Lecture 1

Introduction to Computational Intelligent (CI)

Computational Intelligent (CI)

Computational intelligence (CI) comprises practical adaptation and self-organization concepts, algorithms and implementations that enable or facilitate appropriate actions (intelligent behavior) in complex and changing environments.

In other words, CI aims to study adaptive techniques that facilitate intelligent behavior of the system in changing or complex environments. These techniques include many features that exhibit an ability to learn, generalization, discovering, or adapt to new situations.

Computational Intelligent (CI)

- CI is defined as:

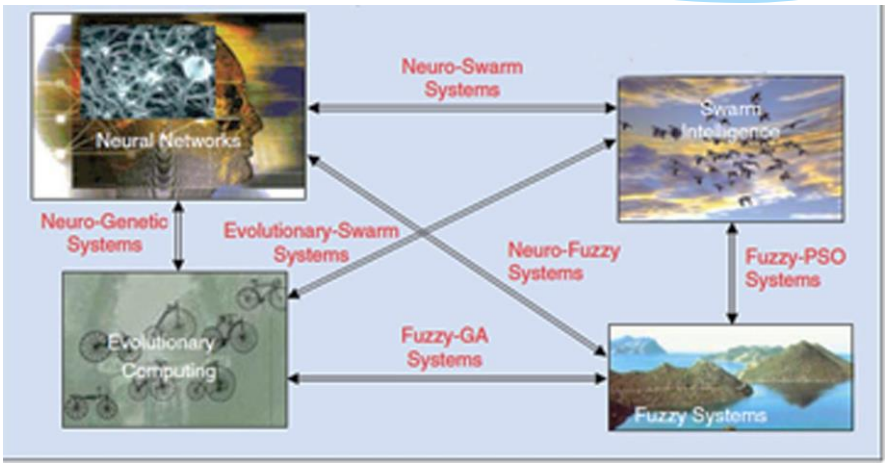
"*The computational models and tools of intelligence capable of inputting raw numerical sensory data directly, processing them by exploiting the representational parallelism and pipelining the problem, generating reliable and timely responses*".

Computational Intelligent (CI)

The paradigms of CI are designed to model the behavior of biological intelligences. CI encompasses several of intelligent models and algorithms, such as **Fuzzy Systems (FSs)**, **Genetic Algorithm (GAs), Neural Networks (NNs) and Swarm Intelligent.**

Also, It is possible to merge among them to generate new systems called **Hybrid Systems**.

Computational Intelligent (CI)



Hybrid Systems: (NNs, GAs, FSs, and Swarm Intelligent)

Lecture 2

Fuzzy Logic (FL)

Fuzzy Logic (FL)

❑ Introduction

Intelligent systems are usually described by analogies with biological systems by, for example, looking at how human beings perform **control tasks**, **recognize patterns, or make decisions**.

There exists a mismatch between humans and machines: **human's reason** in uncertain, imprecise, fuzzy ways **while machines and the computers** that run them are based on binary reasoning.

Fuzzy Logic (FL)

❑ Introduction

Fuzzy System (FS) is a way to make machines more intelligent enabling them to reason in a fuzzy manner like humans.

Fuzzy Logic (FL) proposed by Lotfy Zadeh in 1965, emerged as a tool to deal with uncertain, imprecise, or qualitative decision-making problems.

FL theory was initially only an extension to the standard mathematical theory of set.

Fuzzy Logic (FL)

❑ Introduction

The idea was to introduce a degree of membership to a set instead of the usual notion of member or not member.

Zadeh's invention of fuzzy logic theory did not, in the start, initials a huge theoretical research from mathematicians, but was found very useful by engineers who were designing controllers.

Fuzzy Logic (FL)

❑ Introduction

One of the first who created such a control system in 1974 was the engineer **Mamdani** who was constructing a controller for a system engine.

Afterwards especially Japanese engineers constructed a lot of different fuzzy logic controller and fuzzy logic products with a tremendous success.

Fuzzy Logic (FL)

❑ Introduction

A fuzzy control system is a knowledge-based system, implementing expertise of a human operator or process engineer that can be easily expressed through a set of linguistic fuzzy rules (IF-THEN rules).

FL evolved as a key technology for developing the knowledge-based systems in the control engineering to incorporate practical knowledge for designing controllers.

Fuzzy Logic (FL)

❑ Introduction

Gradually, FL progressed as a powerful technique for other fields as well and its applications have rapidly expanded in adaptive control systems and system identification.

FL has the advantages of easy implementation, robustness, and ability to approximate to any nonlinear mapping.

Fuzzy Logic (FL)

❑ Fuzzy Set Theory

- Classical Set Theory

✓ Sets are defined by a simple statement describing whether an element having a certain property belongs to a particular set.

✓ When a set **A** is contained in an universal space **X**, then we can state explicitly whether each element **x** of space **X** "is ***or*** is not" an element of **A**.

Fuzzy Logic (FL)

❑ Fuzzy Set Theory

- Classical Set Theory

✓ Set **A** is well described by a function called **characteristic function A**. This function, defined on the universal space **X**, assumes:

value **1** for those elements **x** belong to set **A**,

value **0** for those elements **x** do not belong to set **A**.

The notations used to express these mathematically are:

A: x→ [0,1]

$$A(x) = \begin{cases} 1 & x \text{ is a member of A} \\ 0 & x \text{ is not a member of A} \end{cases} \quad \ldots(1)$$

Fuzzy Logic (FL)

❑ Fuzzy Set Theory

- Classical Set Theory

✓ The set **A** can be represented for all elements $x \in X$ by its characteristic function $\mu_A(x)$ defined as:

$$\mu_A(x) = \begin{cases} 1 & \text{if} \quad x \in X \\ 0 & \text{otherwise} \end{cases} \qquad \ldots(2)$$

✓ Thus, in classical set theory $\mu_A(x)$ has only two values **0 ('false')** and **1 ('true').** Such sets are called **Crisp Sets**.

Fuzzy Logic (FL)

❑ Fuzzy Set Theory

- Fuzzy Set Theory

✓ As said before, in classical set theory, the membership of elements in a set is evaluated in binary terms according to a bivalent condition — an element either belongs or does not belong to the set, as given by Eq.2.

✓ By contrast, fuzzy set theory permits the gradual evaluating of the membership of elements in a set; this is described with the aid of a membership valued in the real unit interval [0, 1].

