{2^{11} - 1}{7 + 5} = 0.62 \cdot \frac{2047}{12} \approx 106$$

Therefore we obtain the following result:

$$enc(x_1^0) = 106_2 = 000\,0110\,1010$$

Decoding:
Now we want to calculate the parameter value of a respective bitstring. For the parameter x_i we know the underlying interval $[x_{i,min}, x_{i,max}]$ and the length l_i of the bitstring representations. We assume to have an encoded value $enc(x_i) \in [0, 2^{l_i} - 1]$. This encoded value is, as above, linearly mapped in the given interval $[x_{i,min}, x_{i,max}]$.

Therefore for a binary string $\mathbf{b} = \langle b_{l_i-1}b_{l_i-2}\dots b_1b_0\rangle_2$, which represents a real value x_i^0 of the parameter x_i, we obtain the following decoding formula (compare (1.1)):

$$x_i^0 = dec(\mathbf{b}) = x_{i,min} + decimal(\mathbf{b}) \cdot \frac{x_{i,max} - x_{i,min}}{2^{l_i} - 1},$$

where $decimal(\mathbf{b})$ is the decimal value of the binary string $\mathbf{b}$.

We can also derive the precision $prec_i$ of the parameter x_i:

$$prec_i = \left[\log_{10} \left(\frac{2^{l_i} - 1}{x_{i,max} - x_{i,min}} \right) \right],$$

where $[\cdot]$ are the gauß-brackets, which means we are rounding to the next lowest integer.

A multiparameter encoding scheme is now obtained by simply concatenating the single-parameter codes. Each subcode has its own length l_i and range $[x_{i,min}, x_{i,max}]$.

1.2. Fitness Evaluation

After encoding the parameters of the optimization problem the next step is the calculation of the fitness values for each chromosome.
Herefore we introduce a fitness function $f\!f$, that operates on binary strings and is equivalent to a function f that operates on the real value x, represented by the string $\mathbf{b}$:

$$f\!f(\mathbf{b}) = f(x).$$

The fitness function is a model for the role of the influence of the environment in the natural evolution. The produced output, the „measure of fitness“, is used when the parents for the next generation are selected.
Since mostly positive fitness values are needed, there are often one or more mappings from the underlying cost function of the optimization problem to a fitness function necessary.

Examples

In a *minimization problem* the task is to minimize the cost function $Q(x)$. Then we can use the following cost-to-fitness-transformation:

$$f\!f(\mathbf{b}) = f(x) = \begin{cases} C_{max} - Q(x), & \text{for } Q(x) < C_{max}, \\ 0, & \text{otherwise} \end{cases},$$

where C_{max} is a selected constant, for example the largest value of Q in the current population or from the last few generations.

For a *maximization problem* the used fitness function is ofttimes the original cost function $Q(x)$.

Another possibility is to sort the chromosomes and take the number in the ranking as their fitness value. In this case, the fitness functions don't have to be accurate as long as they provide the correct ranking of the chromosomes.

1.3. Selection mechanism

In natural selection, the „survival of the fittest“ is the mechanism with which the new population will be created. The analogous part in the generic algorithms is the selection mechanism which selects (on base of the fitness value) which chromosomes will participate in the creation of the new population, so they will be „parent chromosomes“.
The selection probability of a chromosome to be chosen is (normally) proportional to their fitness value.

1.3.1. Roulette-wheel & Ranking selection

The fitness values are assumed to be positive, if not, we have to apply a convenient mapping.
Let *pop_size* describe the number of chromosomes in the population (population size) and i be an index to identify every chromosome. First we calculate the probabilities of selection:

- Calculation of the fitness values $ff(\mathbf{b}_i)$ for all chromosomes.
- Computation of the total fitness of the population:

$$F = \sum_{i=1}^{pop\_size} ff(\mathbf{b}_i)$$

- Calculation of the selection probability p_i for all chromosomes:

$$p_i = \frac{ff(\mathbf{b}_i)}{F}$$

- Computation of the cumulative probability q_i for all chromosomes:

$$q_i = \sum_{j=1}^{i} p_j$$

and afterwards we let our roulette wheel spin. It simulates a roulette wheel where every chromosome has a box with a box size proportional to the fitness value. Now we spin our wheel *pop_size* times and choose one parent chromosome each time:

- Generation of a random (float) number $r \in [0, 1]$.
- Set $q_0 := 0$ and select the k-th chromosome, which fulfills:

$$q_{k-1} < r \leq q_k, \qquad \text{where } k \in \{1, \dots, pop\_size\}$$

Clearly some of the chromosomes can be selected more than one time, so the chromosomes with higher fitness get more copies (probably), average fitness chromosomes stay even and the ones with lesser fitness die off after a few generations.
One disadvantage of this approach is the (small) probability that the best chromosome may not be selected, what would cause a stochastic error. This is for example corrected in the *elitist strategy*, where the best chromosome of each generation is copied to the succeeding generation before performing the roulette wheel strategie. But this approach improves the local search at the expense of a global search since it increases the speed of domination of the population by the best chromosome. On the other side, it appears to improve the performance of genetic algorithms.
If the best chromosome has a much higher fitness than all the others, the parental chromosomes are mostly a copy of the best one. To handle this problem the chromosomes can be sorted, for example on the basis of their fitness values and instead of their fitness values their number in the ranking is used for the random wheel probability calculation. This approach is called *ranking selection.*

