

Angelo Antona

CREATIVE PROGRAMMING & TECHNOLOGIES

MUSIC ENGINEERING
— POLITECNICO DI MILANO

ANGELO ANTONA

Some Elementary Definitions

Computing → the term refers to the elaboration of informations performed by a system in order to obtain a certain result.

Creativity → the definition of "creativity" changed during the years

- before 1950 → it didn't refer to psychological properties, but to new changes in a culture.
- 1950 → Cyffert on refers in psychometrics, defined creativity on the production of a person to have new ideas.
- 1953 → Schemas offered that having new ideas wasn't enough to define "creativity": those ideas should have a certain value, resulting in a work that is useful for a certain group of people.
- after 1953 → Margaret Boden affirmed that "creativity" is the ability to generate novel and valuable ideas (where "valuable" doesn't mean only "useful").

In general we can define two types of creativity:

- Historically - Creative → to create something completely new or find an answer with the creation of the world.
- Practically - Creative → refers to the natural process of change in nature (Latin concept of "creare") according to this definition, any action can have an effect.

The definition that professor Pease → "creativity is a consequence of 'creativity'; 'creativity' is a consequence of emotion, learning, experiences." According to that definition (and to the professor) computer cannot be creative.

Another definition of creativity could be the one of Boden: «creativity is not a particular ability that only a tiny elite possess, but is a part of human daily life». According to this definition there are three types of creativity:

- Combinational Creativity → produces unfamiliar combinations of familiar ideas and it works by making associations between ideas that were previously unlinked.
- Expansive Creativity → it consists in producing new ideas staying inside a pre-existing "conceptual space" that is defined by a set of generative rules (ex. creating new songs or a certain musical genre).
- Transformational Creativity → in this case, the pre-existing "conceptual space" is transformed by altering one or more of its generative rules (ex. inventing a new musical genre).

Art → we can see Art as a subset of Creativity. The definition that professor Beer is: «we all know that some is not truth, Art is to be that makes us realize truth». Art is an art → tools have been important in the development of artistic movements: new tools open the possibility to explore new artistic fields.

Let's see some definitions in the artistic field:

- Connective Art → refers to any activity where the artist uses a system (ex. computer program or procedural rule) which is not in motion with some degree of autonomy contributing to the artistic work.
- New Media Art / Digital Art → there are themes used to describe art that is made or presented using digital technology. In today's digital art are there some important "three points":
 - Interactivity: the artist and the audience are engaged with the physical world.
 - Embodiment: body and mind are not separated as reported independent entities.
 - The cyborg: the artistic piece should exhibit a certain grade of autonomy.
 - Hybridity: between disciplines and between physical and digital worlds.
 - Narrative: there should be an evolution in time.
- Interactivity: the viewer has the power to be an active participant in the unfolding of the artistic piece's flow of events.

Computer Music → is the application of computing technology in music composition.

Visual Art → traditionally it is related to painting, drawing, printmaking, sculpture... today it is also related to digital imagery, digital film and video installations, immersive luminous chore performances...

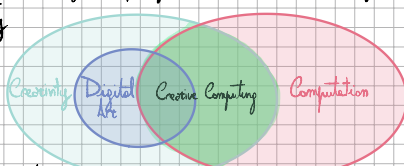
- Robotic Art → any discourse that employs some form of robotic or automated technology.
- Evolutionary art → the art of the system constructing itself. They are typically based on evolution systems like Genetic Algorithms.
- Interactive art → users are no longer passive observers of the artistic piece, instead the audience takes part to the evolution of the system.

Creative Computing → it is defined as a meta-technology that combines knowledge in computing and other disciplines. The aim of creative computing is to help the artist to be creative, not to substitute the artist.

In this view, the result of creative programming is creativity itself. The study of "how to compute creativity" is called Computational Creativity.

The way in which creative programming combine computing with other disciplines are:

- Multidisciplinarity → the knowledge may collaborate but they maintain a separation in their disciplines.
- Interdisciplinarity → it could fuse different knowledge deeply and finally end up with the extension of existing disciplines.
- Transdisciplinarity → focuses on all the disciplines in the world, which makes it a more holistic approach than interdisciplinary combination.



creativity in human culture new situation

Protocols

Society of Motion Picture and Television Engineers

SMPT-E Time Code → it was originally used to sync audio and video signals. This can done through a sequence of data sent at specific time intervals.
A timescode contains:

binary-coded decimal hour:minute:second:frame ID	32 bit for use by users (User Bit)	color-frame flag	drop-frame flag	binary group flags for interpretation of U-bit
--	------------------------------------	------------------	-----------------	--

information about time

used in video applications to indicate the frame color

manage the compression needed in some video segments in which the nominal speed is different from the actual speed.

SMPT-E can be incorporated into the media, so that each frame will have its timescode, making synchronization easier. The negative aspect of the protocol is that it introduces latency.

● **Modi:** is a protocol that allows musical instruments and programs to communicate. It is characterized by a non-realtime asynchronous 31.25 Kbit/sec



status bytes	Data Bytes 1	Data Bytes 2
1 S S S S n n n n h o x x x x x x x x o y y y y y y y y		

Message type (Channel number byte) Assume different meanings based on the type of each message

↑ Note Off = 8n → Data1 = midi note (0,127) → Data2 = velocity (0,127)

↑ Note On = 8m → Data1 = midi note (0,127) → Data2 = velocity (0,127)

↑ Polyphonic Aftertouch = An → Data1 = midi note (0,127) → Data2 = pressure (0,127)

↑ Channel Aftertouch = Dm → Data1 = pressure (0,127)

↑ Pitch Wheel = Em → Data1 = Style (0,127) → Data2 = Style (0,127)

↑ Control Change = Bm → Data1 = type of control → Data2 = value

↑ Program Change = Cm → Data1 = patch number

Running status: in order to decrease the amount of data that we need to transmit we can use the running status. In this state several messages are organized using the same status byte → note off becomes note on with 0 velocity.

● **Channel Mode messages:** are a special type of Control Change message. They are used to perform operation like setting master/poly master turn on/off the Omni mode... This kind of message has the side effect of turning off all the control notes.

● **Program Change:** is a kind of message used to change between patches or presets of an instrument. The messages are organized in a single channel. Programs are grouped in banks of 128 patches. We can switch between different banks using a Control Change message: bank select.

● **Midi system messages:** these are global messages that can be heard by all midi channels. They can be:
- system exclusive → used for device-specific data transfer.
- system common → used for transfer data to all midi devices

- This protocol working with a very real transport mechanism. This is problematic with heavy data streams. Furthermore, the transmission rate is low.
- It has an integer representation of pitch that makes difficult to implement it on wind instruments or guitars.
- In general, the problem with MIDI is that it is not programmable.

● **Open Sound Control (OSC):** it is a communication protocol designed to move this communication between computer, synth and other device easier.

It does not specify a "message syntax" like midi does, instead, OSC provides only the communication protocol.

Its main characteristics are:

URL-style symbolic naming scheme → the receiver are communicated using a URL-style syntax making easier the comprehension and the managing of the system.

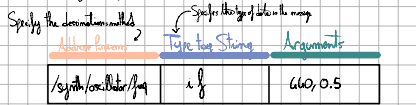
Different type of data → it's possible to handle various data types:

Data type	Type tag
int 32	i
float 32	f
osc-string	s
osc-blob	b

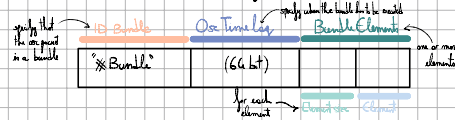
High resolution numeric data
Symbolic organized data

● **Pattern matching language:** it's possible to group messages that have to be sent simultaneously. It's not only the individual OSC message and the OSC bundles.

● **OSC message:** is the elementary block of OSC data transmission. They must have a 4-bit multiple length in order to be compatible with UDP/TCP. They are built in the following way:



OSC Bundle → it can contain messages or other bundles:



● **Network transmission protocol:** → use UDP or TCP.