Fuzzy Logic (FL)

❑ Fuzzy Set Theory

- Fuzzy Set Theory

✓ Fuzzy sets allow an object to be a partial member of a set. In Fig. 1, if **X** suggests a collection of objects denoted by *x*, usually **X** is referred to as the "*universe of discourse*", and then a fuzzy set **A** in **X** is defined by a set of ordered pairs:

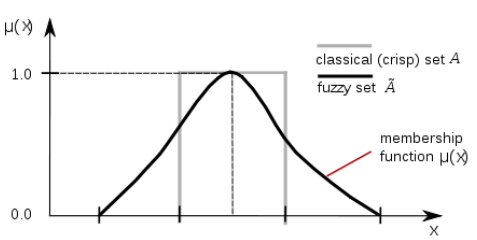
$$A = \{(x, \mu_A(x) / x \in X\} \qquad \ldots(3)$$

where the function $\mu_A(x)$ is called **membership function** of the object ***x*** in ***A***.

Fuzzy Logic (FL)

❑ Fuzzy Set Theory

- Fuzzy Set Theory



Membership function of a fuzzy set

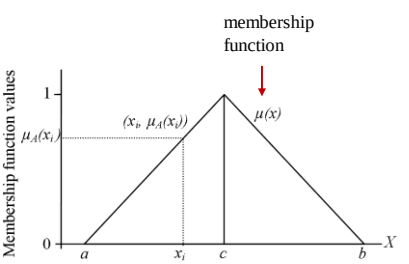


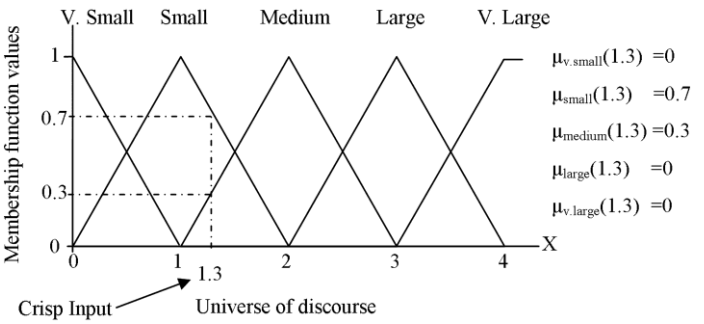
Fig.1 Membership function from the pair (x, μA(x))

Fuzzy Logic (FL)

❑ Fuzzy Set Theory

- Fuzzy Set Theory

✓ FL analyzes information using fuzzy sets, each of which is represented by a linguistic term such as "*small*", "*medium*" or "*large*". For example:



Fuzzy Logic (FL)

❑ Membership Functions (MFs)

A Membership Function (MF) is the curve that defines how each point in the universe of discourse is mapped to a membership value in the range of [0…1]. The membership function (μ) for a given input (x_i) can be written as:

$$0 \le \mu(x_i) \le 1 \qquad \ldots(4)$$

The membership function represents a "*degree of belonging*" for each object to a fuzzy set, and provides a mapping of objects to a continuous membership value in the interval [0…1].

Fuzzy Logic (FL)

❑ Membership Functions (MFs)

When a membership value is close to the value 1 ($\mu_A(x) \longrightarrow 1$), it means that input x belongs to the set A with a high degree, while small membership values ($\mu_A(x) \longrightarrow 0$) indicate that set A does not suit input x very well.

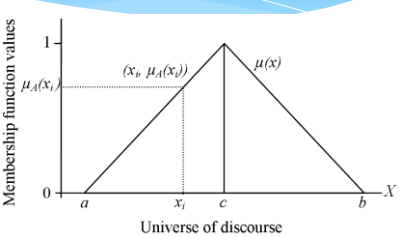
There are different forms of membership functions that can be used in FL, such as *Triangular*, *Trapezoidal*, Gaussian, Bell etc. The choice of membership function type is based on the designer experience and the problem under consideration.

Fuzzy Logic (FL)

❑ Membership Functions (MFs)

1. Triangular MFs

A triangular MF is specified by three parameters {a, b, c} as follows:



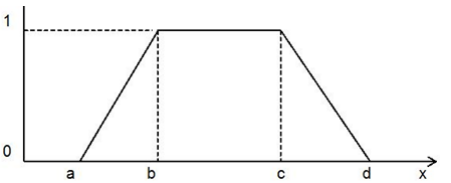
$$\mu(x;a,c,b) = \begin{cases} 0 & x \le a \\ \dfrac{x-a}{c-a} & a < x \le c \\ \dfrac{b-x}{b-c} & c < x < b \\ 0 & b \le x \end{cases} \quad \ldots(5)$$

where ***a*** and ***b*** correspond to the boundary of the triangular function curve, and ***c*** is the center of the function curve.

Fuzzy Logic (FL)

❑ Membership Functions (MFs)

2. Trapezoidal MFs



A trapezoidal MF is specified by four parameters {a, b, c, d} as follows:

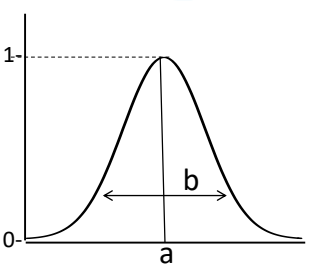
$$\mu(x;a,c,b,d)=\begin{cases}0 & x\le a\\ \dfrac{x-a}{b-a} & a<x<b\\ 1 & b\le x\le c\\ \dfrac{d-x}{d-c} & c<x<d\\ 0 & d\le x\end{cases} \qquad \ldots(6)$$

where ***a*** and ***d*** correspond to the boundary of the trapezoidal function curve, and ***b, c*** are the centers of the function curve.

Fuzzy Logic (FL)

❑ Membership Functions (MFs)

3. Gaussian MFs



A Gaussian MF is specified by two parameters {a, b} as follows:

$$\mu(x;a,b)=e^{-\frac{1}{2}\left(\frac{x-a}{b}\right)^2} \qquad \ldots(7)$$

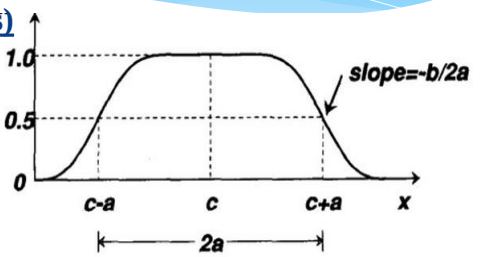
where ***a*** is the center and ***b*** is spreading of the function curve.