1.3.2. Tournament selection

In the tournament selection mechanism several „tournaments“ of selected size *size_t*, for example *pop_size*, are being simulated. The chromosomes which are taking part in a tournament are selected randomly.
There are different possibilities of the tournament selection mechanism:

- „One-on-one“: We have one chromosome against another. With a specified probability p_t the chromosome with higher fitness „wins“ and is being selected. In the other case the chromosome with lower fitness is selected as winner. Normally p_t is bigger than 0.5 to favour the chromosomes with higher fitness.
  A tournament mostly has more than one level, so the winner of all tournament rounds is chosen as one parent. Therefore the bigger the tournament, the higher the probability that the weaker chromosomes are not getting selected. Normally there are as many tournaments simulated as there will be parental chromosomes (since there will be only one winning chromosome per tournament).

- „Everyone fights everyone“: In this variant in the set of the $size_t$ randomly chosen chromosomes the one with the best fitness is being chosen as a parent. Again there will be as many tournaments as there will be parents.

1.4. Crossover operation

In this part the chromosomes of the new population will be created.
Assume, two parents always have two children. If other, we just have to adapt the size of the parental set of chromosomes and the crossover mechanism.
In nature, the new chromosomes inherit genes from both parents, what is achieved by crossover, meaning to exchange parts of the chromosomes between the parents. This can be a „one-point crossover“, where all data following the „crossover point“ is exchanged,

or „n-point crossover“, where the parts between the crossover points are exchanged. But empirical studies have shown that multiple-point crossover may degrade the perfomance of genetic algorithms.

The occurring of a crossover is defined by a *crossover probability* p_c, which determines the expected number $p_c \cdot pop\_size$ of chromosomes which will undergo the crossover operation.
In applications, p_c is usually between 0.6 and 1.0.

Crossover procedure

For every chromosome in the new population the following steps will be executed:

- Generation of a random (float) number $r \in [0, 1]$.
- If $r < p_c$, then the chromosome will be selected for crossover.

After the selection of all „crossover-chromosomes“, they will be mated randomly and for each pair the crossover operation is performed.
All chromosomes have the same length $l = \sum_{j=1}^{param\_size} l_j$, where l_j is the length of the bit representation of the j-th parameter and $param\_size$ is the number of parameters.

In case of the *one-point crossover*, for each pair a random integer number $pos \in \{1, \ldots, l-1\}$ will be generated and the parts $\langle b_{pos+1} \ldots b_l \rangle$ and $\langle c_{pos+1} \ldots c_l \rangle$ of two parental chromosomes $\langle b_1 \ldots b_{pos} b_{pos+1} \ldots b_l \rangle$ and $\langle c_1 \ldots c_{pos} c_{pos+1} \ldots c_l \rangle$ will get exchanged. Now the new chromosomes

$$\begin{aligned} &\langle b_1 \ldots b_{pos} c_{pos+1} \ldots c_l \rangle \quad \text{and} \\ &\langle c_1 \ldots c_{pos} b_{pos+1} \ldots b_l \rangle \end{aligned}$$

will be stored in the population instead of the parental ones.

1.5. Mutation operation

With the crossover operation only an improvement of the current gene potential is possible. But sometimes the population does not (yet) contain all the information to solve the problem optimaly, so crossover alone can't produce a satisfactory solution. Therefore as a source of spontaneously generated new properties (defined for example by bits), mutation is needed and applied with a (low) probability p_m. This probability defines the totally expected number $p_m \cdot l \cdot pop\_size$ (where l is the number of bits in a chromosome) of mutated bits and is usually chosen between 0.001 and 0.01.
In binary representation (see 1.1.), a mutation is a change from 0 to 1 or vice versa.

Mutation procedure

After the crossover operations, the following procedure is executed for every bit in every chromosome.

- Generation of a random (float) number $r \in [0, 1]$.
- If $r < p_m$, then the bit will be mutated.

If the mutation rate p_m is too high (> 0.1), the genetic algorithm will become similar to a random search technique, therefore p_m is normally kept rather low.