Every OSC server has a set of potential OSC Methods, which are distinctive of OSC messages. Methods are organized in a three called OSC address space. Leaves are methods while the bundle makes are the OSC containers. When a server receive a message, it increases the correct method dispatching.

DMX (Digital Multiplexed signal) → it is a communication protocol mainly used to control light dimmer and smoke machines and other devices in theaters, concerts and live events. DMX transmits messages on a serial line. Each DMX message is composed by 512 channels and each channel containing an integer value updated in [0,255] (8-bit). Channels are used to specify which device should read a certain parameter value. a set of successive channels is assigned to each device that has to be controlled. for example, if we are controlling a light we could use a channel configuration like this:

light fixture

Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
light intensity	Red color	Green color	Blue color	Lens focus	Horizontal movement	Vertical Movement

- If we want to control more than one light we should be careful to put them in the correct order: the first channel will control the first light, the second channel will control the second light and so on.
- The total amount of bytes that constitutes the DMX channel (512 bytes) is called DMX universe. More than one DMX universe can be used simultaneously.

Art-Net → it is a protocol developed to transmit DMX on ethernet network. It reduces practically the amount of cables needed for the implementation of a DMX universe, allowing to control thousands of DMX universes simultaneously.

This protocol is based on UDP and each device in the network is identified by an IP-address.

- Several types of messages are used in Art-Net:
 - Data messages → messages are sent in packages called ArtDMX, which contain the values for all the channels of a specific DMX universe. These packages are sent at regular intervals ensuring that all the devices are synchronized.
 - ArtPoll → it is a message to detect all the devices of the Art-Net. When a device receives this message it answers with an ArtAck reply providing its information (its IP-address).
 - ArtSync → can be used to synchronize packet sending.
- The Art-Net architecture is usually composed by the following devices:
 - Art-Net controller → device that generates and transmits the Art-Net messages.
 - Art-Net compatible devices → devices that are mutually compatible with the Art-Net protocol.
 - Art-Net media → adapters used to connect "legacy DMX devices" to the Art-Net.

The world described through data and signals

- Differences between Novel Instrument Design and Immersion are:
 - Novel Instrument Design → it aims at creating new instruments that expand artistic ability.
for NID it's important that the interface is as intuitive as possible: the learning curve should have a high slope
 - Interactive are → it is more focused on the message and the user experience rather than the medium. The interaction is designed for the unique piece in a specific context. It does not exist in that specific moment for that specific people.
The interface system should be the best to make the user do the experience the artist has designed. The learning curve should be zero.
 - Hybrid Context → it is the one in which an artist creates its own interactive system for its own performance. The interactive system should be the best to make the artist's experience that he desires and to make them communicate with the audience correctly.
- Audio descriptors → the classical characteristics of sound like waveform, spectrum, envelope are too general. We need features that are able to capture more specific elements. These features can be classified in 3 levels:
 - Low-level features → are the ones that we can directly extract from the audio file (e.g. spectral centroid, spectral flux)
 - Mid-level features → are based on music theory (e.g. tonality)
 - High-level features → are the ones that are closely connected to our perception (e.g. genre, mood).

Computer Vision → it consists in developing techniques to help computer "see" and understand the content of digital images.

This aim can be achieved through machine learning. There are different techniques:

- Object classification → identify the broad category of the photo.
- Object identification → identify the type of object contained in the photo.
- Object recognition → identify if an object is in the photo.
- Object detection → identify the position of the objects in the photo.
- Object landmark detection → identify the position of the key points of the objects in the photo.
- Object segmentation → identify which pixels in the image belong to the object.
- Object recognition → identify which objects are in the photo and where they are located.
- Motion Detection → in order to detect the movement of a person in space we have to detect the changes in the pixel values for a sequence of frames.
if the value of a pixel changes remarkably, there has been motion in the video. To see "how much" in the kinetic movement that the object has done we just have to count how many pixels have remarkably changed in just one frame.
- Motion capture → it consists in tracing movement by following the human skeleton movements or the movement of some key points of the face.

How to manage a creative project

Creative projects are not functional to solve a problem, they are built to create experiences. The steps in creative project management are:

- conceptualizing: creating the concept of the project, describing the theme, the motivation and the user experience.
- design: based on the concept, create the design of the project deciding techniques and tools to be used.
- prototyping: first implementation of the main feature of the user experience.
- production: final realization.

Agents

- Definition of Agent → an agent is anything that can be viewed as ^{perceiving its environment through sensors} ^{acting on that environment through actuators.} ^{in which this agent is living} ^{Sensors and actuators can be digital or physical} ^{system that lives in the virtual environment but performs actions on it can be considered an agent. The agents are not free} ^{agent's behavior is described by the agent function that maps the 'input sequence' to 'an action'} ^{In order to model the interaction between the system and the agent we must learn the fundamentals of interaction design.}

Interaction design → this is the 'art' of 'designing a system of some sort that a person can interact with in a way that is meaningful to them'. Interaction happens via messages sent from system to user and vice versa. These messages can be text, speech, colours, mechanical input/feedbacks.

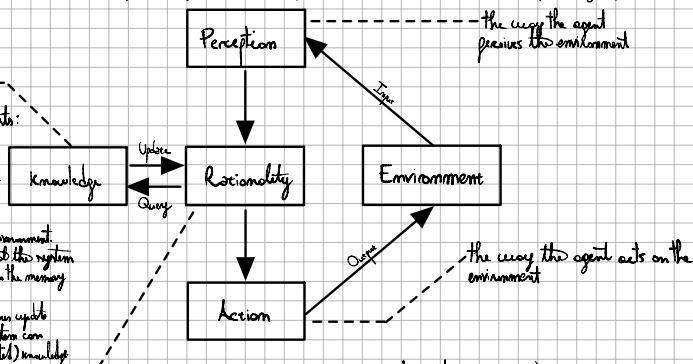
There are different modes of interaction:

- Discrete (used) → the communication is unidirectional system → user. The user learns watching the system (ex. website).
- Semi-conversational model → the user communicates something to a system and the system communicates something back that allows the user to carry on with a secondary task (ex. classical musical instruments).
- Full-fledged conversational model → the user and the system can fully and concurrently interact and they both learn and adapt (ex. groups).

Architecture of an agent

The knowledge base of itself and of the environment. We can identify two concepts:

- Initial knowledge → it is the knowledge contained in the agent's memory. It can be:
 - encoded → it is the knowledge that the authors encoded in the system.
 - implicit knowledge → the knowledge that the agent acquires directly from the environment.
 - learned knowledge → the knowledge that the system extracts from the data already stored in the memory using machine learning algorithms.
- Knowledge update → it is the continuous update of the knowledge of the system. A system can have static (no update) or dynamic (updated) knowledge.



It is the 'reasoning process' that makes the agent do the right thing. It consists in two elements:

- ① Take a decision.
- ② Perform a set of actions, implemented through an agent function. The agent function tries to maximize the desirability of the output (it's necessary to implement a measure of desirability = performance).

 In an interactive application, performance evaluation can be performed in two ways:

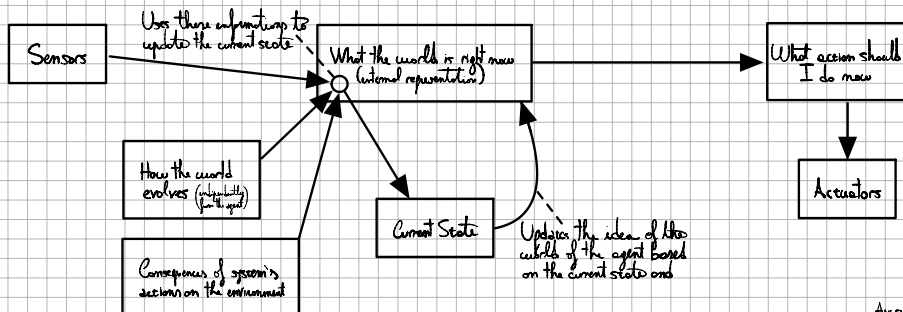
- internal evaluation (ex. robot decides to not jump on the table).
- external evaluation (ex. someone asked the robot that tells it if that something's wrong).