Fuzzy Logic (FL)

❑ Membership Functions (MFs)

4. Bell MFs

A Bell MF is specified by three parameters {a, b, c} as follows:



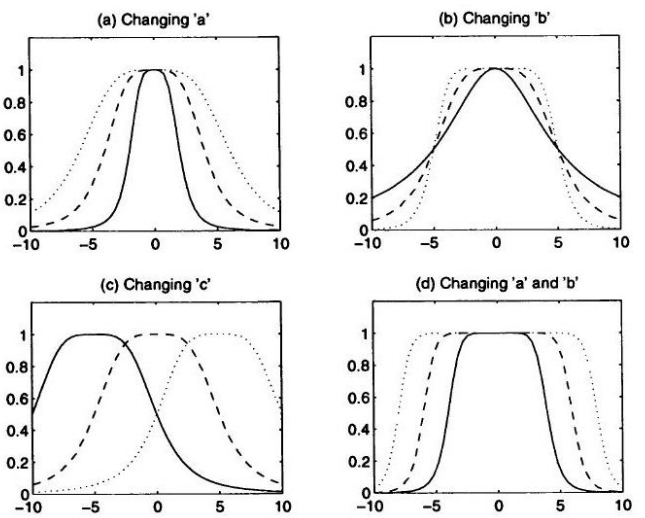
$$\mu(x;a,b,c)=\frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}} \qquad \ldots(8)$$

where ***c*** determines the center of the MF; ***a*** is the half width; and ***b*** (together with ***a***) controls the slopes at the crossover points.

Fuzzy Logic (FL)

❑ Membership Functions (MFs)

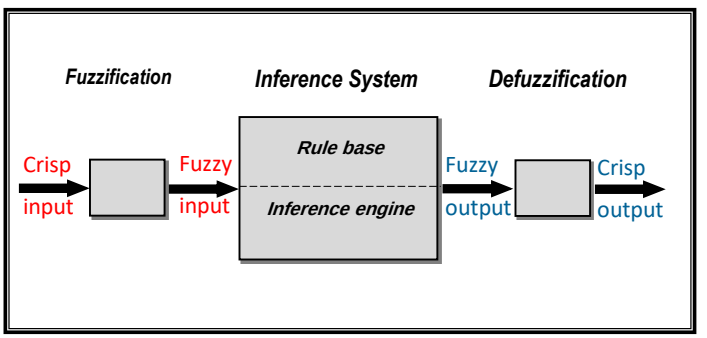
4. Bell MFs



Fuzzy System (FS)

❑ Fuzzy Structure

The four typical components required to design a fuzzy system are:



Fuzzy System (FS)

❑ Fuzzy Structure

1. Fuzzification

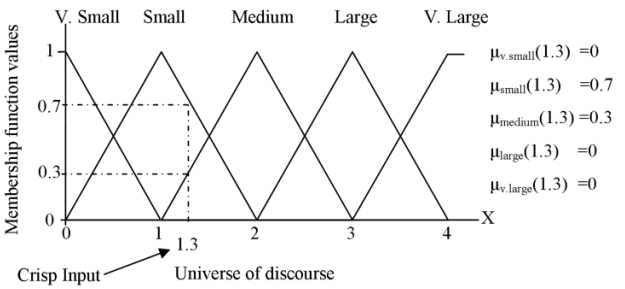
The fuzzification module converts the crisp numerical values into the degrees of membership (i.e., fuzzy values) related to the corresponding fuzzy sets.

The fuzzy values are compatible with the fuzzy set representation in the rule base.

The degree of membership returned by any membership function represents a fuzzy value that is always in the range [0…1].

Fuzzy System (FS)

Most variables in a fuzzy system have multiple membership functions attached to them; subsequently fuzzification will result in the mapping of a single crisp input value into several degrees of membership. The fuzzification process is illustrated in the following figure.



Chapter 3

Genetic Algorithms (GAs)

Genetic Algorithms (GAs)

❑ Introduction

Genetic algorithm is the first population-based optimization method.

They behave as a computational analog of adaptive systems by representing a general-purpose search algorithm that uses principles from natural population genetic to evolve solutions of problems.

Each genetic structure of the population, which is called a chromosome, represent an individual solution to the mentioned problem.

Genetic Algorithms (GAs)

❑ Introduction

The chromosome fitting capabilities are evaluated by an appropriate fitness function, determining their matching power during the competition process.

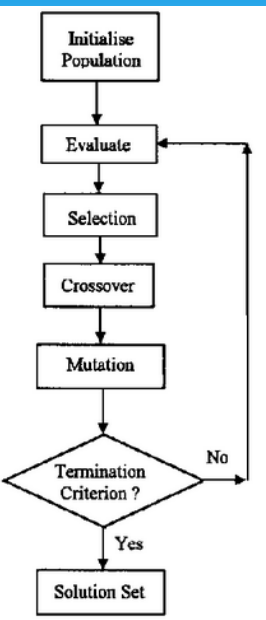
At each generation step new members of the population are created by applying genetic operator, such as Crossover and Mutation.

Genetic algorithms perform parallel random search on a set of solutions (population) to produce the better of them.

Genetic Algorithms (GAs)

❑ Foundations of GAs

The basic steps of the genetic algorithm are the following:



Genetic Algorithms (GAs)

❑ Foundations of GAs

1. Initialization

The first step of GAs is to produce the initial population of solution randomly.

Each chromosome consists of a number of genes. The chromosome is often referred to as the genotype of an individual. The following notation is used to describe the chromosome of individual number i.

$$chromosomre_i = gene_1 \ gene_2 \ \dots \ gene_L$$

where L is the length of this individual.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

1. Initialization

A number of chromosomes is called the population size. Choosing this size for a GA is a fundamental decision faced by all GA users. The population size affects both the ultimate performance and the efficiency of GAs.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

2. Encoding

In order to be able to use a GA to find solutions to problems, it is necessary to construct an encoding such that these possible solutions can be expressed by the chromosome.

The choice of an encoding is very important in order to achieve good performance. Generally, there are many different kinds of encoding, such as:

Genetic Algorithms (GAs)

❑ Foundations of GAs

2. Encoding

✓ **Binary encoding**

In this type of encoding every chromosome is a string of bits, 0 or 1, as shown in the following figure.

✓ **Integer encoding**

Every chromosome is a string of integer numbers, as shown in the following figure.

✓ **Real encoding**

Here, very chromosome a string of real numbers, as shown in the following figure.

Genetic Algorithms (GAs)

❑ Foundations of GAs

2. Encoding

Chromosome 1	1	1	0	0	1	0
Chromosome 2	1	1	0	0	1	0

(a) Binary encoding

Chromosome 1	1	5	3	2	6	4
Chromosome 2	8	3	6	7	2	9

(b) Integer encoding

Chromosome 1	2.9	1.2	1.0	2.3	0.5	4.1
Chromosome 2	8.0	1.3	6.1	7.9	0.2	1.9

(c) Real encoding

Examples of chromosome encoding

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

3. Evaluation

An evaluation of individuals is a measure of credit or goodness to be optimized that is entirely domain dependent.

The fitness function is used to evaluate the performance of each individual, such that every string in the population tested concerning its fitness for finding a solution for the coded problem, the formula of this function:

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

3. Evaluation

The formula of this function: $f: C \longrightarrow R^{+}$

where: f is a fitness function; C is a chromosome.

