AI Lecture Notes

▼ Chapters

What is ML?

Definition

- Arthur Samuel (1959)
    - “the field of study that gives computers the ability to learn without being explicitly programmed

- Herbert Simon (1970)
    - “Learning is any process by which a system improves performance from experience”
- Tom Mitchell (1998)
    - “a well-posed learning problem is defined as follows: He says that a computer program is set to learn from an experience E with respect to some task T and some performance measure P if its performance on T as measured by P improves with experience E”
- Ethem Alpaydin (2010)
    - Programming computers to optimize a performance criterion using example data or past experience

Necessity

- Better computational systems
    - to difficult or expensive to be constructed manually
        - systems that automatically adapt
            - ex: spam filters
        - systems that discovers information in large database → data mining
            - ex: financial analysis, test/image analysis
- understanding the biological systems

How to design ML?

Improve of task T

- establish the goal (what has to be learned) - *objective function* - and its *representation*
- select a learning algorithm to perform the inference of the goal based on experience

Respect a performance metric P

- evaluation of the algorithm’s performance

Based on experience E

- select an experience database

Examples

- Example 1
  - T: playing checkers
  - P: percent of winning games
  - E: playing the game
- Example 2
  - T: handwritten recognition
  - P: percent of correct recognised words
  - E: database of images with different words
- Example 3
  - T: separate the spams
  - P: percent of correct classified emails
  - E: database with annotated emails

Objective Function

What is the function that must be learned?

- ex: checkers game → a function that:
  - selects the next move

    - evaluates a move in order to identify the best next move
    - a linear combination of #white_pieces, #black_pieces, #white_compromised_pieces, #black_compromised_pieces

Representation of objective function

- different representations:
    - table
    - symbolic rules
    - numeric functions
    - probabilistic functions
- there is a trade-off between
    - how expressive is the meaning of the representation
    - ease of learning
- objective function computation
    - polynomial time
    - non-polynomial time

The Learning Algorithm

Selection

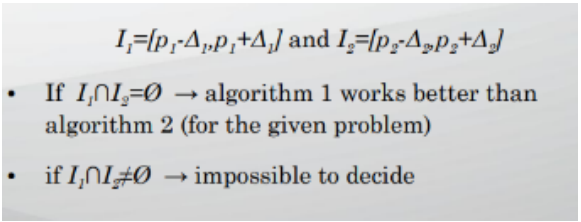
- according to the training data
- induce the hypothesis definition that
    - match the data
    - generalize the unseen data
- main principles
    - error minimisation (cost function - loss function)

Evaluation

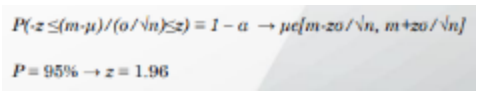
- experimental
  - by comparing different methods on different data (cross-validation)
  - collect data based on performances
    - accuracy, training time, testing time
  - statistical analyse of differences
- theoretic
  - mathematical analysis of algorithms and theorems proving
    - computation complexity
    - ability to match the training data
    - complexity of the most relevant sample for learning

Comparison between 2 algorithms

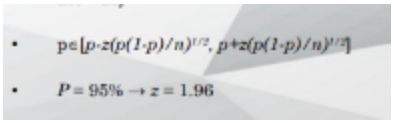
- performance measures
  - parameters of a statistic series
  - proportion (percent) computed for a statistical series (ex: accuracy)
- comparing based on confidence intervals
  - for a problem and 2 solving algorithms with performances p1 and p2
  - confidence intervals



  - confidence interval for mean
    - for a statistical series of n data, with computed mean m and dispersion o, determine the confidence interval of the mean p



- confidence interval for accuracy
  - for an accuracy p computed for n data, determine the confidence interval of accuracy



Training database

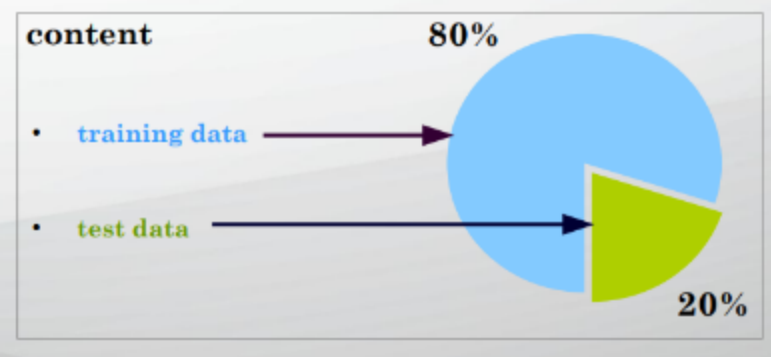
Choose the training database based on

- direct experiences
  - pairs (in, out) that are useful for the objective function
  - eg: checkers game → board game annotated by correct or incorrect move
- indirect experience
  - useful feedback (unlike i/o pairs) for the objective function
  - eg: checkers game → sequence of moves and the final score of the game

Data sources

- random generated examples
  - positive and negative examples
- positive examples collected by a learner
- real examples

Content



Characteristics

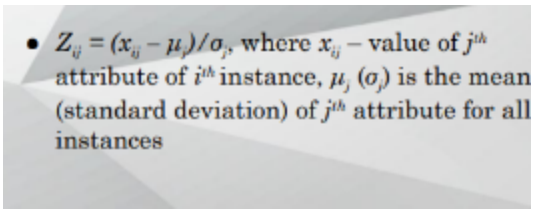
- independent data
  - otherwise → collective learning
- training and test data must respect the same distribution law
  - otherwise → transfer learning / inductive transfer
    - vehicle recognition → truck recognition
    - text analysis
    - spam filters

Characteristics extracted (attributes) from raw data

- quantitative characteristics → nominal or rational scale
  - continuous values → weight
  - discrete values → # of computers
  - range values → event times
- qualitative characteristics
  - nominal → color
  - ordinal → sound intensity (low, medium, high)
- structured
  - trees - root is a generalisation of children (vehicles → car, bus, tractor, truck)

Data Transformation

- standardisation → numerical attributes
  - remove the scale effect (different scale and units)
  - raw values tranformed in z scores



- selection of some attributes

ML Classification - goal oriented

Intelligent systems for prediction

- aim: predict the output for a new input based on a previously learned model
- ex: predicting sales of a product for a time in the future based on price, calendar month, region, average income

Intelligent systems for classification

- aim; classify an object into one or more - known or unknown - categories based on their characteristics
- ex: diagnostic systems for cancer: malign, benign or normal

Intelligent systems for regression

- aim: estimation of the (uni or multi variabe) function's shape based on a previouslt learned mode
- ex: estimate the function that models the edge of a surface

Intelligent systems for planning

- aim: generate a sequence of optimal actions for performing a task
- ex: planning the moves of a robot from a position to a source of energy

ML classification - based on learning experience

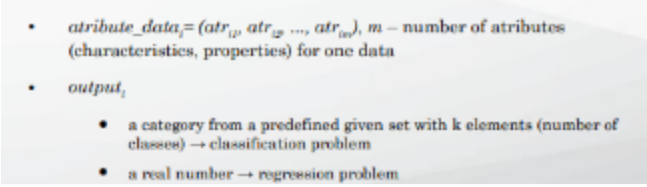
Intelligent systems with supervised learning

Aim

- to provide a correct output for a new input

Definition

- for a given data set (examples, instances, cases)
  - training data - pairs (attribute_data, output) where
    - i = 1,N (N = number of training data pairs)



  - test data - a set (attribute_Data), i=1,n (n=number of test pairs)
- determine
  - a function (unknown) that maps the input attributes to the output on the training data set
  - the output (class/value) associated with a test data (new) using the detected function

Other names

- classification (regression), inductive learning

Process → 2 steps

- training
  - learning the classification model - using an algorithm
- testing
  - test the model with the new data (not seen in the training step)

Feature

- data is labeled (for learning and sometimes also for testing)

Suitable problem types

- regression, classification

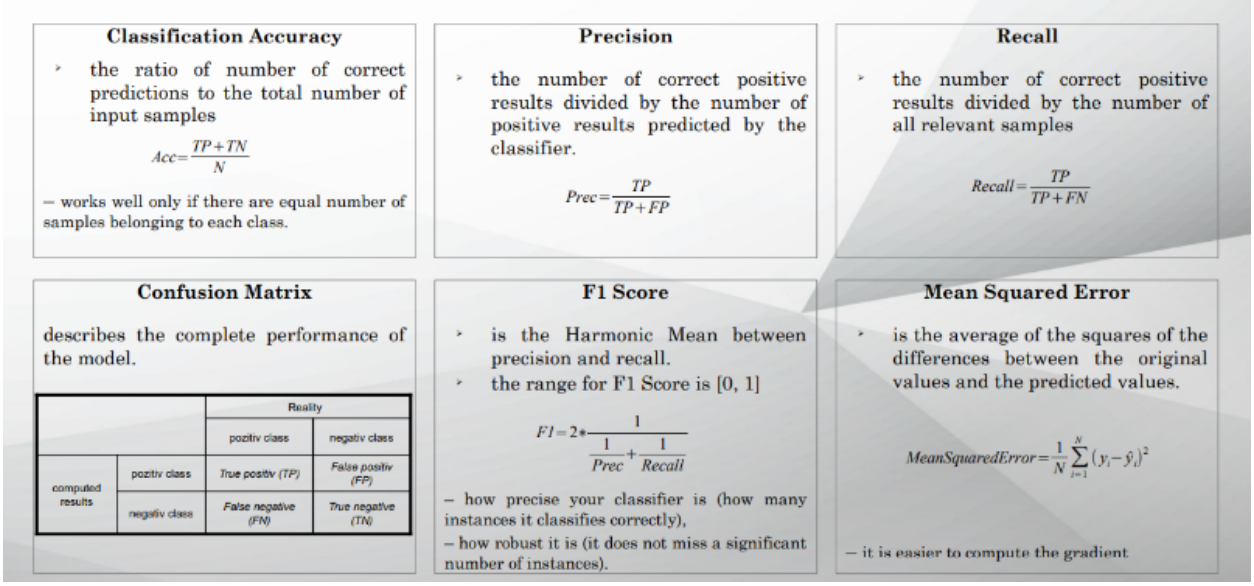
Learning quality

- we consider a performance measure for the algorithm that is evaluated during both steps, training and testing, separately
- Evaluation methods

for a large dataset	for a small dataset	for a very small dataset
disjunctive sets for training and testing	cross validation with h equal sub sets of the dataset	leave one out cross validation

- Difficulites: over-fitting - good performance on training data but very poor on testing data
- performance measures:
  - statistical
  - efficiency
  - robustenss
  - scalability
  - interpret-ability
  - compactness

Statistical Metrics



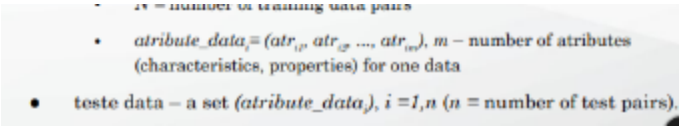
Intelligent systems with unsupervised learning

Aim

- to detect a model or a internal util structure of the data

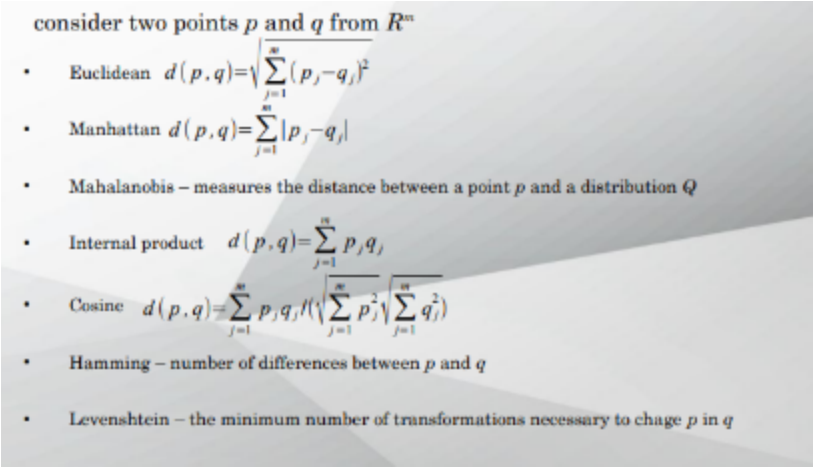
Definition

- for a given data set (examples, instances, cases)
  - training data - a set {attribute data | i=1,N} where
    - N = number of training data pairs



  - determine
    - an unknown function that groups the training data in several classes
      - the number of classes k can be predefined or unknown
      - the data from one class are similar
    - the output is the class the new input data (new) using the detected function

Examples of distances



Distance vs Similarity

- distance → min
- similarity → max

Other names

- clustering

Process → 2 steps

- training
  - learning (determine), using an algorithm, the existing clusters
- testing
  - test the model with the new data (not seen in the training step)

Feature

- db is not labeled (for learning and sometimes also for testing)

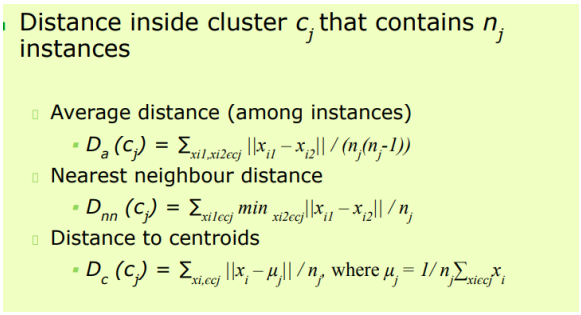
Suitable problem types

- Identifying some classes (clusters)
  - Genome analysis

    - Image processing
    - Social network analysis
    - Market segmentattion
    - Astronomic data analysis
    - Computer clustering
- Dimensional reduction
- Identifying data properties (causes, explanation, etc)
- Modeling data density

Learning quality

- internal criteria - high similarity withing a cluster and a reduced one between clusters
    - distance inside the cluster



    - distance between the clusters

- Simple link
  - $d_s(c_{j1}, c_{j2}) = min_{xi1 \epsilon cj1,\ xi2 \epsilon cj2} \{||x_{i1} - x_{i2}||\}$
- Complete link
  - $d_{co}(c_{j1}, c_{j2}) = max_{xi1 \epsilon cj1,\ xi2 \epsilon cj2} \{||x_{i1} - x_{i2}||\}$
- Average link
  - $d_a(c_{j1}, c_{j2}) = \Sigma_{xi1 \epsilon cj1,\ xi2 \epsilon cj2} \{||x_{i1} - x_{i2}||\} / (n_{j1} * n_{j2})$
- Link between centroids
  - $d_{ce}(c_{j1}, c_{j2}) = ||\mu_{j1} - \mu_{j2}||$

  - Davies-Boulding index

    - $DB = 1/nc * \Sigma_{i=1,2,\ldots,nc} max_{j=1,2,\ldots,nc,\ j \neq i}((\sigma_i + \sigma_j)/d(\mu_i, \mu_j))$
    - where:
      - nc – # of clusters
      - μ_i – centroid of cluster i
      - σ_i – average of distances between elements form cluster i and the centroid μ_i
      - $d(\mu_i, \mu_j)$ – distance between centroid μ_i and centroid μ_j

  - Dunn index

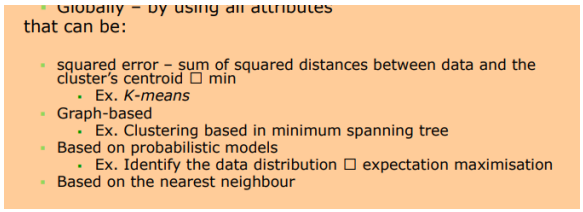
    - Identifies the dense clusters and well separated
    - $D = d_{min} / d_{max}$
    - where:
      - d_{min} – minimal distance between 2 elements from different clusters - intra-cluster distance
      - d_{max} – maximal distance between 2 elements from the same cluster - inter-cluster distance

- external criteria - using benchmarks made from already grouped data sets
  - comparison with known data - almost impossible in practice
  - precision
  - recall
  - f1-measure

How the clusters are forming

- hierarchic clustering
  - creates a dendogram (taxonomic tree)

        - create the clusters (recursively)
        - k is unknown
    - agglomerative clustering (bottom-up) → small clusters to large clusters
    - divisive clustering (top-down) → large clusters to small clusters
    - eg: clustering ierhahic agglomerative
- non-hierarchical (partitioned) clustering
    - partitional → determine a data separation → all the clusters in the same time
    - optimises an objective function defined
        - locally - by using some features only
        - globally - by using all attributes



    - required to fix k apriori → fix the initial clusters
        - algorithm is run more times with different parameters and the most efficient version is selected
    - ex: k-means, aco
- clustering based on data density
    - data density and data connectivity
        - cluster formation is based on data density from a given region
        - cluster formation is based on data connectivity from a given region
    - function of data density
        - tries to model the data distribution
    - advantage:

        - modelling of clusters of any shape
- clustering based on a grid
    - is not a distinct approach
        - can be hiearchic, partitional or density-based
    - involves data space segmentation in regular areas
    - objects are placed on a multi-dimensional grid
    - ex: aco

How the algorithms work

Agglomerative clustering

- initially, each instance form a cluster
- compute the distance between any 2 clusters
- reunion the closest 2 clusters
- repeat steps 3 and 4 until a single cluster is obtained or other stop criterion is satisfied

Divisive clustering

- establish the number of clusters
- intitialise the centre of each cluster
- determine a data separation
- re-computer the centre of each cluster
- repeat steps 3 and 4 until the partition is unchanged (algorithm converges)

How the algorithm takes into account the attributes (features)

- monotonic → attributes are taken into account one by one
- polytonic→ attributes are simultaneously taken into

How the data belong to clusters

Exact clusterning (hard clustering)

- each instance has associated a label

Fuzzy clustering

- associates to each input xi a degree (probability) of membership fij to a certain class cj → an instance that can belong to several clusters

Algorithms

- agglomerative hierachical clustering
- k-means
- ama
- probabilistic models
- nearest neighbour
- fuzzy
- artificial neural network
- evolutionalry algorithms
- aco

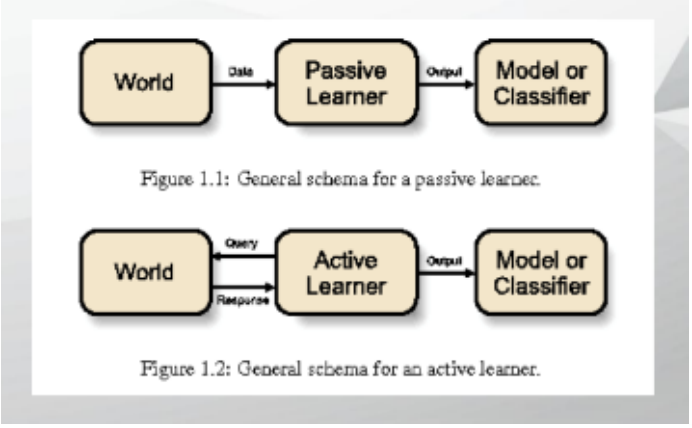
Intelligent systems with active learning

Definition

- The algorithm can receive supplementary information during the learning step in order to improve its performance

Example

- example: what is the subset of the training data that will facilitate the learning process



Intelligent systems with reinforcement learning

Aim

- learning, over a period of time, a course of action (behavior) that maximizees long-term rewards (earning)

Characteriscs

- interaction with the enviroment (actions → rewards)
- decision sequence

Problem's type

- ex: traing a dog (good or bad dog)

Supervised learning

- decision → consequence (malignant or benign cancer)

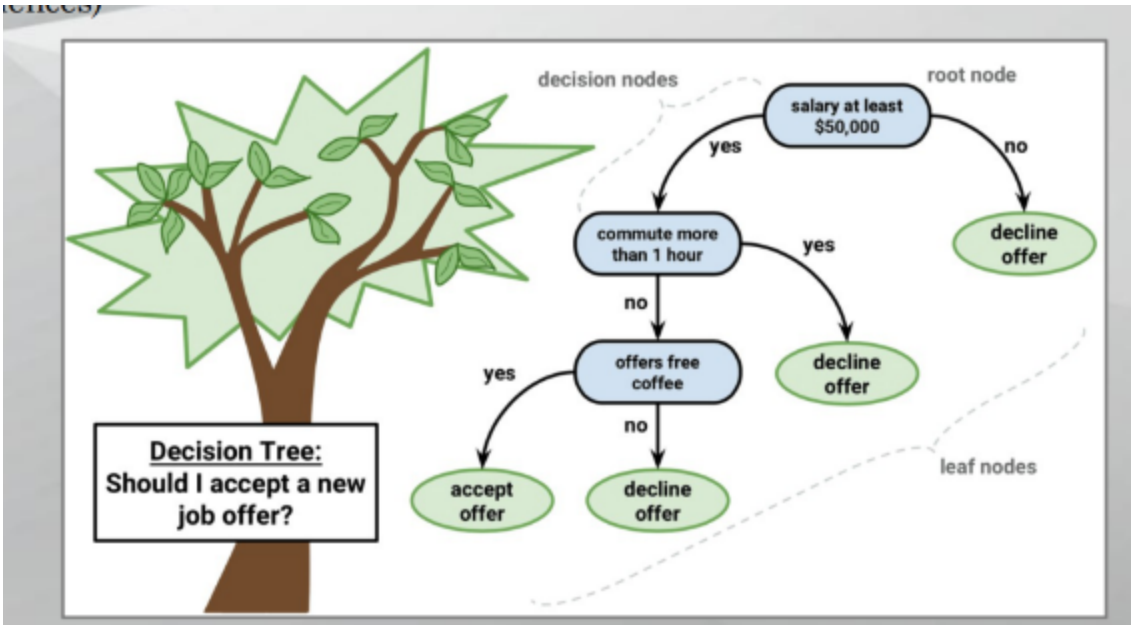
Deicision trees (DT)

Aim

- divide a collection or articles in smaller sets by successibly applying some decision rules → asing more question each question is addresed based on the answer of the previous question

Definition

- decision tree → a special graph (bi-color and oriented tree)
  - contains tree node types:
    - decision nodes → possibilities of decider (a test on an attribute of an item that must be classified
    - hazard nodes → random events outside of control of decider (exam results, therapy consequences)
    - result nodes → final states that have a utility or a label
  - decision and hazard node alternate on the tree levels
  - result nodes → leaf / terminal nodes
  - oriented edges of the tree of decisions (can be probabilistic)
- each internal node corresponds to an attributes
- each branch under a node (attribute) corresponds to the value of that attribute
- each leaf corresponds to a class



Problems solved with DT

Properties

- problem's instances are represented by a fixed number of attributes, each attribute having a finite number of values
- objective function takes discrete values
- DT represents a disjunction of more conjuctions, each conjunction being "attribute ai has value vj"
- training data could contain errors
- training data could be incomplete
    - some data don't have all attributes

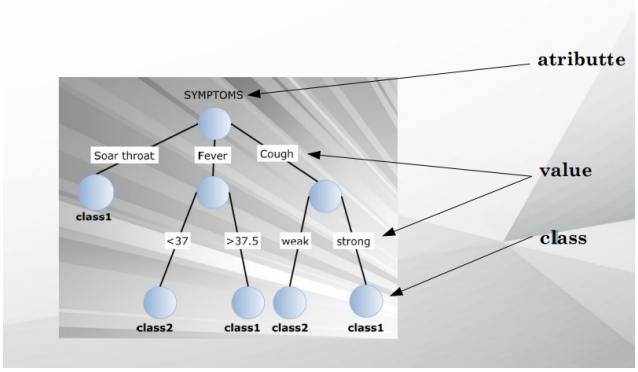
Classification problem

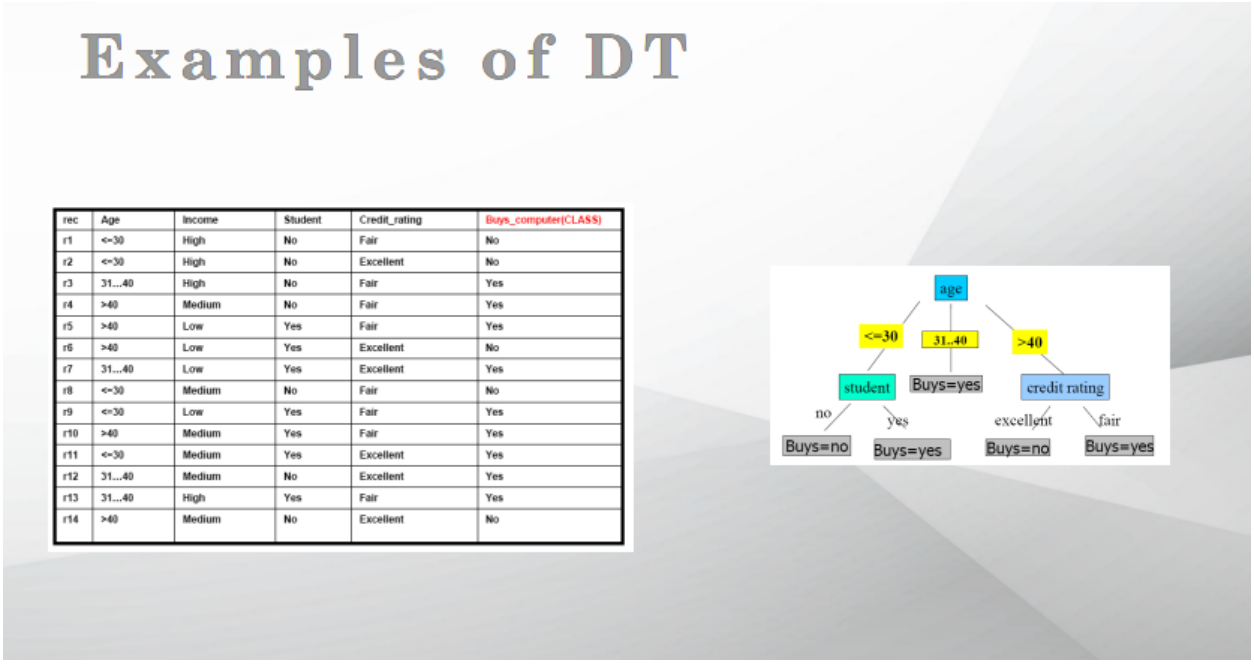
- binary classification
    - instances are [(attribute_ij, value_ij), class_i)], i=1,2,...,n, j=1,2,...n and class_i takes 2 values
- multi-class (k-class)
    - instances are [(attribute_ij, value_ij_, class_i], i=1,2,...,n, j=1,2,...,m and class_i taking k values

Regression problems

- instead of labeling each node by the label of a class, each node has associated a real value or a function that depends on the inputs of that node
- input space is split in decision regions by parallel cutting to Ox and Oy
- discrete outputs are transformed in continuous functions
- quality of problem solving
    - prediction error (square or absolute)

Example of DT





rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

Example of train data for DT

Credits

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Tree construction (induction)

- based on training data
- works bottom-up or top-down (splitting)

Split the training data into subsets based on the charactheristics of data

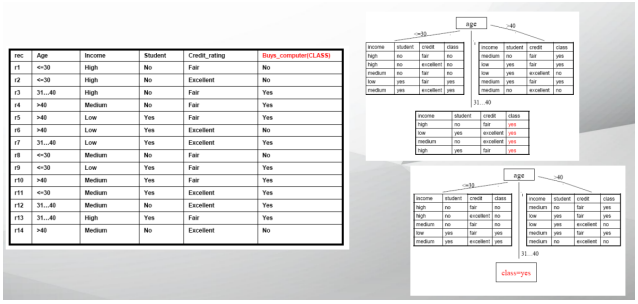
- a node - question related to a property
- branches of a node - possible answers to the question of the node
- initially, all examples are located in the root
  - an attribute gives the root label and its values give the branches
- on the next levels, examples are partitioned based on their attributes → order of attributes
  - for each node, an attribute is (recursively) chosen - its values → branches
- splitting - greedy decision making

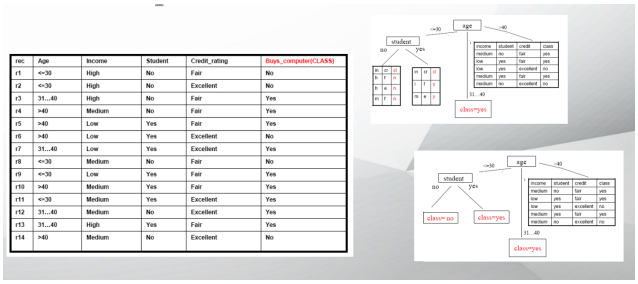
Iterative process

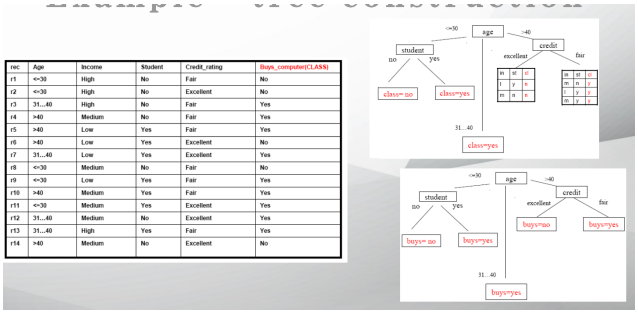
- stop conditions

- all examples from a node belong to the same class → node is a leaf and is labeled class_i
- there are no left examples → node becomes a leaf and is labeled by the majority class of training data
- there are no attributes

Example







ID3 / C4.5 algorithm

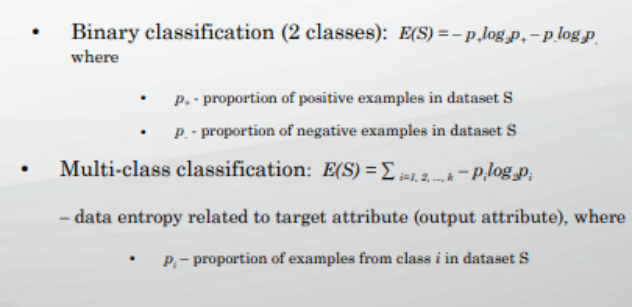
```
generate(D,A) //D- a partioning of training data, A - list of at
{
```

```
    create a new node N
    if examples from D belong to a single class C then
        node N becomes a leaf and is labeled C
        return node N
    else
        if A == empty then
            node N becomes a leaf and is labeled by majority cla
            return node N
        else
            separation_attributes = AttributeSelection(D,A)
            label node N by separation_attribute
            for all possible values vj of separation_attribute
                let Dj - set of examples from D that have separa
                if Dj == empty then
                    add a leaf (to node N) by majority class of
                else
                    add a node (to node N) return by generate(Dj
            return node N
}

AttributeSelection(D,A)
{
    selection the attribute that corresponds to a node (root or
    - random
    - attribute with fewest/most values
    - based on a preestablished order
    - information gain
        - gain rate
        - distance between partitions created by the attribute
```

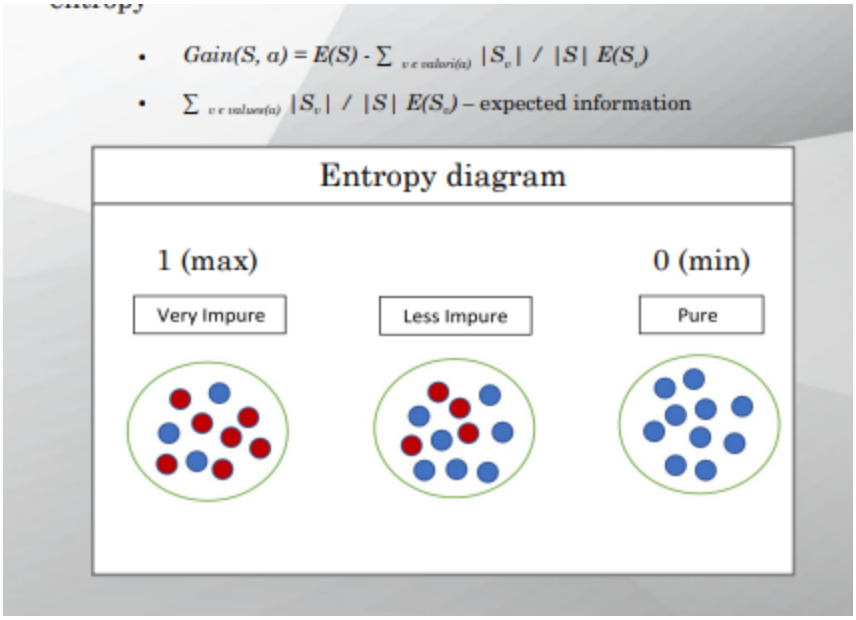
Information gain

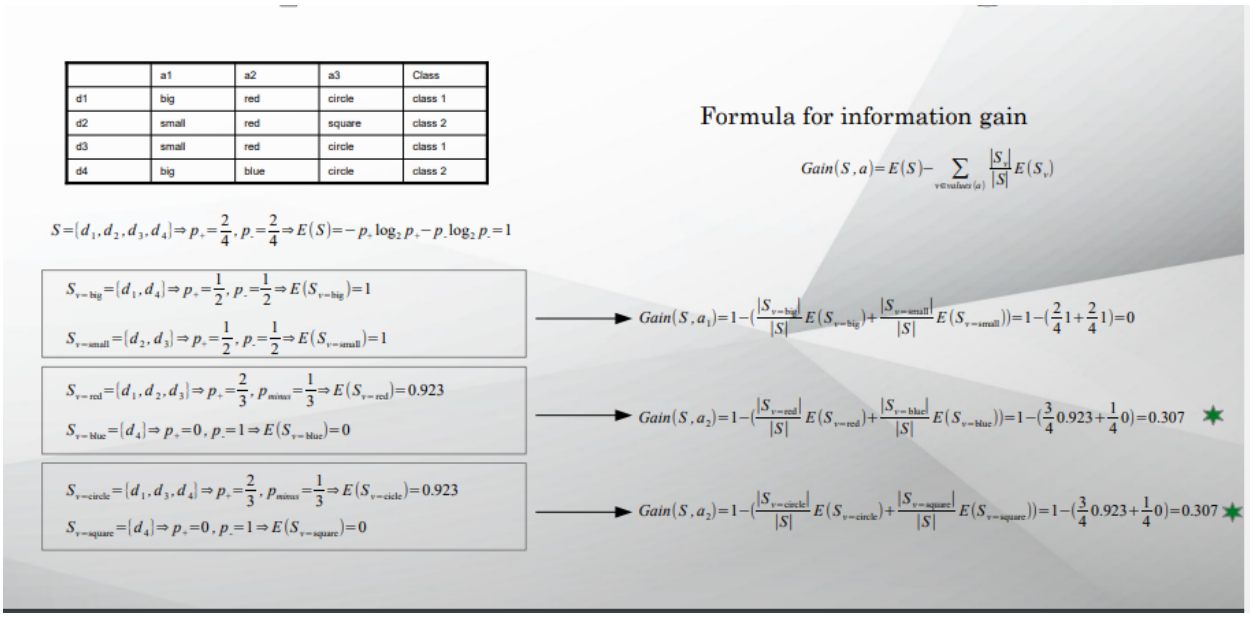
- an impurity measure
  - 0 (minimum) - if all examples belong to the same class
  - 1 (maximum) - if all examples are uniform distributed over classes