Types of Agents' behaviour → there are 4 types of possible behaviours:

- Fixed: some exact behaviour.
- Periodic: the behaviour follows a periodic pattern.
- Chaotic: the behaviour is random.
- Complex: the behaviour is random but it follows some rule of recurrence (ex. automatic music composition).

Types of agents → we can distinguish different types of agents depending on their architecture:

- Cognitive agents: maintain a complex internal representation of the world (i.e. robot that knows how objects are positioned in a room. They are 'intelligent' but not so 'reactive' because of the large amount of information they have to analyse).
- Reactive Agents: they have not a complex representation of the world and just focus on behavioural rules. There are:
 - Reactive agents with no memory = Reflex Agents → they have no internal memory. These agents select actions on the basis of the current percept and ignoring the rest of the perceived history. This is done simply by applying a rule. The main problem with this type of system is that they work correctly only in cases in which the environment is fully observable.
 - Reactive agents with memory → they have internal memory that keeps trace of the memory (it still is a form of cognitive memory). Thanks to this internal state we can manage the case of not fully observable environment in better ways. A scheme of a general reactive agent with memory is the one showed below.



Despite their simplicity, reactive agents can be used to build chaotic systems. Two examples are shown below.

• Hénon Attractor
$$\begin{cases} x_{n+1} = 1 + y_n - a x_n^2 \\ y_{n+1} = b x_n \end{cases}$$

We can "set" the behavior of this system by fixing the variables a, b and fixing the initial values x_0 and y_0 .

Gingerbreadman function
$$\begin{cases} x_{n+1} = 1 - y_n + |x_n| \\ y_{n+1} = x_n \end{cases}$$

In this case we have to set the initial values x_0 and y_0 .

In both cases we can see a pattern even though the system is not well determined. This happens because the term "chaotic" doesn't mean "non-deterministic".

Multi-Agent creative systems

It is possible to see a system constituted by multiple agents as a flock of birds: each bird has not the overall vision of the flock's shape. Each bird can be seen as a reactive agent that reacts with actions to specific inputs. Rather than a "single intelligence", a *swarm intelligence* emerges. ^{swarm}

There are different possible types of interaction between agents:

- No interaction $\rightarrow$ agents don't interact
 - Personal goal $\rightarrow$ each agent has a personal goal and interacts with other agents to reach that goal. An example could be an automatic negotiation system, in which the agent interact with others to get the most convenient price.
 - Collective goal $\rightarrow$ the complex system of agents has a goal and each single agent give their personal contribute to reach the goal. An example could be a system in which robot carry heavy objects.
 - Particle system $\rightarrow$ a particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system. The particle system is composed by particles that can be seen as reactive agents. Interaction between these agents can be of three types:
 - Avoid collision (repulsion) $\rightarrow$ to implement it we define a "repulsive zone". When an object enters the repulsive zone of another one is subjected to a repulsion force that can be static or dynamic.
 - Attraction $\rightarrow$ the same of repulsion case but with opposite force.
 - Connection: particles are connected through spring ^{malleable}.
- $\rightarrow$ this can be modelled using location, velocity, acceleration and lifespan for each single particle.

Boids $\rightarrow$ this term derives from the word "birds" written in that way to resemble the New York is dialed. It is a simulation model developed by Craig Reynolds in 1986 used to represent the collective movement of flocks.

To understand this mechanism let's consider ants:

- when an ant leaves the nest in search of food, she usually lay down a pheromone trail that the other ants are able to follow.
- if she reaches some food, she will return to the nest, using the same path, leaving even more pheromone on her way back.
- if she does not find any food, she will not lay down any pheromone in the way back to the nest and the previous trail will evaporate away.
- some ants can leave trail to find other routes and that is where fluctuation comes in.

This organization is thought to arise from the intentionless dynamics of the swarming creatures, and not by any central leadership. The biological process through which swarming animals communicate is called *stigmergy*.

Reynolds implements this mechanism using reactive agents called boids. The interaction between boids is modelled using 3 forces:

- Separation: avoidance $\rightarrow$ steer to avoid colliding with your neighbours. It is obtained applying repulsive force.
- Alignment: copy $\rightarrow$ steer in the same direction of your neighbours.
- Cohesion: centre $\rightarrow$ steer towards the centre of your neighbours. It is a matter of attraction forces.

We will use the steering force formula to implement the rules of flocking: $\text{resultant direction} = \text{steer} = \text{desired-velocity} - \text{current velocity/direction of the boid}$.

Each boid must observe the behaviour of its neighbours to determine its behaviour. The different forces (attraction, repulsion, connection) are weighted and summed to get the resultant behaviour.

From these simple rules a complex behaviour will emerge.

Cellular automata $\rightarrow$ it is a technique based on Turing's theories that consists in a universe, meaning, on m-dimensional grid of cells.

- Each cell of the universe is characterized by a finite number of states (in the easiest case just 0-1).
- Each cell has a neighborhood, represented as a list of adjacent cells.
- The state of each cell change at each iteration following a transition rule based on the state of the cells of the neighborhood.

We can create different types of cellular automata changing the m-dimension, the transition rule or the states. Let's see some examples.

- Wolfram model
 - Dimensions: one-dimensional model
 - States of each cell: 0, 1
 - Transition rule:
- | neighbourhood | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|-----------------|-----|-----|-----|-----|-----|-----|-----|-----|
| resulting state | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

- Game of Life
 - Dimensions: 2-dimensional model
 - States of each cell: 0, 1
 - Transition rule:
 - Death $\rightarrow$ if a cell is alive (1) it will die if it has less than 2 or more than 3 alive neighbours.
 - Birth $\rightarrow$ if a cell is dead (0) it will come to life if it has exactly 3 alive neighbours.
 - Stasis $\rightarrow$ in all the other cases, the cell's state doesn't change.

It was to emulate "how things evolve in nature".

Grammar and Automata-Based Agents

The simplest way to implement "creativity" in a digital system is to introduce "uncertainty". This randomness is described by a distribution probability. Random Walk $\rightarrow$ it is a technique to build creative systems using randomness and coherence, basically, it follows a random cycle in which each step is randomly determined using a set of constraints. Let's consider different cases, from the simplest to the most complex (considering algorithmic music composition):

- Simple uniform probability distribution $\rightarrow$ it's the most simple random noise generator that we can get. Obviously, the result we get is simple and boring.
 - Random cycle 1D: we can introduce a simple constraint to get a more coherent melody. For example, we could introduce the rule that the next note is always one semitone higher or lower to respect to the current note (probability of $\frac{1}{2}$ for each possibility).
 - Random cycle 2D: it's possible to use the same principle to generate pictures system (e.g. generate random points on a grid). In this case, the constraints should manage direction and distance.
- Something of random cycle: a "completely random" movement could be too "not predictable" for some applications. We can solve this problem in two ways:
- using a different distribution probability (e.g. gaussian)
 - using the feeling noise: it introduces "local coherence".

- The Levy flight $\rightarrow$ it is a specific type of Random Walk characterised by frequent small steps and less frequent long steps. This behaviour is inspired by animal's research patterns like bees or birds, combining local exploration with global search. The aim is to introduce constrained creativity that also consider the system history. The algorithm to determine the distance for each step is the following (Monte Carlo Method):

1. Generate a random number $R_1 \rightarrow$ it is in the interval 0-1 and it represent the length of the (possible) step.
2. Calculate the probability using the formula $P = (1 - R_1)^{1/\alpha}$ (if R_1 is really small, P is really big).
3. Generate another random number R_2 . It will be used as a filter.
4. Compare R_2 and P :
 - if $R_2 < P$, the value R_2 is accepted and the process ends (this will happen mostly when R_2 is very small $\Rightarrow P$ is very big).
 - if $R_2 \geq P$, the process restarts from point 1 (this will happen mostly when R_2 is very big $\Rightarrow P$ is very small).