This fitness value has to correlate in some manner with the suitability of the chromosome by computing a fitness function, and it is used by the selection mechanism in determining which chromosome will survive and recombine or kill and penalize.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

4. Selection

Selection is the process of choosing the parents from population for the GA operations.

The purpose of selection of parents is to create the next generation that based on the fitness of individuals.

The probability of selection of any individual is proportional to its fitness. Thus, fitter individuals are more likely to be selected for reproduction.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

4. Selection

There are many strategies to select the parents, such as:

✓ ***Roulette Wheel Selection (RWS)***

The simplest selection scheme is Roulette wheel selection, which is use to select the parents that have a higher fitness with a higher probability.

Genetic Algorithms (GAs)

❑ Foundations of GAs

✓ *Roulette Wheel Selection (RWS)*

RWS method sums up the fitness of all individuals and calculates each individual's percentage of the total fitness.

The percentages of the total fitness are then used as the probabilities to select some individuals from the set population for represent the parents.

Genetic Algorithms (GAs)

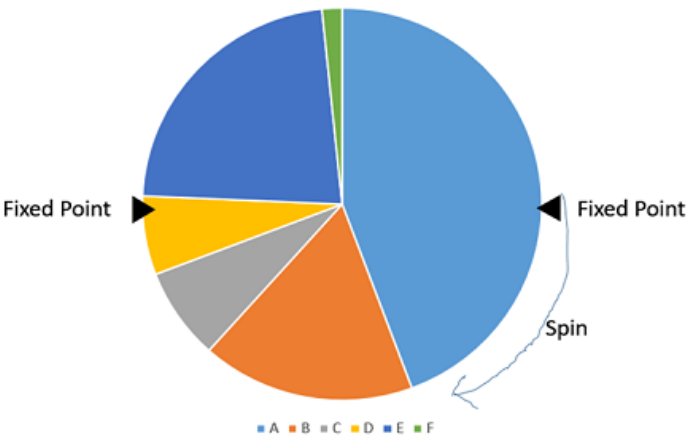
❑ Foundations of GAs

The following Table gives an example of RWS. From the Table, the parents are the first and third individuals, with a high probability

I	*Chromosome i*	*Decimal value* (x_i)	*Fitness i (Fi)* = (x_i^2)	$F_i / \sum F_i$
1	110101	53	2809	0.56
2	011010	26	676	0.14
3	100110	38	1444	0.29
4	000111	7	49	0.01
			$\sum F_i = 4978$	

Example of Roulette Wheel Selection

Example2 of RWS



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Genetic Algorithms (GAs)

❑ Foundations of GAs

✓ *Tournament selection*

Tournament selection simply picks a many different individuals at random and selects the individual with the large fitness. This process is repeated until we have selected the needed number of individuals.

For example, the binary tournament selection can be described more mathematically like the following:

Genetic Algorithms (GAs)

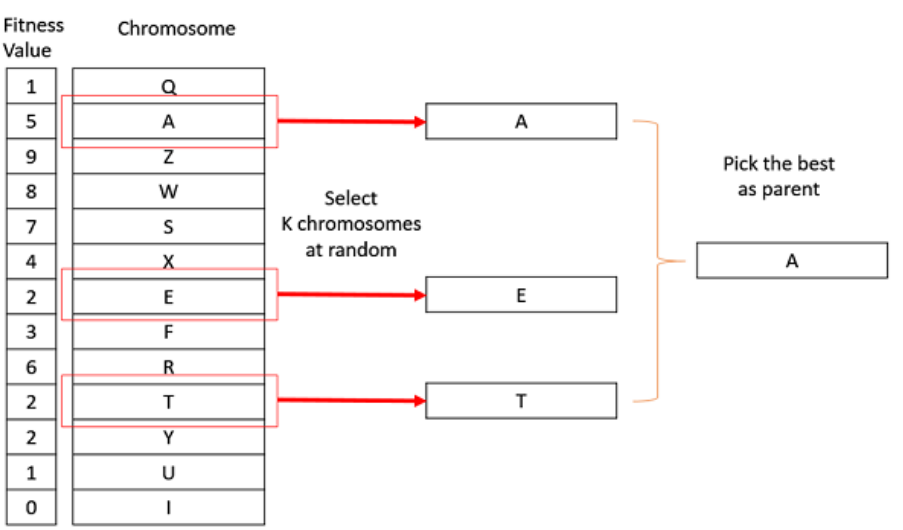
❑ Foundations of GAs

✓ ***Tournament selection***

$$Seclet_n = \begin{cases} ind_i & fitness(ind_i) > fitness(ind_j) \\ ind_j & otherwise \end{cases}$$

where n = { 1, .. , N }, random numbers i, j ∈ {1,.., n} and i, j and N are number of individuals selection.

Example of Tournament



Genetic Algorithms (GAs)

❑ **Foundations of GAs**

5. Crossover

Crossover is the main genetic operator. It operates on two individuals at a time and generates two offspring (new points in the search space to be tested).

The purpose of crossing strings in the GAs is to test new parts of target regions then testing the same string over and over again is successive generation.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

5. Crossover

The crossover rate controls the frequency with which the crossover operator is applied. The higher the crossover rate, the more quickly new structures are introduced into the population, If the crossover rate is too high, high-performance structures are discarded fast than selection can produce improvement and if the crossover rate is too low, the search may stagnate due to the lower exploration.

Genetic Algorithms (GAs)

❑ Foundations of GAs

5. Crossover

There are several types of crossover operators, such as: -

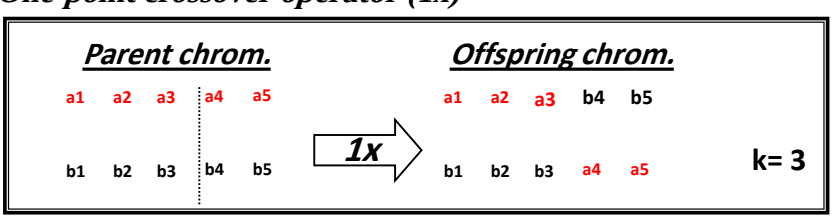
i. One-point crossover operator (1x)

In this operator, every pairs of individuals (parents) one position in individual genetic code is chosen. All genes after that position are exchanged among individuals. The example of 1X crossover can be seen in the following figure.

Genetic Algorithms (GAs)

❑ Foundations of GAs

i. One-point crossover operator (1x)



One-point crossover operator (1x)

where k is the crossover site; k∈{1, .. , L-1}, and L is the chromosome length.