- based on data entropy
  - expected number of bits required by coding the class of an element from data



- information gain of an attribute
  - how the elimination of attribute a reduces the dataset's entropy

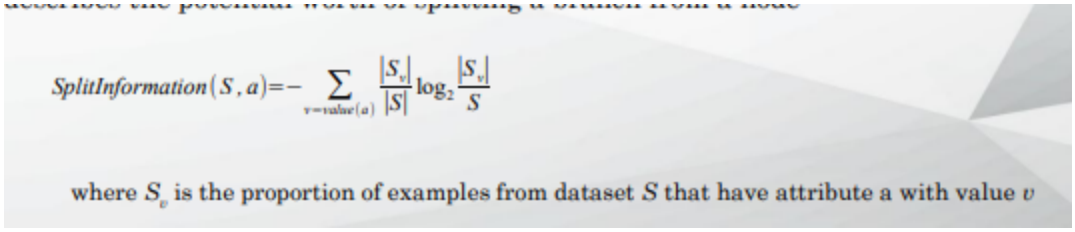




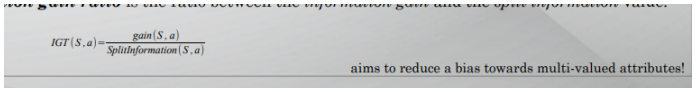
	a1	a2	a3	Class
d1	big	red	circle	class 1
d2	small	red	square	class 2
d3	small	red	circle	class 1
d4	big	blue	circle	class 2

Gain rate

- penalises an attribute by integrating a new term that depends on spreading degree and uniformity degree of separation - split information
- Split information
  - entropy related to possible values of attribute a
  - describes the potential worth of splitting a branch from a node

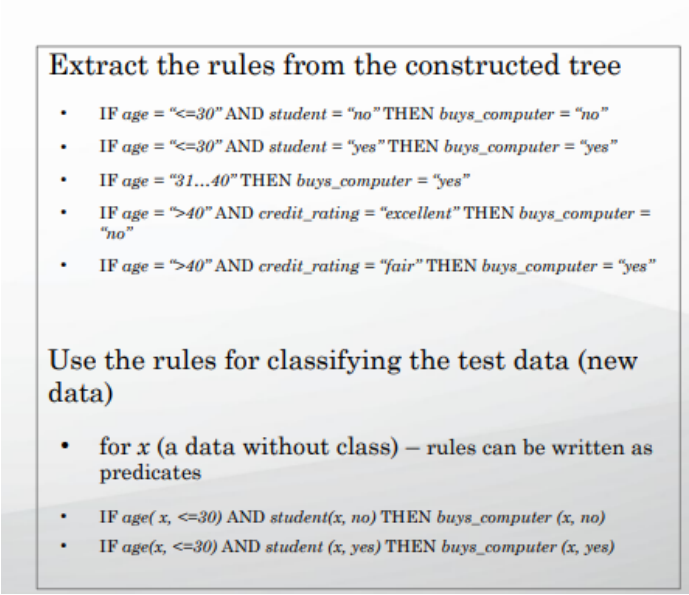


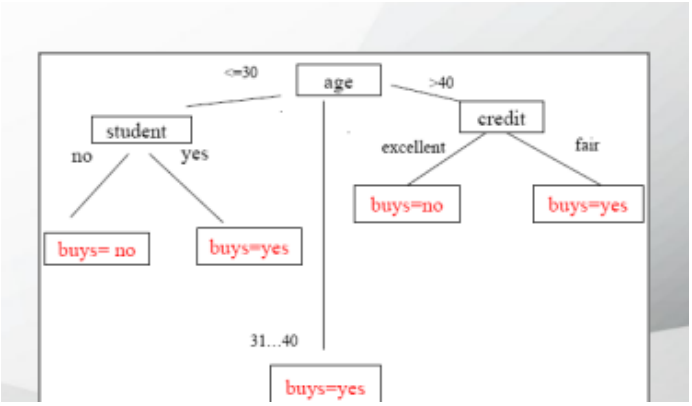
- information gain ration is the ratio between the information gain and the split information value



Using the tree as a problem solver

- all decisons performed along a path from the root to a leaf from a rule
- rules from dt are used for labeling new data





Difficulties & Solutions

- underfitting - DT constructed on training data is too simple → large classification error during training and testing
- overfitting - Dt constructed on training data matches the training data, but it cannot generalize new data
- solution
  - prunning - remove some branches (not useful, redundant) → smaller tree

    - cross-validation

Pruning

- identify and move/eliminate branches that reflect noise or exceptions
- pre-prunning
    - increasing the tree is stopped during construction by stopping the division of nodes that become leaf labeled by majority class of examples from that node
- post-prunning
    - after the Dt is constructed, eliminate the branches of some classes

Advantages & Disadvantages

Advantage

- easy to understand and interpret
- can use nomial or categorized data
- decision logic can be easy followed (rules are visibile)
- works better with large data

Disadvantages

- Instability due some changes to the training data
- complexity due to the representation
- difficult to use
- the dt construction is expensive
- the dt construction requires a lot of information

ANN

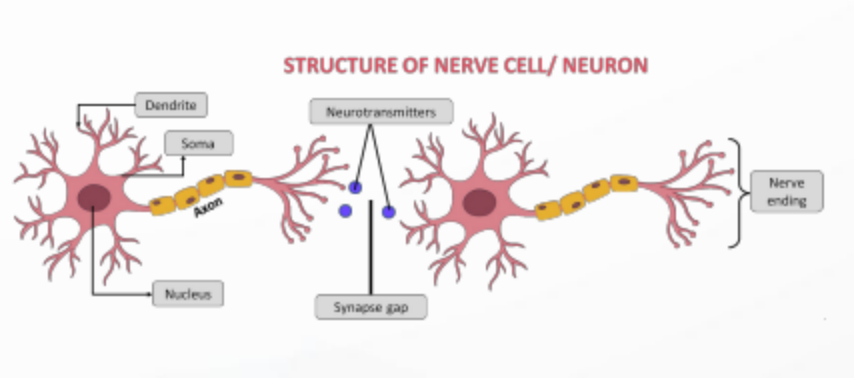
Aim

- binary classification for any input data (discrete or continous)
  - data can be separated by:
    - a line → ax+by+c=0 (if m=2)
    - a plane → ax + by + cz + d = 0 (if m=3)
    - a hyperplan → sum(a_i*x_i) + b = 0 (if m>3)
  - how do we identify the optimal values of a,b,c,d,ai?
    - artificial neural networks (anns)
    - suport vector machines (svms)

Why

- some tasks can be easily done by humans, but they are difficult to be encoded as algorithsm
  - shape recognition
    - old friends
    - handwritten
    - voice
  - rational proceeses
    - car driving
    - piano playing
    - basketball palying
    - swmming
- such tasks are too difficult to be formalized and done by a rational process

The learning process (brain)

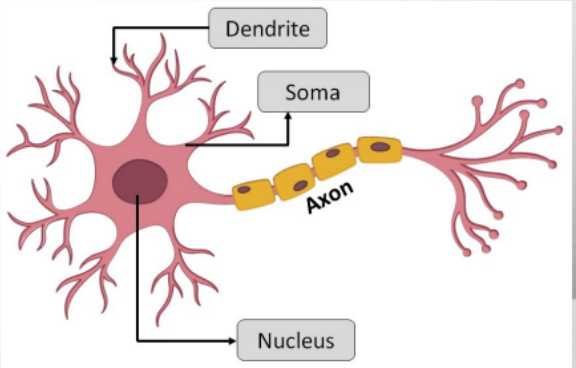


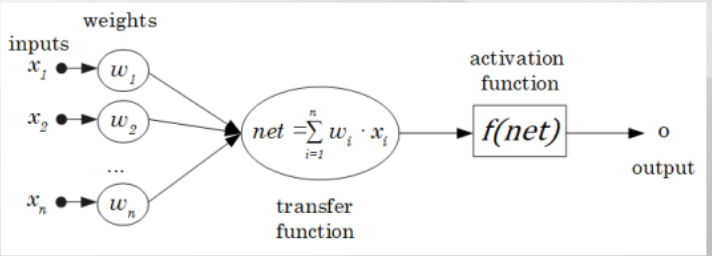
- human brain : ~ 10.000.000.000 of neurons connected through synapses
- a neuron
  - has a body (soma), an axon, and more dendrites
  - can be in a given state
    - active - if the input information is over a given stimulartion threshold
    - passive - otherwise
- synapse
  - link between the axon of a neuron and the dendrites of other neurons
  - take part to information exchange between neurons
  - 5.000 connections / neuron (average)
  - durint a life, new connections can appear
- brain → neural network
  - complex system, non-linear and parallel that processes information
  - information is stored and processed by the entire network, not only by a part of the network
- model for information processing:
  - learning
  - storing
  - memorizing

- learning
  - a basic characterstic
  - useful connections become permanent (others are eliminated)
- memory
  - short time memory
    - immediately → 30s
    - working memory
  - long term memory
    - capacity
    - increasing along life
    - limited → learning a poetry strophe by strophe
    - influenced by emotional states

Biology vs artificial

BNN	ANN
Soma	Node
Dendrite	Input
Axon	Output
Activation	Processing
Synapse	Weighted connection

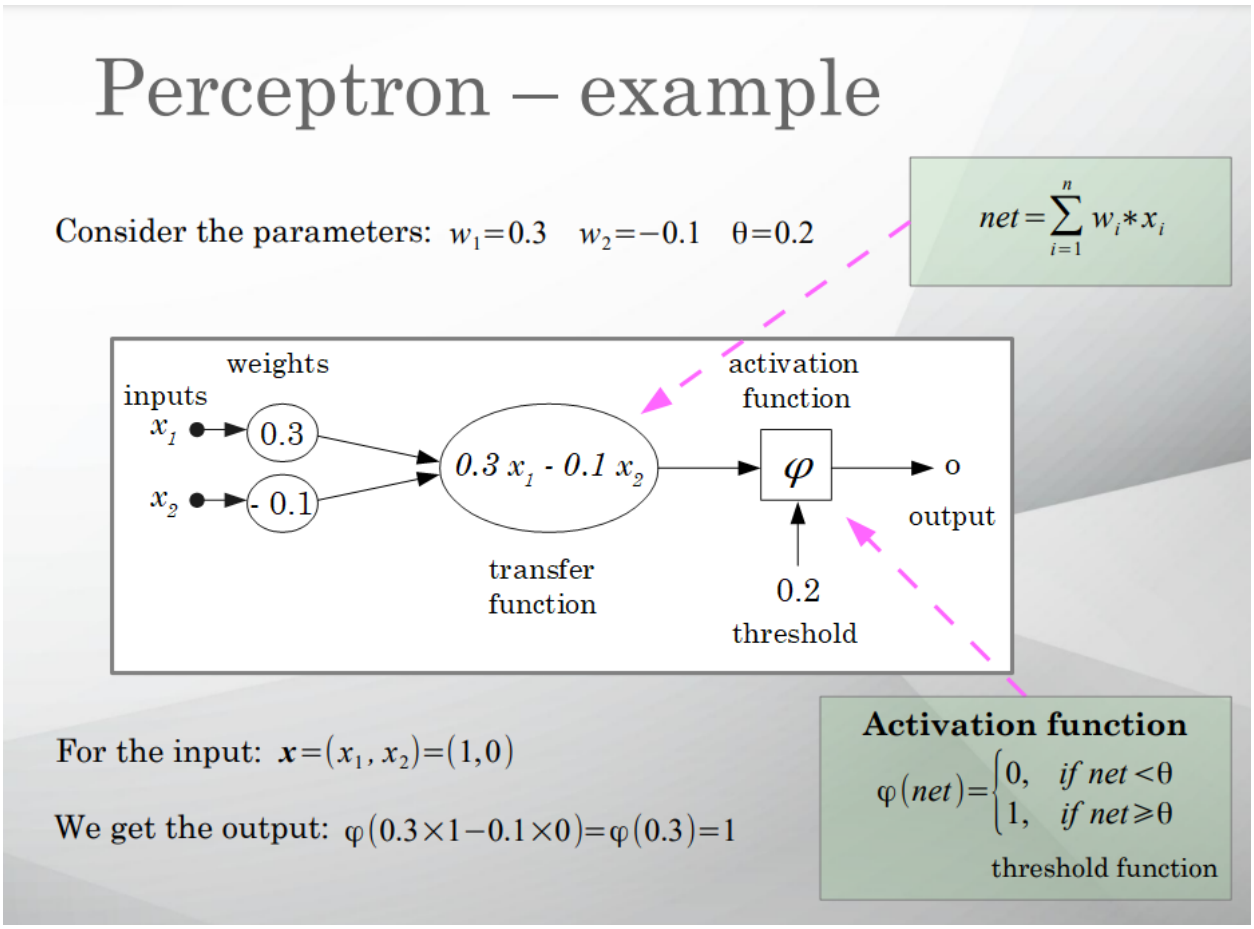




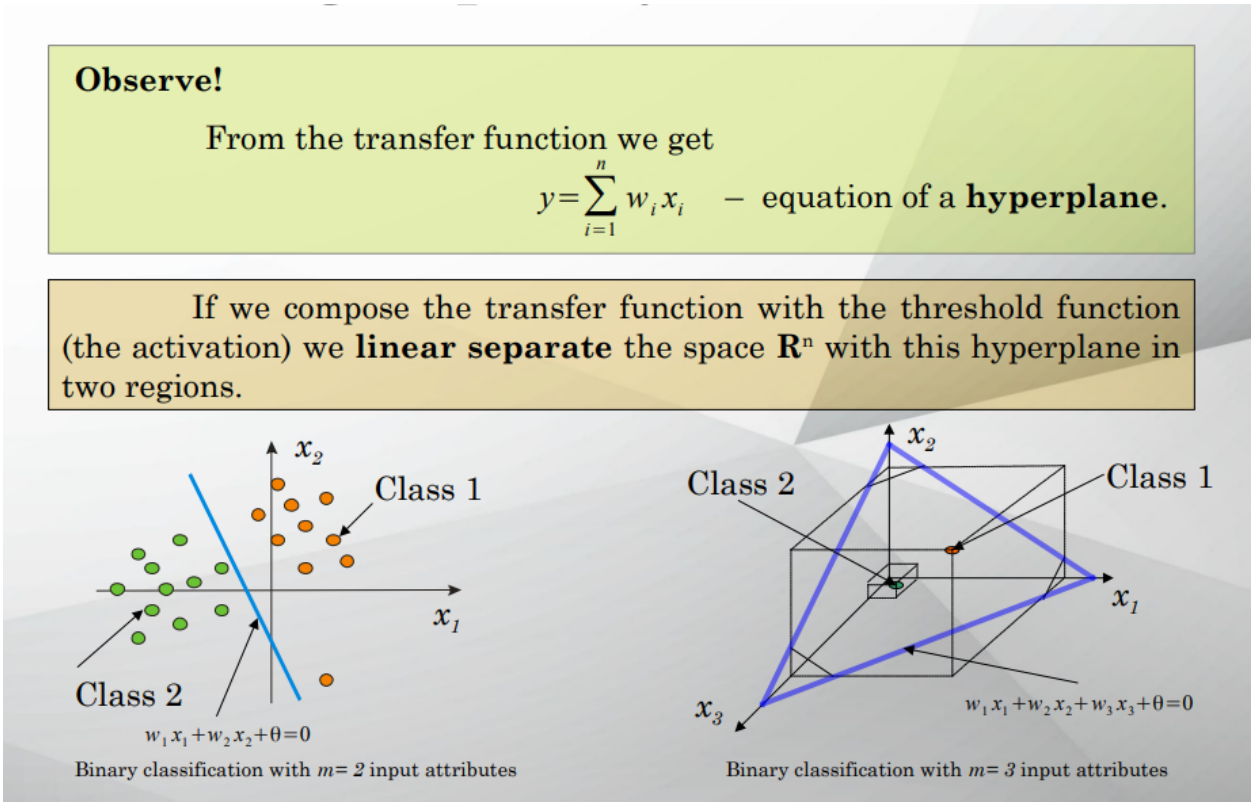
The Perceptron

Info

- it is the first model of a neuron
- invented by Frank Rosenblatt (1928-1971)
- the original preceptron was designed to take a number of binary inputs and produce one binary output
- uses different weights to represent the importance of each input, and that the sum of the values should be greater then a threshold value before making a decision like true or false
- the original algorithm
    1. set a treshold value
    2. multiply all inputs with its weights
    3. sum all results
    4. activate the output

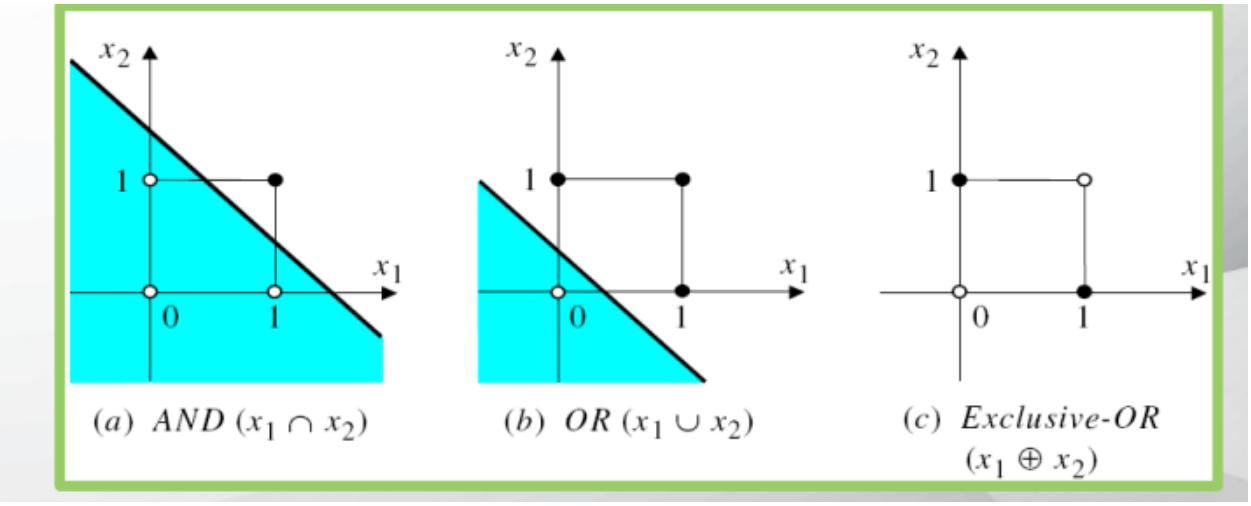


Solving capacity



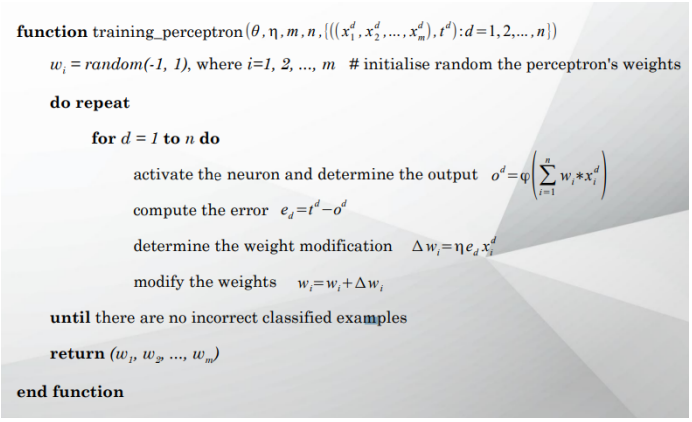
Limits

- non-linear separable data can not be classified
- a perceptron can learn AND and OR operations, but it cannot learn XOR operations (it is not linear separable)
- solutions
  - neuron with continuous threshold
  - more neurons



Learning

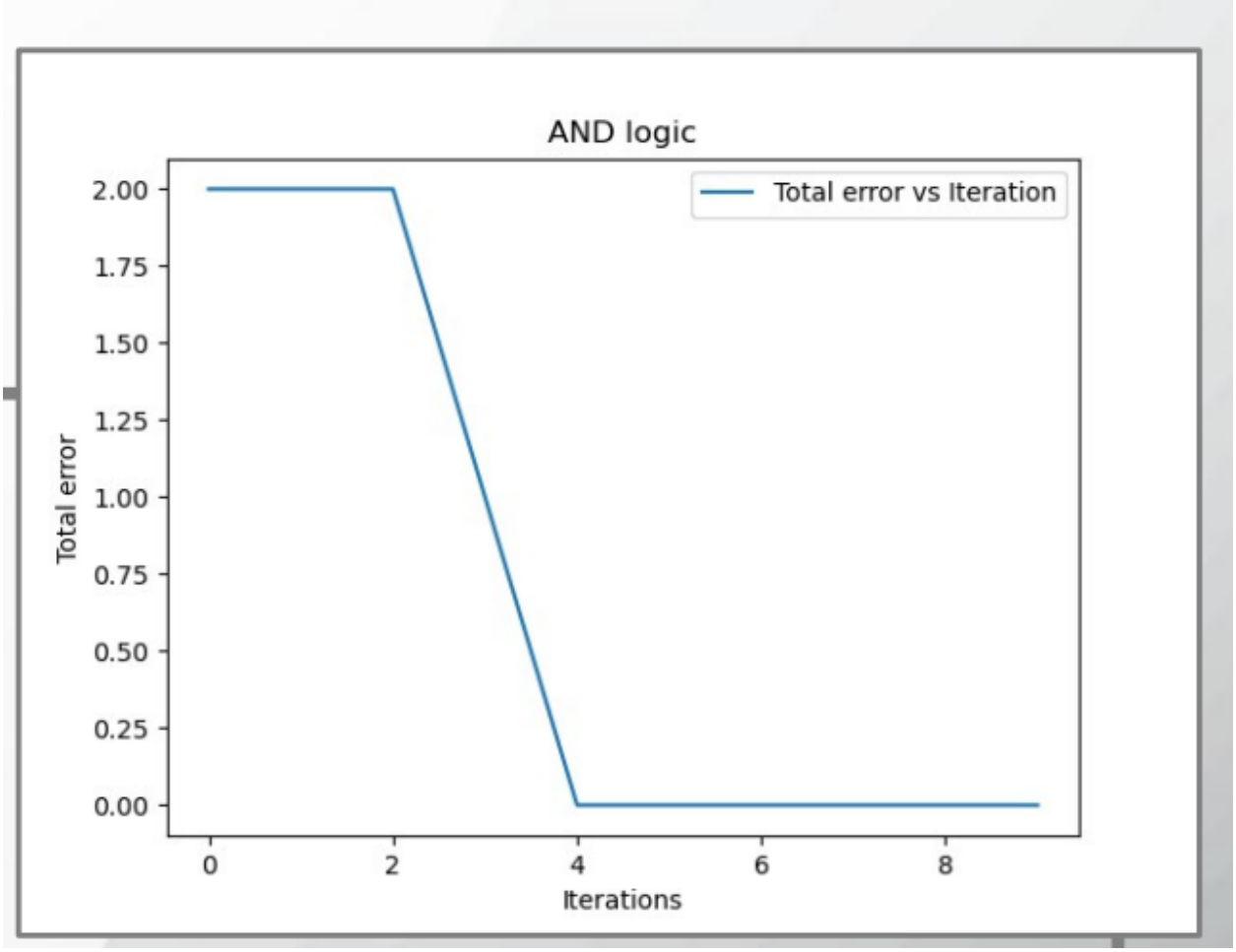
- aim: to identify the optimal weights (w1,w2,...wm) by minimising the errors
- training the data set of n data with m - number of attributes
- the error is the difference between the real output y and the output o computed by the perceptron for the input
- based on error minimisation associated to an instance of train data
- modify the weights based on error associated to an instance of train data




```
from numpy import random , dot, array
import matplotlib as mpl

def threshold(x):
```

```
    if x < 0.2 :
        return 0
    return 1

class Perceptron:
    def __init__ (self, noInputs, activationFunction, learningRa
        self.noInputs = noInputs
        self.weights = random.rand(self.noInputs)
        self.activationFunction = activationFunction
        self.output = 0
        self.learningRate = learningRate
        self.errorVariation = []
    def fire(self, inputs):
        self.output = self.activationFunction(inner(array(inputs
        return self.output

    def trainPerceptronRule(self, inputs, output):
        totalError = 0
        for i in range(len(inputs)):
            error = output[i] - self.fire(inputs[i])
            delta = self.learningRate * array(inputs[i])
            self.weights += delta
            totalError += error**2
        self.errorVariation.append(total_error)


def test1():
# AND logic

p = Perceptron(2,threshold,0.1)
x = [[1,1],[1,0],[0,1],[0,0]]
t = [1,0,0,0]
noIterations = 10
for i in range(noIterations):
    p.trainPercetron(x,t)
print(p.weights)
```

```
for j in range(len(x)):
    print(x[j],p.fire(x[j]))
```



Neuron training

Perceptron's rule → Perceptron's algorithm

1. start by some random weights
2. determine the quality of the model create for these weights for a single input data
3. re-compute the weights based on the model's quality
4. repeat (from step 2) until a maximum quality is obtained

Delta's rule →> algorithm of gradient descent

1. start by some random weights
2. determine the quality of the model create for these weights for all input data
3. re-compute the weights based on the model's quality
4. repeat (from step 2) until a maximum quality is obtained

Differences	Perceptron's algorithm	Gradient descent algorithm (delta rule)
What does o^d represent?	$o^d=sign(\mathbf{w}x^d)$	$o^d=\mathbf{w}x^d$ or $o^d=sig(\mathbf{w}x^d)$
Convergence	After a finite # of steps (until the perfect separation)	Asymptotic (to minimum error)
Solved problems	With linear separable data	Any data (linear separable or non-linear)
Neuron's output	Discrete and with threshold	Continue and without threshold

Gradient descent

- based on the error associated to the entire set of training data
- modify the weights in the direction of the steeptest slope of error reduction E(w) for the entire set of train data

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{n}(t^d - o^d)^2$$

- how to determine the steepest slope? → derive E based on w (establish the gradient of error e)

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_m} \right)$$

- error's gradient is computed basd on the neuron's activation function (that function must be differentiable → continous)
- how are the weights modified?

$$? \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \text{ where } i = 1,2,\ldots,m$$

- linear function

$$f(net) = \sum_{i=1}^{m} w_i x_i^d$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial \frac{1}{2} \sum_{d=1}^{n} \left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^{n} \frac{\partial \left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^{n} 2\left(t^d - o^d\right) \frac{\partial \left(t^d - \mathbf{w}\mathbf{x}^d\right)}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \sum_{d=1}^{n} \left(t^d - o^d\right) \frac{\partial \left(t^d - w_1 x_1^d - w_2 x_2^d - \ldots - w_m x_m^d\right)}{\partial w_i} = \sum_{d=1}^{n} \left(t^d - o^d\right)\left(-x_i^d\right)$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^{n} \left(t^d - o^d\right) x_i^d$$

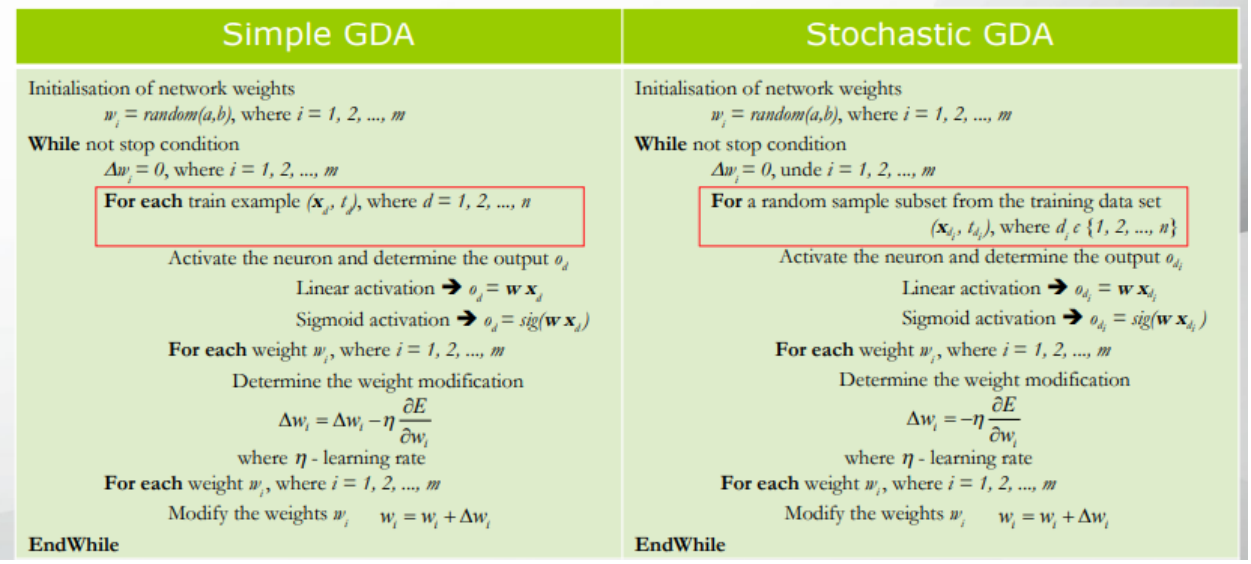
- sigmoid function

$$f(net) = \frac{1}{1+e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1+e^{-\sum_{i=1}^{m} w_i x_i^d}} \qquad y = s(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{\partial s(z)}{\partial z} = s(z)(1-s(z))$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial \frac{1}{2} \sum_{d=1}^{n} \left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^{n} \frac{\partial \left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^{n} 2\left(t^d - o^d\right) \frac{\partial \left(t^d - sig(\mathbf{w}\mathbf{x}^d)\right)}{\partial w_i} = \sum_{d=1}^{n} \left(t^d - o^d\right)\left(1-o^d\right) o^d \left(-x_i^d\right)$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^{n} \left(t^d - o^d\right)\left(1-o^d\right) o^d x_i^d$$

Simple and stochastic GDA



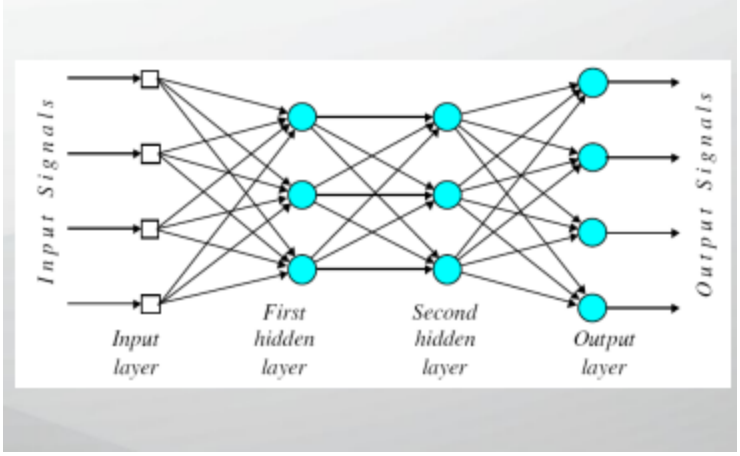
Common types of Neural Network Arhitectures

- FeedForwared Neural networks
- convolutional neural networks
- reccurect neural networks
- long shor-term memory networks
- autoencoders
- generative adversial networks

Feed Forward Artifical Neural Networks

- a structure similar to biological neural network
  - 1943, neurophysiologist Warrent McCulloch and mathematician Walter Pitts
  - the orginal ANN was built with electrical circuits

    - a set of nodes (units, neurons, processing elements) located in a graph with more layers
- in a feed forward network the information moves in only one direction, forward, from the input nodes through the hidden nodes (if any) to the output nodes
- the most simple type of ann
    - the percetrons are arranged in layers
        - the first layer is taking inputs
        - the last layer is producing outputs
        - between them there are hidden layers
- the data flow goes in one direction
    - each perceptron is connected with every perceptron on the next layer
    - there is no connection between perceptrons in the same layer