Noise $\rightarrow$ it is a sequence of random values that can be used to produce noise and sound effects. There are different types of noise:

- White noise $\rightarrow$ it's the most simple type of noise in which all the values have the same probability. There is no correlation between consecutive values and that makes this type of noise not so appropriate to most natural events.
- Perlin Noise $\rightarrow$ it's a noise which introduces local correlation between consecutive values. In this way, the generated values are not completely random but gradually variable.

The algorithm is the following:

1. Definition of a grid $\rightarrow$ the input domain space (e.g. x, y) is divided in regular cells forming a grid. Each cell has vertices. A **coarse gradient vector** is assigned to each of the vertices.

e.g. cell with vertices at $\{(1,1), (1,3), (2,2), (2,3)\}$

coarse gradient vectors: $[0.7, 0.5]$ $[0.4, 0.8]$ $[0.1, -0.5]$ $[-0.6, -0.1]$

2. Calculation of directional vectors $\rightarrow$ for an input point (x, y) inside a cell we calculate the directional vector that connect that point to each one of the vertices of the cell.

e.g. point $(1.3, 2.4) \Rightarrow$ calculation of directional vectors: $V_1 \rightarrow (1.3 - 1, 2.4 - 2) = (0.3, 0.4)$
 $V_2 \rightarrow (1.3 - 1, 2.4 - 3) = (0.3, -0.3)$
 $V_3 \rightarrow (1.3 - 2, 2.4 - 2) = (-0.7, 0.4)$
 $V_4 \rightarrow (1.3 - 2, 2.4 - 3) = (-0.7, -0.3)$

3. Dot product $\rightarrow$ for each vertex of the cell we calculate the scalar product {gradient vector * directional vector}. The result will be a "big" value if the two vectors have similar directions, it will be a "small" value if they are not aligned.

e.g. scalar product for each vertex $\rightarrow V_1: (0.7 \times 0.3) + (0.5 \times 0.4) = 0.56$

$V_2: (-0.6 \times 0.3) + (0.1 \times -0.3) = -0.36$

$V_3: (-0.1 \times 0.4) + (-0.6 \times 0.4) = -0.64$

$V_4: (-0.6 \times 0.4) + (-0.1 \times -0.3) = -0.45$

4. Smooth interpolation $\rightarrow$ the scalar products are interpolated using a function called smoothstep. The final value for the point (x, y) is the linear combination of the scalar products performed for each corner of the cell.

Grammar-Based Agents $\rightarrow$ creativity often requires something more than just randomness and memory: we need a grammar.

If we consider human natural language, it is constituted by syntax (how to combine words), semantics (meaning of sentences), pragmatics (how to use language in context). We are now interested in SYNTAX. We want to model it using a formal grammar.

Formal Grammar $\rightarrow$ a formal grammar is a set of rules that transform non-terminal symbols into terminal symbols. Those rules are of the type:

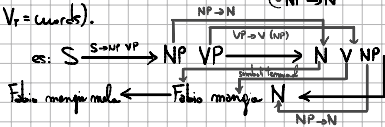
it must be a non-terminal symbol (left side) $\rightarrow$ right side, it can be a non-terminal or a terminal symbol

- Non-terminal symbols are contained in the set V . e.g. $V = \{S, VP, NP, V, AP, ADV, \dots\}$. The set of rules is of the type R :
 - $S \rightarrow NP VP$
 - $VP \rightarrow V (NP)$
 - $AP \rightarrow (ADV) A$
 - $PP \rightarrow P NP$
 - $NP \rightarrow N$

- Terminal symbols are contained in the set V_t . (in the case of natural language $V_t = \text{words}$).

- So, a grammar can be defined as a set $G = \{V, V_t, R, S\}$.

- The process ends when there are only terminal symbols.



- In the case more rules share the left side, it is possible to define the probability for each rule to be applied (es. $R_1: S \rightarrow NP \Rightarrow P(R_1) = 0.6$).
 $R_2: S \rightarrow VP \Rightarrow P(R_2) = 0.4$
- Application of a formal grammar to music $\rightarrow$ we can model musical languages using a formal grammar:
 - we have to define a set of terminal symbols $V_T = \{w, h, q, o, \& \{w, h, q, o\}, \& \{h, q\}\}$
left side: note, right side: note, note and pitch, pitch
 - we also need a set of non-terminal symbols $V_N = \{S, H, H, Q, Q, \&\}$
area of music, area of music, area of music
 - and a set of rules: $R = \{S \rightarrow H, SH, H \rightarrow HH, QHQ, w, h, q, o, \& \{h, q\}, H \rightarrow QQ, OQO, h, sh, iq, iq, Q \rightarrow q, oo, to, to, b, \& O \rightarrow o, \& o\}$
area of music, area of music, area of music

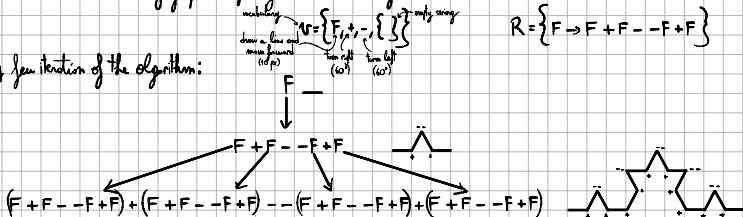
le grammatiche formali sono perfette per descrivere sistemi con strutture gerarchiche e, come possiamo vedere, la musica è per una sua parte una struttura gerarchica: Boccherini $\rightarrow$ Menuetto $\rightarrow$ Menuetto (HH)
 $\downarrow (QQ)$
 due quarti di battuta + 4 ottavi di battuta
 $\downarrow (OO)$
 note di $\frac{1}{4}$ + terzine di $\frac{1}{8}$ + note di $\frac{1}{4}$ + quarti + terzine + note di $\frac{1}{4}$
 (q) (q) (o) (h) (o) (o)

- L-Systems $\rightarrow$ it is a model to apply grammar to images. Unlike formal grammars, in L-Systems there is no difference between terminal and non-terminal symbols: all symbols corresponds to an action. An L-System is formally defined as:

$L = \{w, w, P\}$
 w : alphabet of symbols
 P : set of transformation rules $\neq \rightarrow X_{X \in w}$
(w: set of all the possible symbols, w: set of all the symbols)
 $R = \{F, F, \& \{F\}\}$
draw a line and move forward (d.p.)
turn right (w)
turn left (w)

- Koch Curve $\rightarrow$ In the context of graphic shapes generation, each symbol is associated to an action. Let's consider the Koch Curve as example:

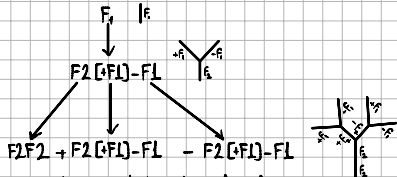
Let's try few iteration of the algorithm:



- Tree Generation $\rightarrow$ we can use L-Systems also to generate realistic trees. The procedure is similar to the Koch Curve. We just have to modify the set of transformation rules and the alphabet:

$w = \{F_1, F_2, \& \{F_1, F_2\}\}$
draw a line and turn left (w)
draw a line and turn right (w)
 $R = \{F_1 \rightarrow F_2 F_2, F_2 \rightarrow F_2 [F_1] - F_1, w = seed = F_1\}$
[F_1] means push the pen, [F_2] means pull the pen, [F_1] means pull the pen, [F_2] means pull the pen

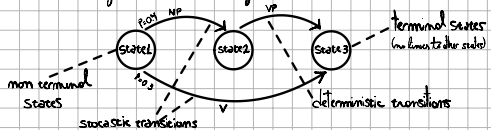
Let's try few iteration of the algorithm:



- Stochastic L-Systems $\rightarrow$ we can also introduce a control variability associating a probability to each rule. (es. $P_1: F \rightarrow F[F_1 F_2] \text{ con } P=0.2$
 $P_2: F \rightarrow F[F_2] \text{ con } P=0.5$
 $P_3: F \rightarrow F[F_1] \text{ con } P=0.3$)

From Transitional Networks to Hybrid Agents

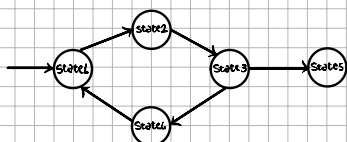
- Transitional network $\rightarrow$ transitional networks are a model used to represent a structural sequence of events. The type of transitional network we will focus on are called "finite state automata" and it is composed by:



Outputs are generated based on $\begin{cases} \text{input} \\ \text{internal state} \end{cases}$

- Autogrammer $\rightarrow$ finite state automata applied in music are called autogrammer. We can use different transitional networks to model different aspect of music (e.g. pitch, volume, chords).