This operator is very fast, but it has problem of decreasing diversity especially when individuals are similar in the population.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

5. Crossover

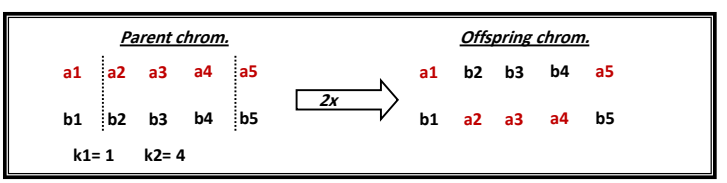
ii. Two-point crossover operator (2x)

In two points crossover chooses two crossover sites at random, instead of only one site in one point crossover. After that, individuals exchange genes between these sites as can be seen in the following figure.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

ii. Two-point crossover operator (2x)



Two-points crossover operator (2x)

where k1,k2 are the crossover sites; k1,k2∈ {1, .. , L-1}, and k1 ≠ k2, when L is the chromosome length.

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

ii. Two-point crossover operator (2x)

Speed and capability characterize this operator. It used with big chromosomes, where it can hereditary two parts from one of parents and one part from the other to each individual and that can help to produce different individuals.

This operator gives good results when the diversity is high in the population but it capability decreased .

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

5. Crossover

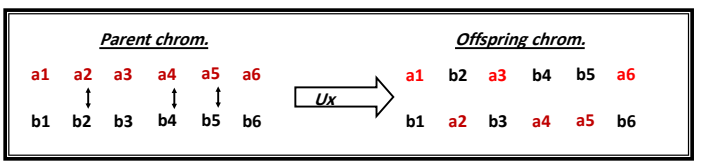
iii. Uniform crossover operator (Ux)

The **Ux** operator quite different than 1x and 2x, since the Ux does not use crossover sites, but randomly shuffles the genes of the parents, by probability of exchanging genes, in order to create two offspring, as can be seen in the following figure.

Genetic Algorithms (GAs)

❑ Foundations of GAs

iii. Uniform crossover operator (Ux)



Uniform crossover operator (Ux)

This operator has a large ability to product different individuals from both parents (to increase the diversity of population); but it is slow when treats with big chromosomes.

Crossover: 1x (k=5)

Chromosome1	11011|00100110110
Chromosome2	11011|11000011110
Offspring1	11011|11000011110
Offspring2	11011|00100110110

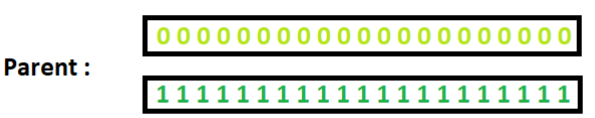
Single Point Crossover

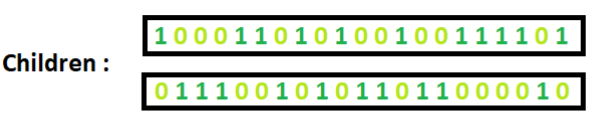
Crossover: 2x (k1=5,K2=10)

Chromosome1	11011|00100|110110
Chromosome2	10101|11000|011110
Offspring1	11011|11000|110110
Offspring2	10101|00100|011110

Two Point Crossover

Crossover: Ux (k=1,5,6,8,10,1316,17,18,19,21)





Uniform Crossover

Genetic Algorithms (GAs)

❑ **Foundations of GAs**

6. Mutation

Mutation is a random change of one or more genes. It prevents falling all solutions in population into a local optimum of solved problem. It has the effect of increasing the genetic diversity of the population by creating new individuals over a long period of time and prevents stagnation in the convergence of the optimization technique.

Genetic Algorithms (GAs)

6. Mutation

Mutation operator increases the variability of the population when each bit position of each chromosome in the new population undergoes a random change with a probability equal to the mutation rate. A low level of mutation rate serves to prevent many given bit position from remaining forever converged to a single value in the entire population and a high level of mutation rate yields an essentially random search.

Genetic Algorithms (GAs)

6. Mutation

There are several types of mutation operators, such as: -

i. One-point mutation operator (1m)

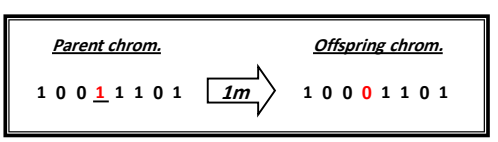
In this operator a single gene in the chromosome at a selected mutation point and changed it value to other value in the range of this gene. When the gene has a binary encode the value is flipping from “1” to “0” or vice versa, as can be seen in the following figure:-

Genetic Algorithms (GAs)

6. Mutation

There are several types of mutation operators, such as: -

i. One-point mutation operator (1m)



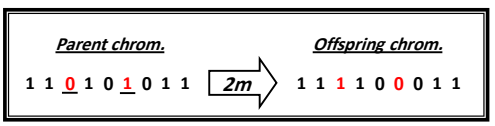
Example: One-point mutation operator (1m)

Genetic Algorithms (GAs)

6. Mutation

ii. two-point mutation operator (2m)

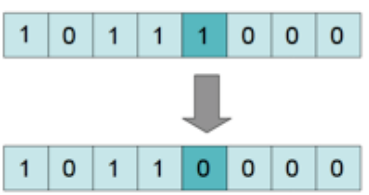
This operator selected two genes in the chromosome at randomly and swapping the values of these genes between them, as can be seen in the following figure:-



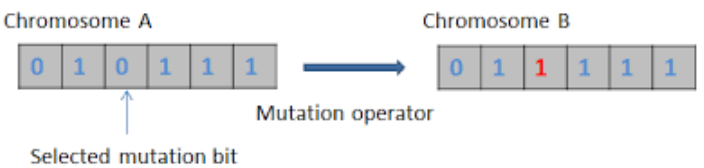
Example: Two-points mutation operator (2m)

One-point mutation operator (1m)

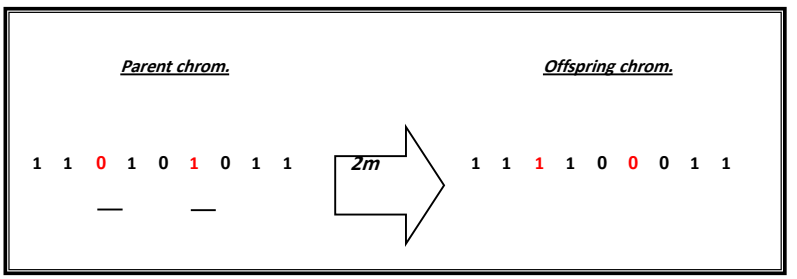
k=5



k=3



two-point mutation operator (2m)



Example: Two-points mutation operator (2m)

mutation operator (um)

(um) k=3,4,5

Before Mutation

A5	1	1	1	0	0	0

After Mutation

A5	1	1	0	1	1	0

Genetic Algorithms (GAs)

7. Replacement

The replacement operator removes few relatively poor individuals, which have a low fitness, from the population, such that, there is space for the new individuals which have a high fitness.

There are many ways for the replacement, such as:

Genetic Algorithms (GAs)

7. Replacement

i. Holland replacement way

This way used to replacement the worst individual in the population by the new individual and evaluation the population new again.