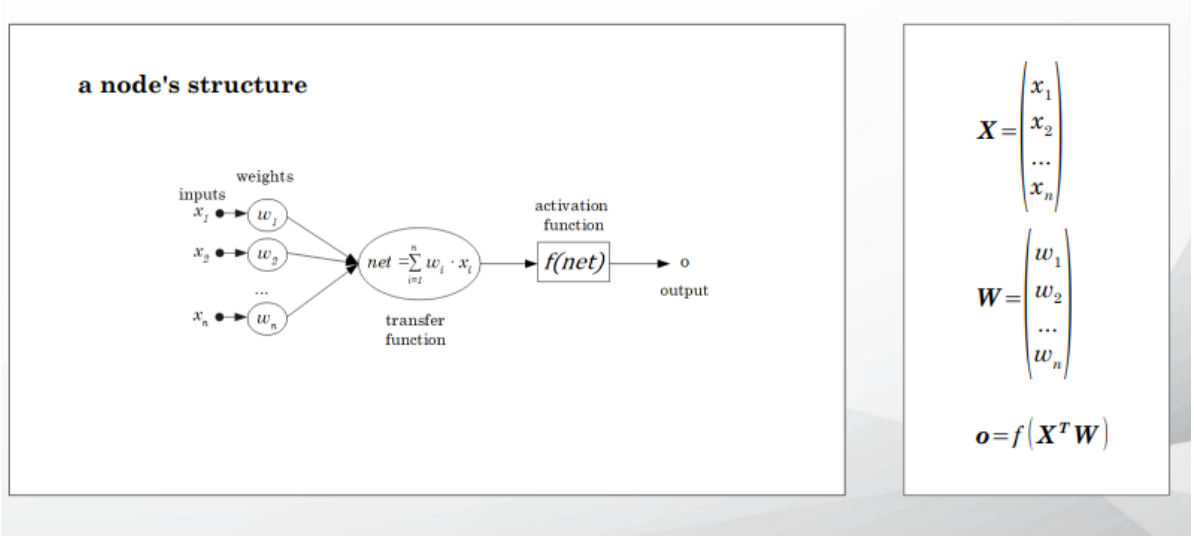


Structure

nodes

- have inputs and outputs
- perform a simple computing through an activation function
- connected by weighted links

- links between nodes give the network structure
- links influence the performed computations



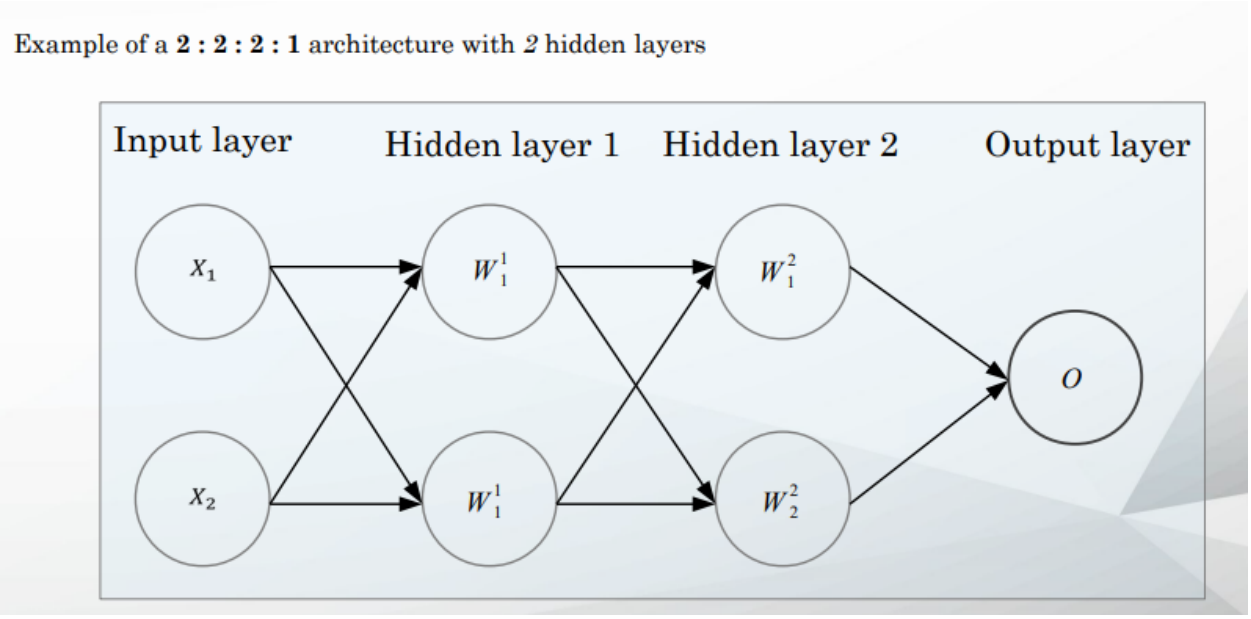
layers

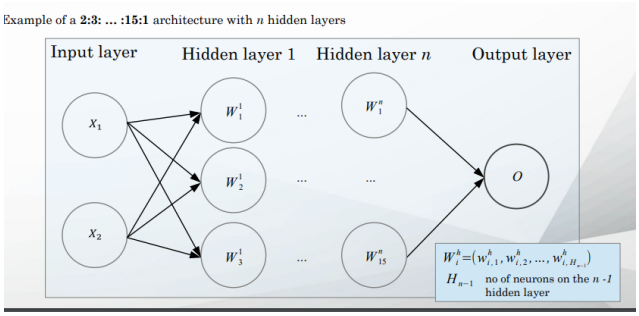
- input layer
  - contains m nodes (m - the number of attributes of a data)
- output layer
  - contains r nodes (r - the number of outputs)
- intermediate layers
  - different structures
  - different sizes

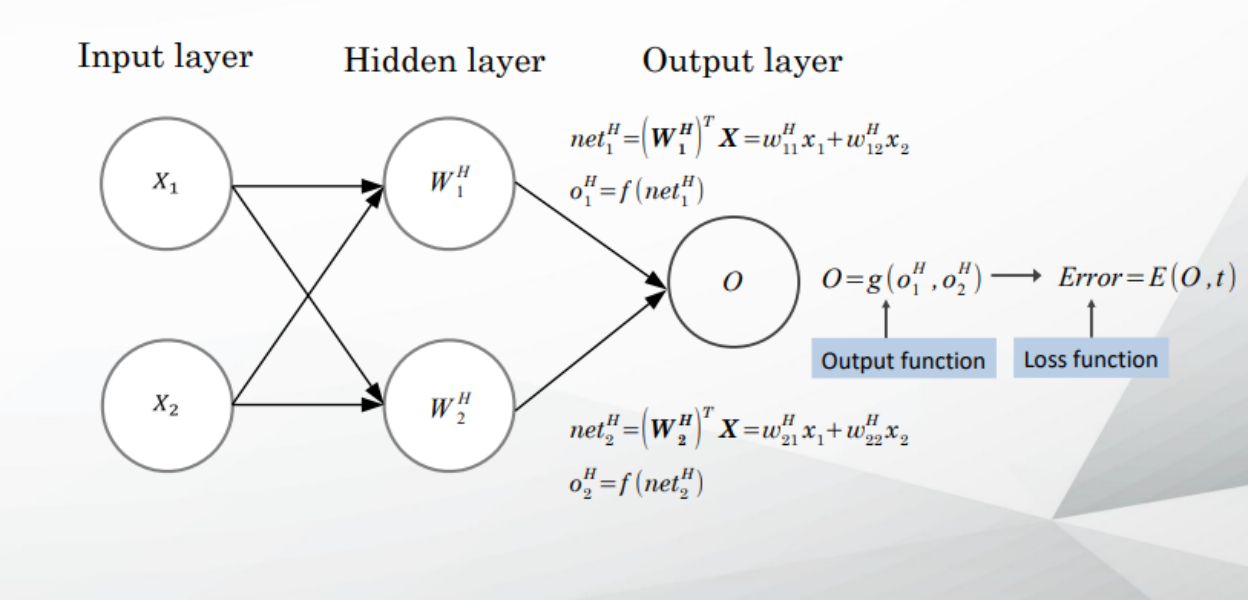
examples

- 6:4:10:2 → input layer with 6 units (artifical neurons), two hidden layers with 4 and 10 units and an output layer with 2 units
- this structure can be used for a problemm with 6 attributes and 2 outputs
- each node from the first layer has one input and one output, each unit from the first hidden layer has 6 weights and one output, each unit from the second

layer has 4 weights and one output, each node from the output layer has 10 weights and one output







The learning process

- a training data set of n data where m - number of attributes, r- number of outputs
- form an ANN with m input nodes, r output nodes and an internal structure
  - some hidden layers, each layer having a given number of nodes
  - with weigthed connections between every 2 nodes of consecutives layers
- determine the optimal weights by minimising error
  - difference between the real output y and the output computed by the network

Problems solved by an ANN

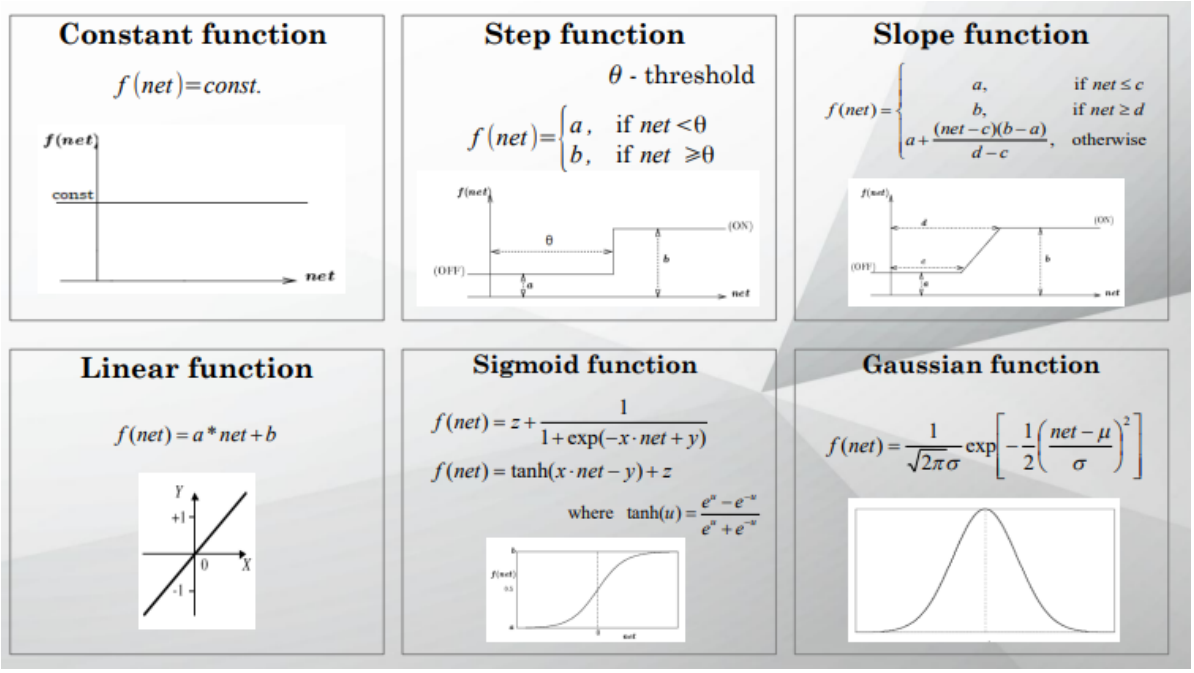
- problem data can be represented by pairs (attribute-value)
- objective function can be:
  - single or multi-criteria
  - discrete or continous
- training data can be noised
- a large training time

Neuron processing

- information is transmitted to the neuron → compute the weighted sum of inputs

$$net = \sum_{i=1}^{n} w_i * x_i$$

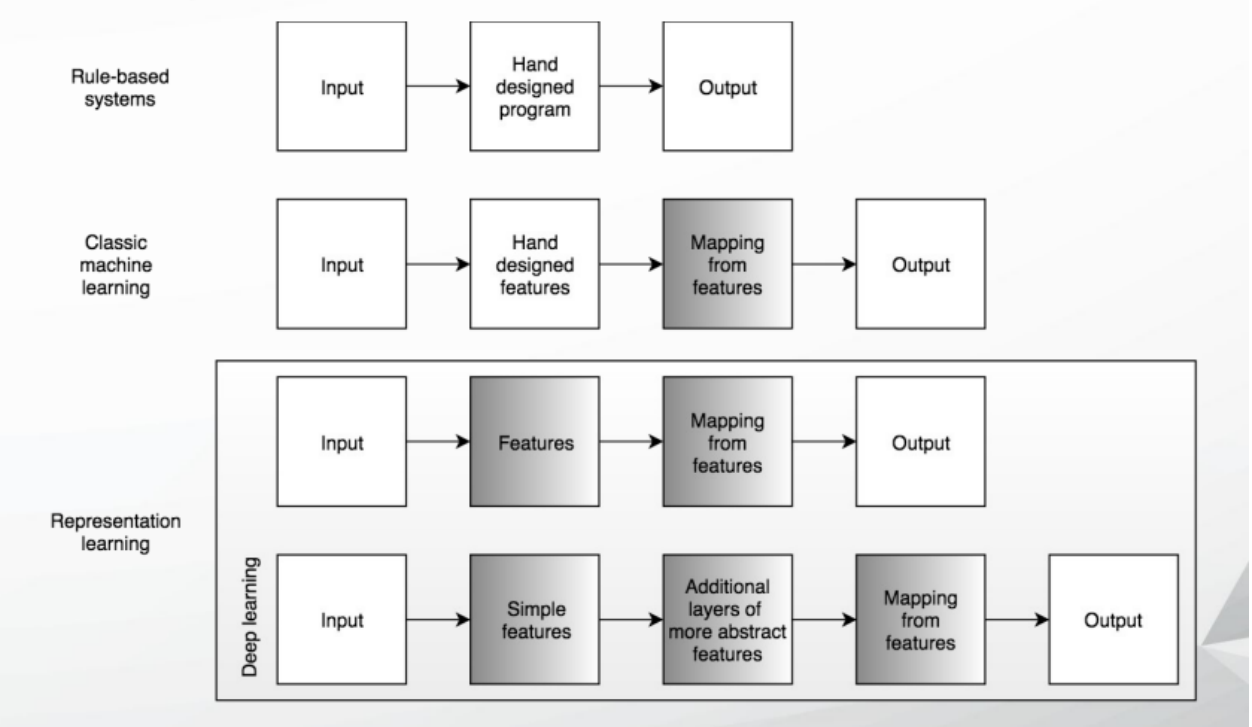
- neuron processed the information → by using an activation function



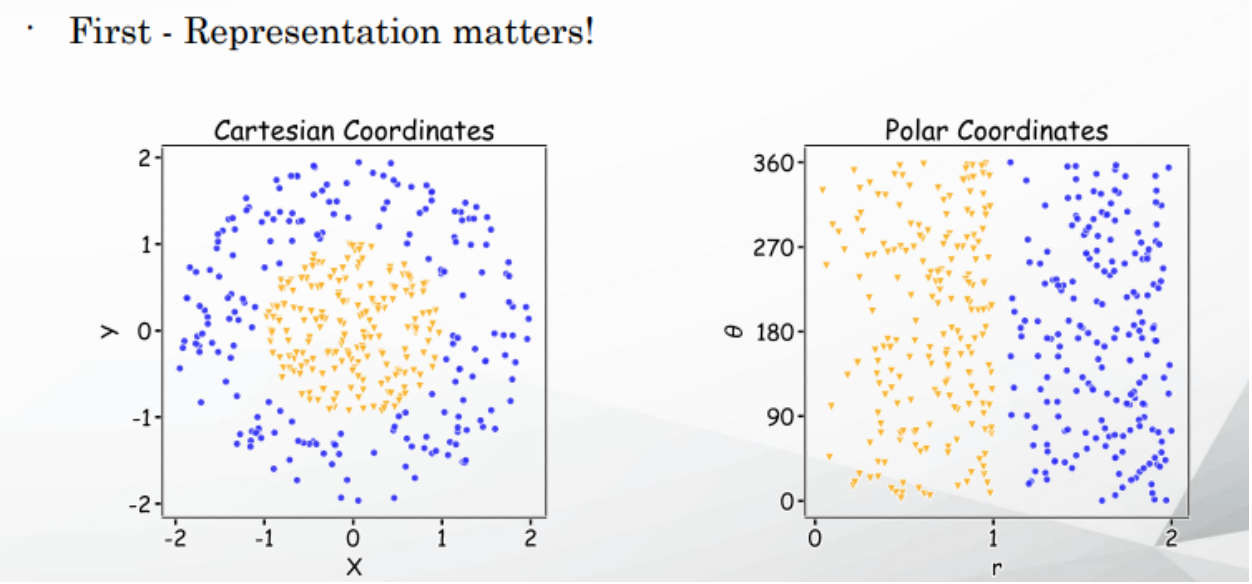
Features

- with more perceptrons that are independent of each other in the hidden layer, the points are classified in more pairs of linearity separable regions, each of it having a unique line separating the region
- by varying the number of nodes in the hidden layer, the number of layers, and the number of input and output nodes, one can classification of points in arbitrary dimension into an arbitrary number of groups
- hence feed-forward networks are commonly used for classifications

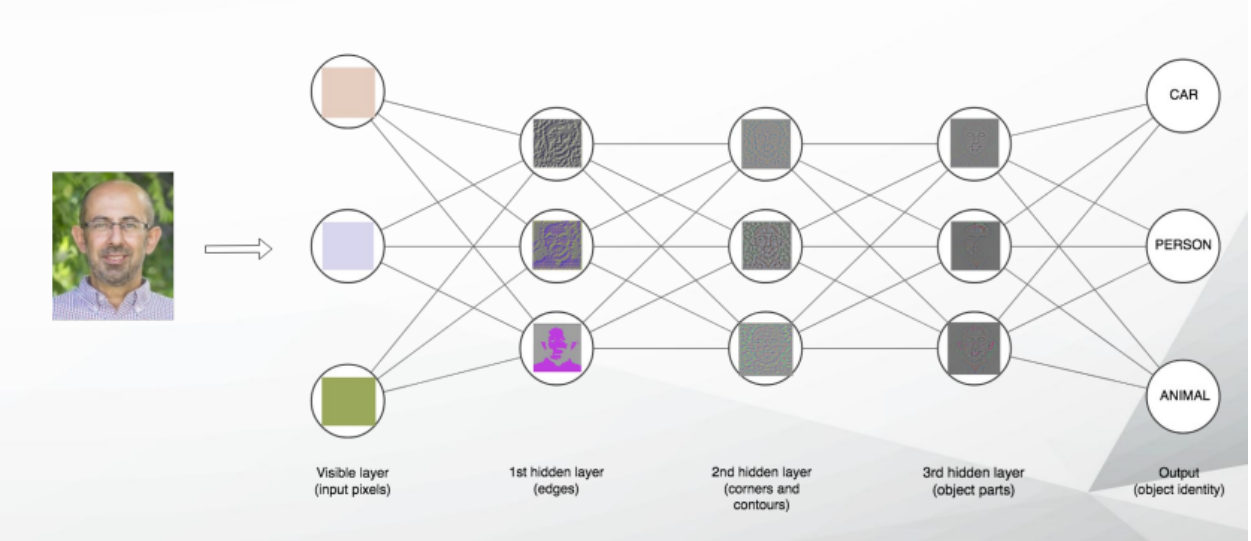
Learning multiple components



Representation



Depth - repeated compositions



Traditional ANNs (like feed-forward

- manually engineer
  - domain specific, enormous human effort
- generic transform
  - maps to a higher-dimensional space
  - kernel methods : RBF kernels
  - over fitting: doesn not generalize well to test set
  - cannot encode enough prior infomration

Design choices

Activation function

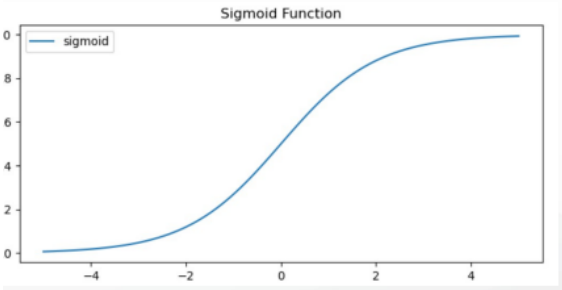
- linearity vs non-linearity
  - linear models
    - can be fit efficienly (via convex optimization)
    - liminted model capacity
  - alternative

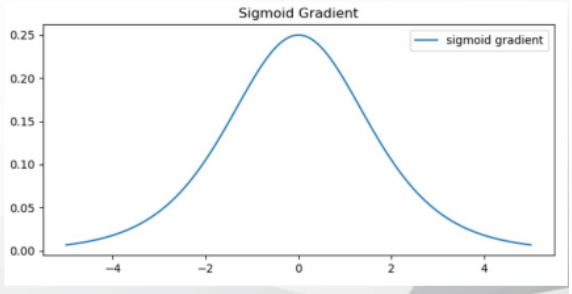
$$f(\boldsymbol{x}) = W^T \phi(\boldsymbol{x})$$

- Where $\phi(\boldsymbol{x})$ is a *non-linear transform*

- the activation function should
  - provide non-linearity
  - ensure gradients remain large through hidden units
- common choices
  - sigmoid

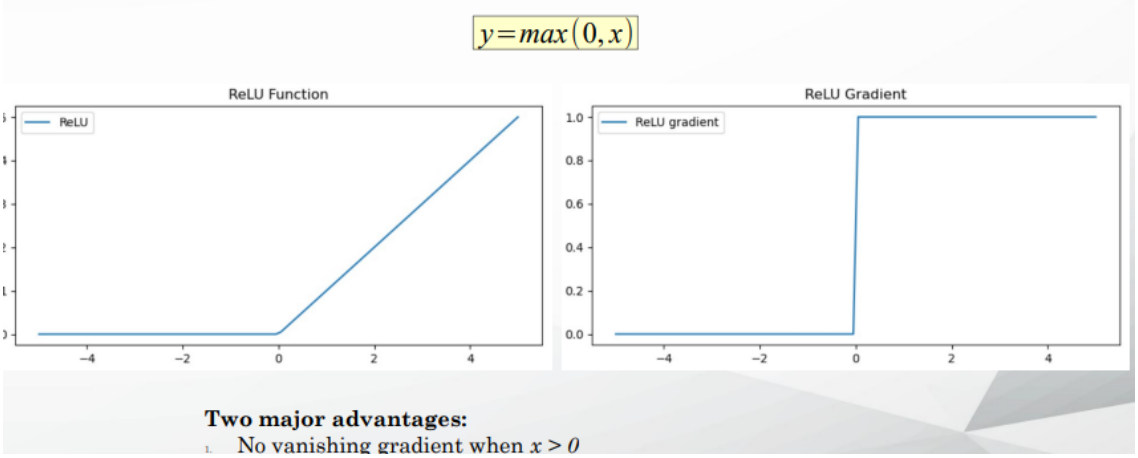
$$y = \frac{1}{1+e^{-x}}$$



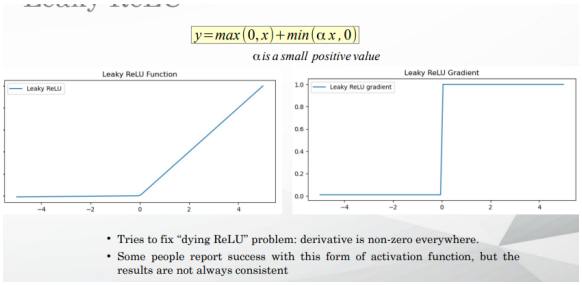


Derivative is **zero** for much of the domain. This leads to "vanishing gradients" in backpropagation.

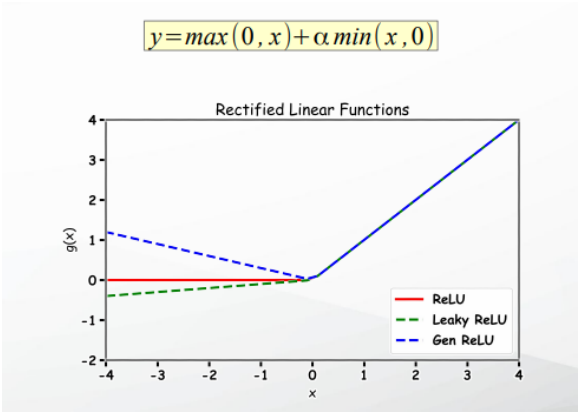
  - relu



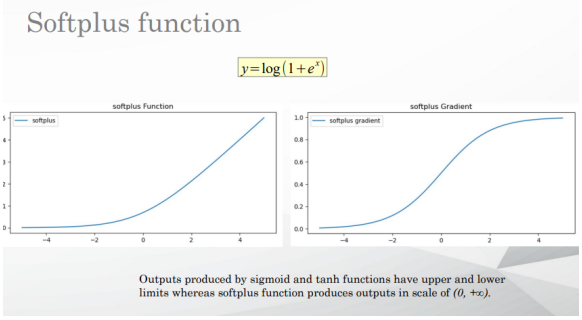
- leaky relu



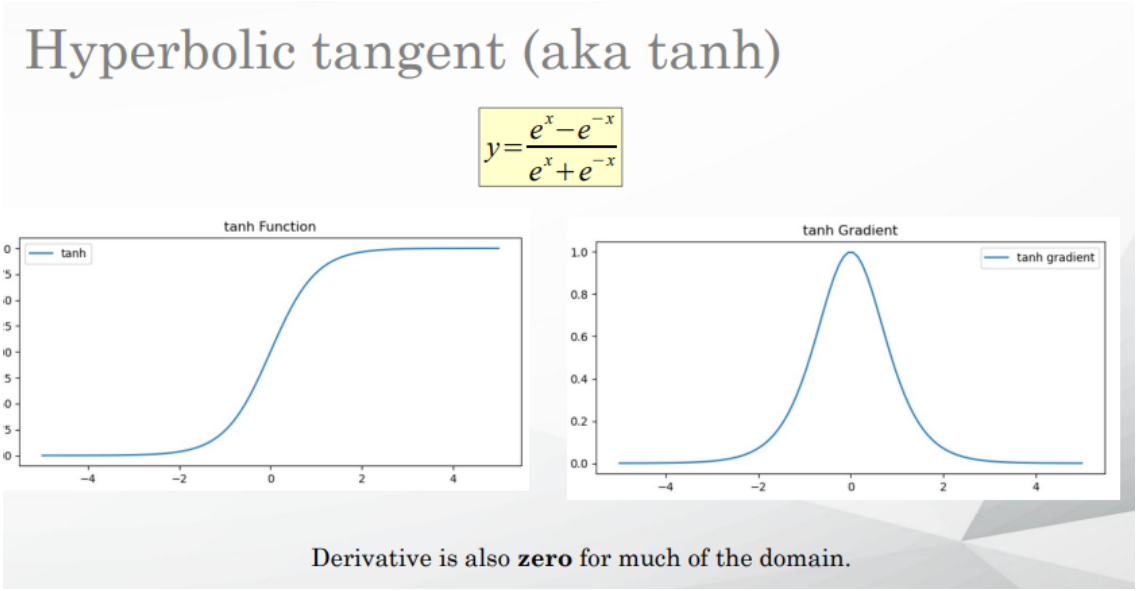
- generalized relu



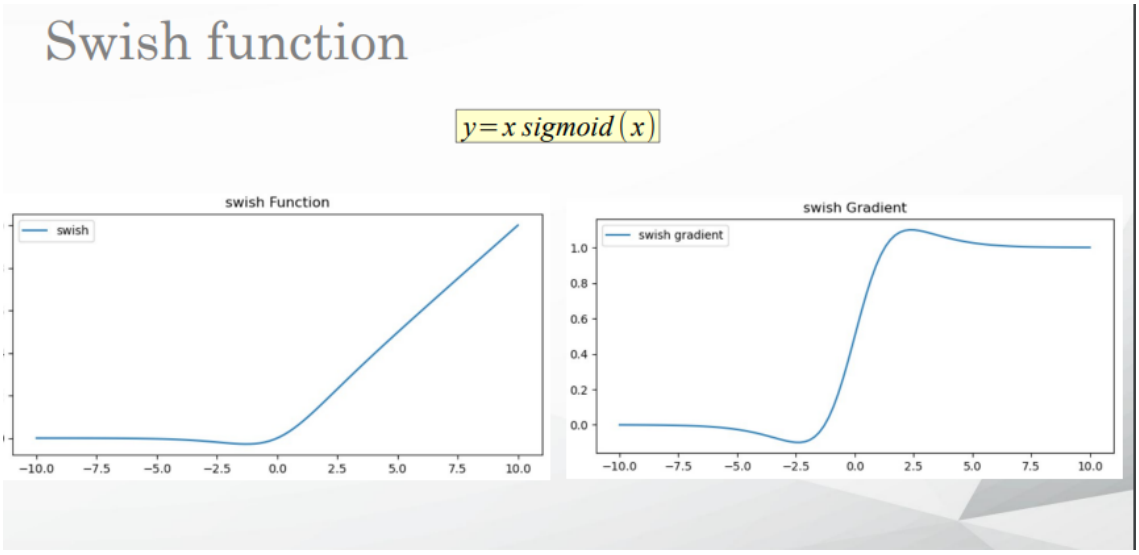
- softplus


Softplus function
$y=\log(1+e^x)$
softplus Function
softplus
softplus Gradient
softplus gradient
Outputs produced by sigmoid and tanh functions have upper and lower limits whereas softplus function produces outputs in scale of (0, +∞).

- tanh


Hyperbolic tangent (aka tanh)
$y=\frac{e^x-e^{-x}}{e^x+e^{-x}}$
tanh Function
tanh
tanh Gradient
tanh gradient
Derivative is also zero for much of the domain.