For example, we can try to exploit patterns of western music composition (e.g. scales) to harmonize notes, considering a certain scale and assigning a note to each note of that scale.



- We need two fundamental functions; in order to simulate more flexibility we can introduce stochastic transitions
 - get next state (curState, count) returns the new state.
 - get current output (curState, old note, count) returns new note, new count.

We could manage the other aspect of music such as bpm or volume using other transitional networks and connecting all those network in a hierarchy.

The main problem of applying transitional networks to music is that it's necessary to write the transition rules by hand. That's why Markov chains are born.

- Markov Model $\rightarrow$ it is a mathematical model used to model sequences of discrete events (e.g. musical notes). It is based on 'probabilistic transitions' between consecutive states, overcoming the limits of classical transitional network.

Imagine to model a melodic generator using Markov chains: each note will be represented by a state (line in transitional networks) but in this case the model determines the probability of the transition between two consecutive states using conditional probability $P(X_i | X_{i-1})$. This means calculating which is the note we should expect given the current sequence of notes.

This method is based on the following elements:

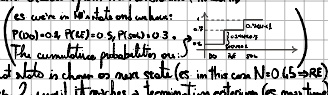
- Transition Matrix $\rightarrow$ it can be defined in different ways:
 - manual definition of the statistical rules.
 - learning over a sample set. In this case an algorithm counts how many times the transition $X_{i-1} \rightarrow X_i$ appears in the sample set, normalizes the probabilities and organizes these normalized probabilities into a transition matrix. This matrix can be fixed or dynamic (probabilities are updated during the composition).

The probability into the transition matrix depends also from the order of the system: $\begin{cases} 1^{\text{st}} \text{ order} \rightarrow P \text{ depends only from the previous state of the system} \\ 2^{\text{nd}} \text{ order} \rightarrow P \text{ depends from the previous states of the system} \end{cases}$

Complexity increases exponentially following the order of the system.

- Generation Algorithm: (stochastically (based on a probability distribution))

- Select the initial status. It can be chosen to be based on deterministic rules.
- Identification of the current status X_i and read the probabilities from the current state to the other states (from the transitional matrix).
- Generation of the next status. It is determined through the following algorithm:
 - generate a random number N between 0 and 1 (e.g. $N=0.65$)
 - read the probability for each possible next state and consider the cumulative probability
 - compare N and the cumulative probability: if N falls into the range of a state S , that state is chosen as next state (e.g. in the case $N=0.65 \rightarrow RE$)
- The selected next status X_{i+1} becomes the current status and the algorithm return from step 2 until it reaches a termination criterion (e.g. max time)



The main problem of this approach is that it's difficult to represent long term musical relations because complexity exponentially increases with the order of the model.

- Variable Length Markov Chain (VLHC) $\rightarrow$ in VLHC the model predicts the probability of next state based on a variable length sequence of past states. This means that if a shorter event sequence is sufficient for a precise prediction, the model will use a shorter context.

The process of building and utilizing a VLHC can be divided into three main components:

- Counting $\rightarrow$ during the building of the transition table, the model counts the occurrence of sequences of varying lengths to determine which contexts are relevant for predicting the next state. This is the name of Markov Chain, but calculated on context of varying lengths.
- Smoothing $\rightarrow$ if there are "unseen sequences" in the train data, the model combines context of different length (e.g. ABC in context, the model uses AB).
- Variable length modelling $\rightarrow$ instead of maintaining separate transition matrices for every possible context length, the VLHC uses a unified representation that dynamically adapts the context length.

Cognitive Agents

Cognitive agents are agents that maintain an internal symbolic representation of the world.

Desire-Belief-Intention model (DBI) → it is composed by:

- Beliefs** → it is the internal knowledge the agent has about the world. Beliefs can be updated during the simulation using:
 - perception: external stimuli update the agency's belief of the world.
 - reasoning: the internal elaboration of data one sensibly helps to maintain a coherence in the belief of this agent.
- Desires** → the objectives that agent wants to achieve. Desires can be in contrast between them.
 - (if there is a big and the small want to do the same action, that is the agent's desire)
 - as it cannot aim and be aware of them time
- Intentions** → are the actions that the agent actually takes to achieve its objectives. Intentions cannot be in conflict, guaranteeing that agents have a realizable plan.

So, the agent "calculates" its intentions starting from its beliefs and desires. The two processes that we used are:

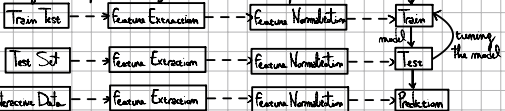
Deliberation → select which desire to pursue or an intention based on the current beliefs.

Mean-end reasoning → generate a plan to achieve the selected desires.

Artificial Intelligence → it is the process of modelling intelligence into machines.

Machines learning → it is a subset of AI techniques. In this case we focus on building systems that are able to learn and perform actions for which they are not specifically programmed. It is complex because in complex contexts a symbolic representation of the world can be impossible.

Machines learning algorithms usually work following the scheme below:



Let's analyze these steps closer.

Feature Extraction → it is the process of describing world through descriptors able to capture very specific facts (ex. HFCC). It is very difficult to manually select the correct type of features for a certain model. For this reason we use automatic feature selection algorithms. These are three families of features selector algorithms:

- Feature Selection algorithms** → they reduce the feature space by selecting a subset of features by removing irrelevant or redundant data. This allows to:
 - improve predictive accuracy, because model can focus on most informative features.
 - decrease computational cost, because there are less features to process.
 - decrease model complexity, making it better work on not previously seen data.

The aim is to make this problem easier (left for the model and for humans). Sometimes the selected features has no meaning for an humans, but we trust this process because it is based on quantitative analysis of data. Let's see three important feature selection algorithms:

Sequential Forward Generation (SFG):

- ① We start with an empty feature set S.
- ② At each iteration of the algorithm, we add the most relevant feature to S. The "best feature" is chosen based on:
- ③ heuristics ends when we reach a certain model accuracy or when a certain number of features has been selected.

(measure of model accuracy, decrease of prediction error, improvement of the metrics)

Sequential Backward Generation:

- ① The set of features S contains all the available features at the beginning.
- ② At each iteration of the algorithm, we remove the less relevant feature from S. "Less relevant feature" is identified based on its impact on model accuracy.
- ③ The algorithm stops when we reach a desired set dimension or when an important accuracy loss is registered.

Bidirectional Generation (BG) → combines SFG and SBG. heuristics ends when both methods converge in an optimal set features.

Feature Reduction algorithms → these are techniques that aim to decrease the number of variables in dataset without losing information by generating a more compact and efficient representation of the features.

An important feature reduction algorithm is Principal Components Analysis (PCA). The aim here is to decrease the dataset size minimizing the lost information by reducing redundancy generated by correlated features.

→ finding a new coordinate system (principal components) that represents dataset in a more compact way.