The disadvantage of this way, it does not compare the cost of the remove individual and the new individual.

Genetic Algorithms (GAs)

ii. Whitley replacement way

This way selected two (or three) individuals from the population randomly and determine the worst among them, compare it with the new individual. If it's cost lower than the cost of the new individual then replacement it by the new individual else does not use the replacement operator.

The problem of this way is that the selection of individuals from the population randomly, which does not give a good chance for selected the worst individual.

Genetic Algorithms (GAs)

8. Termination (stopping criterion)

The GA iteratively performs operators on each generation of individuals to produce new generation. This loop continues until some halt criterion is satisfied.

The termination can be one or more of the following criterion:-

Genetic Algorithms (GAs)

8. Termination (stopping criterion)

- A predetermined number of generations to be run, than to take the best solution.
- Stopping when the fitness of the number of the population is within the user specified range.
- When all individuals in the generation are identical, which means the evaluation function for all individuals are the same.
- When a relative convenience of a good solution (among several ones) to the problem domain is achieved.

```
Procedure GA;
  Begin
    Generate initial population;
    Evaluate each individual's fitness;
    Repeat
      Selection individuals;                /* Parents */
      Apply crossover operator;
      Apply mutation operator;
      Evaluate each individual's fitness;  /*Offspring*/
      Apply replacement policy;
    Until (terminating condition);
  End;
```

Procedure of Genetic Algorithm

Apply the GA for three generations to find the maximum decimal number for six bit binary code. Consider the following initial population:

no	chromosome
1	101010
2	110101
3	101101
4	110011
5	111000

Using the following parameters:

1. Selection : RWS
2. Crossover: 1x (k=3)
3. Mutation: 1m (k=4)
4. Replacement: Holland way

H.W. In one of companies, the manager wants to select the best employee. The selection depends on some of criteria, i.e.:

1. **The certificate:** it is evaluated by 30% from the total evaluation. It's divided to (Bachelor, Master, and PhD) and assigning to it two genes (01, 10, and 11), respectively.
2. **The years of service:** it is evaluated by 25% from the total evaluation. It's divided to (Less than 5 years, 5-to-10 years, 10-to-15 years, and more than 15 years) and assigning to it two genes (00, 01, 10, and 11), respectively.
3. **The experience:** it is evaluated by 25% from the total evaluation. It's divided to (acceptable, good, v. good, and excellent) and assigning to it two genes (00, 01, 10, and 11), respectively.
4. **The acknowledgement:** it is evaluated by 20% from the total evaluation. It's divided to (one, 2-4, 5-7, more than 7) and assigning to it two genes (00, 01, 10, and 11), respectively.

Please use the GA for **two loops only** to choose the best employee in the company. When using the GA, it should take in your account the following:

- The selection is ***Roulette Wheel***
- The crossover is 2x (k1=3, and k2=6).
- The mutation is not use here
- The replacement is ***Holland Way***

FYI, the initial population is:

no	chromosome
1	10 01 10 00
2	11 10 11 10
3	01 11 11 01
4	01 10 11 10
5	10 10 11 10

1. **The certificate:** it is evaluated by 30% from the total evaluation. It's divided to (Bachelor, Master, and PhD) and assigning to it two genes (01, 10, and 11), respectively.
2. **The years of service:** it is evaluated by 25% from the total evaluation. It's divided to (Less than 5 years, 5-to-10 years, 10-to-15 years, and more than 15 years) and assigning to it two genes (00, 01, 10, and 11), respectively.
3. **The experience:** it is evaluated by 25% from the total evaluation. It's divided to (acceptable, good, v. good, and excellent) and assigning to it two genes (00, 01, 10, and 11), respectively.
4. **The acknowledgement:** it is evaluated by 20% from the total evaluation. It's divided to (one, 2-4, 5-7, more than 7) and assigning to it two genes (00, 01, 10, and 11), respectively.

Lecture 4

Neural Networks (NNs)

Neural Networks (NNs)

❑ **Introduction**

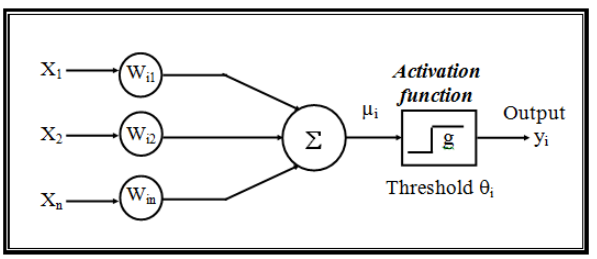
- ✓ A neural network is an information-processing system that has performance characteristics in common with biological neural networks.
- ✓ Neural networks have been developed as generalizations of mathematical models of neural biology. They can be considered as a parallel distributed for storing experimental knowledge and making it available for use.
- ✓ They represent mathematical models of brain-like systems where knowledge is received through a learning process.

Neural Networks (NNs)

❑ Structure of Neuron Cell

✓ A neural network is composed of a large number of high-interconnected processing elements (neurons) working in parallel to solve a specific problem.

✓ As illustrated in the following figure a *neuron i* consists of a set of *n* connecting links that are characterized by weights *Wij.*



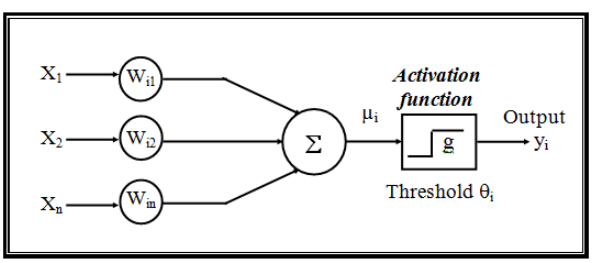
Basic neuron model

Neural Networks (NNs)

❑ Structure of Neuron Cell

✓ Each input signal x_j applied to the link j is multiplied by its corresponding weight w_{ij} and transmitted to *neuron i*. These links product are accumulated by the adder as expressed by the formula:

$$\mu_i = \sum_{j=1}^{n} (x_j * w_{ij})$$



Basic neuron model

Neural Networks (NNs)

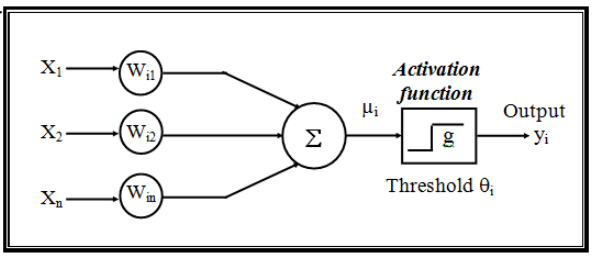
❑ Structure of Neuron Cell

✓ Each an activation function g ( ) provides the output y_i of the unit as:

$$y_i = g(\mu_i - \theta_i)$$

✓ θ_i denotes the threshold which is an external parameter of *neuron i* and is used to apply an affine transformation of the net input to the output.