- swish


Swish function
$y=x\,sigmoid(x)$
swish Function
swish
swish Gradient
swish gradient

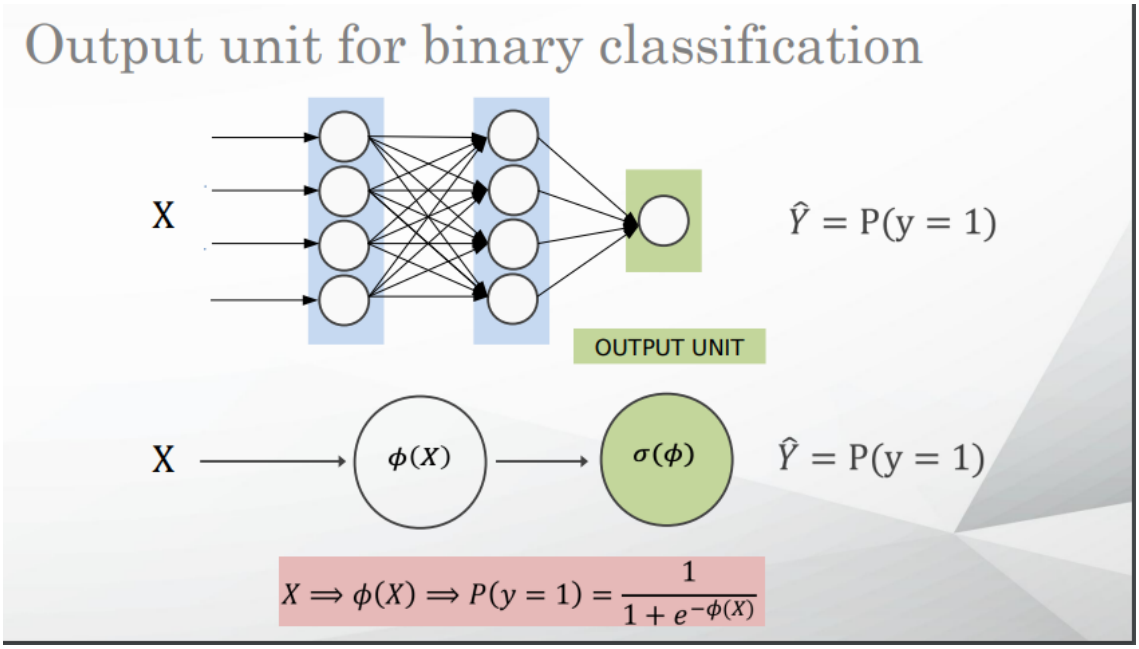
Loss functions

- likelihood for a given point
- assume independence - likelihood for all measurements
- maximize the likelihood, or equivalently maximize the log-likelihood
- turn all this into a loss function
- common loss function
    - mean absolute error loss (mae)
        - also called l1 loss
        - compute the average of the sum of absolute differences between
            - loss (x,y) = $|x-y|$
        - used for
            - regression problems
                - especially when the distribution of the target variable has outliers, such as small or big values that are a great distance from the mean value
    - mean squared error loss
        - alos called l2 loss
        - computes the average of the squared differences between actual values and predicted values
        - loss(x,y) = (x-y)^2
        - mse is the default loss function for most regression problems
    - negative log-likelihood loss
    - cross-entropy loss
        - is the difference between two probability distributions for a provided set of occurences or random variables
        - in the discrete setting, give two probability distributions p and q , their cross-entropy is defined as

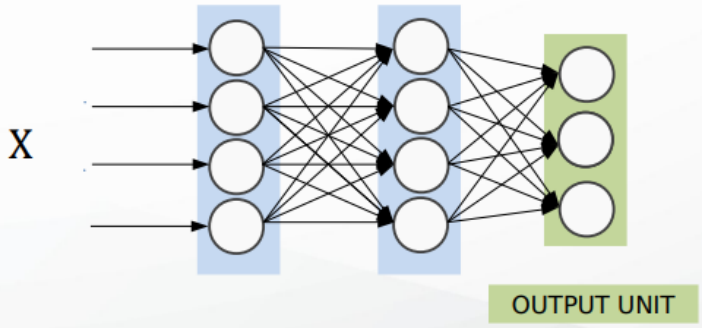
$$H(p,q) = -\sum_{x \in X} p(x) \log(q(x))$$

- to compute the loss we apply the cross-entropy operator between the desired output and to the real output of our model after we used the softmax operator on the output
- used in classifcations

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE
Continuous	Arbitrary	-	GANS



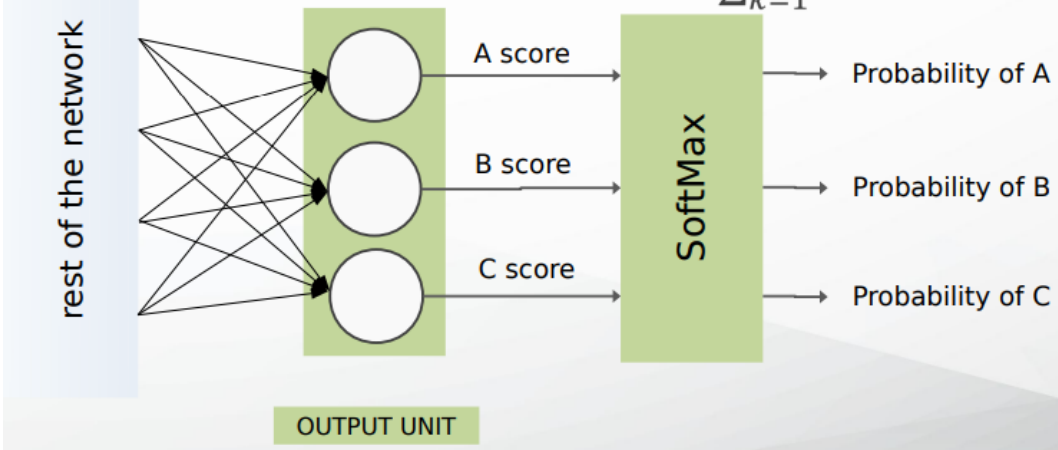
Output unit for multi class classification

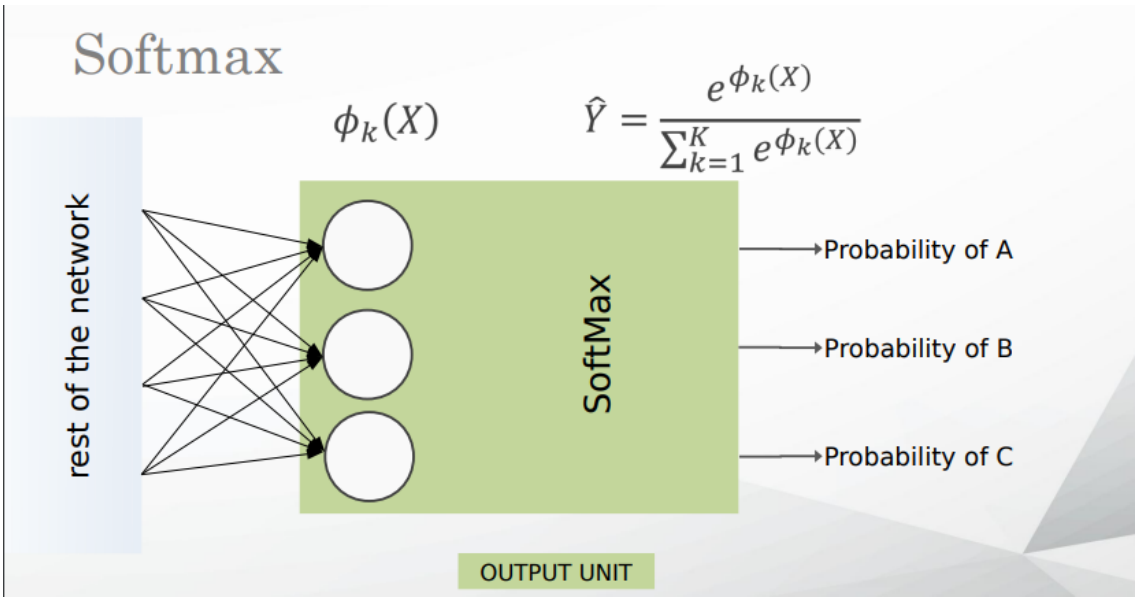


Softmax

$\phi_k(X)$

$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$

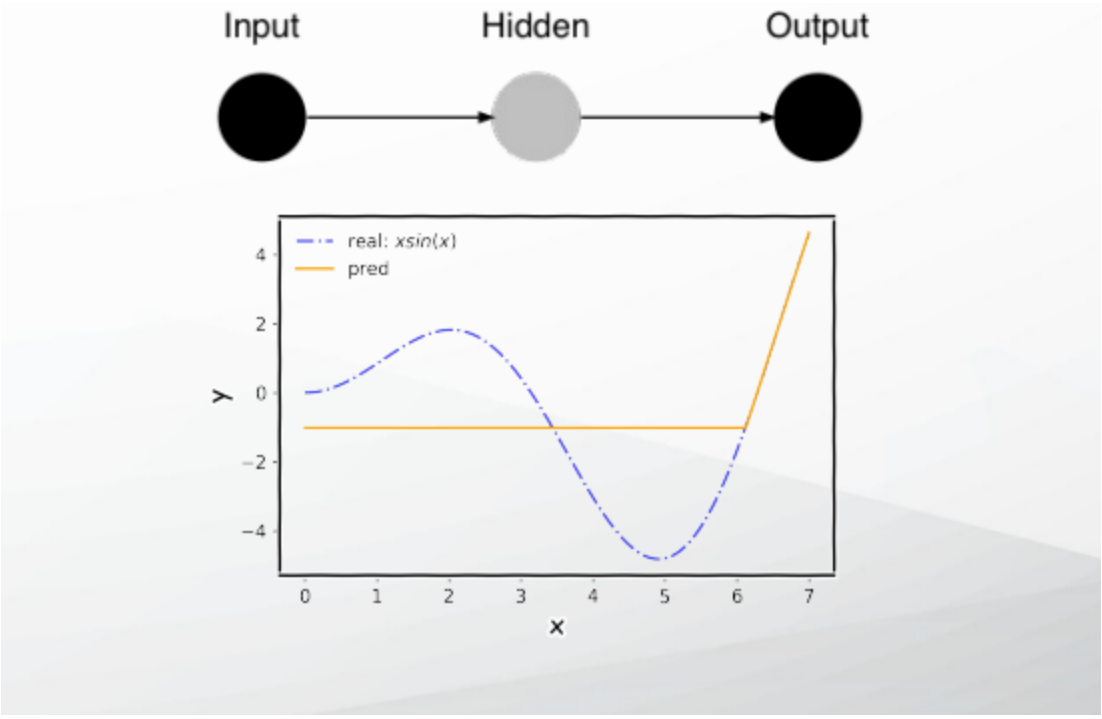


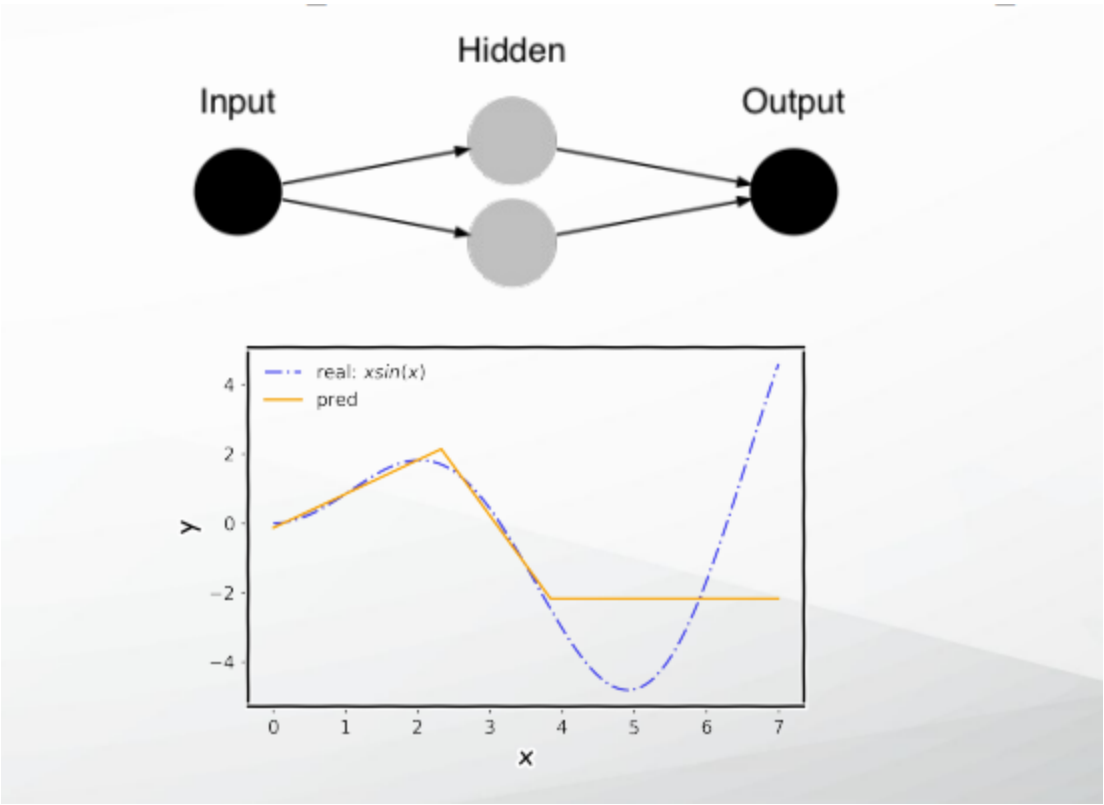


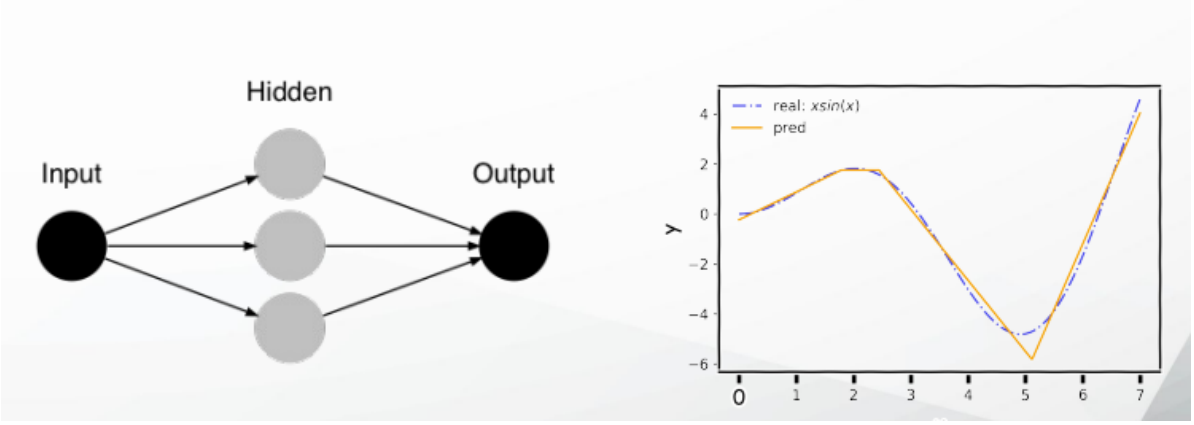
- hinge embedding loss

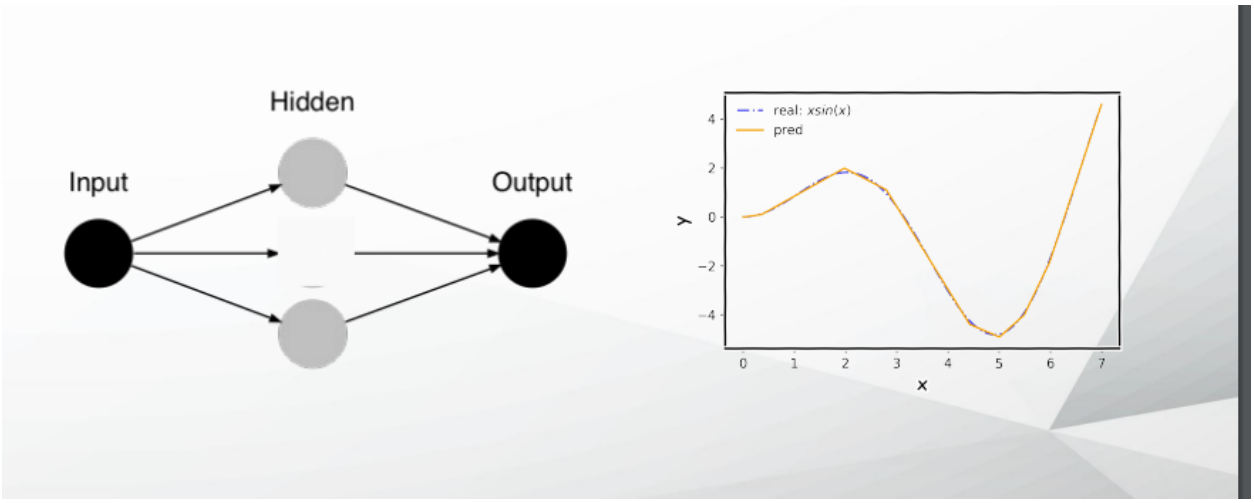
Arhitecture

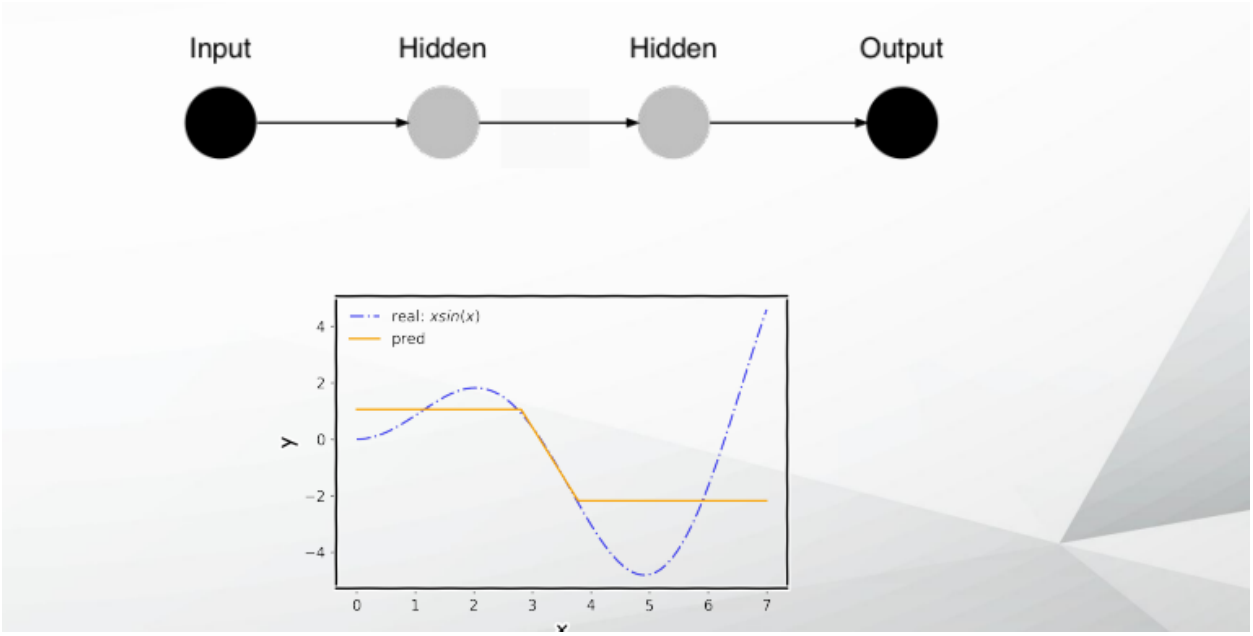
Performance example

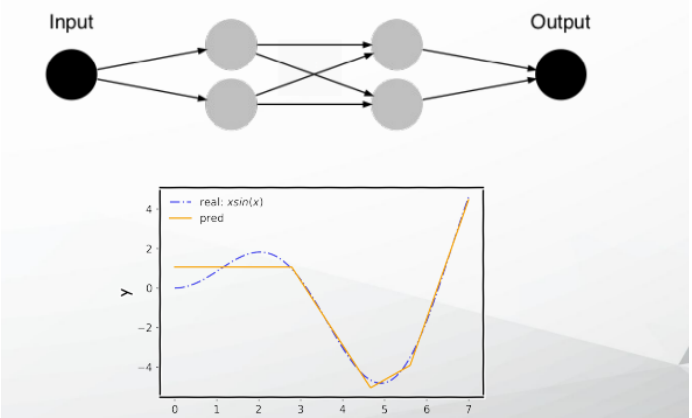



Hidden
Input
Output
real: xsin(x)
pred
y
x


Hidden
Input
Output
real: xsin(x)
pred
y


Input
Hidden
Output
real: xsin(x)
pred
y
x


Input
Hidden
Hidden
Output
real: xsin(x)
pred
y
x


Input
Output
real: xsin(x)
pred
y

Universal Approximation Theorem

- think of a neural network as function approxiamtion
- nn:
  - one hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy
- why deeper?
  - shallow net may need more width
  - shallow net may overfit more

Training a feed-forward ANN

Initialisation of network weights $w^1_{1,1}, w^1_{1,2}, w^1_{1,3}, \dots, w^n_{1,1}, \dots, w^n_{H_{n-1},H_n}, w^O_{1,1}, w^O_{1,2}, \dots, w^O_{r,H_n}$

While not stop condition

For each train example (x^d, t^d), where $d = 1, 2, \dots, n$

Activate the network and determine the output o^d :

forward propagate the information and determine the output of each neuron

Modify the weights:

establish and backward propagate the error:

- establish the errors of neurons from the output layer $e_r, r=1,\dots,m$
- backward propagate the errors in the entire network → distribute the errors on all connections of the network
- modify the weights

EndWhile

Back-propagation of errors in an ANN

- one of the first algorithms for fine tunning of the weights in an ANN
- very popular
- advantages
  - a gradient descent method
  - does not require normalization of input vectors (normalization could improve performance)

- limitations
  - is not guaranteed to find the global minimum of the error function, but only a local minimum
  - is has trouble crossing plateus in the error function landscapes
  - requires the derivatives of activation functions to be known at network desing time

Computing the derivaties of error

We consider an error function for the model $E(X, W, t)$

Compute the derivative of this function with respect to every weight $w^l_{j,k}$ (the weight from layer l, between neuron j from layer l and neuron k from layer l-1).

In general it is:

$$\frac{\partial E}{\partial w^l_{j,k}} = \frac{\partial E}{\partial o^l_j}\frac{\partial o^l_j}{\partial w^l_{j,k}} = \frac{\partial E}{\partial o^l_j}\frac{\partial o^l_j}{\partial net^l_j(X^l)}\frac{\partial net^l_j(X^l)}{\partial w^l_{j,k}}$$

Observe that in the last derivative we have a sum, but only one term depends on the weight so:

$$\frac{\partial net^l_j(X^l)}{\partial w^l_{j,k}} = \frac{\partial}{\partial w^l_{j,k}}\left(\sum_{q=1}^{n_{l-1}} x^l_q w^l_{j,q}\right) = \frac{\partial(x^l_k w^l_{j,k})}{\partial w^l_{j,k}} = x^l_k = o^{l-1}_k$$

Here: $X^l = (x^l_1, x^l_2, \ldots, x^l_{n_l-1})$ is the input for layer l (aka the output o^{l-1} from the previous layer)

For the first hidden layer we have a different situation: the input for this layer is the actual input in the network.

If we evaluate further the partial derivative we get:

$$\frac{\partial o^l_j}{\partial net^l_j(X^l)} = \frac{\partial \phi(net^l_j(X^l))}{\partial net^l_j(X^l)}$$

which is the partial derivative of the activation function ϕ - hence the importance of the derivative

In practice we begin from the last layer because the partial derivatives for this layer are straight forward, and we move backward from layer to layer until we reach the first hidden one.

Deep Learning

- Directly learn ϕ

$$f(\boldsymbol{x}, \eta) = W^T \phi(\boldsymbol{x}, \eta)$$

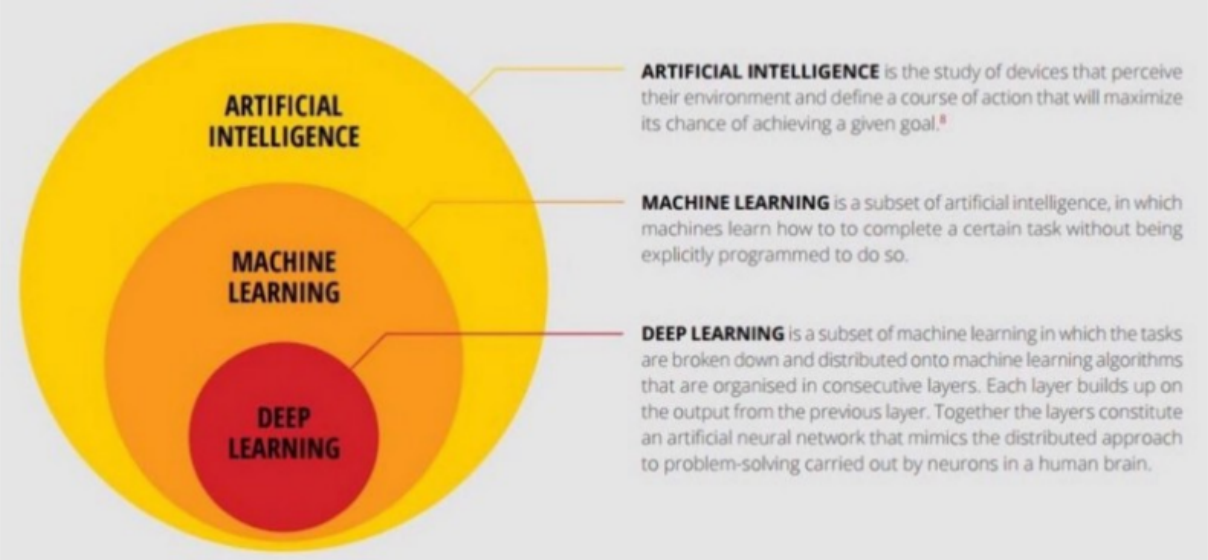
  - $\phi(\boldsymbol{x}, \eta)$ is an automatically-learned **representation** of x
- For **deep networks**, ϕ is the function learned by the **hidden layers** of the network
- η are the learned weights

Non-convex optimization

- Can encode prior beliefs, generalizes well

Definition

- a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks



- it mirros the functioning of our brains
- similar to how system structured where each neuron connected each other and passing information

Deep convolutional Neural Networks

- convnets are deep artifical neural networks used:
  - to clasify images
  - cluster them by similarity
  - perform object recognition within scenes
- algorithms that can identify faces, individuals, street signs, tumors, perform optical character recognition (OCR)
- pre-processing required in a ConvNet is much lower as compared to other classification algorithms
- arhitecture of a ConvNet is analogous to that of the connetivity pattern of Neurons in the Human Brain - Visual Cortex
- a Conv Net capturres the spatial and temporal dependencies in an image

Tensors

- an n-th rank tensor in m-dimensional space is a mathematical object that has n indices and m^n components and obeys certain transformation rules
- generalization of scalar, vectors, and matrices to an arbitrary number of indices
- used in physics such as elasticity, fluid mechanics and general relativity

Notations

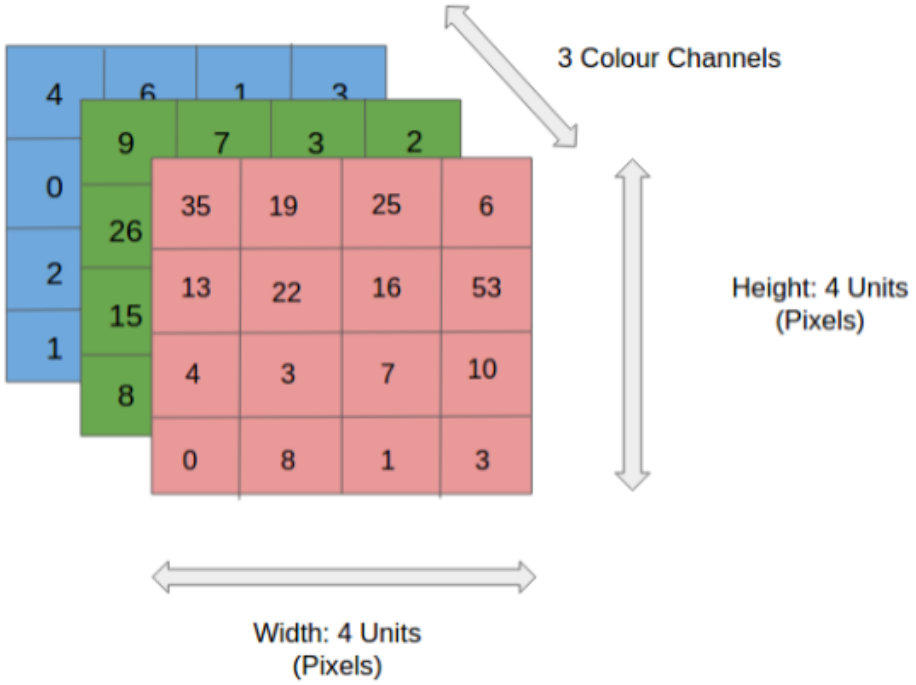
- $a_{ijk\ldots}$, $a^{ijk\ldots}$, $a_i^{jk\ldots}$, etc., may have an arbitrary number of indices
- a tensor (rank $r+s$) may be of mixed type (r,s): r "contravariant" (upper) indices and s "covariant" (lower) indices.

- the positions of the slots in which contravariant and covariant indices are placed are significant!

$$a^{\lambda}_{\mu\nu} \neq a^{\nu\lambda}_{\mu}$$

Images as tensors

- convNets ingest and process images as tensors
- reduce images → form easy to process (without losing critical features



Convolution operation

- an operation on two functions x(t) (input) and w(t) (kernel) of a real-valued argument

$$s(t) = \int x(a)w(t-a)da$$

notation:

$$s(t) = (x \circledast w)(t)$$

if t is discreet the integral turns into a sum

$$s(t) = (x \circledast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- in practice we have two tensors:
  - the input - multidimensional array of data
  - the kernel - multidimensional array of parameters
- stored separately → these functions are zero everywhere but in the finite set of points
- we can implement the infinite summation as a summation over a finite number of elements
- convolutions can be over more than one axis at a time

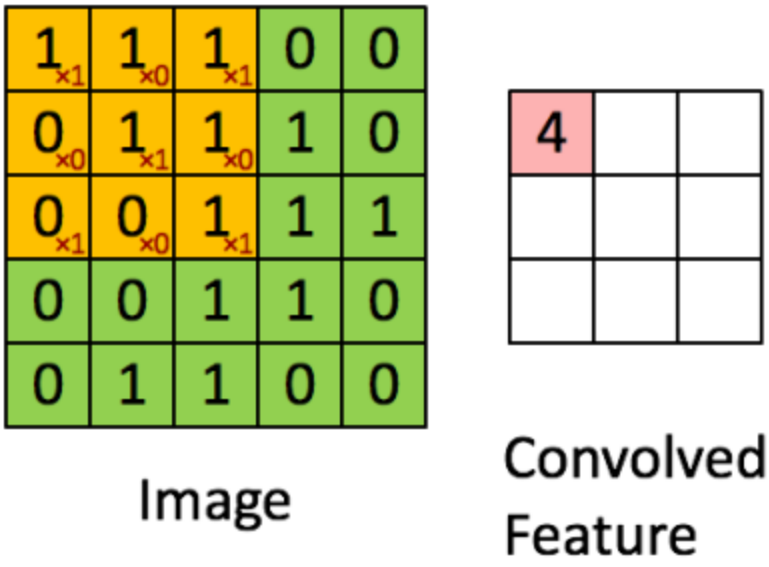
$$S(i,j) = (I \circledast K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Using a kernel

- we want to apply a filter over the image
- aim: extract the high-level features

- example: image dimensions = 5 (height) x 5 (breath) x 1 (number of channels, ex: rbg)
- kernel / filter

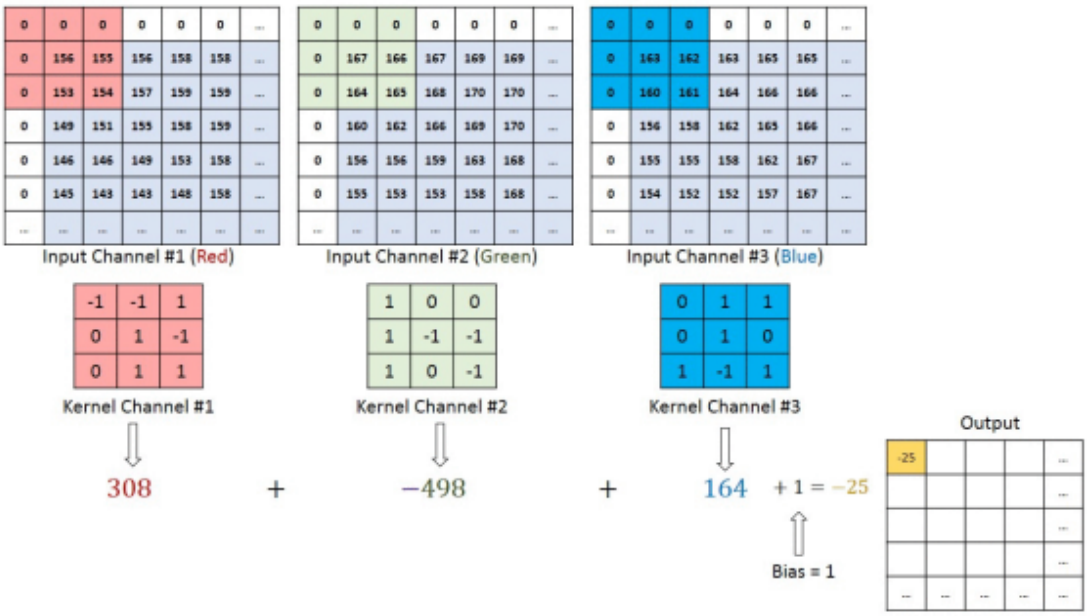
$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



Movement of the kernel

- kernel shifts
- every time perfoming a matrix multiplitcation operation between k and the portion p of the image over which the kernel is hovering

3 channles



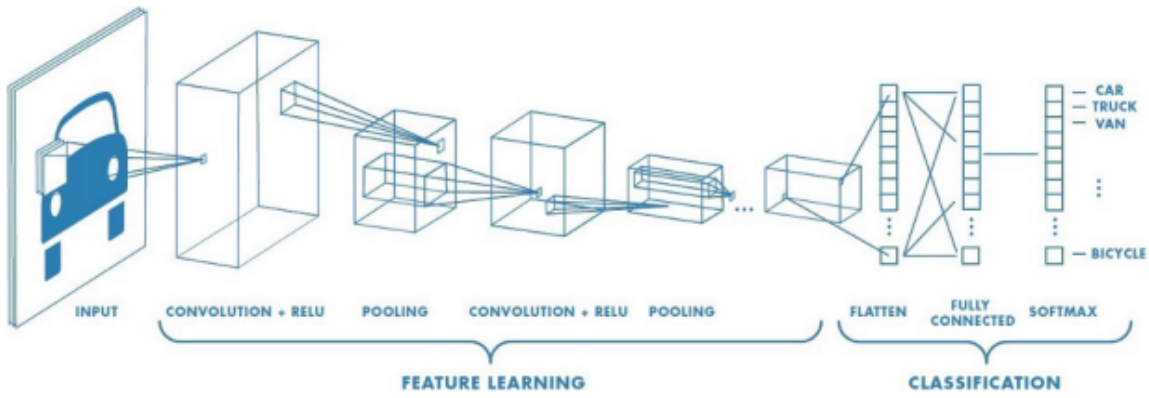
Layers of a convolutional network

- typical three stages:
  - first: several convolutions
  - second: a nonlinear activation function → detector stage
  - third: pooling function to modify the output
- examples
  - max pooling (Zho and Chellappa, 1988 - maximum output withing a rectangular neighborhood), the average of a rectangular neighbourhood, L2 norm of a rectangular neighbourhood, a weighted avearage based on the distance from the central pixel

Training the network

- similar to ANN
- after computing the error, a gradient descent methos is applied to all layers

Feature learning



Vertical line detector

- applying this filter to an image wiell result in a feature map that contains only vertical lines. It is a verical line detector

1.0	0.0	-1.0
2.0	0.0	-2.0
1.0	0.0	-1.0

Table: A 3x3 element Sobel filter for detecting vertical lines

- dragging this filter systematically across pixel values in an image can only highlight line pixel

Horizontal line detector

-2.0	-2.0	-4.0	-2.0	-2.0
-1.0	-1.0	-2.0	-1.0	-1.0
0.0	0.0	0.0	0.0	0.0
1.0	1.0	2.0	1.0	1.0
2.0	2.0	4.0	2.0	2.0

Table: A horizontal line detector

- combining the filters' results (combining both feature maps, will result in all of the lines in an image being highlighted)
- a suite of tens or even hundreds of other small filters can be designed to detect other features in the image

- combining the filters' results (combining both feature maps, will result in all of the lines in an image to being highlighted)
- a suite of tens or even hundreds of other small filters can be designed to detect other features in the image
- the values of the filter weights to be learned during the training of the network
- training under gradient descent
  - the network will learn what types of features to extract from the input
  - extracted features will be those that minimize the loss function (extract features that are most useful for clasyfying images as dogs or cats

Gradient descent on Convolution layer

- the input $X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$
- the filter $F = \begin{bmatrix} F_{1,1} & F_{1,2} \\ F_{2,1} & F_{2,2} \end{bmatrix}$
- the output is $O = X \circledast F$
- $\frac{\partial E}{\partial O}$ is the gradient of loss from previous layer

After we apply the convolution on X and F we have:

$$O_{1,1} = x_{1,1} * F_{1,1} + x_{1,2} * F_{1,2} + x_{2,1} * F_{2,1} + x_{2,2} * F_{2,2}$$

$$O_{1,2} = x_{1,2} * F_{1,1} + x_{1,3} * F_{1,2} + x_{2,2} * F_{2,1} + x_{2,3} * F_{2,2}$$

$$O_{2,1} = x_{2,1} * F_{1,1} + x_{2,2} * F_{1,2} + x_{3,1} * F_{2,1} + x_{3,2} * F_{2,2}$$

$$O_{2,2} = x_{2,2} * F_{1,1} + x_{2,3} * F_{1,2} + x_{3,2} * F_{2,1} + x_{3,3} * F_{2,2}$$

we compute the partial derivative of O with respect of F

$$\frac{\partial O_{1,1}}{\partial F_{1,1}} = x_{1,1},\ \frac{\partial O_{1,1}}{\partial F_{1,2}} = x_{1,2},\ \frac{\partial O_{1,1}}{\partial F_{2,1}} = x_{2,1},\ \frac{\partial O_{1,1}}{\partial F_{2,2}} = x_{2,2}$$

similar we compute for $O_{1,2}$, $O_{2,1}$, and $O_{2,2}$

the gradient to update the filter will be

$$\frac{\partial E}{\partial F} = \frac{\partial E}{\partial O} * \frac{\partial O}{\partial F} \tag{1}$$

if we expand Equation 1

$$\begin{aligned}\frac{\partial E}{\partial F_{1,1}} &= \frac{\partial E}{\partial O_{1,1}} * \frac{\partial O_{1,1}}{\partial F_{1,1}} + \frac{\partial E}{\partial O_{1,2}} * \frac{\partial O_{1,2}}{\partial F_{1,1}} \\ &+ \frac{\partial E}{\partial O_{2,1}} * \frac{\partial O_{2,1}}{\partial F_{1,1}} + \frac{\partial E}{\partial O_{2,2}} * \frac{\partial O_{2,2}}{\partial F_{1,1}} \\ &= \frac{\partial E}{\partial O_{1,1}} * x_{1,1} + \frac{\partial E}{\partial O_{1,2}} * x_{1,2} \\ &+ \frac{\partial E}{\partial O_{2,1}} * x_{2,1} + \frac{\partial E}{\partial O_{2,2}} * x_{2,2}\end{aligned}$$

we observe that this is actually a convolution product between the input and the loss gradient

$$\begin{bmatrix} \frac{\partial E}{\partial F_{1,1}} & \frac{\partial E}{\partial F_{1,2}} \\ \frac{\partial E}{\partial F_{2,1}} & \frac{\partial E}{\partial F_{2,2}} \end{bmatrix} = X \circledast \begin{bmatrix} \frac{\partial E}{\partial O_{1,1}} & \frac{\partial E}{\partial O_{1,2}} \\ \frac{\partial E}{\partial O_{2,1}} & \frac{\partial E}{\partial O_{2,2}} \end{bmatrix} \quad (2$$

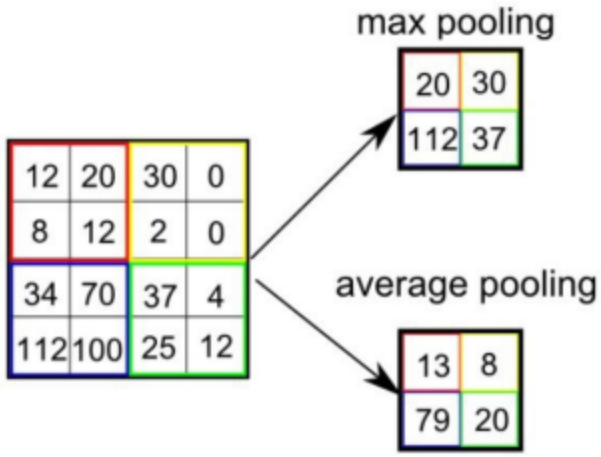
similar we get for the derivative of E with respect of X

$$\begin{bmatrix} \frac{\partial E}{\partial x_{1,1}} & \frac{\partial E}{\partial x_{1,2}} & \frac{\partial E}{\partial x_{1,3}} \\ \frac{\partial E}{\partial x_{2,1}} & \frac{\partial E}{\partial x_{2,2}} & \frac{\partial E}{\partial x_{2,3}} \\ \frac{\partial E}{\partial x_{3,1}} & \frac{\partial E}{\partial x_{3,2}} & \frac{\partial E}{\partial x_{3,3}} \end{bmatrix} = \begin{bmatrix} F_{2,2} & F_{2,1} \\ F_{1,2} & F_{1,1} \end{bmatrix} \circledast \begin{bmatrix} \frac{\partial E}{\partial O_{1,1}} & \frac{\partial E}{\partial O_{1,2}} \\ \frac{\partial E}{\partial O_{2,1}} & \frac{\partial E}{\partial O_{2,2}} \end{bmatrix}$$

observe that matrix F is flipped over 180^o degrees in this formula.