1.6. Summary

After presenting the main features of a generic algorithm, we can present the basic steps of a genetic algorithm:

1. Initialization of the population of *pop_size* chromosomes with binary values, which are randomly generated.
2. Calculation of the fitness value for every chromosome.
3. Creation of a new population:
   a) Selection of a parental population of *pop_size* chromosomes (the new generation; it will get changed in Step b) and c))
   b) Crossover operations in the parental population with probability p_c.
   c) Mutation operations in the crossovered population with probability p_m.
4. Calculation of the fitness value for every chromosome.
5. If STOPPING_CRITERIA is fulfilled, then STOP and return the best chromosome (point), otherwise go to step 3.

In the initialization part we should also choose an appropriate population size: the more chromosomes are used, the more memory is needed but if on the other hand too less chromosomes are used, the search space is not filled out properly and it's less likely to find a satisfactory solution.
The algorithm will stop if we found a solution that is probably optimal, so for example if there is no significant change between the fitness value of the best chromosome of the last and the newest population. Another approach could be to define a fixed amount of population creations. The STOPPING_CRITERIA can also be a mix or some similar criteria of both proposed possiblities.

By changing the parameters, a genetic algorithm can be tuned to nearly every grade between *exploration* (looking for new paths that are unknown so far) and *exploitation* (using the information about good solutions so far and search in their neighbourhood). As random search relies completely on exploration and „hill climbing“ fully on exploitation, a genetic algorithm can be tuned to vary between random search and hill climbing. This can be achieved by modifying these parameters:

- mutation probability: mutation encourages exploration, therefore higher values means more exploration, lower values decreases exploration.
- elitist strategy: promotes exploitation, because it supports the domination of the (locally) best chromosome(s).

In summary we can find the following properties of genetic algorithms:

Advantages:

- The only requirement for the fitness function is the ability to estimate a value for every chromosome (therefore nearly any kind of fitness function (continous, step function,...) can be used (and optimized!)).
- In a chromosome any information can be encoded, therefore it is also possible to generate for example an *evolving neural network* by encoding the structure and parameters of an artificial neural network in a single chromosome).

Disadvantages:

- Usually genetic algorithms take a long time to converge.
- There's no guarantee that the obtained solution is optimal or even close to an optimal one.

2. Analysis of a genetic algorithm

For the analysis of the behaviour of a genetic algorithm, e.g. to find out, for which „kind“ of problems or rather solutions the genetic algorithm is fitting, we use the concept of a „schema“.
A *schema* is a similarity template for some chromosomes, or better, it is a representation of a set of similar chromosomes. For example the schema $(*11*0)$ contains the chromosomes $(01100), (01110), (11100)$ and (11110). So in this case, the chromosomes with $1's$ at the 2nd and 3rd position and a 0 in the 5th position are similar and described by the schema $S_1 = (*11*0)$. Usually $*$ is used as a place holder for 1 or 0.
There are two characteristics of a schema S:

- the order $o(S)$, what is simply the amount of fixed digits; in our example: $o(S_1) = 3$,
- the length $\delta(S)$, which describes the distance between the first and the last fixed digit position; again for our example: $\delta(S_1) = 5 - 2 = 3$.

As the genetic algorithm processes now all the individual chromosomes, there is a parallel processing of the chromosomes described by the schemata, what is called *implicit parallelism.*
One special kind of schemata are *building blocks.* These are schemata with

- fitness higher than average,
- short length,
- low order, so less fixed positions.

Because of having higher fitness values than average the solution will probably be a chrosomome from a building block schema.
They have automatically increasing numbers in subsequent generations, since the order and the length of these schematas are preserved. This makes it easier to generalize, for which applications the given genetic algorithm can be used for.
Therefore, to find an appropriate genetic algorithm for a given problem we should consider an encoding method such that short, low-order schemata exist that are relevant to the underlying problem and can be distinguished from unrelated ones. This is called *selection of meaningful building blocks*.
Another fundamental guideline in choosing the encoding method is *selection of minimal alphabets*. To concentrate on the important data of the problem and improve the finding of an optimal solution, it states that we should use the smallest (possible) alphabet that permits a natural expression of the underlying problem.

References

[Buckley:2005]	James J. Buckley: *Simulating Fuzzy Systems.* Springer-Verlag, 2005.
[Buckley:2002]	James J. Buckley, Esfandiar Eslami: *An Introduction to Fuzzy Logic and Fuzzy Sets.* Springer-Verlag, 2002.
[Driankov:1993]	Dimiter Driankov, Hans Hellendoom, Michael Reinfrank: *An Introduction to Fuzzy Control.* Springer-Verlag, 1993.
[Piegat:2001]	Andrzej Piegat: *Fuzzy Modeling and Control.* Springer-Verlag, 2001.
[Waszczyszyn:1999]	Z. Waszczyszyn: *Neural Networks in the analysis and design of structures.* Springer-Verlag, 1999.
[Gorzal:2002]	Marian B. Gorzałczany: *Computational Intelligence Systems and Application.* Springer-Verlag, 2002.
[Waszczyszyn:1999]	Z. Waszczyszyn: *Neural Networks in the analysis and design of structures.* Springer-Verlag, 1999.