✓ The activation function defines the output of neuron.



Basic neuron model

Neural Networks (NNs)

❑ Structure of Neuron Cell

✓ There are many several types of activation functions, such as:

Activation Function	Formula
Sign Function	$g(x)=\begin{cases} 0 & x<0 \\ 1 & x \geq 0 \end{cases}$
Sigmoid Function	$g(x)=\frac{1}{1+e^{-Bx}}$
Pulse Function	$g(x)=\begin{cases} 1 & a \leq x \leq b \\ 0 & otherwise \end{cases}$
Gaussian Function	$g(x)=\frac{(x-c)^2}{e^{2\sigma^2}}$

Some of Activation Functions

Neural Networks (NNs)

❑ **The Learning Methods For Neural Networks**

There are two different types of learning methods have been constructed to give the neural networks the ability to adjust themselves intelligently.

1. Supervised Learning

In this type of learning, data are presented together with teacher information in order to associate the data with the teacher signal.

Supervised learning algorithms adjust the weights using input-output data to match the input-output characteristics of a network to desired characteristics. One of the most popular of these algorithms is the back-propagation learning algorithm.

Neural Networks (NNs)

❑ **The Learning Methods For Neural Networks**

There are two different types of learning methods have been constructed to give the neural networks the ability to adjust themselves intelligently.

2. Unsupervised Learning

The neural network is presented for some data without getting any teacher information. This type of learning is often used for data clustering and data analysis.

Neural networks that use unsupervised learning use the redundancy in the data in order to build up clusters or feature maps based on a familiarity distance. K-Means clustering algorithm represents one of unsupervised learning algorithms.

Neural Networks (NNs)

❑ The Back-propagation (Bp) Network

Back-propagation (Bp) is a multiple layer network. it has one input layer and one output layer as well as one or more hidden layers.

In order to train a neural network to perform some task, it must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced.

Neural Networks (NNs)

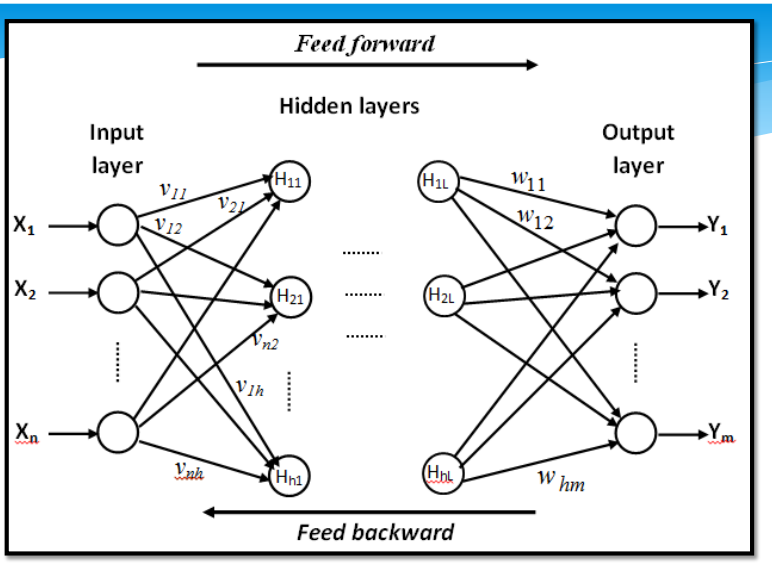
❑ The Back-propagation (Bp) Network

This process of Bp requires that the neural network compute the error derivative of the weights.

In other words, Bp algorithm must calculate how the error changes as each weight is increased or decreased. It looks for the minimum of the error function in weight space using the method of gradient descent learning.

The Bp network undergoes supervised training with a finite number of pattern pairs. Each one of these pairs consists of an input pattern and a desired or target output pattern.

The following figure explains the general structure of the Bp network.



The structure of Back-propagation network

where X is the input vector; Y is the output vector; and H is the hidden layers.

Neural Networks (NNs)

❑ The Back-propagation (Bp) Network

The learning process of Bp algorithm involves two steps:

1. **Feed forward**
   Entering each one of the pattern pairs (input, output) and computing the actual output.
2. **Feed backward**
   Adjustment all the weights depending on the difference between the desired output and the actual output.

This process repeated as many times as needed until the error between the desired and the actual outputs reaches to the minimum value.

Neural Networks (NNs)

❑ **The gradient descent learning:**

This is based on the minimization of errors ***E*** defined in terms of weights and the activation function of the network.

- The activation function of the network is required to be differentiable because the updates of weights is dependent on the gradient of the error ***E.***
- If $\Delta \boldsymbol{w_{ij}}$ is the weight update of the link connecting the i^{th} and j^{th} neurons of the two neighboring layers. So, the $\Delta \boldsymbol{w_{ij}}$ is defined as : $\Delta w_{ij} = -\eta \, (\partial E \,/\, \partial w_{ij})$
  where η is the learning rate parameter and $(\partial E \,/\, \partial w_{ij})$ is error gradient with reference to the weight w_{ij}.

Note: the Bp learning is the example of Gradient descent learning.

Neural Networks (NNs)

❑ **The Back-propagation (Bp) Network**

The Bp algorithm is based on the gradient descent technique for solving an optimization problem, which involves the minimization of the network cumulative error ***E***, it defines as:

$$E = \sum_{k=1}^{n} E(k)$$

where ***n*** is the number of training patterns presented to the network for training purposes. ***E(k)*** represents the vectorial difference between the target output and the actual output vectors of the network, it defines as:

$$E(k) = \frac{1}{2} \sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

where $t_i(k)$ is the target output vector, and $o_i(k)$ is the actual output vector.

Neural Networks (NNs)

❑ The Back-propagation (Bp) Network

So, the minimization of the network error becomes:

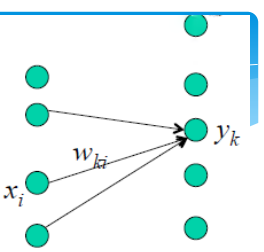
$$E = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q}[t_i(k) - o_i(k)]^2$$

where the index i represents the i^{th} neuron of the output layer composed of the a total number of q neurons.