Pooling

- helps to make the representation approximately invariant to small transaltions of the input
- useful property if we care more about whether some feature is present than exactly where it is
- essential for handling inputs of varying size



Max Pooling

- return the maximum value from the portion of the image covered by the kernel

Average pooling

- returns the average of all the values from the portion of the image covered by the kernel

Training

- in an NxN pooling block, backward propagation of error is reduced to a single value, value of the “winning unit”

Solving Search problems - Unifromed Search Strategies

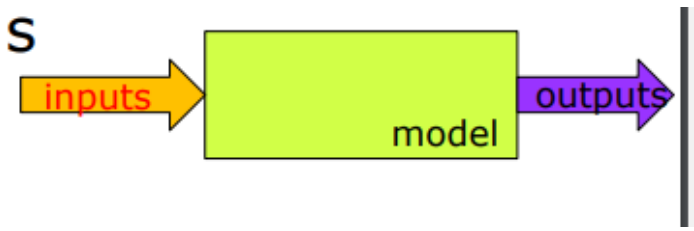
Problems

- two problem types
  - solving in a deterministic manner
  - computing the sinus of an angle or the square root of a number
  - solving in a stochastic manner
    - real-world problems → design of ABS
    - involve the search of a solution → AI’s methods

Typology

Search/Optimization problems

- planning, satellite’s design

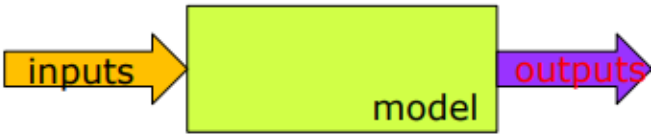


Modelling problems

- predictions, classifications

Simulation problems

- Game theory



Problem solving

Identification of a solution

- in computer science (AI) → search process
- in engineering and mathematics → optimisation process

How?

- representation of (partial) solutions → points in the search space
- design of a search operators → map a potential solution into another one

Steps in problem solving

- problem definition
- problem analysis
- selection of a solving technique
    - search

    - knowledge representation
    - abtraction

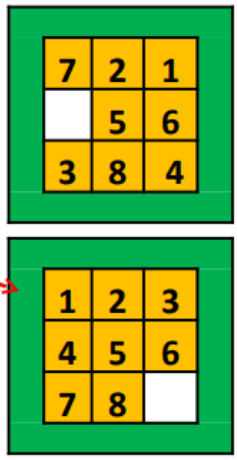
Solving problems by search

- based on some objectives
- composed by actions that accomplish the objectives
    - each action changes a state of the problem
- more actions that map the initial state of problem into a final state

Problem definition - involves

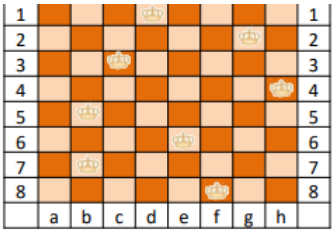
- a search space
    - all possible states
    - representation
        - explicit - construction of all possible states
        - default - by using some data structures and some functions (operators)
- one or more intial states
- one or more final states
- one or more paths
    - more successful path
- a set of rules (actions)
    - successor functions (operators) - next state after a given one
    - cost function that evaluates
        - how a state is mapped into another state
        - an entire path
    - objective functions that check if a state is final or not
- examples

- Puzzle game with 8 pieces
  - State's space – different board configurations for a game with 8 pieces
  - Initial state – a random configuration
  - Final state – a configuration where all the pieces are sorted in a given manner
  - Rules -> white moves
    - conditions: move inside the table
    - Transformations: the white space is moved up, down, to left or to right
  - Solution - optimal sequence of white moves



Examples

- Queen's problem
  - State's space – different board configurations for a game with n queens
  - Initial state – a configuration without queens
  - Final state – a configuration n queens so that none of them can hit any other in one move
  - Rules -> put a queen on the table
    - conditions: the queen is not hit by any other queen
    - Transformations: put a new queen in a free cell of the table
  - Solution - optimal placement of queens



Problem analyse

- the problem can be decomposed?
  - the sub-problems are independent or not?
- the possible state's space is predictable?

- we want a solution or an optimal solution?
- the solution is represented by a single state or by more successive states?
- we require some knowledge for limiting the search or for identifying the solution?
- the problem is conversational or solitary?
    - human interaction is required for problem solving?

Selection of a solving technique

- solving by moving rules (and control strategy) in the search space until we find a path from the initial state to the final state
- solving by search
    - examination of all possible states in order to identify
        - a path from the initial state to the final state
        - an optimal state
    - the search space = all possible states and operators that maps the states

more selection strategies → how we select one of them?

- computational complexity (temporal and spatial)
    - strategy's performance depends on
        - internal factors
            - time for running
            - memory for running
            - size of input data
        - external factors
            - computer's performance
            - compiler's quality
        - can be evaluated by complexity → computational efficiency

        - spatial → required memory for solution identification
            - S(n) - memory used by the best algorithms A that solves the decision problem f with n input data
        - temporal → required time for solution identification
            - T(n) → running time (number of steps) of the best algorithm A that solves a decision problem f with n input data
- completeness → the algorithms always ends and finds a solution (if it exists)
- optimal → the algorithms finds the optimal solution (the optimal cost of the path from the initial state to the final state)

problem solving by search can be performed y:

- step by step construction of solution
    - problem's components
        - initial state
        - operators
        - final state
        - solution = a path (optimal cost) from the initial state to the final state
    - search space
        - all the states that can be obtained from the initial state (by using the operators)
        - a state = a component of solution
    - example
        - traveling salesman problem (TSP_
    - algorithms
        - main idea: start with a solution's component and adding new components until the complete solution is obtained
        - reccursive ⇒ until a condition is satistfied

        - the search’s history (path from inotial state to the final state) is retained in LIFO/FIFO containers
    - advantages
        - do not require knowledge
- optimal solution identification
    - problem’s compoents
        - conditions (constraints) that must be satisfied by the solution
        - evaluation function for a potential solution → optimum identification
    - search space
        - all possible and complete solutions
        - state = a complete solution
    - example
        - queens’s problem
    - algorithms
        - main idea: start with a state that doesnt respect some conditions and change it for eliminating these violations
        - iterative → a single state is retained and the algorithm tried to improve it
        - the searches history is not retained
    - advantages
        - simple
        - requires a small memory
        - can find good solution in (continous) search spaces very large (where other algorithms can not be utilised)

Solving problem by search involves:

- very complex algorithms (np-complete problems)

- search in an exponential space

Topology of search strategies

Solution generation

- constructive search
  - solution is identified step by step
  - ex: tsp
- pertrubative search
  - a possible solution is modified in order to obtain another possible solution
  - ex: sat - propositional satisfaction problem

Search space navigation

- systematic search
  - the entire search space is visited
    - solution identification (if it exists) → complete algorithms
- local search
  - moving from a point of the search space into a neigbour point → incomplete algorithms
  - a state can be visited more times

Certain items of the search

- deterministic search
  - algorithms that exatcly identify the solution
- stochastic search
  - algorithms that approximate the solution

Search space exploration

- sequential search

- parallel search

Number of objectives

- single-objective search
  - the solution must respect a single condition / constraint
- multi-objective search
  - the solution must respect more conditions / constraints

Number of solutions

- single-modal search
  - there is a single optimal search
- multi-modal search
  - there are more optimal solutions

Algorithm

- search over a finite number of steps
- iterative search
  - the algorithm converge through the optimal solutions
- heuristic search
  - the algorithms provide an approximation of the solution

Search mechanism

- traditional search
- modern search

Where the search takes place

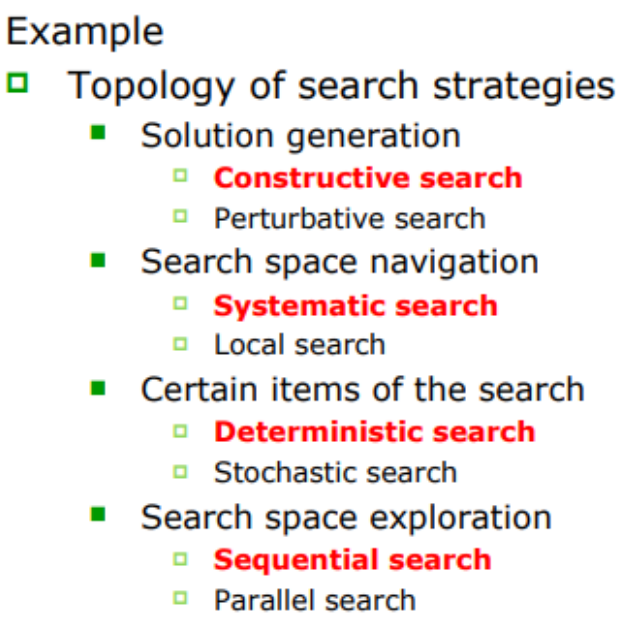
- local search
- global search

Type(linearity) of contraints

- linear search
- non-linear search
  - classical (deterministic)
    - direct - based on evaluation of the objective function
    - indirect - based on derivative (I and/or II) of the objective function
  - enumeration-based
    - how solution is identified
      - uniformed - the solution is the final state
      - informed - deals with an evaluation function for a possible solution
    - search space type
      - complete - the space is finite (if solution exists, then it can be found)
      - incomplete - the space is infinite
  - stochastic search
    - based on random numbers

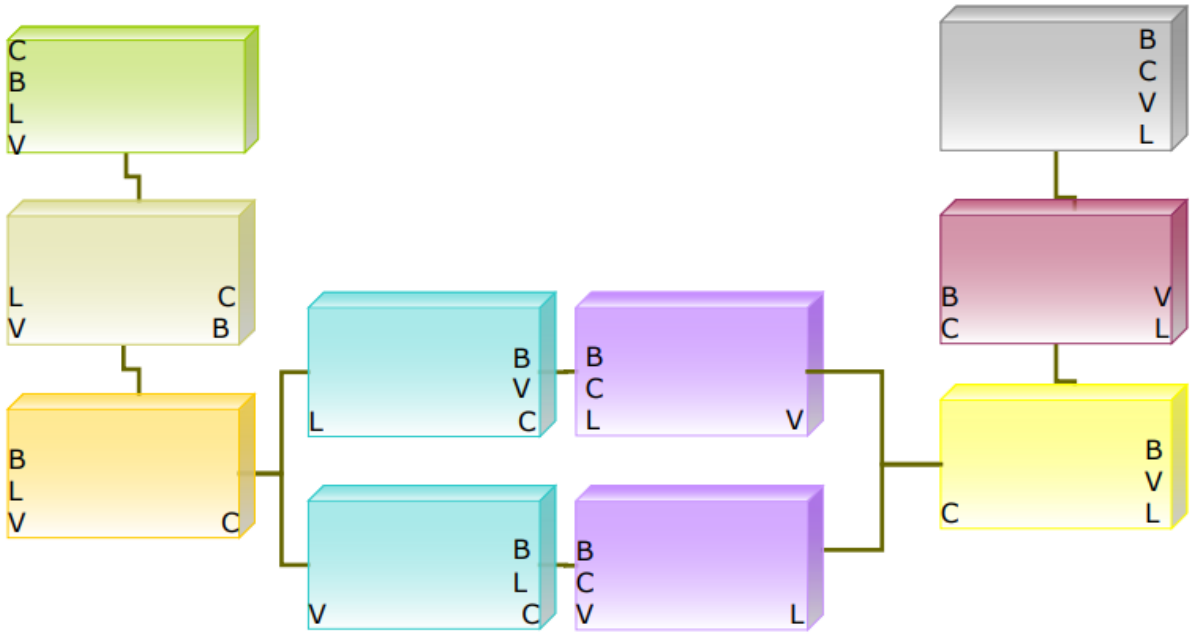
Agents involved in the search

- search by a single agent → without obstacle for achieving the objectives
- adversarial search → the opponent comes with some uncertainty



Examples

- constructive, global ,determinist, sequential search
- problem “capra, varza, lupul”
- input
  - a goat, a cabbgae and a wolf on a river-side
  - a boar with a boater
- output
  - move all the passengers on the other side of the river
  - take into account
    - the boat has only 2 places
    - it is not possible to rest on the same side
      - the goat and the cabbage
      - the wolf and the goat



Search strategies - basic elements

Abstract data types (ADTs)

- adt list →linear structure
- adt tree → hierarchial structure
- adt graph → graph based structure
- adt
  - domain and operations
  - representation

Uninformed search strategies (USS)

Charactheristics

- are not based on problem specific information
- are general
- blind strategies

- brute force methods

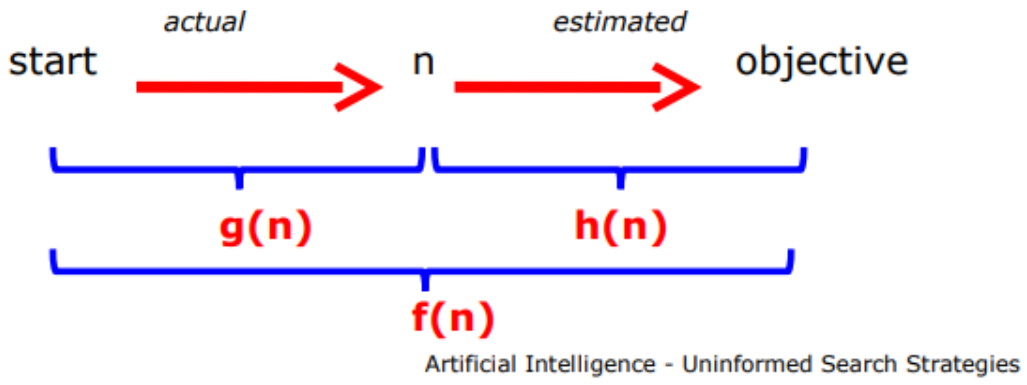
Topology

- order of node exploration
  - uss in linear structures
    - linear search
    - binary search
  - uss in non-linear structures
    - breath-first search
      - uniform cost search (branch and bound)
    - depth first search
      - limited depth first search
      - iterative deepening depth-first search
    - bidirectional search

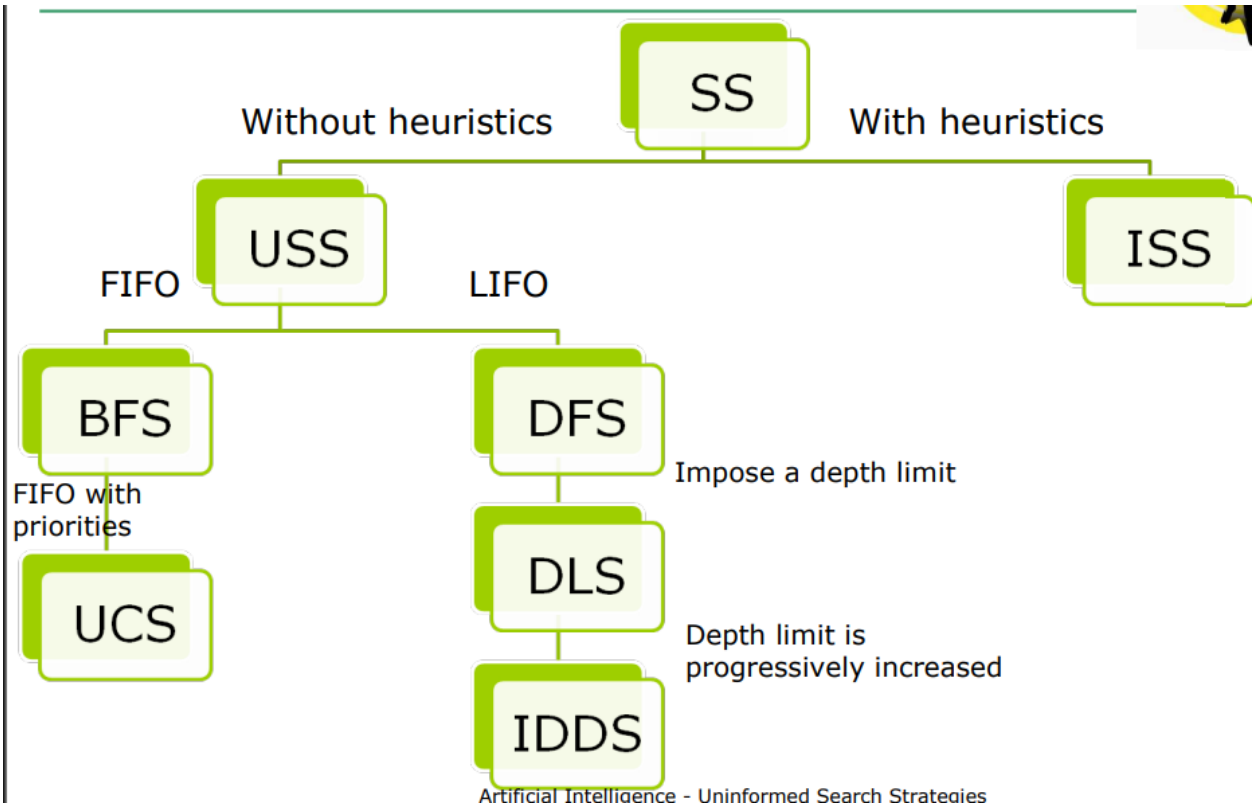
Search strategies in tree-based structures

Basic elements

- f(n) - evaluation function for estimating the cost of a solution through a node (state) n
- h(n) - evaluation function for estimating the cost of a solution path from node (state) n to final node (state)
- g(n) - evaluation function for estimating the cost of a solution path from the intial node (state) to node (state) n
- f(n) = g(n) + h(n)



USS in tree-based structures



SS	Time complexity	Space complexity	Completeness	Optimality
BFS	$O(b^d)$	$O(b^d)$	Yes	Yes
UCS	$O(b^d)$	$O(b^d)$	Yes	Yes
DFS	$O(b^{dmax})$	$O(b*d_{max})$	No	No
DLS	$O(b^{dlim})$	$O(b*d_{lim})$	Yes, if $d_{lim} > d$	No
IDS	$O(b^d)$	$O(b*d)$	Da	Yes
BDS	$O(b^{d/2})$	$O(b^{d/2})$	Yes	Yes

Artificial Intelligence - Uninformed Search Strategies

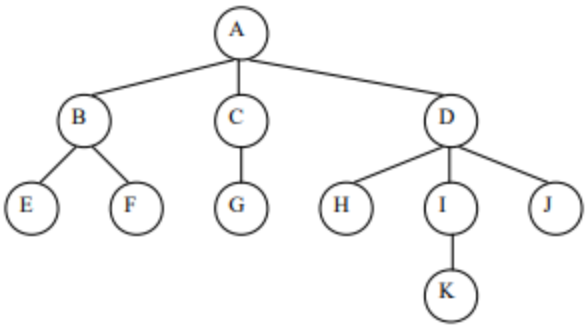
Breath first search - BFS

Basic elements

- all the nodes of depth d are visited before all the nodes of depth d+1
- all the children of current node are added into a FIFO list (queue)

Example

- visiting order: a,b,c,d,e,f,g,h,i,j,k



Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	C, D, E, F
A, B, C	D, E, F, G
A, B, C, D	E, F, G, H, ,I, J
A, B, C, D, E	F, G, H, I, J
A, B, C, D, E, F	G, H, I, J
A, B, C, D, E, F, G	H, I, J
A, B, C, D, E, F, G, H	I, J
A, B, C, D, E, F, G, H, I	J, K
A, B, C, D, E, F, G, H, I, J	K
A, B, C, D, E, F, G, H, I, J, K	Φ

e do

ce - Uninformed Search Strategies

Algorithm

```
bool BFS(elem, list) {
    found = false;
    visited = null;
    toVisit = (start); //FIFO list
    while((toVisit != null) && (!found)){
        node = pop(toVisit);
        visited = visited U {node};
        if (node == elem)
        {
            found = true;
        }
```

```
        else
        {
            aux = null;
            for all (unvisited) children of node do
            {
                aux = aux U {child};
            }
        }
        toVisit = toVisit U aux;
    } //while
    return found;
}
```

Search analysis

- time complexity
  - b - ramification factor (number of children of each node)
  - d - length (depth) of solution
  - T(n) = 1 + b + b^2 + ... b^d ⇒ O(b^d)
- space complexity
  - S(N) = T(N)
- completness
  - if solution exists, then bfs finds it
- optimality
  - no

Advantages

- finds the shortest path to the objective node (the shallowest solution)

Disadvantages

- generate and retain a tree whose size exponentially increases (with depth of objective node)

- exponential time and space complexity
- Russel&Norving experiment
- works only for small search spaces

Applications

- identification of connex components in a graph
- identification of the shortest path in a graph
- optimisation in transport networks → Ford-Fulkerson
- Serialisation / deserialization of a binary tree (vs. serialization in a sorted manner) allows efficiently reconstructing the tree
- collection copy (garbage collection) → algorithm Cheney

Uniform Cost Search - UCS

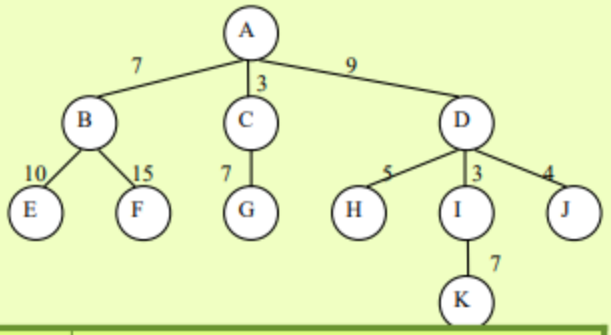
Basic elements

- bfs + special expand procedure (based on the cost of links between nodes)
- all the nodes of depth d are visited before all nodes of depth d+1
- all children of current node are added into FIFO ordered list
  - the of minimum cost are firstly expanded
  - when a path to the final state is obtained, it became a candidate to the optimal solution
- branch and bound algorithm

Example

- visiting order: a,b,c,d,e,f,g,i,j,k

solution



visited	toVisit
Φ	A
A	C(3), B(7), D(9)
A, C	B(7), D(9), G(3+7)
A, C, B	D(9), G(10), E(7+10), F(7+15)
A, C, B, D	G(10), I(9+3), J(9+4) ,H(9+5), E(17), F(22)
A, C, B, D, G	I(12), J(13) ,H(14), E(17), F(22)
A, C, B, D, G, I	J(13) ,H(14), E(17), F(22), **K(9+3+7)**
A, C, B, D, G, I, J	H(14), E(17), F(22), **K(19)**
A, C, B, D, G, I, J, H	E(17), F(22), **K(19)**
A, C, B, D, G, I, J, H, E	F(22), **K(19)**
A, C, B, D, G, I, J, H, E, F	K(19)
A, C, B, D, G, I, J, H, E, F, K	Φ

- Uninformed Search Strategies 30

Algorithm

```
bool UCS(elem, list) {
    found = false;
    visited = null;
    toVisit = (start); //FIFO sorted List
    while((toVisit != null) && (!found)){
        node = pop(toVisit);
        if(node == elem)
            found = true;
        else{
            aux = null
            for all (univisited) children of node do
                aux = aux U {child};
```

```
            }
        }
        toVisit = toVisit U aux;
        TotalCostSort(tovisit);
    } //while
    return found;
}
```

Search analysis

- time complexity
  - T(n) = 1 + b + b^2 ... + b^d = O(b^d)
- space complexity
  - S(n) = T(n)
- completness
  - if solution exists, then ucs finds it
- optimality
  - no

Advanatages

- finds the shortest path to the objective node

Disadvantages

- exponential time and space complexity

Applications

- shortest path → Dijkstra algorithm

Depth-first search - DFS

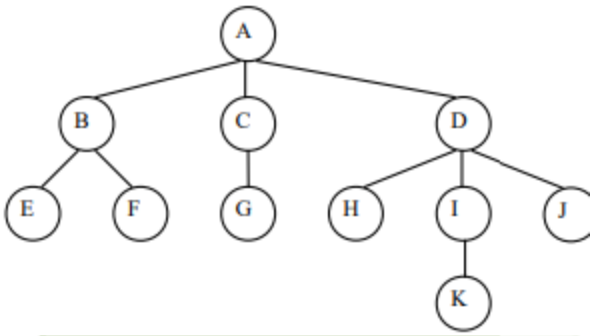
Basic elements

- expand a child and depth search untill

    - the final node is reached or
    - the node is a leaf
- coming back in the most recent node that must be explored
- all the children of the current node are added in a LIFO list (stack)

Example

- visiting order: a,b,c,d,e,f,g,i,k,j



Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	K, J
A, B, E, F, C, G, D, H, I, K	J
A, B, E, F, C, G, D, H, I, K, J	Φ

Algorithm

```
bool DFS(elem, list){
    found = false;
    visited = null;
```

```
    toVisit = {start} ; //LIFO list
    while((toVisit != null) && (!found)){
        node = pop(toVisit);
        visited = visited U {node};
        if (node == elem)
            found = true
        else
        {
            aux = null
            for all (unvisited) children of node do{
                aux = aux U {child};
            }
            toVisit = aux U toVisit;
        }
    } //while
    return found;
}
```

Complexity analysis

- time complexity
  - T(n) = 1 + b + b^2 + ... + b^dmax ⇒ O(b^dmax)
- space complexity
  - S(n) = b*dmax
- completness
  - no → the algorithm does not end for infinte paths (there is no sufficient memory for all the nodes that are visited already)
- optimality
  - no → depth search can find a longer path than the optimal one

Advantages

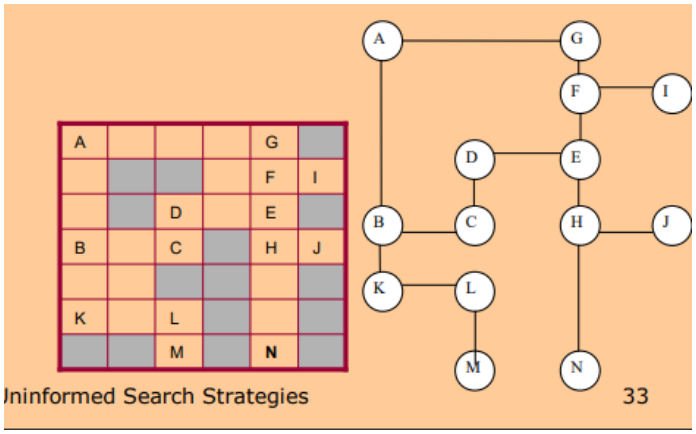
- finding the shortest path with minimal resources (recursive version)

Disadvantages

- dead paths
  - infinite cycles
  - longer solution than the optimal one

Applications

- max problem
- identification of connex components
- topological sorting
- testing the graph planarity



Muchia	Muchii vizitate deja	Muchii de vizitat	înapoi	Noduri vizitate
	Φ	AB, AF	Φ	A
AB	AB	BC, BK, AF	Φ	A, B
BC	AB, BC	CD, BK, AF	Φ	A, B, C
CD	AB. BC, CD	DE, BK, AF	Φ	A, B, C, D
DE	AB, BC, CD, DE	EF, EH, BK, AF	Φ	A, B, C, D, E
EF	AB, BC, CD, DE, EF	FI, FG, EH, BK, AF	Φ	A, B, C, D, E, F
FI	AB, BC, CD, DE, EF, FI	FG, EH, BK, AF	Φ	A, B, C, D, E, F, I
FG	AB, BC, CD, DE, EF, FI, FG	GA, EH, BK, AF	Φ	A, B, C, D, E, F, I, G
GA	AB, BC, CD, DE, EF, FI, FG	EH, BK, AF	GA	A, B, C, D, E, F, I, G
EH	AB, BC, CD, DE, EF, FI, FG	HJ, HN, BK, AF	GA	A, B, C, D, E, F, I, G, H
HJ	AB, BC, CD, DE, EF, FI, FG, HJ	HN, BK, AF	GA	A, B, C, D, E, F, I, G, H, J
HN	AB, BC, CD, DE, EF, FI, FG, HI, HN	BK, AF	GA	A, B, C, D, E, F, I, G, H, N

```
bool DFS_edges(elem, list) {
    discovered = null;
    back = null;
    toDiscover = null; //LIFO
    for (all neighbours of start) do
        toDiscover = toDiscover U {(start, neighbour)};
    found = false;
    visited = {start};
    while ((toDiscover != null) && (!found)){
    {
        edge = pop(toDiscover);
        if (edge.out !E visited){
            discovered = discovered U {edge};
            visited = visited U {edge.out};
            if(edge.out == end)
                found = true;
            else{
                aux = null;
                for all neigbours of edge.out do{
                    aux = aux U {(edge.out, neighbour)};
            }
            toDiscover = aux U toDiscover;
        }
        else
            back = back U {edge}
    } //while
    return found;
}
```

Depth-limited search - DLS

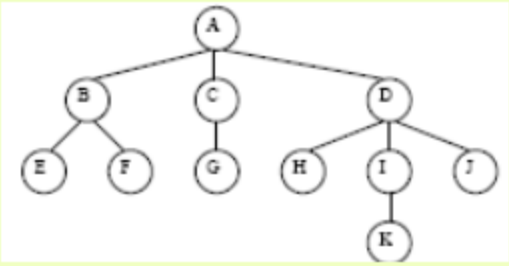
Basic elements

- dfs + maximal depth that limita the search (dlim)

- solved the completeness problems of DFS

Example

- dlim = 2
- visiting order: A,B,E,F,C,G,D,I,J



Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	J
A, B, E, F, C, G, D, H, I, K, J	Φ

Algorithm

```
bool DLS(elem, list, dlim){
    found = false;
    visited = null;
    toVisit = {start}; //LIFO list
    while((toVisit != null) && (!found)) {
```

```
        node = pop(toVisit);
        visited = visited U (node);
        if (node.depth <= dlim) {
            if(node == elem)
                found = true;
            else{
                aux = null
                for all (unvisited) children of node do{
                    aux = aux U {child}
                }
            toVisit = aux U toVisit;
            } // if found
        } // if dlm
    } // while
    return found;
}
```

Complexity analysis

- time complexity
  - T(n) = 1 + b + b^2 + ... b^dlim ⇒ O(b^dlim)
- space complexity
  - S(n) = b*dlim
- completeness
  - yes, but ↔ dlim > d, where d = length (path) of optimal solution
- optimality
  - no → dls can find a longer path than the optimal one

Advantages

- solves the completness problems of DFS

Disadvantages

- how to choose a good limit dlim?