The algorithm is the following:

- ① Center the data → we subtract the average of each feature to each value of that feature. In this way features will have zero-mean.
- ② Calculate covariance matrix → the covariance matrix measures how variables (in this case features) are correlated.
- ③ Calculate eigenvalues and eigenvectors of the cov-matrix → we use them to determine the new principal components. Normally:
 - eigenvalues → they measure the total data variance for each dimension (if eigenvalue is a high value, the corresponding dimension captures a lot of information).
 - eigenvectors → they represent the corresponding directions for each eigenvalue.
- ④ Ordering principal components → principal components (axes) are ordered based on their variance:
 - most variance in the new principal component is put in first axis

⑤ Project data → original data is projected in the principal components space: $x_{ij} = X(i) \cdot v_j$.

In step 6 we decide how many principal components (coordinates) we should keep based on how much cumulative variance we want to keep. For example, if our aim is to keep the 95% of the variance, we keep the first N components (ordered by decreasing variance) that allow us to keep that level of variance in our dataset.

- Feature Normalisation $\rightarrow$ most of times, datasets contain features highly varying in magnitudes, units and ranges. Since most of the machine learning algorithms use distance between two data points in their computations, this is a problem. We need to bring features to the same level of magnitudes.

There are three methods:

Standardisation (Z-score normalisation) $\rightarrow$ features are scaled so that they will have the properties of a standard normal distribution ($\mu=0, \sigma=1$).

The normalised features vector is obtained through the formula:
$$z = \frac{x - \mu}{\sigma}$$
 where μ = mean of all features, σ = standard deviation of all features

If data doesn't follow a normal distribution, don't worry well

Min-Max scaling $\rightarrow$ data is scaled in a fixed range, usually from 0 to 1:
$$z = \frac{x - \min}{\max - \min}$$

- Mean Normalisation $\rightarrow$ normalises data between -1 and 1 with $\mu=0$:
$$z = \frac{x - \text{mean}(x)}{\text{max}(x) - \text{min}(x)}$$

Usually, the best one is chosen by trying them all.

Train the model $\rightarrow$ train consists in 'learning a symbolic representation of the world'. There are two kind of learning process:

- Unsupervised Learning $\rightarrow$ it is a type of learning in which the model works on unlabeled data. The aim is to find patterns and structures in data.

There are three main unsupervised learning techniques:

- Clustering: it consists in grouping objects based on their similarity. The aim is to create clusters having high inner-cluster similarity, low inner-cluster similarity.

A very used clustering algorithm is the k-means algorithm:

- 1) Select k random centroids
 - 2) Assign each element of the dataset to the cluster of the nearest centroid
 - 3) Re-calculate centroids for each cluster and return to point 2.
- The algorithm ends when convergence is achieved.

In step 2, we can calculate distance on different ways:

- Euclidean distance $\rightarrow$ direct distance between two points.

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Manhattan distance $\rightarrow$ is under 'discretised' distance measure.

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

t-Distributed Stochastic Neighbor Embedding (t-SNE): it is a dimensionality reduction technique used to visually explore high dimensional complex data preserving similarity relations between points. The algorithm is:

- 1) Build a probability distribution in the high dimensional space: for each point we measure the probability that another point is near (based on a Gaussian kernel).
- 2) Sampling from the previously calculated probability distribution creates a similar distribution in the low-dimensional space, trying to preserve the same probability of similarity.
- 3) Optimisation: we try to minimise the Kullback-Leibler divergence between the two distributions.

Note that this is not a clustering method: it doesn't identify explicit clusters, it just helps to visualize structure and relations in the data.

- Supervised Learning $\rightarrow$ it is a type of learning in which model is trained using labelled data: for each input element there is a label (correct classification) that guides the model. In this way, model learns a function $f(x)$ that associates an output y to each input x .

In this case we divide the training data in train set and test set.

There are different types of supervised learning:

- Classification $\rightarrow$ the aim is to predict a category or a discrete value (ex: 'male' or 'female'). Some classification algorithms are:

- K-Nearest Neighbours (KNN) $\rightarrow$ the basic idea is to classify a new point x comparing it with the already classified data. The algorithm to classify a new point is the following:

- 1) Calculate the distance (Euclidean or Manhattan) between x and all the points in the dataset.
- 2) Select the k-nearest points to x .
- 3) The class assigned to the new point will be the one shared by the majority of the k-nearest points.

In this algorithm k is a crucial parameter:

- if we choose too big k $\rightarrow$ we use the model precision
- if we choose too small k $\rightarrow$ model becomes too variable to noise.

- Support Vector Machine (SVM) $\rightarrow$ it tries to divide two classes finding an optimal hyperplane. Hyperplane is chosen in order to maximize the margin between the two classes.

There are two cases:

- Linear data $\rightarrow$ in that case SVM try to find an hyperplane (or line) that divides points of the two classes.
- Not Linear data $\rightarrow$ in this case, SVM uses a kernel function to transform data in a multi-dimensional space in which the dataset becomes linearly separable.

The boundary between the two classes (hyperplane) is chosen based on the support vectors. These are the closest points to the hyperplane.

The hyperplane equation is: $f(x) = W^T \cdot x + b$

Regression $\rightarrow$ it is used to predict a continuous value. The aim is to find a function $f(x)$ that maps the inputs x on a continuous output domain y (a scalar computation). Let's see two regression algorithms:

- KNN regression $\rightarrow$ it is a KNN algorithm adapted for regression for each new x point to predict, this algorithm is the following:

- 1) Calculate the distance between x and all the points in the train dataset.
- 2) Select the k-nearest points (as in supervised regression we use mean or median as model)
- 3) Calculate the output y as the average of these k points.

Polynomial regression $\rightarrow$ it tries to model the relation between the input x and the output y using a polynomial function of order M . The function is $y(x, w) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M$ where w_i are the weights that determine the shape of the polynomial. The aim is to find the weights that minimize the difference between $y(x, w)$ and the actual value y . This is achieved by minimizing the loss function = quadratic mean error $E(w) = \frac{1}{2} \sum_{i=1}^n (y_i(x_i) - y_i)^2$

- Reinforcement Learning $\rightarrow$ the agent 1) takes an action $\rightarrow$ 2) receives rewards or punishment $\rightarrow$ 3) adapts its behaviour

Deep Learning

Deep Learning techniques are a sub category of machine learning techniques. It is useful in managing complex problems such as artificial vision or vocal recognition.

Let's compare this previously seen methods with this new one:

rule-based systems

They simulate a rule-based way of reasoning. They can be useful and more reliable in cases of simple problems, but they just could fit complex problems because is difficult for manually build a symbolic representation of the world.

(classical) machine learning systems

They simulate the way in which a human brain approaches a "single specific problem". These systems learn automatically, but they need to be fed with "manually" defined features.

deep learning systems

They simulate the way in which brain works. They automatically learn how to extract features, and can be used with not structured data (ex. images) being capable to automatically adapt to different problems.

Simple neuron → In order to understand how deep learning works, let's analyze the fundamental element of it: the neuron. A simple neuron is based on the following algorithm:

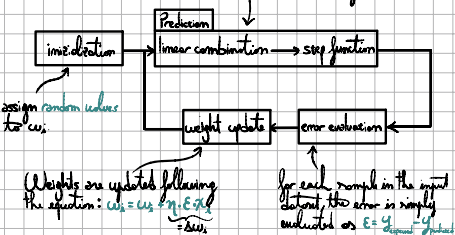
initialization → prediction → evaluate the prediction → improve model

This simple scheme can be implemented in different ways. Let's see two of them:

Perceptron → it is used to perform binary classification, obtaining a boolean output (0 or 1).