A simple Linear Model

- Outputs y_k are linear combinations of input variables x_i

$$y_k = \sum_i w_{ki} x_i$$



- Error function for a particular input x_n has the form

$$E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$$

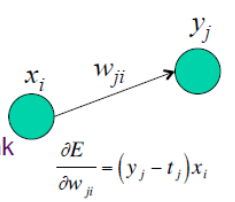
  - where $y_{nk}=y_k(\mathrm{x}_n, \mathrm{w})$

Subscript n is for a particular input $\mathbf{x}_n$ which is ignored below

- Gradient of Error function a weight w_{ji}

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni}$$

  - a local computation involving product of
    - error signal y_{nj}-t_{nj} associated with output end of link w_{ji}
    - variable x_{ni} associated with input end of link



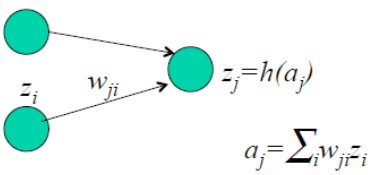
General Feed-Forward Network: Forward Propagation

- Each unit computes weighted sum of its inputs

$$a_j = \sum_i w_{ji} z_i$$

- z_i is activation of a unit (or input) that sends a connection to unit j and w_{ji} is the weight associated with the connection
- Transformed by nonlinear activation function

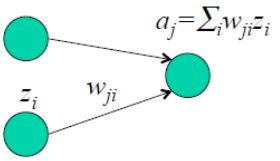
$z_j = h(a_j)$



Evaluation of Derivative E_n a weight w_{ji}

- By chain rule for partial derivatives

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$



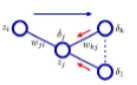
Define $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$

$a_j = \sum_i w_{ji} z_i$

we have $\frac{\partial a_j}{\partial w_{ji}} = z_i$

- Substituting $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$
- Thus required derivative $\frac{\partial E_n}{\partial w_{ji}}$ is obtained by
  - Multiplying value of δ for the unit at output end of weight by value of z for unit at input end of weight
- Need to figure out how to calculate $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$

Calculation of Error for hidden unit δ_j



- For output unit $\delta_k = y_k - t_k$ Since $E = \frac{1}{2}\sum_k (y_k - t_k)^2$ and $y_k = a_k = \sum_i w_{ki} z_i$ $\delta_k \equiv \frac{\partial E}{\partial a_k}$
- For hidden unit j
  - By chain rule

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

We are summing partial derivatives over several variables a_k

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k}$$

$$a_k = \sum_i w_{ki} z_i = \sum_i w_{ki} h(a_i)$$

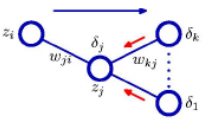
$$\frac{\partial a_k}{\partial a_j} = \sum_k w_{kj} h'(a_j)$$

- Substituting
  - We get the backpropagation formula for for error derivatives at stage j

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

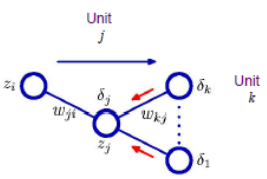
Input to activation from earlier units

error derivative at later unit k



Blue arrow for forward propagation
Red arrows indicate direction of information flow during error backpropagation

Error Backpropagation Algorithm



- Backpropagation Formula

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

- Value of δ for a particular hidden unit can be obtained by propagating the δ's backward from units higher-up in the network

1. Apply input vector x_n to network and forward propagate through network using

$$a_j = \sum_i w_{ji} z_i \quad \text{and} \quad z_j = h(a_j)$$

2. Evaluate δ_k for all output units using $\delta_k = y_k - t_k$
3. Backpropagate the δ's using

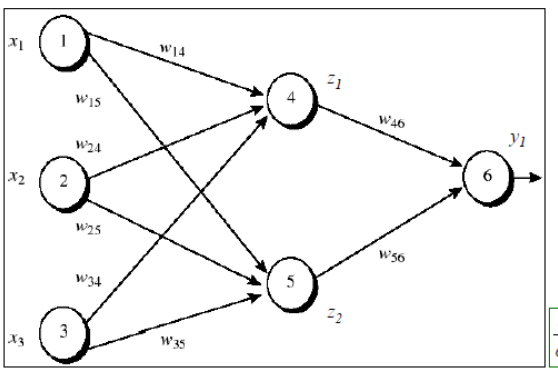
$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

to obtain δ_j for each hidden unit

4. Use $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$

to evaluate required derivatives

A numerical example



$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i$$
$$z_j = \sigma(a_j)$$
$$y_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j$$

Errors

$$\delta_j = \sigma'(a_j) \sum_k w_{kj} \delta_k$$

$$\delta_k = \sigma'(a_k)(y_k - t_k)$$

Error Derivatives

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

- First training example, $x = [1 \quad 0 \quad 1]^T$ whose class label is $t = 1$
- The sigmoid activation function is applied to hidden layer and output layer
- Assume that the learning rate η is 0.9

Outputs, Errors, Derivatives, Weight Update

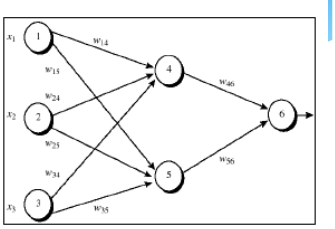
$$\delta_k = \sigma'(a_k)(y_k - t_k) = [\sigma(a_k)(1 - \sigma(a_k))](1 - \sigma(a_k))$$
$$\delta_j = \sigma'(a_j) \sum_k w_{jk} \delta_k = \left[\sigma(a_j)(1 - \sigma(a_j))\right] \sum_k w_{jk} \delta_k$$

Initial input and weight values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	w_{04}	w_{05}	w_{06}
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Net input and output calculation

Unit	Net input a	Output $\sigma(a)$
4	0.2 + 0 -0.5 -0.4 = -0.7	$1/(1+e^{0.7})=0.332$
5	-0.3 +0+0.2 +0.2 =0.1	$1/(1+e^{0.1})=0.525$
6	(-0.3)(0.332)-(0.2)(0.525)+0.1 = -0.105	$1/(1+e^{0.105})=0.474$



Errors at each node

Unit	δ
6	(0.474)(1-0.474)(1-0.474)=0.1311
5	(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065
4	(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087

Weight Update*

Weight	New value
w_{46}	-03+(0.9)(0.1311)(0.332)= -0.261
w_{56}	-0.2+(0.9)(0.1311)(0.525)= -0.138
w_{14}	0.2 +(0.9)(-0.0087)(1) = 0.192
w_{15}	-0.3 +(0.9)(-0.0065)(1) = -0.306
w_{24}	0.4+ (0.9)(-0.0087)(0) = 0.4
w_{25}	0.1+ (0.9)(-0.0065)(0) = 0.1
w_{34}	-0.5+ (0.9)(-0.0087)(1) = -0.508
w_{35}	0.2 + (0.9)(-0.0065)(1) = 0.194
w_{06}	0.1 + (0.9)(0.1311) = 0.218
w_{05}	0.2 + (0.9)(-0.0065)=0.194
w_{04}	-0.4 +(0.9)(-0.0087) = -0.408

* Positive update since we used $(t_k - y_k)$