Applications

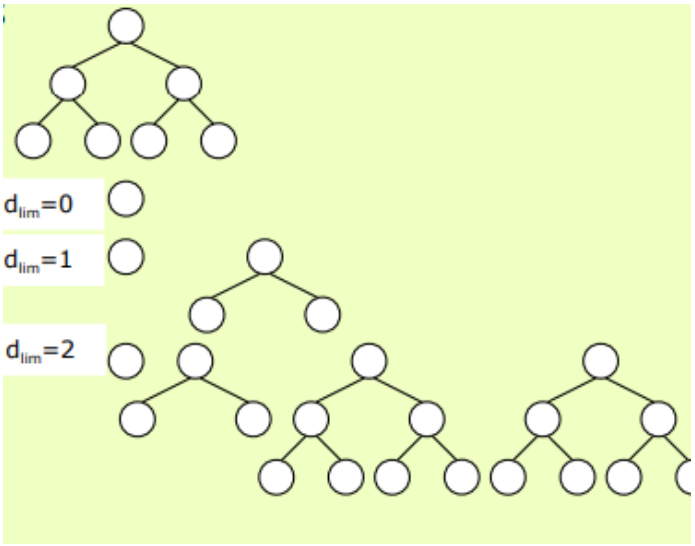
- identificaiton of bridges in a graph

Iterative deepening depth search - IDDS

Basic elements

- U DLS(dlim) where dlim = 1,2,3,4,...,dmax
- solves the identification of the optimal limit dlim from DLS
- usually, it works when:
  - the search space is large
  - the length (depth) of solutio is known

Example



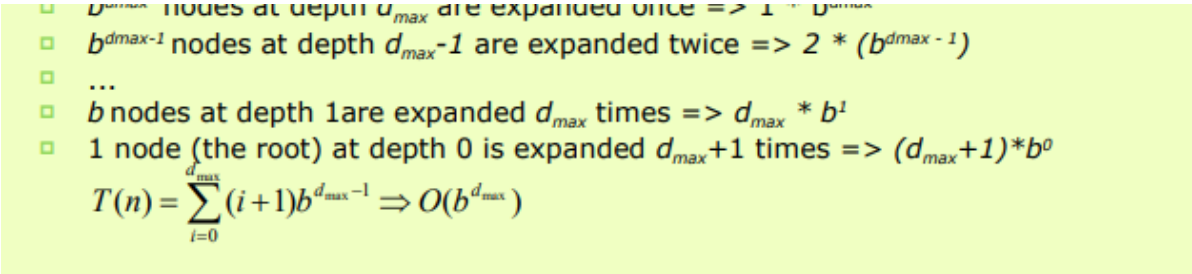
Algorithm

```
bool IDS(elem, list){
    found = false;
    dlim = 0;
    while ((!found) && (dlim < dmax)){
```

```
            found = DLS(elem, list, dlim);
            dlim++;
        }
        return found;
    }
```

Complexity analysis

- time complexity



- space complexity
  - S(n) = b*dmax
- completness
  - yes
- optimality
  - yes

Advatnages

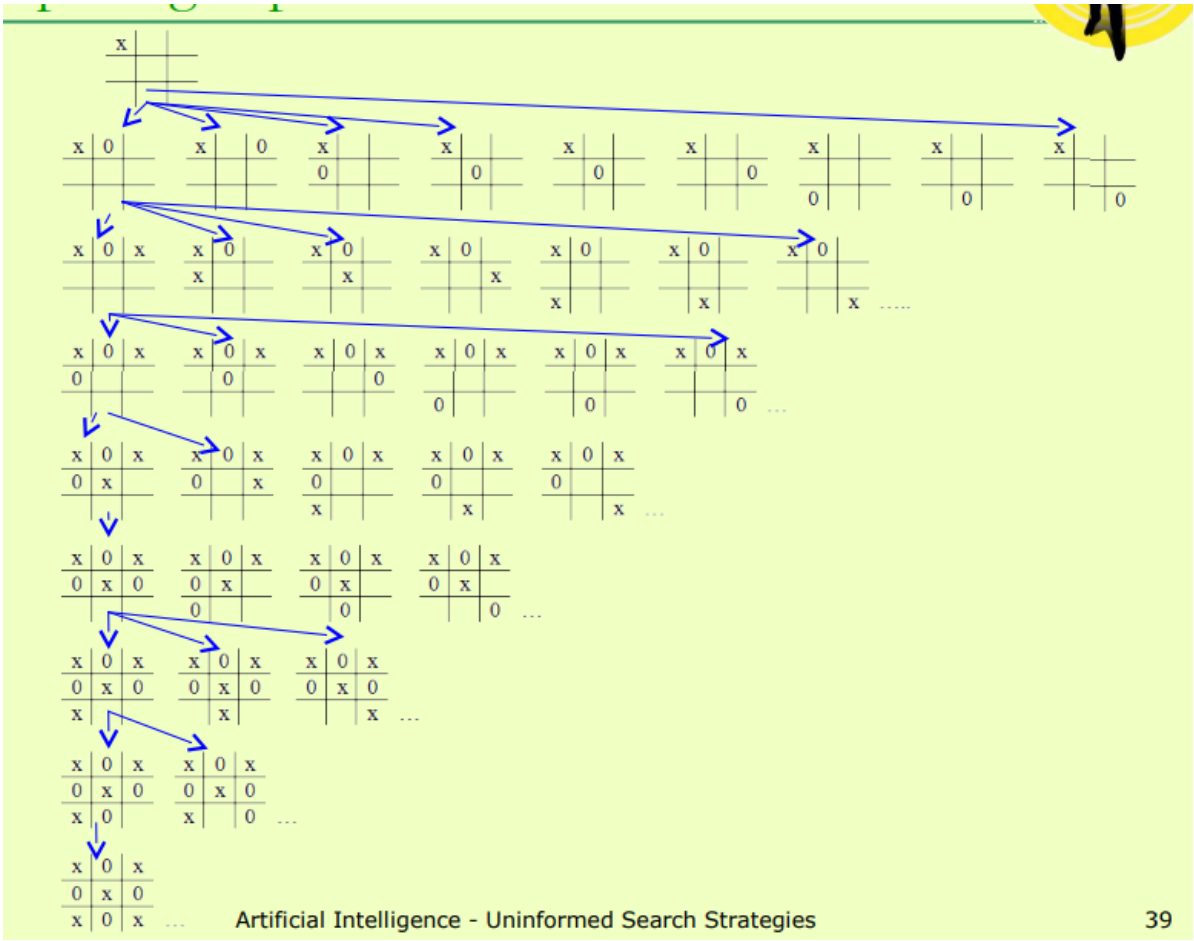
- requires linear memory
- the goal state is obtained by a minimal path
- faster than bfs and dfs

Disadvantages

- rquires to know the solution depth

Applications

- tic tac toe game

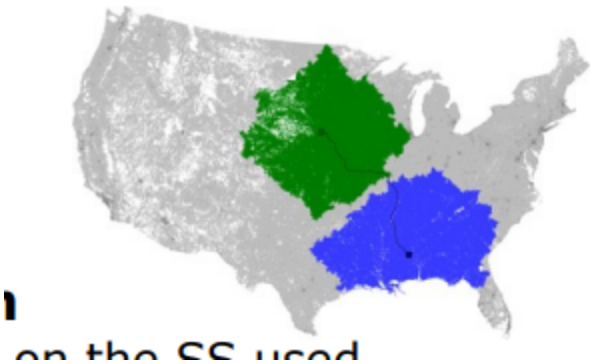


Bi-directional search - BDS

Basic elements

- 2 parallel search strategies
  - forwared: from root to leaves
  - backward: from leaves to root
- any ss can be used in a direction
- requires establishing
  - the parents and the children of each node
  - the meeting point

Example



Algorithm

- depends on the SS used

Complexity analysis

- time complexity
  - O(b^(d/2)) + O(b^(d/2) ⇒ O(b^(d/2))
- Space complexity
  - S(n) = T(n)
- Optimality
  - yes
- Completeness
  - yes

Advantages

- good time and space complexity

Disadvantages

- each state must be reversed
  - from dead to tail
  - from tail to head

- difficult to implement
- identification of parents and children for all the nodes
- the final state must be known

Applications

- partitioning problem
- shortest path

Local Search

- simple loach search - a single neghibour state is retained
  - hil climibing → selects the best neighbour
  - simulated annealing → selects probabilistically the best neigbhour
  - tabu search → retains the list of visited solutions
- beam local search - more states are retained (a population of states)
  - evolutionary algorithms
  - particle swarm optimisation
  - ant colony optimization

Nature-inspired search

Best method for solving a problem

- human brain
- mechanism of evolution

Simulation of nature

- by machines' help → the artificial neural networks simulate the brain

    - flying vehicles, dna computers, memebrane-based computers
- by alhorithms' help
    - evolutinoary algorithms simulate the evolution of nature
    - particle swarm optimisation simulates the collective and socia behaviour
    - ant colony optimisation

Solving search problems - Evolutionary algorithms

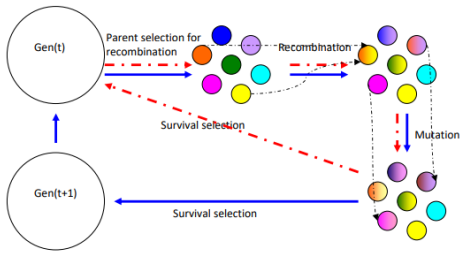
Main charactheristics of EAs

- iterative and parallel processes
- based on random search
- bio-inspired - involve mechanims as:
    - natural selection
    - reporduction
    - recombination
    - mutation

Evolutionary metaphor

Natural evolution		Problem solving
Individual	↔	Possible solution
Population	↔	Set of possible solutions
Chromosome	↔	Coding of a possible solution
Gene	↔	Part of coding
Fitness	↔	Quality
Crossover and Mutation	↔	Search operators
Environment	↔	Problem

- General sketch of an EA
  - Generational
  - Steady-state



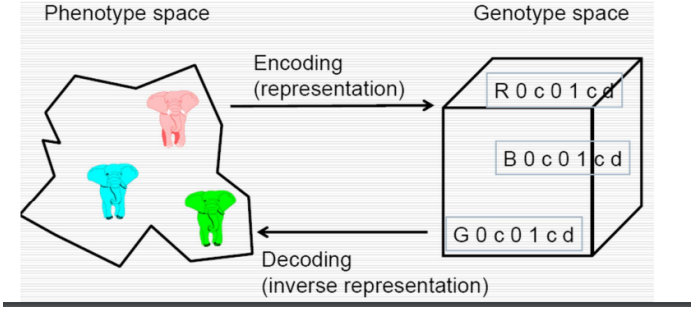
Algorithm Design

- chromosome representation
- population model
- fitness functions
- genetic operators
  - selection
  - mutation
  - crossover
- stop condition

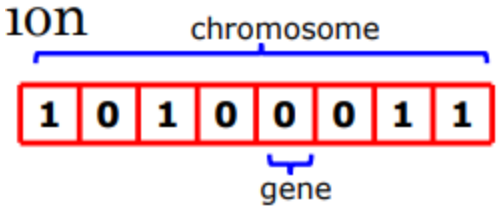
Chromosome Representation

- 2 levels of each possible solution
  - external level → phenotype
    - individual - original object in the context of the problem
    - the possible solutions are evaluated here
    - ant, knapsack, elephant, towns
  - internal level → genotype
    - chromosome - code associated to an object
      - composed by genes in loci (fix positions) and having some values (alleles)

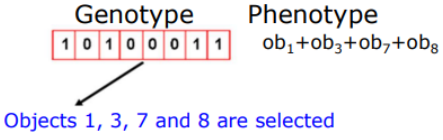
        - the possible solutions are searched here
        - one dimensial vector (with numbers, bits, characters), matrix
- representation must be representative for :
    - problem
    - fitness function
    - genetic operators



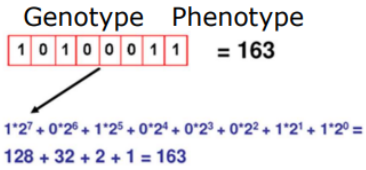
- linear
    - discrete
        - binary → knapsack problem
            - genotype : bit-strings



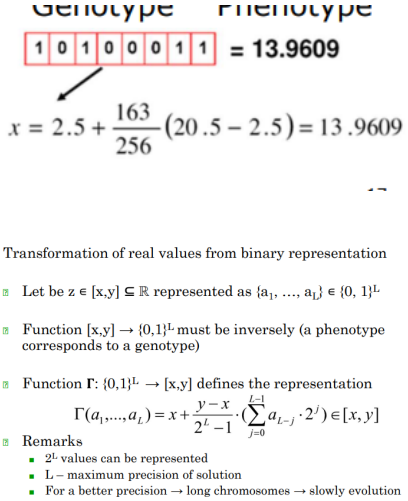
            - phenotype :
                - boolean elements: knapsack problem



- integers



- real number from a range (ex. [2.5,20.5])

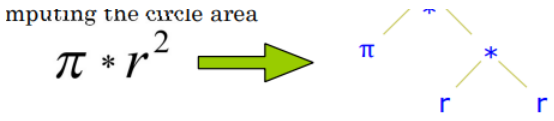


- not-binary
  - integers
    - random → image processing
      - genotype
        - vector of integers from a given range
      - phenotype
        - utility of number in the problem
      - ex: pay a sum s by using different n coins
        - genotype → vector of n integers from range [0, S/value of current coin]

        - phenotype → how many coins of each type must be considered
    - permutation → travelling salesman problem
        - genotype
            - permutation of n elements (n-number of genes)
        - phenotype
            - utility of permutation in the problem
        - ex: traveling salesman problem
            - genotype → permutation of n elements
            - phenotype → visiting order of towns
- class-based → map colouring problem
    - similar to the integer one, but labels are used instead of numbers
    - genotype
        - vectors of labels from a given set
    - phenotype
        - labels' meaning
    - ex: map coloring
        - genotype → vector of n colors (n - number of countries)
        - phenotype → what color has to be used for each country

- continous → function-optimization
    - genotype
        - vector of real numbers
    - phenotype
        - number meaning
    - ex: functio optimisation f:R^n → R

        - genotype → more real numbers
        - phenotype → values of functio f arguments
- tree based → regression problems
    - genotype
        - trees that encode S-expressions
        - internal nodes : functions
            - mathematical
                - artihmetic operators
                - boolean operators
            - statements
                - of a give programming language
                - of other language type
        - leaf → terminals
            - real or boolean values, constants or variables
            - sub-programs
    - phenotype
        - meaning of S-expresssions
    - ex: computing the circle area



Population

Aim

- keeps a collection of possible solutions (candidate solutions)
- repetitions are allowed

- is entirely utilised during selection for recombination

Properties

- (ussualy) fixed dimension u
- divesity
  - number of differeent fitnesses/phenotyes/genotypes

Remarks

- represents the basic unit that evolves
  - the entire population evolves, not only the individuals

Intialisation

- uniformly distributed in the search space (if it is possible)
- binary strings
  - randomly generation of 0 and 1 with a 0.5 probability
- array of real numbers uniformly generated (in a given range)
- permutations
  - generation of identical permutations and making some changes
- trees
  - full method - complete trees
    - nodes of depth d < D_max are randomly intialised by a function from function set F
    - nodes of depth d = D_max are randomly initialised by a terminal from the terminal set T
  - grow method - incomplete trees
    - nodes of depth d < D_max are randomly initialised by an element from F U T
    - nodes of depth d = Dmax are randomly intialised by a terminal from the terminal set T

    - ramped half and half method
        - 1/2 population is inititalised by full methods
        - 1/2 by grow methods
        - by using different depths

Population model

- generational ea
    - each generation creates u offsprings
    - each individual survives a generation only
    - set of parents are totally replaced by set of offspring
- steady-state ea
    - each generation creates a single offspring
    - a single parent (the worst one) is replaced by the offspring
- generation gap
    - proportion of replaced population
    - 1 = u/u , for generational model
    - 1/u, for steady-state

Fitness function

Aim

- reflects the adaptation to enviorment
- quality function or objective function
- associates a value to each candidate solution
    - consequences over selection → the more different values, thebetter

Properties

- costly stage

    - unchanged individuals could not be re-evaluated

Typology

- number of objectives
    - one-objective
    - multi-objective → pareto fronts
- optimisation direction
    - maximisation
    - minimisation
- degree of precision
    - determinstic
    - heuristic’

Examples

- knapsack problem
    - representation → linear, discrete and binary
    - fitness → abs (knapsack’s capacity - weight of selected objects) → min
- problem of paying sum s by using different coins
    - representations → linear , discrete and integer
    - fitness → abs (sum to be paid - sum of selected coins) → min
- tsp
    - representation → linear, discrete, integer, permutation
    - fitness → cost of path → min
- numerical function optimization
    - representation → linear, continous, real
    - fitness → value of function → min/max
- computing the circle’s area

- representation -. tree-based
- fitness → sum of square errors → min

Selection

Aim

- give more reproduction / survival chances to better individuals
  - weaker individuals have chances also because they could contain useful genetic material
- Orients the population to improve its quality
- parent selection (from cucrrent generaton) for reproduction
- survival selection (from parents and offspring) for next generation

Properties

- works at population-level
- is based on fitness only (is independent to representation)
- helps to escape from local optima (because its stochastic nature

Winner strategy

- deterministic - the best wins
- stochastic - the best has more chances to win

Mechanism

- selection for recombination
  - based on entire population
    - proportional selection (based on fitness)
    - rank-based selection
  - based on part of population
    - tournament based selection

- survival selection
  - age-based selection
  - fitness-based selection

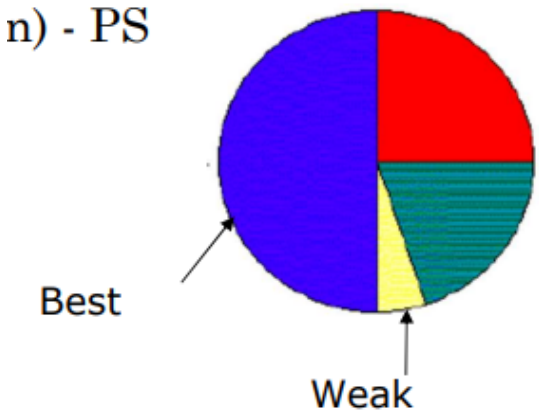
Recombination selection

- proportional selection (fitness-based selection) - PS
  - main idea
    - roulette algorithm for entire population
    - estimation of the copies # of an individual (selection pressure)

$$E(n_i) = \mu \frac{f(i)}{\langle f \rangle}$$, where:

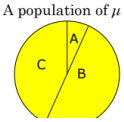
- μ = size of population,
- $f(i)$ = fitness of individual i,
- ⟨f⟩ = mean fitness of population

  - better individuals
    - have more space on roulette
    - have more chances to be selected



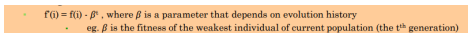
Have more chances to be selected

Ex. A population of $\mu = 3$ individuals

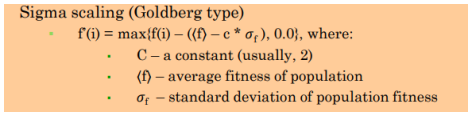


	f(i)	$P_{selPS}(i)$
A	1	1/10=0.1
B	5	5/10=0.5
C	4	4/10=0.4
Sum	10	1

- advantages
  - simple algorithm
- disadvantages
  - premature convergence
  - low selection pressure when fitness function are very similar (at the end of a run)
  - real result are different to theoretical probabilistic distribution
  - works at the entire population level
- solutions
  - fitness scaling
    - windowing



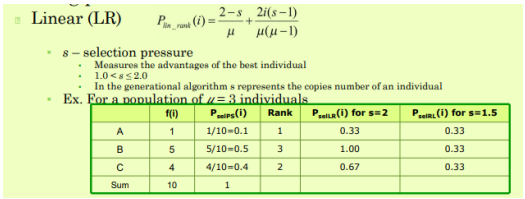
    - sigma scaling



    - normalisation
      - starts by absolute fitnesses
      - standardize these fitness such at the fitnesses
        - belong to [0,1]
        - bet fitness is the smallest one (equal to o)
        - sum of them is 1
  - another selection mechanism
- ranking selection - rs
  - main idea

- sort the entire population based on fitness
  - increases the algorithm complexity, but it is negligible related to the fitness evaluation
- each individual receives a rank
- computes the selection probablities based on these arnks
  - best individual has rank u
  - worst individual has rank 1
- tries to solve the problems of proportional selection by using relative fitness (instead of absolute fitness)

- ranking procedures
  - linear



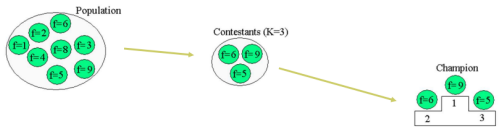
	f(i)	$P_{selPS}(i)$	Rank	$P_{selLR}(i)$ for s=2	$P_{selRL}(i)$ for s=1.5
A	1	1/10=0.1	1	0.33	0.33
B	5	5/10=0.5	3	1.00	0.33
C	4	4/10=0.4	2	0.67	0.33
Sum	10	1			

  - expoenential

$$P_{\exp\_rank}(i) = \frac{1-e^{-i}}{c}$$

    - best individual can have ore than 2 copies
    - c - normalisation factor
      - depends on the population size
      - must be choosen such as the sum of selection probabilities to be 1
- advantages
  - keep the selection pressure constant

    - disadvantages
        - works with the entire population
    - solutions
        - another selection procedure
- tournament selection
    - main idea
        - chooses k individuals → sample of k individuals (k - tournament size)
        - selects the best individual of the sample
        - probability of sample selection depends on
            - rank of individual
            - sample size (k)
                - the larker k is, the greater selection pressure is
            - choosing manner - with replacement (steady-state model) or without replacement
                - selectio without replacement increases the selection pressure
            - for k=2 the time required by the best individual to dominate the population is the same to that from linear ranking selection with s = 2*p, p - selection probablity of the best individual from the population



    - advantages
        - does not work with the entire population
        - easy to implement
        - easy to control the selection pressure by using parameter k

- disadvantages
  - the real results of this selection are different to theoretical distribution (similarly to roulette selection)

Survival selection

- survival selection (selection for replacement)
  - based on age
    - eliminates the oldest individuals
  - based on fitness
    - proportional selection
    - ranking selection
    - tournament selection
    - elitism
      - keep the best individuals from a generation to the next one (if the offspring are weaker thank parents, thank keep the parents

Variation operators

Aim

- generation of new possible solutions

Properties

- works at individual level
- is based on individual representation (fitness independent)
- helps the exploration and exploitation of the search space
- must produce valid individuals

Typology

- arity criterion

- arity 1 → mutation operators
- arity > 1 → recombination / crossover operators

Mutation

Aim

- reintroduce in population the lost genetic material
- unary search operator (continous space)
- introduces the diversity in population (discrete space)

Properties

- works at genotype level
- based on random elements
- responsible to the exploration of promising regions of the search space
- responsible to escape from local optima
- must introduce small and stochastic changes for an individual
- size of mutation must be controllable
- can probabilistic take place at the gene level

Representations

- binary representation
  - strong mutation - bit-flipping
  - weak mutation
- integer representation
  - random resetting
  - creep mutation
- permutation representation
  - insertion mutation

    - swap mutation
    - inverse mutation
    - scramble mutation
    - k-opt mutation
- real representation
    - uniform mutation
    - non-uniform mutation
        - gaussian mutation
        - caucht mutation
        - laplace mutation
- tree-based representation
    - grow mutation
    - shrink mutation
    - switch mutation
    - cycle mutation
    - koza mutation
    - mutation for numerical terminals

Mutation (binary representation)

A chromosome $c = (g_1, g_2, ..., g_L)$ becomes $c' = (g_1', g_2', ..., g_L')$,
where $g_i, g_i' \in \{0,1\}$, for $i = 1, 2, ..., L$.

- strong mutation - bit flipping
    - main idea
        - changes by probability pm (mutation rate) all the genes in their complement
            - 1 → 0, 0 → 1

- ex: a chromosome L = 8 genes, pm = 0.1

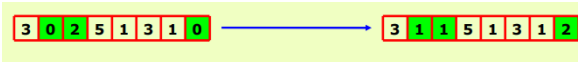
- weak mutation
  - main idea
    - changes by probability pm some of the genes in 0 or 1
      - 1 → 0/1, 0 → 1/0
    - ex: a chromosome of L= 8 genes, pm=0.1

Mutation (Integer representation_

A chromosome $c = (g_1, g_2, ..., g_L)$ becomes $c' = (g_1', g_2', ..., g_L')$,
where $g_i, g_i' \in \{val_1, val_2, ..., val_k\}$ for $i = 1, 2, ..., L$.

- random resetting mutation
  - main idea
    - the value of a gene is changed (by probability pm) into another value (from the definition domain)

- creep mutation
  - main idea
    - the value of a gene is change by adding a positive/negative value
      - new value follows a symmetric distribution
      - the performed change is very small



Mutation (permutation representation)

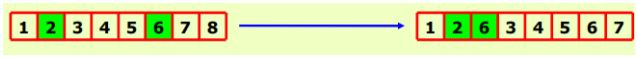
A chromosome $c = (g_1, g_2, ..., g_L)$ with $g_i \neq g_j$ for all $i \neq j$ becomes

$c' = (g_1', g_2', ..., g_L')$, where $g_i, g_i' \in \{val_1, val_2, ..., val_L\}$,

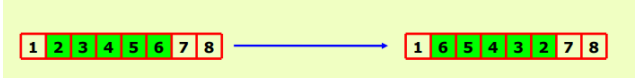
for $i = 1, 2, ..., L$ s. a. $g_i' \neq g_j'$ for all $i \neq j$.

- swap mutation
  - main idea
    - randomly choose 2 genes and swap their values

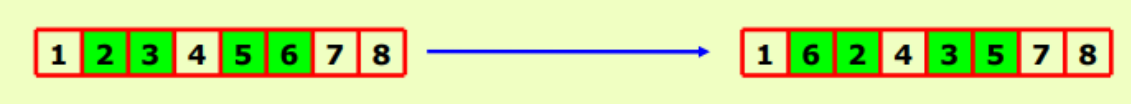
- insertion mutation
  - main idea
    - randomly choose 2 genes gi and gj with j>i
    - insert gene gj after gene gi



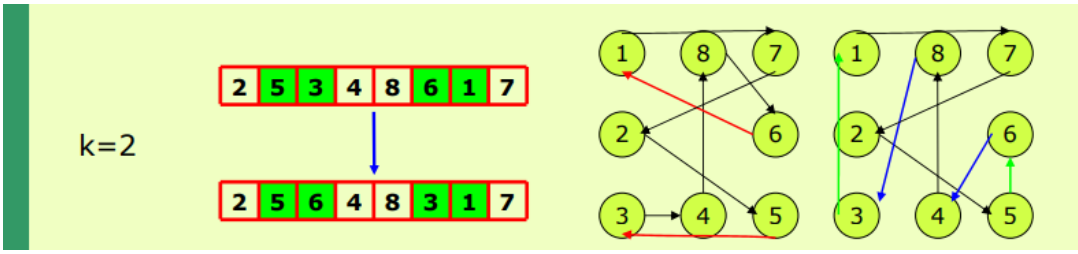
- inversion mutation
  - main idea
    - randomly choose 2 genes and ivnerse the order of genes between them (sub-string of genes)



- scramble mutation
  - main idea
    - randomly choose a (continous or discontinous) sub-array of genes and re-organise that genes



- k-opt mutation
  - main idea
    - choose 2 disjoint sub-strings of length k
    - interchange 2 elements of these sub-strings

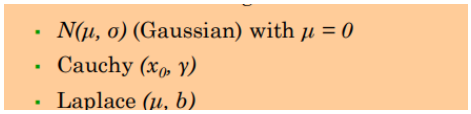


Mutation (real representation)

A chromosome $c = (g_1, g_2, ..., g_L)$ becomes $c' = (g_1', g_2', ..., g_L')$, where $g_i, g_i' \in [a_i, b_i]$, for $i=1, 2, ..., L$.

- uniform mutation
  - main idea
    - gi' is changed by probability pm into a new value that is randomly uniform generated in [ai,bi] range

- non-uniform mutation
  - main idea
    - the value of a gene is probabilistically pm changed by adding a positive/negative value
      - the added value belongs to a distribution of type



      - and it is re-introduced in [ai,bi] range (if it is necessary) - clamping

Recombination

Aim

- mix the parents information

Properties

- the offspring has to inherit something from both parents
  - selection of mixed information is randomly performed
- operator for exploitation of already discovered possible solutions
- the offspring can be better, the same or weaker than their parents
- its effects are reducing while the search converges

Types

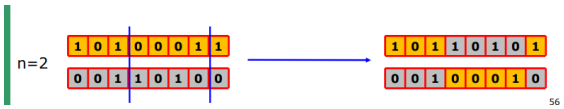
- binary and integer representation
  - with cutting points
  - uniforme
- permutation representation
  - order crossover (version 1 and version 2)
  - partially mapped crossover

    - cycle crossover
    - edge-based crossover
- real representation
    - discrete
    - arithmetic
        - sinular
        - simple
        - complete
    - geometric
    - shuffle crossover
    - similated binary crossover
- tree-based representation
    - sub-tree based crossover

Recombination (binary and integer representation)

- From 2 parent chromosomes
    - $p_1 = (g_1^1, g_2^1, ..., g_L^1)$ and $p_2 = (g_1^2, g_2^2, ..., g_L^2)$
- 2 offspring are obtained
    - $c_1 = (g_1', g_2', ..., g_L')$ and $c_2 = (g_1'', g_2'', ..., g_L'')$,
    - where $g_i^1, g_i^2, g_i', g_i'' \in \{0, 1\} / \{val_1, val_2, ..., val_k\}$, for $i = 1, 2, ..., L$

- n-cutting point crossover
    - main idea
        - choose n cutting-points (n<L)
        - cut the parents through these points
        - put togheter the resulted parts, by alternating the parents



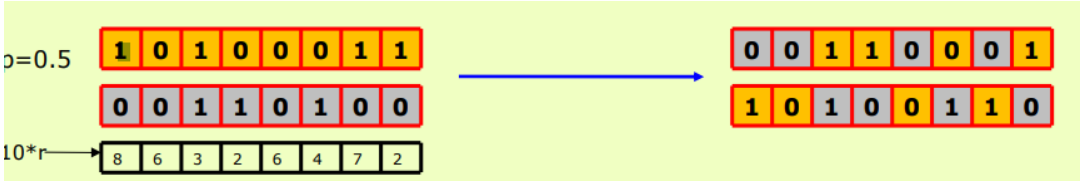
- properties

- Average of values encoded by parents = average of values encoded by offspring
  - Eg binary representation on *4* bits of integer numbers – XO with $n = 1$ after second bit
    - $p_1 = (1, 0, 1, 0), p_2 = (1, 1, 0, 1)$
    - $c_1 = (1, 0, 0, 1), c_2 = (1, 1, 1, 0)$
    - $val(p_1) = 10, val(p_2) = (13) \rightarrow (val(p_1) + val(p_2)) / 2 = 23 / 2 = 11.5$
    - $val(c_1) = 9, val(c_2) = (14) \rightarrow (val(c_1) + val(c_2)) / 2 = 23 / 2 = 11.5$
  - Eg. Binary representation on *4* bits for knapsack problem ($K=10$, *4* items of weight and value: *(2,7), (1,8), (3,1), (2,3)*)
    - $p_1 = (1, 0, 1, 0), p_2 = (1, 1, 0, 1)$
    - $c_1 = (1, 0, 0, 1), c_2 = (1, 1, 1, 0)$
    - $val(p_1) = 8, val(p_2) = 18 \rightarrow (val(p_1) + val(p_2)) / 2 = 26 / 2 = 13$
    - $val(c_1) = 10, val(c_2) = 16 \rightarrow (val(c_1) + val(c_2)) / 2 = 26 / 2 = 13$
- Probability of $\beta \approx 1$ is the largest one

$$\beta = \left| \frac{val(d_1) - val(d_2)}{val(p_1) - val(p_2)} \right|$$

  - Contracting crossover $\beta < 1$
    - Offspring values are between parent values
  - Expanding crossover $\beta > 1$
    - Parent values are between offspring values
  - Stationary crossover $\beta = 1$
    - Offspring values are equal to parent values

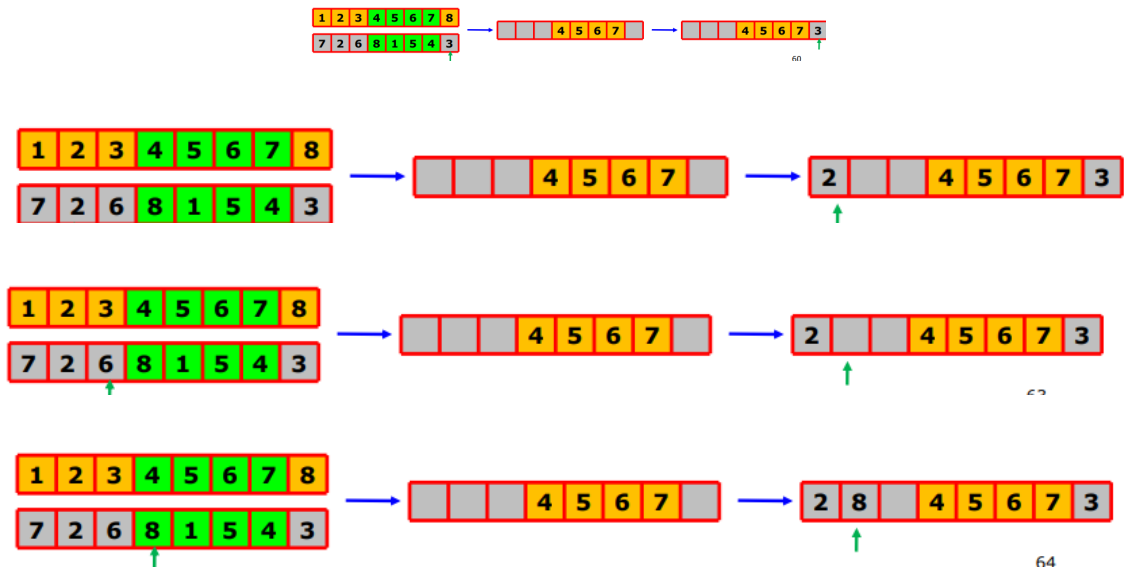
- uniform crossover
  - main idea
    - each gene of an offspring comes from a randomly and uniform selected parent:
      - for each gene a uniform random number r is generated
      - if r < probability p (usually, p=0.5) c1 will inherit that gene from p1 and c2 from p2, otherwise c1 will inherit from p2 and c2 from p1

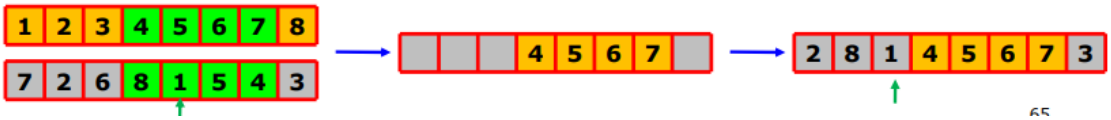


Recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, ..., g_L^1)$ and $p_2 = (g_1^2, g_2^2, ..., g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', ..., g_L')$ and $c_2 = (g_1'', g_2'', ..., g_L'')$,
  - Where $g_i^1, g_i^2, g_i', g_i'' \in [1, L] \cap \mathbb{Z}$, for $i = 1, 2, ..., L$.