The behaviour of a perceptron is the following:

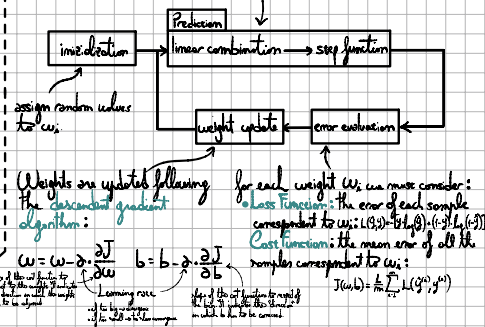
- ① A linear combination of the input is performed: $Z = x_1 \cdot w_1 + x_2 \cdot w_2 + \dots$
- ② The result is fed into the **step function**: $g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$



Logic Regression → it is used to perform binary classification obtaining a probabilistic output ($\in [0,1]$).

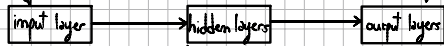
The Logic Regression algorithm is the following:

- ① A linear combination of the input is performed: $Z = x_1 \cdot w_1 + x_2 \cdot w_2 + \dots$
- ② The result is fed into the **sigmoid function**: $g(z) = \frac{1}{1 + e^{-z}}$



Multi-layer Perceptron (MLP) = Deep Learning → in order to solve non linear problems we have to concatenate neurons in a structure made like this:

The input layer is the input interface of this system. It identifies low-level features such as borders or pixels. It receives data such as speckograms or pixels.



The more we go deep in the model, the more features become abstract and generalized:

- mid-level features → identify shapes or other structures/curves.
- hi-level features → identify complex objects/complex texts/images.

the output layer produces the final results. Regarding the quantity of neurons, there are just two cases:

- classification → num. out. neurons = num. classes
- regression → num. out. neurons = cardinality of domains of output.

In the final layer we use the softmax function to get a probability distribution: $\hat{z}(x) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$

It's possible to use different activation functions such as:

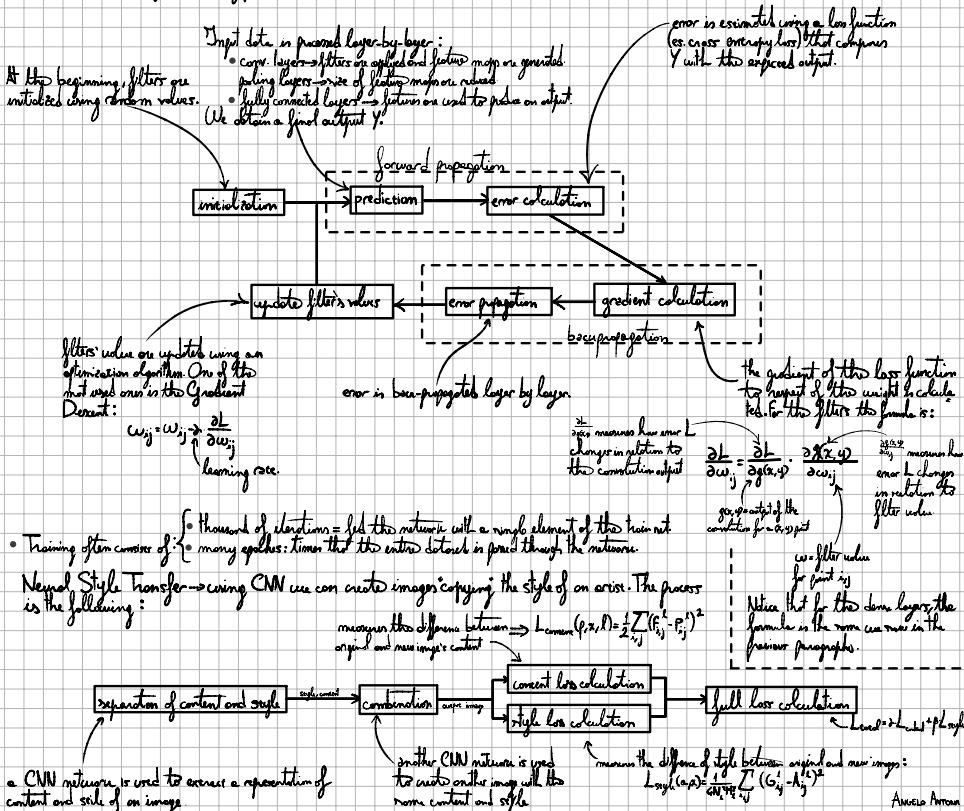
- Step function → some of perceptron.
- Sigmoid function → some of logic regression: $g(z) = \frac{1}{1 + e^{-z}}$
- ReLU (rectified linear unit) → the most used: $g(z) = \max(0, z)$
- ...

- The parameters that we can tune in a deep learning neural network are the number of levels, the number of neurons for each level, the activation function for each layer, the connection structure.

Convolutional NN

Convolutional Neural Network (CNN) → they are a specific type of Deep Learning Neural Network designed to work with grid structured data such as images.

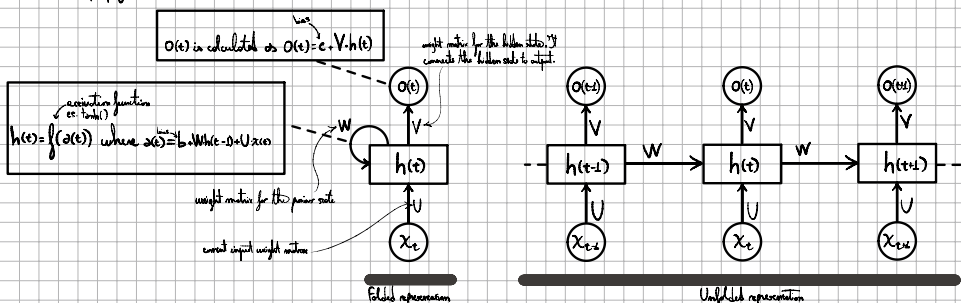
- **Convolution** → The fundamental operation in this type of networks. In convolution $g(x, y) = \sum_{i=1}^I \sum_{j=1}^J w(i, j) \cdot f(x-i, y-j)$. The way in which this operation is implemented in the neural network is the following:
 - ① **Definition of the filter** → the filter is defined as a little matrix (2D for B&W, 3D for R&G&B). The values inside this filter matrix (also called kernel) are usually initialized and then updated during the training.
 - ② **Selection of the receptive field** → we select the region of the image (same size of the filter) where we have to apply the filter.
 - ③ **Element-wise multiplication and sum** → we perform an element-wise multiplication between the kernel and the receptive field. We sum all the results to obtain a single scalar number (we also add the bias).
 - ④ **Activation function** → the scalar number is fed into the activation function (e.g. ReLU).
 - ⑤ **Shift the filter** → we apply the filter to another part of the image (select another receptive field) and start again from point ③. The process ends when the filter has been applied to the entire image.
- **Structure of Convolutional Neural Networks** → we can use different types of layers in CNN:
 - **Convolutional levels** → they apply filters on data using convolution. The "designer" of a layer measures how many filters are applied.
 - **Pooling layers** → they decrease the dimensionality of the feature map, maintaining the most important information. There are different types of pooling:
 - **max pooling** → considers only the max value in each sub-region of the feature map. It is useful to highlight the most evident features.
 - **average pooling** → considers the average value of the values of each sub-region of the matrix.
 - **sum pooling** → for each matrix sub-region it considers the sum of the values.
 - **Flattening layers** → it transforms multi-dimensional feature maps into mono-dimensional vector. This is necessary to connect convolutional layers to dense layers.
 - **Dense layers** → these layers are the ones that perform classification or regression as we have seen in the previous paragraphs.
- **Training of CNN** → during the training, the CNN learns the optimal values for the filters and the optimal weights for the dense layers. Let's see the steps of the training process:



Recurrent Neural Network

RNN are designed to manage sequential data. This makes them suitable for language recognition, automatic translation,...