- order crossover
  - main idea
    - offspring keep the order of genes from parents
    - choose a substring of genes from p1
    - copy the substring from p1 into offspring d1 (on coressponing posistion
    - copt the genes of p2 in offspring d1:
      - starting with the first position after substring
      - respecting gene's order from p2 and
      - re-loading the genes from start (if the end of chromosome is reached)

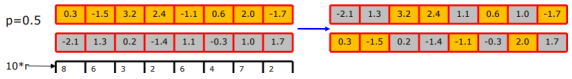




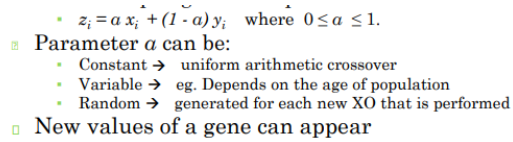
Recombination (real representation)

- From 2 parent chromosomes
  - $p_1 = (g_1^1, g_2^1, ..., g_L^1)$ and $p_2 = (g_1^2, g_2^2, ..., g_L^2)$
- 2 offspring are obtained
  - $c_1 = (g_1', g_2', ..., g_L')$ and $c_2 = (g_1'', g_2'', ..., g_L'')$,
  - Where $g_i^1, g_i^2, g_i', g_i'' \in [a_i, b_i]$, for $i = 1, 2, ..., L$

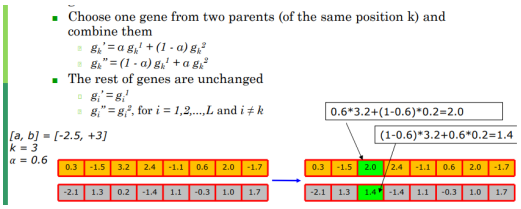
- discrete crossover
  - main idea
    - each gene offspring is taken (by the same probability, p-0.5) from one of the parents
    - similarly to uniform crossover from binary/integer representation
    - the absolute values of genes are not changed



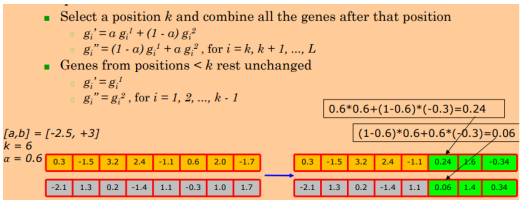
- arithmetic crossover
  - main idea
    - create offspring between parents



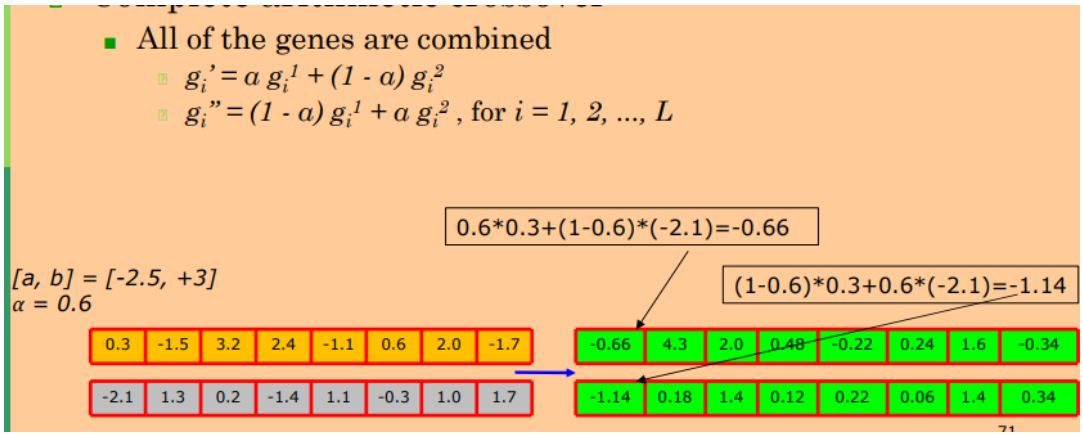
  - types
    - singular arithmetic crossover



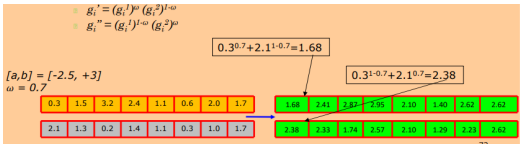
- simple arithmetic crossover



- complete arithmetic crossover

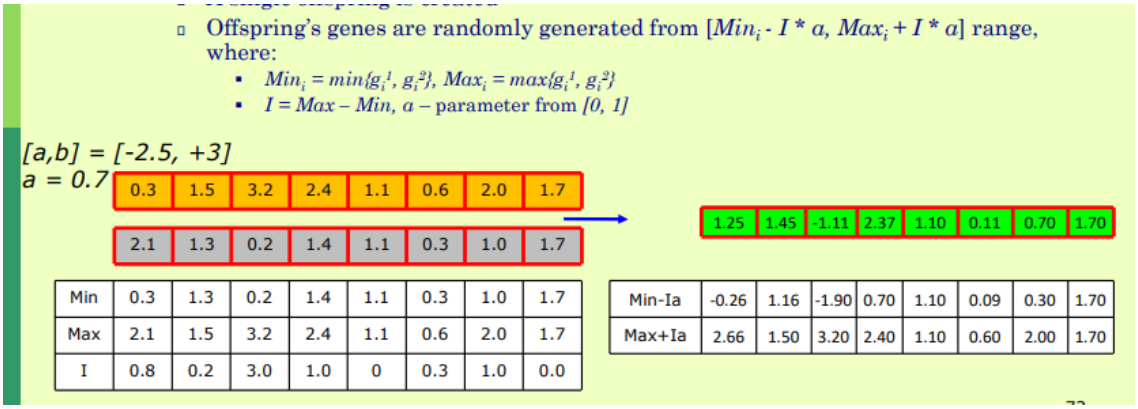


- geometric crossover
  - main idea
    - each gene of an offspring represents the product between parent's genes, each of them by a given exponent w and 1-w, respectively (where w is a real positive number ≤1)



- blend crossover

- main idea
  - a single offspring is created



Recombination or mutation

- intense debate
  - which is the best operator? which is the most necessary opeartor? which is the most important operator?
  - depend on problem ,but in general, is better to use both, each of them having another role (purpose).
  - ea's with mutations only are possible, but Ea with with crossover only are not possible
- search aspects
  - exporation → discovering promising regions in the search space (accumulating useful information about the problem)
  - exploitation → optimizing in a promising region of the search space (by using the existent information)
  - cooperation and competition must exist between these 2 aspects
- recombination
  - expoitation operator → performs a large jump into a region somewhere between the regions associated to parents
  - binary/n-ary operator that can combine information from 2/more parents

    - operator that does not change the frequency of values from chromosomes at the population levels
- mutation
    - exploration operator → performs small random diversions, remaining in a neighbourhood of parent
    - operator that can introduce new genetic information
    - operator that change the frequency of values from chromosomes at the population level

Stop conditnon

- choosing a stop conditnon
    - an optimal solution was found
    - the physical resources were ended
        - a given number of fitness evaluation has been performed
    - the user resources were ended
        - several generations without improvements have been born

Evaluation

- after more runs
    - statistical measures are computed
        - average of solutions
        - median of solutions
        - best solution
        - worst solution
        - standard deviation of solutions - for comparasions
    - the number of independent runs must be large enough

Analysis of complexity

- the costly part → fitness evaluation

Advantages

- ae have a general sketch for al the problems only representation and fitness function are changed
- ae are able to give better results than clasical optimisation methods because
  - they do not require linearization
  - they are not based on some presumptions
  - they do not ignore some possible solutions
- ae are able to explore more possible solutions than human can

Disadavantages

- large running time

Applications

- vehicle design
  - material composition
  - vehicle shape
- enginnering desing
- robotics
- hardware evolution
- telecommunicaton optimisation
- biometric invetions
- traffic and transporation routing
- pc games
- crypthography
- genetics

- chemical anaylsis of kinematics
- financial and marketing strategies

Types

- evolutionary strategies
- evolutionary programming
- genetic algorithms
- genetic programming

Particle Swarm Optimization & Ant Colony optimisation

Nature-inspired algorithms

- swarm intelligence (collective intelligence)
  - a group of individuals that interact in order to achieve some objectives by collective adaptation to a global or local environment
  - a computational metaphor inspired by
    - birds flying v shape
    - ants that are searching for food
    - bees swarms that are constructing their nests
    - schools of fish
  - because
    - control is distributed among more individuals
    - individuals local communicate
    - system behavior transcends the individual behaivor
    - system is robust and can adapt to environment changes
  - social insects (2% total)

        - ants
            - 50% of social insects
            - 1 ant has ~ 1mg → total weight of ants ~ total weight of humans
            - live for over 100 millions of years
        - termites
        - bees
    - swarm(group)
        - more individuals, apparently un-organized, that are moving in order to form a group, but each individual seems to move in particular direction
        - inside the group can appear some social processes
        - the collection is able to do complex tasks
            - without external guide or external control
            - without a central coordination
        - the collection can have performances better than the independent individuals
    - collective adaptation → self-organization
        - set of dynamic mechanisms that generate a global behaviour as a result of interacton among individual components
        - rules that specify this interaction are executed based on local infomration only, withoout global references
        - global behavior is an emergent property of the system (and not one external imposed)

PSO - theoretical aspects

- proposed by
    - kennedy and eberhart in 1995
    - inspired by social behaviour of bird swarms and school of fish

- search is
  - cooperative, guided by the relative quality of individuals
- search operators
  - a kind of mutation
- special elements
  - optimisation method based on
    - populations of particles that search the optimal solution
    - cooperation (instead of concurrence, like in EAs case)
  - each particle
    - moves (in the search space) and has a velocity
    - retains the place where it has obtained the best result
    - has associated a neighbourhood of particles
  - particles cooperate
    - exchange information among them (Regarding the discovering performed in the places already visited)
    - each particle knows the fitness of neighbours such as it can use the position of the best neigbour for adjusting its velocity
- main idea
  - cognitive behaviour → an individual remembers past knowledge (has memory)
  - social behaviour → an individual relies on the knowledge of other members of the group

PSO - algorithm

General sketch

1. creation of the initial population of particles
2. evaluation of particles

3. for each particle
4. if the stop condition are not satisfied, go back to step 2, otherwise STOP/

Creation of the initial population of particles

- each particle has associated
  - a position - possible solution of the problem
  - a velocity - changes a position into another position
  - a quality function (fitness)
- each particle has to
  - interact (exchange information) with its neighhbours
  - memorise a previous position
  - use the information in order to take decisions
- intialisation
  - random positions
  - null / random velocities

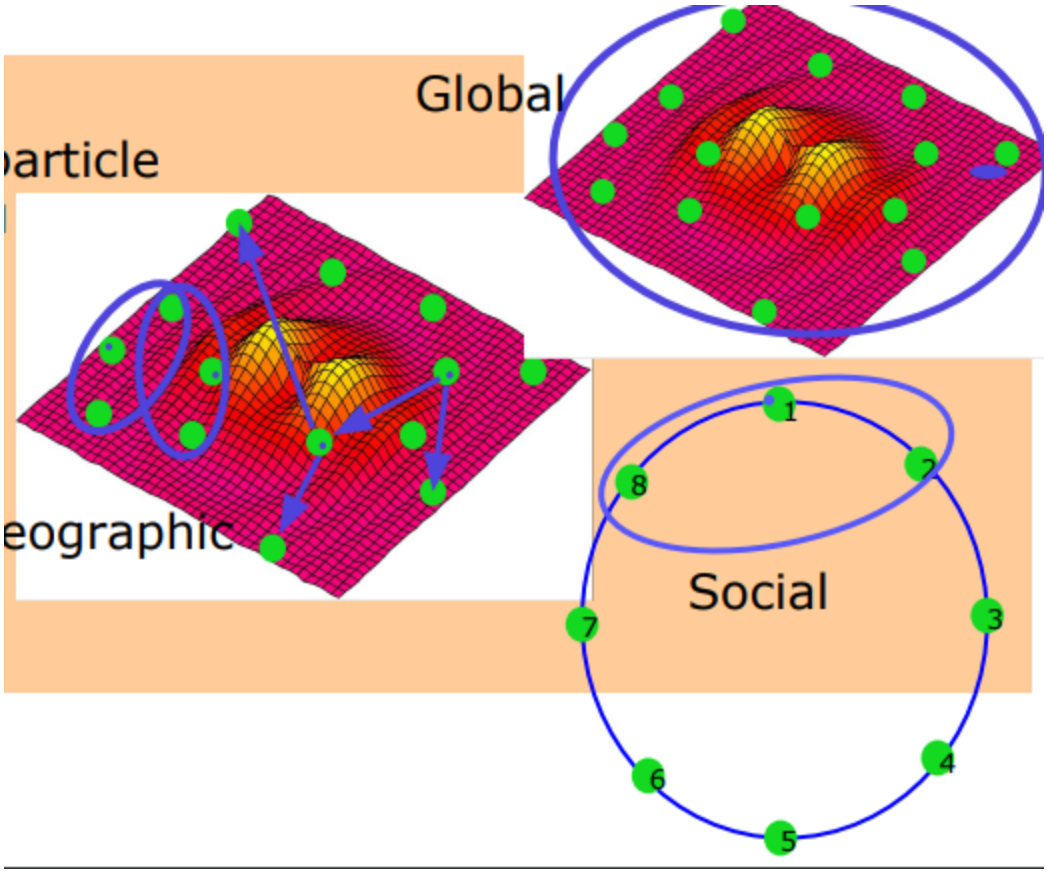
Evaluation of particles

- depends on problem

For each particle p

- update the memory
  - identify the best particle of the swarm (g_best) / of current neighbourhood (l_best)
  - identify the best position (with the best fitnesss) reached until now (p_best)
  - neighbourhood for a particle
    - spread
      - local

        - global
    - type
        - geographical
        - social
        - circular



- update the velocity v_i and position x_i = (x_i1, x_i2, ...,x_id) (on each dimension)

dimension)
- $v_{id} = w * v_{id} + c_1 * rand() * (p_{Best\ d} - x_{id}) + c_2 * rand() * (g_{Best\ d} - x_{id})$
- $x_{id} = x_{id} + v_{id}$
- where:
  - $i=1,N$ (N – total number of particles/swarm size);
  - $d = 1,D$
  - w – inertia coefficient (Shi, Eberhart)
    - $w*v_{id}$ inertial factor → forces the particle to move in the same direction until now (*audacious*)
    - Balance the search between global exploration (large w) and local exploration (small w)
    - Can be constant or descending (while the swarm is getting old)
  - c_1 - cognitive learning coefficient
    - $c_1 * rand() * (p_{Best\ d} - x_{id})$ – cognitive factor → forces the particle to move towards its best position (conservation)
  - c_2 - social learning coefficient
    - $c_2 * rand() * (g_{Bestd} - x_{id})$ – social factor → forces the particle to move towards the best neighbour (follower)
  - c_1 and c_2 can be equal or different ($c_1 > c_2$ şi $c_1 + c_2 < 4$ – *Carlise, 2001*)

- each component of velocity vector must belong to a given range [-v_max, v_max] in order to keep the particles inside the search space

PSO - Properties

PSO principles

- proximity - the group has to performed computing in space and time
- quality - the group has to be able to answer the quality of the environment
- stability - the group has not to change its behaviour at each environment
- adaptability - the group has to be able of changing its behaviour when the cost of change is not prohibit

Differences from EAs

- there is no recombination operator - information exchange takes place based on particle's experience and based on the bet neighbour (not based on the parents selected based on quality only)
- position update ~ mutation
- selection is not utilized - survival is not based on quality (fitness)

PSO versions

- PSO binary and discrete

    - version for a discrete search space
    - position of a particle
        - possible solution of the problem → binary string
        - changes based on the velocity of particle
    - velocity of a particle
        - element from a continous space
        - changes based on standard PSO principles
        - can be viewed as changing probability of the coressponding bit from the particle's solution

$$x_{ij} = \begin{cases} 1, & \text{if } \tau < s(v_{ij}) \\ 0, & \text{otherwise} \end{cases}, \text{where } s(v_{ij}) = \frac{1}{1+e^{-v_{ij}}}$$

- PSO with more social learning coefficients
- PSO with heterogenous particles
- Hierarchic PSO

Risks

- particles have the trend to group in the same place
    - raid convergence and the imposibility to escape from local optima
    - solution
        - re-initialisation of some particles
- particles move through infeasible regions

Analysis of PSO algorithm

- dynamic behaviour of the swarm can be analysed by 2 index
    - dispersion index

        - measures the spreading degreee of particle around best particle of the swarm
        - average of absolute distances (on each dimension) between each particle and the best practice of the swarm
        - explains the cover degree (small or spread) of the search space
    - velocity index
        - measures the moving velocity of the swarm into a iteration
        - average of absolute velocities
        - explain how the swarm moves (aggresive or slow)

PSO - applications

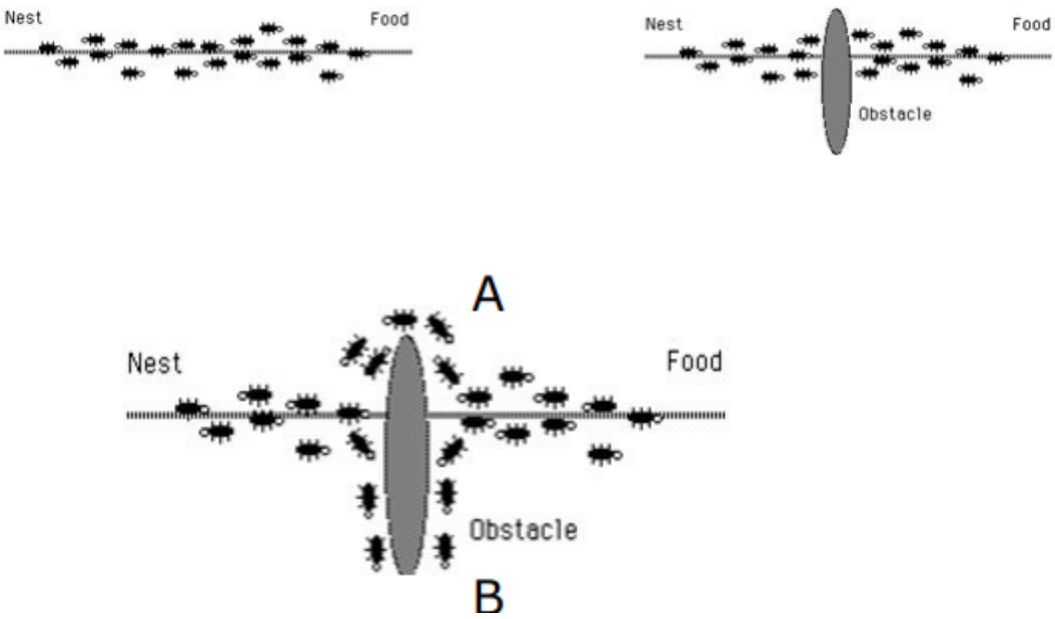
- control and design of antenna
- biological, medical and pharmaceutical applications
    - analysis of tremor in parkinson’s diseases
    - cancer classification
    - prediction of protein structure
- network communcation
- combinatorical optimisation
- financial optimisation
- image and video analysis
- robotics
- planning
- network secturity, intrusion detection, crypthography
- signal processing

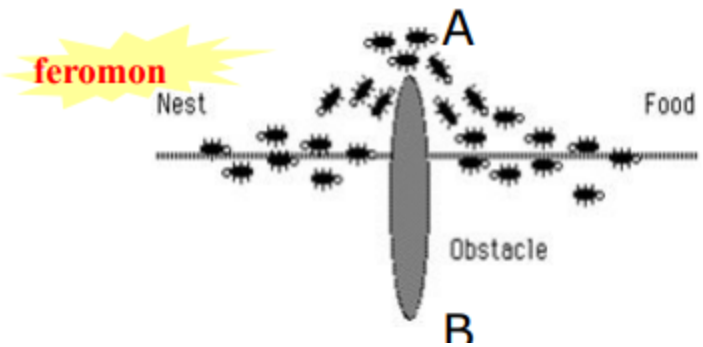
ACO - theoretical aspects

- proposed

    - by colorni and dorigo in 1991 for solving discrete optimisation problems - tsp - as a comparasion for EAs
    - inspired by social behaviour of ants that search a path from their nest and food
    - why ants?
        - colony system
        - labor division
        - social behaviour is very complex
- search
    - cooperative, guided by the relative quality of the individuals
- search operators
    - construtive ones, adding elements in solution
- special elements
    - the optimisaton problem must be transformed into a problem of indentyfin the optimal path in an oriented graph
    - ants construct the solution by walking through the graph and put pheromones on some edges
    - optimisation method based on
        - ant colonies that search the optimal solution
        - cooperation
    - each ant
        - moves (in the search space) and puts some pheromones on its path
        - memorises the path
        - selects the path based on
            - the existing pheromones on that path
            - heuristic information associated to that path

        - cooperates with other ants through the pheromone trail (that corresponds to a path) that
            - depends on the solution quality
            - evaporates while the time is passing
- natura ants
    - an ant colony start to search for some food
    - at a moment, an obstacle appears
    - the ants surround the obstacle either on path A or path B
    - because path A is shorter, the ants of this path will perform more are and therefore, will put more pheromones
    - pheromone concentration will quickly increase on path A (relative to path B) such as the ants from path B will re-orientet to path A
    - because the ants do not follow path B and because the pheromone trail evaporets, the trail from path B will disappear
    - therefore, the ants will take the shortest path (path a)





- artificial ants look like natural ants
  - walk from their nest towards food
  - discover the shortest path based on pheromone trail
    - each ant performed some random moves
    - each ant put some pheromones on its path
    - each ant detects the path of “boss ant” and tends to follow it
    - increasing the pheromone of a path will determine to increase the proabability to follow that path by more ants
  - they have some improvements
    - has memory
      - retains performed moves → has a proper state (retaining the history of decisions)
        can come back to their nest (based on pheromone trail)
    - are not completly blind - can appreciate the quality of their neighbour space
    - perform move in a discrete spcae
    - put pherommones based on the identified solution, also
- pheromone trail plays the role of
  - a collective dynamic and distributed memory
  - a repository of the most recent ants experiences of searching food

- ants can indirectly communicate and can influence each-other by changing the chemical repository in order to identify the shortest path from nest to food

ACO - algorithm

```
while iteration < maximum # of iterations
1. initialisation
2. while # of steps required to identify the optimal solution is
    - for each ant of the colony
        - increase the partial solution by an element (ant moves
        -  change locally the pheromone trail based on the last
3. change the pheromone trail on the paths travesed by
    - all ants / best ant
4. return the solution identified by best ant
```

- 3 versions - differences:
  - rules for transforming a state into another state (moving rules for ants)
  - moment when the ants deposits pheromones
    - while the solution is constructed
    - at the end of solution's construction
  - which ant deposits pheromonts
    - all the ants
    - the best ant only
- versions
  - ant system (AS)
    - all ants deposit pheromones after a solution is completely constructed (global collective update)
  - maxmin ant system (MMAS) ~ AS, but
    - the best ant only deposits pheromones after a solution is completely constructed (global update of the leader)

        - deposited pheromones is limited to a given range
    - ant colony system (ACO) ~ AS, but
        - all the ants deposit pheromones at each step of solution construction (collective local update)
        - the best ant only deposits pheromone after the solution is complete constructed (global update of the leader)

ACO - Example

Traveling salesman problem - TSP

- finds the shortest path that visits only once all the n given cities
- Initialization
    1. t := 0 (time)
    2. for each edge (i,j) 2 elements are initialised

$\tau_{ij}^{(t)} = c$ (intensity of pheromone trail on edge (I,j) at time t)

$\Delta\tau_{ij} = 0$ (quantity of pheromone deposited on edge (i,j) by all the ants)

    c. m ants are randomly placed in n city-nodes (m ≤ n)
    d. each ant updates its memory (list of visited cities)
        a. add in the list the starting city
- while # number of steps required to identify the optimal solution is not performed (# of steps = n)
    1. for each ant of the colony
        a. increase the partial solution by an element (ant moves one step)

- Increase the partial solution by an element (ant moves one step)
  - Each ant k (from city i) selects the next city j:

$$j = \begin{cases} \arg\max_{l \in permis_k} \left\{ [\tau_{il}]^{\alpha} [\eta_{il}]^{\beta} \right\}, & \text{if } q \le q_0 \\ J, & \text{otherwise} \end{cases}$$

Random proportional rule

- where:
  - q – random uniform number from [0,1]
  - q_0 – parameter, $0 \le q_0 \le 1$ ($q_0 = 0 \rightarrow$ AS/MMAS, otherwise ACO)
  - J is a city selected by probability

Pseudo-random proportional rule

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}^{(t)}]^{\alpha} [\eta_{ij}]^{\beta}}{\sum_{s-allowed_k(t)} [\tau_{is}^{(t)}]^{\alpha} [\eta_{is}]^{\beta}}, & j - allowed \\ 0, & otherwise \end{cases}$$

where:
- p_{ij}^{k} probability of transition of ant k from city i to city j
- $\eta_{ij} = \frac{1}{d_{ij}}$ - visibility from city I towards city j (attractive choice of edge (*i,j*))
- $allowed_k$ – cities that can be visited by ant k at time t
- α – controls the trail importance (how many ants have visited that edge)
- β – controls the visibility importance (how close is the next city)

ii. chance the local pheromone trail based on the last element added in solution

$$\tau_{ij}^{(t+1)} = (1-\varphi)\tau_{ij}^{(t)} + \varphi * \tau_0$$

- where:
  - φ – pheromone degradation coefficient; $\varphi \in [0,1]$; for $\varphi = 0 \rightarrow$ AS/MMAS, otherwise ACO
  - τ_0 – initial value of pheromone
  - *(i,j)* – last edge visited by ant

- chance the pheromone trail from
  1. paths covered by all ants

For each edge

- Compute the unit quantity of pheromones put by the k[th] ant on edge (i,j)
  - $\Delta\tau_{ij}^{k} = \begin{cases} Q \\ L_k \\ 0 \end{cases}$ if the k^{th} ant used the edge (i,j)
  - Q – quantity of pheromone deposited by an ant.
  - L_k – length (cost) of tour performed by the k^{th} ant
- Compute the total quantity of pheromone from edge (ij) $\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$
- Compute the intensity of pheromone trail as a sum of old pheromone evaporation and the new deposited pheromone

$$\tau_{ij}^{(t+n)} = (1-\rho) * \tau_{ij}^{(t)} + \Delta\tau_{ij}$$

  - Where ρ $(0<\rho<1)$ – evaporation coefficient of pheromone trail from a complete tour to another complete tour

2. the best path of the best ant (MMAS)

For each edge of the best path

- Compute the unit quantity of pheromone deposited by the best ant on edge (ij)
  - $\Delta\tau_{ij} = \frac{1}{L_{best}}$
  - L_{best} – length (cost) of the best path
    - Of current iteration
    - Over all executed iteration (until that time)
- Compute the intensity of pheromone trail as sum of old pheromone evaporation and the new deposited pheromone
  - Where ρ $(0<\rho<1)$ – – evaporation coefficient of pheromone trail from a complete tour to another complete tour
  - τ_{min} şi τ_{max} – limits (inferior and superior) of pheromone;
    - For $\tau_{min} = -\infty$ and $\tau_{max} = +\infty \rightarrow$ ACO, otherwise MMAS

$$\tau_{ij}^{(t+n)} = \left[(1-\rho) * \tau_{ij}^{(t)} + \rho * \Delta\tau_{ij}^{best}\right]_{\tau_{min}}^{\tau_{max}}$$

ACO - properties

- properties
  - iterative algorithm

    - algorithm that progressively constructs the solution based on
        - heuristsic information
        - pheromone trail
    - stochastic algorithm
- advantages
    - run continous and real-time adaptive change input
    - positive feedback haleps to quickly discover of solution
    - distributed computing avoids premature convergence
    - greedy heuristic helps to identify an acceptable solution from the first stages of search
    - collective interaction of individuals
- disadvantages
    - slowly convergence vs other heuristic search
    - for tsp instances with more than 75 cities it finds weak solutions
    - in as there is no central process to guide the search toward good solutions

ACO - applications

- optimal paths in graphs
- problem of quadratic assigments
- problem of network simulation
- transport problems

Support Vector Machines

Definition

- developed by Vapnik in 1970
- popularised after 1992

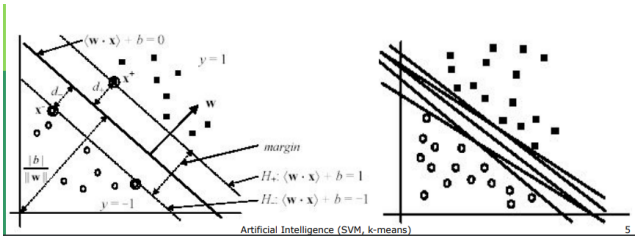
- linear classifiers that identify the hyperplace that separates the positive and negative classes
- have a theoretical foundation
- work very well for large data (text mining, image analysis)

  - SVM finds a linear function $f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$, ($\mathbf{w}$ –weight vector) such as

$$y_i = \begin{cases} 1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

  - $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ ☐ decision hyperplane that separates the two classes

- there are more hyper-planes
  - which is the best hyper-place
- svm searches the hyper-plane with the largest margin (that minimises the generalisation error)
  - SMO (sequential minimal optimization) algorithm

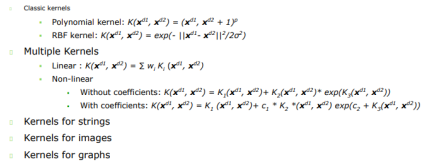


Solved problems

- classification problems
  - more cases (based on the data type)
    - linear separable
      - separable → error = 0
      - non-separable
        - constraints are relaxed → some errors are allowed

        - c - penalisation coefficient
    - non-linear separable
        - input space is transformed (mapped) into a space of more dimensions (feature space) by using kernel functions - in this new space the data becomes linear separable
        - in svms the kernel function computes the distance among 2 points
            - kernel ~ similarity function

        - possible kernels



SVM parameters setting

- penalisation coefficient C
    - C - small → slowly convergence
    - C - large → fast convergence
- kernel parameters
    - if m (# of attributes) is larger then n (# of data)
        - svm with a linear kernel (svm without kernel

$$K(x^{d1}, x^{d2}) = x^{d1} \cdot x^{d2}$$

    - if m is large and n is medium

        - sm with gaussian kernel/rbf kernel
    - if m is small and n is large
        - add new attributes and use svm with linear kerne

Structured SVMs

Machine Learning

- simple svm
    - any type of inputs
    - numerical outputs
- structured svm
    - any type of inputs
    - any type of outputs

Applications

- natural language processing
    - automatic translation
    - syntactic and/or morphologic analysis of sentences
- bioinformatic
    - prediction of secondary structures
    - prediction of enzyme function
- speech processing
    - automatic transcriptions
    - tranformation of texts in voice
- robotics
    - planning

Advantages

- can work with any type of data
  - kernel functions that creates new attributes
- if the problem is convex svm finds a unique solution → global optima
  - ann can ssociate more solutions → local optima
- automatic selection of learnt model (by support vectors)
  - in ann hidden layers have to be configured apriori
- avoid overfitting
  - ann have overfitting problems even when cross-validation is involved

Difficulties

- real attributes only
- binary classification problems only
- difficult mathemmatical background

Unsupervised learning - Agglomerative hierarchical clustering

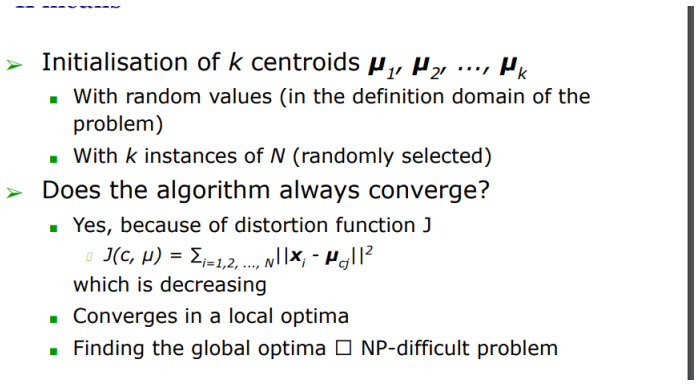
Distance between 2 clusters c_i and c_j:

- Simple link ☐ minimal distance between the objects of 2 clusters
  - $d(c_i, c_j) = min_{xi1 \in ci,\ xi2 \in cj} sim(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
- Complete link ☐ maximal distance between the objects of 2 clusters
  - $d(c_i, c_j) = max_{xi1 \in ci,\ xi2 \in cj} sim(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
- Average link ☐ mean of distances between the objects of 2 clusters
  - $d(c_i, c_j) = 1 / (n_i * n_j) \sum_{xi1 \in ci} \sum_{xi2 \in cj} d(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
- Average link over group ☐ distance between the means (centroids) of 2 clusters
  - $d(c_i, c_j) = \rho(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j)$, ρ – distance, $\boldsymbol{\mu}_j = 1/n_j \sum_{xi \in cj} \mathbf{x}_i$

Unsupervised learning - K-Mean

K-means (Lloyd algorithm / Voronoi iteration)

- Suppose that k clusters will form
- Initialise *k* centroids $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$
  - A centroid $\boldsymbol{\mu}_j$ ($i=1,2, \ldots, k$) is a vector of *m* values (*m* - # of features)
- Repeat until convergence
  - Associated to each instance the nearest centroid ☐ for each instance $\boldsymbol{x}_i$, $i = 1, 2, \ldots, N$
    - $c_i = arg\ min_{j = 1, 2, \ldots, k} ||\boldsymbol{x}_i - \boldsymbol{\mu}_j||^2$
  - Re-compute the centroids by moving them in the mean of instances associated to it ☐ for each cluster c_j, $j = 1, 2, \ldots, k$
    - $\boldsymbol{\mu}_j = \Sigma_{i=1,2, \ldots N} 1_{ci=j}\, \boldsymbol{x}_i \,/\, \Sigma_{i=1,2, \ldots N}\, 1_{ci=j}$



- Initialisation of *k* centroids $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$
  - With random values (in the definition domain of the problem)
  - With *k* instances of *N* (randomly selected)
- Does the algorithm always converge?
  - Yes, because of distortion function J
    - $J(c, \mu) = \Sigma_{i=1,2, \ldots, N} ||\boldsymbol{x}_i - \boldsymbol{\mu}_{cj}||^2$

    which is decreasing
  - Converges in a local optima
  - Finding the global optima ☐ NP-difficult problem

Unsupervised algorithms - Clustering based on minimum spanning tee

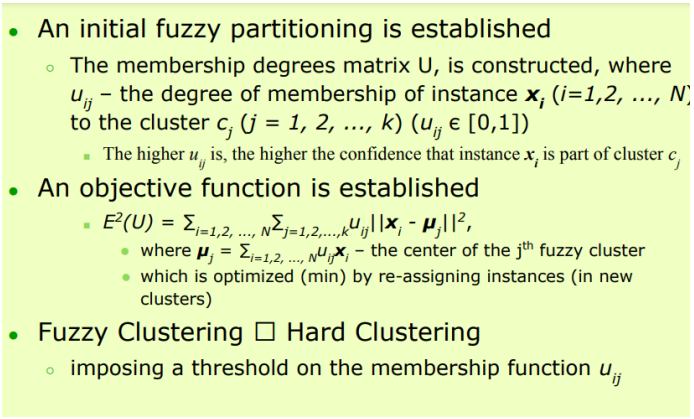
- construct the minimum spanning tree of data
- eliminate from the tree the longest edges and form clusters

Unsupervised learning - Nearest neighbour

- some instances are labeled

- it is repeated until all instances are labeled
  - an unlabeled instance will be included in the closest instance cluster
  - if the distance between the unlabeled instance and the labeled one is less than a threshold

Unsupervised learning - Fuzzy clustering



Rule Based Systems

- knowledge based system → rule based systems in uncertain environments
- computational systems - composed of 2 parts
  - knowledge base
    - specific information of the domain
  - inference engine
    - rules for generating new information
    - domain-independent algorithms

Knowledge base

Content

- information (in a given representation) about environment

- required information for problem solving
- set of propositions that describe the environment

Classification

- perfect information
  - classical logic
- imperfect information
  - non-exact
  - incomplete
  - incommensurable

Sources of uncertainty

- imperfection of rules
- doubt of rules
- using a vague language

Modalities to express the uncertainty

- probabilities
- fuzzy logic
- bayes theorem
- theory of Dempster-Shafer

Modalities to represent the uncertainty

- by using a single value → certainty factors, confidence, truth value
- by using more values → logic based on ranges
  - min → lower limit of uncertainty (confidence, necessity)
  - max → upper limit of uncertainty (plausibility, possibility)

Fuzzy Systems

- why fuzzy?
  - problem: translate in C++ code the following sentences
    - george is tall
    - it is cold outside
- when fuzzy is important
  - natural queries
  - knowledge representation for KBS
  - fuzzy control - then we dead by imprecise phenomena (noisy phenomena)

Theory of possibility

History

- Parminedes (400 B.C.)
- Aristotle
  - "Law of the Excluded Middle“ – every sentence must be True or False
- Plato
  - A third region, between True and False
  - Forms the basis of fuzzy logic
- Lukasiewicz (1900)
  - Has proposed an alternative and systematic approach related to bi-valent logic of Aristotle – trivalent logic: true, false or possible
- Lotfi A. Zadeh (1965)
  - Mathematical description of fuzzy set theory and fuzzy logic: truth functions takes values in [0,1] (instead of {True, False})

    He as proposed new operations in fuzzy logic

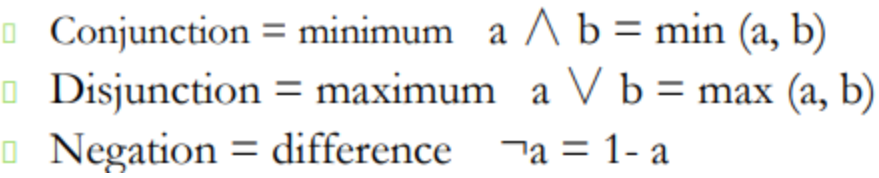
    He has considered the fuzzy logic as a generalisation of the classic logic
  - He has written the first paper about fuzzy sets

Fuzzy logic

- generalisation of boolean logic
- deals by the concept of partial truth
  - classical logic - all things are expressed by binary elements
    - 0 or 1, white or black , yes or no
  - fuzzy logic - gradual expression of a truth
    - values between 0 and 1

Logic vs algebra

- logical operators are expressed by using mathematical terms (George Boole)



Content and design

Main idea

- c.f. to certainty theory
  - popescu is tall
- c.f. to uncertainty theory
  - c.f. to probability theory
    - there is 80% chance that Popescu is young
  - c.f. fuzzy logic
    - Popescu’s degree of membership to the group of young people is 0.8

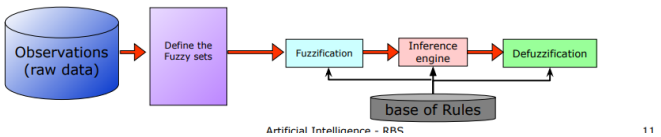
Necessity

- real phenomena involve fuzzy sets
- example
  - the room’s temperature can be: low, medium or high

- these sets of possible temperatures can overlap
- a temperature can belong to more classes depend on the person that evaluates that temperature

Steps for constructing a fuzzy system

- define the inputs and the outputs - by an expert
  - raw inputs and outputs
  - fuzzification of inputs and outputs
    - fix the fuzzy variables and fuzzy sets based on membership functions
- construct a base of rules - by an expert
  - decision matrix
- evaluate the rules
  - inference - transform the fuzzy inputs into fuzzy outputs by applying all the rules
- aggregate the results
- defuzzification of the result
- interpret the result



Elements from probability theory (fuzzy logic)

Fuzzy facts (fuzzy sets)

- set defintion - 2 possibilities
  - by enumeraton of elements
    - ex: set of students = {ana, maria, ioana}
  - by specifying a property of elements

    - ex: set of even numbers = {x | x = 2n, where n = 2k}
- characteristic function u for a set
    - let x a universal set and x an element of this set (x E X)
    - classical logic

        - Let R a subset of X: R⊂X, R – regular set
        - Every element x belong to set R
        - μ_R : X → {0, 1}, where $\mu_R(x) = \begin{cases} 1, & x \in R \\ 0, & x \notin R \end{cases}$

    - fuzzy logic

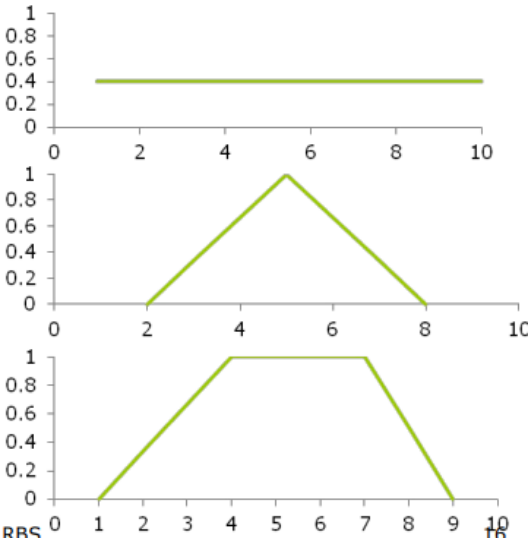
        - Let F a subset of X (a universe) : F⊂X, F – fuzzy set
        - Every element x belongs to F by a given degree of membership $\mu_F(x)$
        - μ_F : X → [0, 1], $\mu_F(x)=g$, where gϵ [0,1] – membership degree of x to F
        - g = 0 → not-belong
        - g = 1 → belong

        - A fuzzy set = a pair (F, μ_F), where

$$\mu_F(x) = \begin{cases} 1, & \text{if } x \text{ is totaly in } F \\ 0, & \text{if } x \text{ is not in } F \\ \in (0,1) & \text{if } x \text{ is part of } F (x \text{ is a fuzzy number}) \end{cases}$$

- representation
    - gradual limits → representation based on membership functions

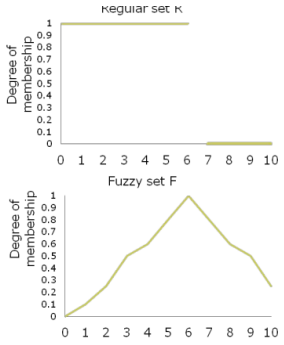
- Singular
  - $\mu(x) = s$, where s is a scalar
- Triangular
  - $\mu(x) = \max\left\{0, \min\left\{\frac{x-a}{b-a}, 1, \frac{c-x}{c-b}\right\}\right\}$
- Trapezoidal
  - $\mu(x) = S(x) = \max\left\{0, \min\left\{\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right\}\right\}$
- Z function
  - $\mu(x) = 1 - S(x)$
- Π function
  - $\mu(x) = \Pi(x) = \begin{cases} S(x), & \text{if } x \le c \\ Z(x), & \text{if } x > c \end{cases}$



- examples/h3

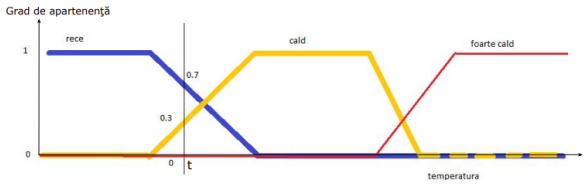
- X –set of natural numbers < 11
- R – set of natural numbers < 7
- F – set of natural numbers that are neighbours of 6

x	$\mu_R(x)$	$\mu_F(x)$
0	1	0
1	1	0.1
2	1	0.25
3	1	0.5
4	1	0.6
5	1	0.8
6	1	1
7	0	0.8
8	0	0.6
9	0	0.5
10	0	0.25

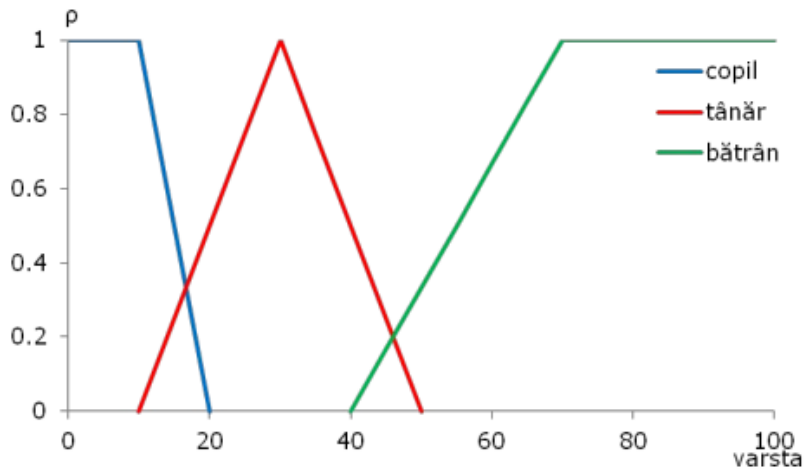


A temperature *t* can have 3 truth values:

- Red (0): is not hot
- Orange (0.3): warm
- Blue (0.7): cold



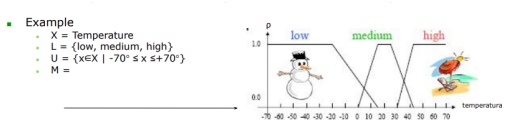
Age of a person



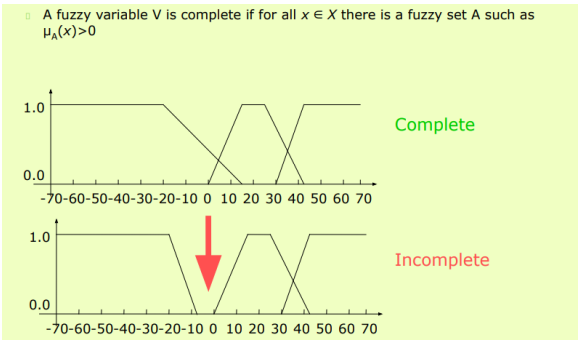
Fuzzy variable

- definitions
  - a fuzzy variable is defined by V = {x, l, u, m}, where:
    - x - name of symbolic variable
    - l - set of possible labels for variable x
    - u - universe of the variable
    - m - semantic regions that define the meaning of labels from l (membership functions)
  - membership functions
    - subjective assessment
      - the shape of the functions are defined by experts
    - ad-hoc assessment
      - simple functions that can solve the problem
    - assessment based on distributions and probabilities of information extracted from measurements

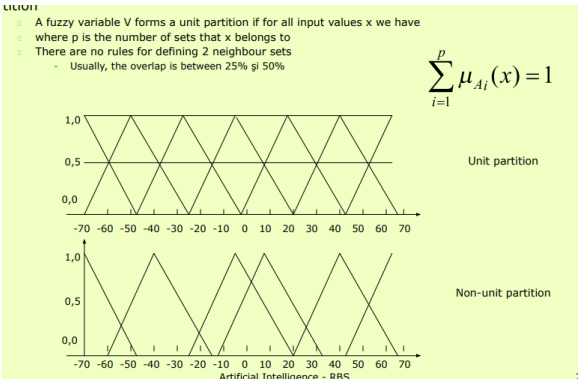
    - adapted assessments
        - by testing
    - automated assessment
        - algorithms utilised for defining functions based on some training data



- properties
    - completeness



    - unit partition



A complete fuzzy variable can be transformed into a unit partition:

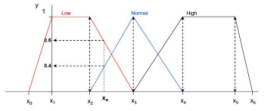
$$\mu_{\hat{A}_i}(x) = \frac{\mu_{A_i}(x)}{\sum_{j=1}^{p} \mu_{A_j}(x)} \; for \; i = 1, \ldots, p$$

Fuzzification of input data

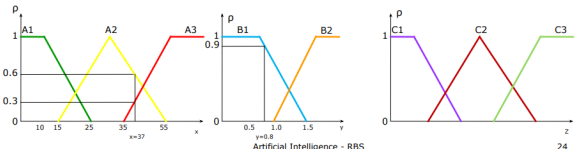
Mechanism

- establish the raw (input and output) data of the system
- define membership functions for each input data
  - each membership function has associated a quality label - linguistic variable
  - a raw variable can have associated one or more linguistic variables
  - example
    - raw variable: temperature T
    - linguistic variable : low → A1, medium → A2, high → A3
- transformt each raw input data into a linguistic data → fuzzification
  - establish the fuzzy sets of that raw input data
  - how?
    - for a given raw input determine the membership degree for each possible set
  - example

- Example
  - $T (= x_0) = 5°$
  - $A_1 \rightarrow \mu_{A1}(T) = 0.6$
  - $A_2 \rightarrow \mu_{A2}(T) = 0.4$



- Example - air conditioner device
  - Inputs :
    - x (temperature – cold, normal, hot) and
    - y (humidity – small, large)
  - Outputs:
    - z (machine power – law, medium, high)
  - Input data:
    - Temperature x = 37
      - $\mu_{A1}(x)=0$, $\mu_{A2}(x)=0.6$, $\mu_{A3}(x)=0.3$
    - Humidity y = 0.8
      - $\mu_{B1}(x)=0.9$, $\mu_{B2}(x)=0$



Base of rules

- construct a base of rules - by an expert

Rules

- definition
  - linguistic constructions
    - affirmative sentences: A
    - conditional senteces: if A then B
  - where A and B are (collections of) sentences that contain linguistic variables
    - A - premise of the rules
    - B - consequence of the rule
- typology
  - non-conditional
    - x is (in) Ai
    - ex: save the energy
  - conditional
    - if x is (in) Ai then z is (in) Ck
    - if x is (in) Ai and y is (in) Bj, then z is (in) Ck
    - if x is (in) Ai or y is (in) Bj, then z is (in) Ck

Example

	Rules of classical logic	Rules of fuzzy logic
R_1	*If temperature is -5, then is cold*	*If temperature is law, then is cold*
R_2	*If temperature is 15, then is warm*	*If temperature is medium, then is warm*
R_3	*If temperature is 35, then is hot*	*If temperature is high, then is hot*

- Database of fuzzy rules
  - R_{11}: if x is A_1 and y is B_1 then z is C_u
  - R_{12}: if x is A_1 and y is B_2 then z is C_v
  - ...
  - R_{1n}: : if x is A_1 and y is B_n then z is C_x
  - R_{21}: if x is A_2 and y is B_1 then z is C_x
  - R_{22}: if x is A_2 and y is B_2 then z is C_z
  - ...
  - R_{2n}: if x is A_2 and y is B_n then z is C_v
  - ...
  - R_{m1}: if x is A_m and y is B_1 then z is C_x
  - R_{m2}: if x is A_m and y is B_2 then z is C_v
  - ...
  - R_{mn}: if x is A_m and y is B_n then z is C_u

Decision matrix of the knowledge database

- example - air conditioner device
  - inputs

        - x (temperatur - cold, normal, hot) and
        - y (humidity - small, large)
    - outputs
        - z (machine power - low, constant, high)
    - rules
        - if temperature is normal and humidity is small then the power is constant

		Input data y	
		Small	Large
Input data x	Cold	Low	Constant
	Normal	Constant	High
	Hot	High	High

Rule evaluation (fuzzy inference)

Fuzzy inference

- rules are evaluated in parallel, each rules contributing to the shape of the final result
- resulted fuzzy sets are defuzzified after all the rules have been evaluated

Evaluation of causes

- for each premise of a rule (if s is (in) A) establish the membership degree of raw data to all fuzzy sets
- a rule can have more premises linked by logical operators and, or, not → use fuzzy operators

- Operator *AND* → intersection (minimum) of 2 sets
  - $\mu_{A \cap B}(x) = min\{\mu_A(x), \mu_B(x)\}$
- Operator *OR* → union (maximum) of 2 sets
  - $\mu_{A \cup B}(x) = max\{\mu_A(x), \mu_B(x)\}$
- Operator *NOT* → negation (complement) of a set
  - $\mu_{\neg A}(x) = 1 - \mu_A(x)$

- result of the premise's evaluation
  - degree of satisfication
  - other names
    - rule's firing strenght
    - degree of fulfillment
- determine the results
  - establish the membership degree of variables (involved in the consequences) to different fuzzy sets
- each output region must be de-fuzzified in order to obtain crisp value
- based on conseqence's type
  - mandani model - consequence of rule: "output variable belong to a fuzzy set"
  - sugeno model - consequence of rule: "output variable is a crisp function that depends on inputs"
  - tsukamoo model - consequence of rule: "output variable belongs to a fuzzy set following a monotone membership function

Mandani model

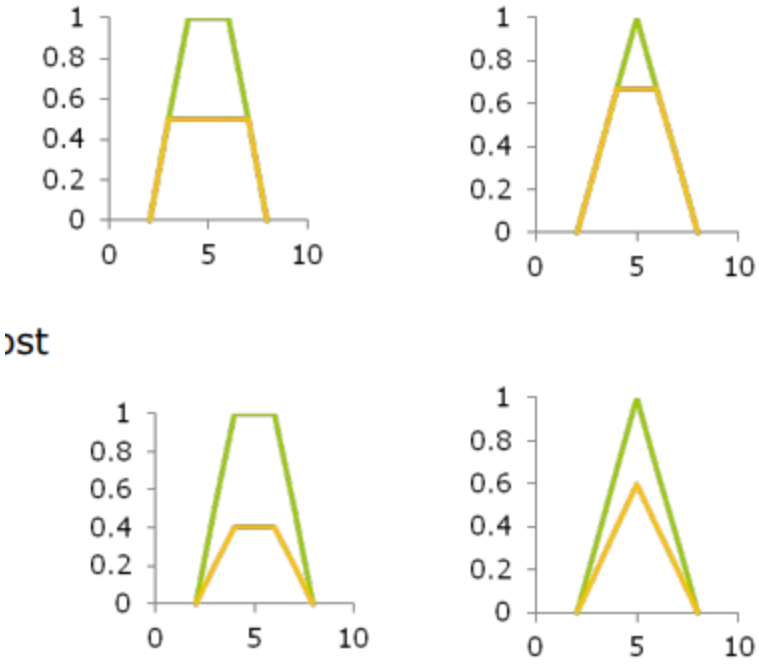
Main idea

- consequence of rule: "output variable belongs to a fuzzy set"
- result of evaluation is applied for the membership function of the consequence

- example
  - if x is in A and y is in B, then z is in C

Classification based on how the results are applied on the membership function of the consequence

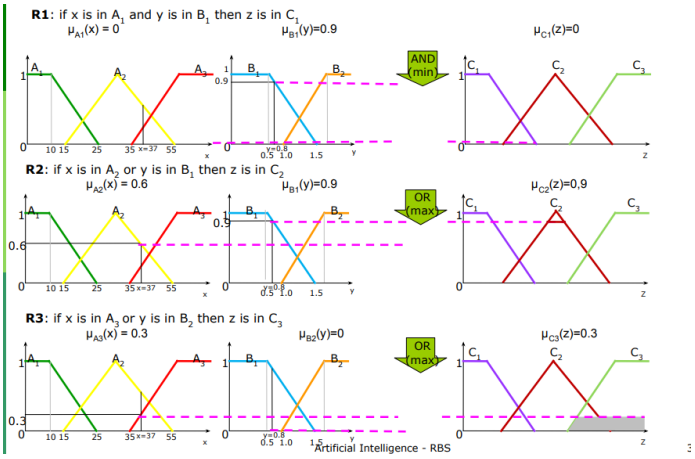
- clipped fuzzy sets
  - membership function of the consequence is cut at the level of the result's truth value
  - advantage → easy to compute
  - disadvantage → some information is lost
- scaled fuzzy sets
  - membership function of the consequence is adjusted by scaling at the level of the result's truth value
  - advantage → few information is lost
  - disadvantage → complicate computing



Example - air conditioner device

- Example – air conditioner device
  - Inputs :
    - x (temperature – cold, normal, hot) and
    - y (humidity – small, large)
  - Outputs:
    - z (machine power – law, constant, high)
  - Input data:
    - Temperature x = 37
      - $\mu_{A1}(x)=0$, $\mu_{A2}(x)=0.6$, $\mu_{A3}(x)=0.3$
    - Humidity y = 0.8
      - $\mu_{B1}(x)=0.9$, $\mu_{B2}(x)=0$

		Input data y	
		Small	Large
Input data x	Cold	Law	Constant
	Normal	Constant	High
	Hot	High	High



Artificial Intelligence - RBS

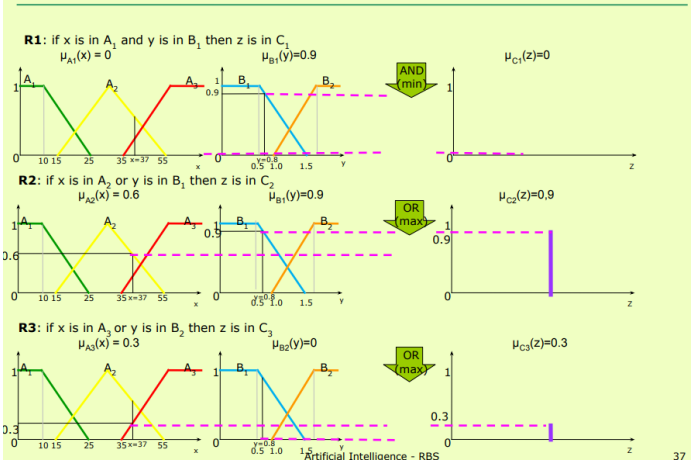
Sugeno model

Main idea

- consequence of rule: “output variable is a crisp function that depends on inputs”

Example

- if x is in A and y is in B then z is f(x,y)



Classificatioon based on charactheristics of f(x,y)

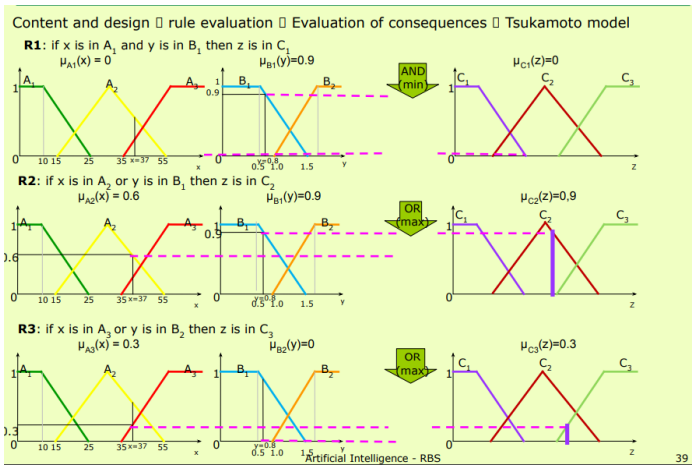
- sugeno model of degree 0
  - f(x,y) = k - constant(membership function of the consequence are singleton - a fuzzy set whose membership functions have value 1 for a single (unique) point of the universe and 0 for all other points
- sugeno model of degree 1
  - f(x,y) = ax + by + c

Tsukamoto model

Main idea

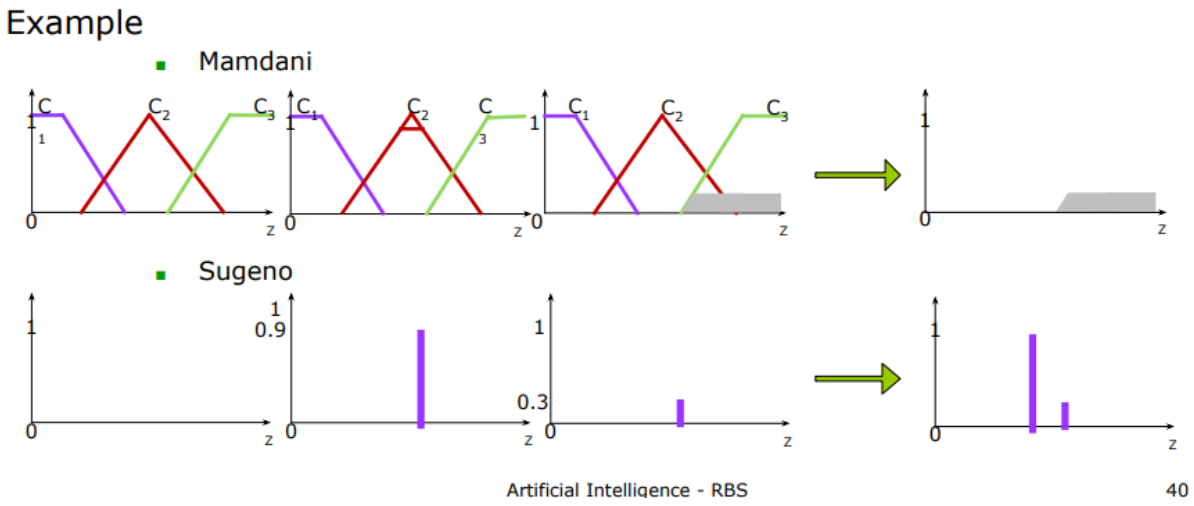
- consequence of rule:
  output variable belongs to a fuzzy set following a montone membership function
- a crisp value is obtained as output → rule's firing strength

Example



Aggregate the results

- union of outputs for all applied rules
- considet the membership functions for all the consequences and combine them into a single fuzzy set (a single result)
- aggregation process have as
  - inputs → membership functions (clipped or scaled) of the consequences
  - outputs → a fuzzy set of the output variable



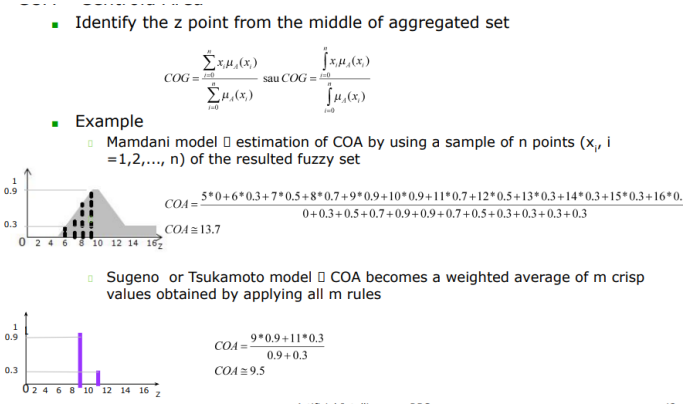
Defuzzification

- transform the fuzzy result into a crisp (raw) value

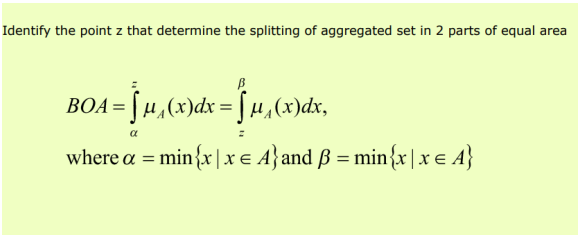
- inference → obtain some fuzzy regions for each output variable
- defuzzification ⟶ tranfsorm each fuzzy region into a crisp value

Methods

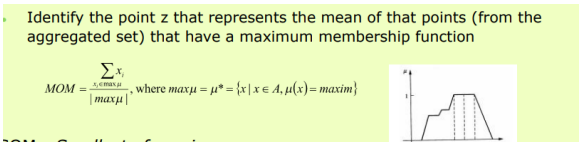
- based on the gravity centre
  - COA - centroid area



  - BOA - bisector of area

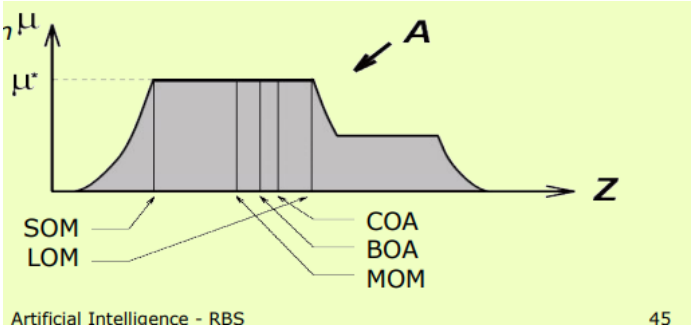


- based on maximum of membership function
  - MOM - mean of maximum



  - SOM - smallest of maximum
    - identify the smallest point z (from the aggregated set) that have a maximum membership function

- LOM - largest of maximum
    - identify the largest point z (from the aggregated set) that have a maximum membership function



Advantages

- imprecision and real-work approximation can be expressed through some rules
- easy to understand, to test and to maintain
- robustness can operate when rules are not so clear
- require few rules the other KBS
- rules are evaluated in parallel

Disadvantages

- require many simulations and tests
- do not automatically learn
- it is difficult to identify the most correct rules
- there is no mathematical model

Applications

- space control
    - altitude of satellites
    - setting the planes

- auto control
  - automatic transmission
  - traffic control
  - anti-braking systems
- business
  - decision systems
  - personal evaluation
  - fond managment
  - market prediction
- industry
  - energy exchange control
  - water purification control
  - ph control
  - chemical distialltion
  - polymer production
  - metal composition
- electronic devices
  - camera exposure
  - humidity control
  - air conditioner
  - shower setting
  - freezer setting
  - washing machine setting
- nourishment
  - cheese production
- military

    - underwater recognition
    - infrared image recognition
    - vessel traffic decision
- navy
    - automatic drivers
    - route selection
- medical
    - diagnostic systems
    - pressure control during anesthesia
    - modeling the neuropathology results of alzheimer patiens
- robotics
    - kinematics (arms)