Forward propagation:



- Training of RNN $\rightarrow$ also in this case, training consists in minimizing a loss function, reducing the error between predictions and expected values. Let's see how this loss is calculated:
 - Loss for a single temporal step $\rightarrow$ often is calculated using the Cross-Entropy loss: $L(t) = -y(t) \cdot \log(\hat{y}(t))$ (individual value at time t)
 - Loss for the entire temporal sequence $\rightarrow L = \sum_t L(t)$ (real value at time t)

So, at the end of the forward propagation, we calculate L . We have to back propagate this value in order to update network parameters. This process for RNN is called:

Backpropagation Through Time (BPTT) $\rightarrow$ error is Backpropagated from time T (last time-step) to $t=1$. Gradient are used to update parameters such as W, U, V, b, c .

The updates of the parameters is performed using a gradient optimization algorithm: $x = x - \alpha \frac{\partial L}{\partial x}$ (where x is one of the parameters of the network and α learning rate).

- Problems of 'classical' RNN: the main problems of this type of networks are:
 - Vanishing Gradient $\rightarrow$ during Backpropagation, gradient can become really small, making the network not capable of learning long-term dependencies.
 - Exploding Gradient $\rightarrow$ during Backpropagation, gradient can become really big. This leads to numerical instability during train.

- Solutions for these problems $\rightarrow$ a solution to these problems could be use the LSTM (Long Short Term Memory) or GRU (Gated Recurrent Unit).
- LSTM $\rightarrow$ this type of Network is characterized by neurons that have a 'more advanced memory management'.

Each neuron in the network is characterized by its state $h(t)$ (memory) and a set of gates that interact with this memory. The gates are:

- Forget Gate $\rightarrow$ decides which informations of the previous state need to be updated. The equation that this gate follows is: $f(t) = \sigma(W_f \cdot [h(t-1), x(t)] + b_f)$. If $f(t)$ is near 0, normally, the information is deleted, otherwise it is saved.
- Input Gate $\rightarrow$ determine the new information that we have to save in memory. The formula is: $i(t) = \sigma(W_i \cdot [h(t-1), x(t)] + b_i)$. The current memory is determined as a linear combination between the past and the new information: $C(t) = f(t) \cdot C(t-1) + i(t) \cdot \tilde{C}(t)$.
- Output Gate $\rightarrow$ decides which information of the hidden state must be sent in output. The formula is: $o(t) = \sigma(W_o \cdot [h(t-1), x(t)] + b_o)$. The current section h is determined applying an activation function to the memory: $h(t) = o(t) \cdot \tanh(C(t))$.

Types of RNN $\rightarrow$ Recurrent Neural Network can be divided in various typology depending on their number of inputs and outputs:

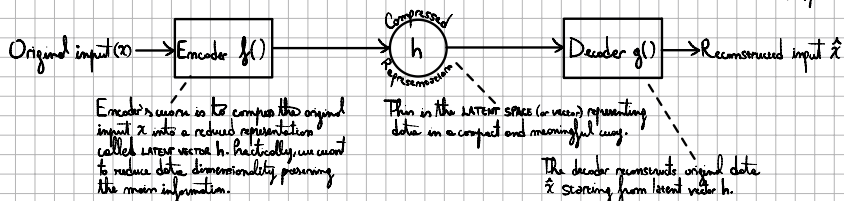
- One-to-one: it is not a properly Recurrent Network. Has an input pattern a single output (e.g. simple image classification).
- One-to-many: in this case, a single input is used to generate a sequence of output (e.g. generate a melody starting from a target emotion).
- Many-to-one: we can obtain a single output starting from an input sequence (e.g. classify the emotion of a song).
- Many-to-many: there are two cases:
 - sync. many-to-many $\rightarrow$ size in = size in (e.g. classify every frame of a video, getting a label for each frame).
 - async. many-to-many $\rightarrow$ size out $\neq$ size in (e.g. translation from a language to another).

Generative Neural Networks

Generative Neural Networks are used to generate new data similar to the one used for training.

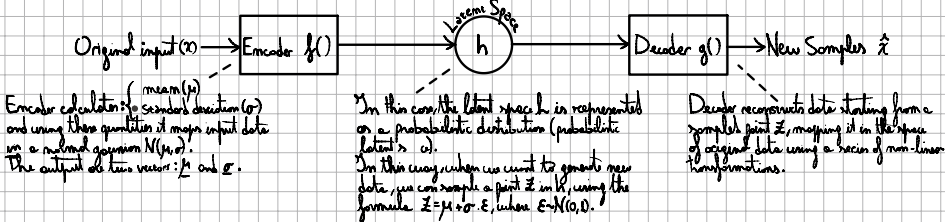
- Differences between discriminative and generative NN $\rightarrow$ in order to understand better the context, let's analyze the difference between the already analyzed NNs and this new model:
 - Discriminative models $\rightarrow$ these are the models that we saw until now. Their aim is to classify and classify data ($P(Y|X)$) (i.e. predict new things from the old).
 - Generative models $\rightarrow$ they focus on modelling data distribution ($P(X)$). They create new samples that are coherent with the train set (but not in original data).
- In the next paragraphs we will see how to build a model that can generate samples, starting from AE that can only represent data in a compressed way, going to VAE that can actually generate new samples.
- Autoencoders $\rightarrow$ this is a type of Generative NN that can be used in different contexts. Before seeing them, let's analyze the general structure:

(unsupervised learning technique)



- Training of autoencoders $\rightarrow$ also in this case, training is performed feeding the network with a train set and minimizing a loss function. Given that $\hat{x} = f(x)$, the loss function will be something like $L(x, \hat{x}) = L(x, g(f(x)))$.
- Autoencoders application $\rightarrow$ it's possible to use autoencoders to perform:
 - dimensionality reduction: as a non-linear version of the PCA. Differently from PCA, autoencoders can learn curves instead of hyperplanes, representing data distribution in a better way.
 - feature learning: can be used to learn important characteristics from data in unsupervised way.
 - denoising: can remove noise from data (e.g. images), learning how to retrieve original data removing undesired components.

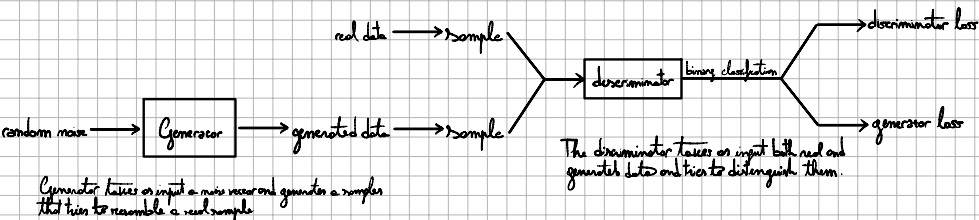
Variational Autoencoders (VAE) $\rightarrow$ they are an advanced version of AE, that can not only reconstruct data, but also generate new samples. The structure is similar to AE:



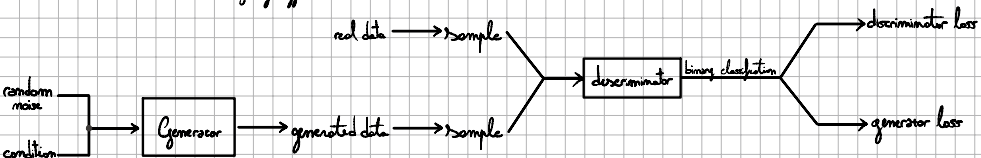
Also in this case, training is performed minimizing a cost function. In this case, the cost function has two components:

- reconstruction loss $\rightarrow$ measures the difference between x and $\hat{x}$: $L_{rec} = ||x - \hat{x}||^2$.
 - regularization loss $\rightarrow$ measures how much generators in the latent space distribution: $L_{reg} = KL(N(\mu, \sigma) || N(0, 1))$.
- (where N is a normal distribution)
- $L_{cost} = L_{rec} + L_{reg}$

- Generative Adversarial Networks (GANs) $\rightarrow$ it is another approach to new data generations. The structure of GANs is the following:



- Training of GANs $\rightarrow$ in this case the aim of training is divided in two parts:
 - ① train the discriminator $\rightarrow$ the aim is to update the weights of the discriminator in order to minimize the error in the binary classification between real and generated data (minimize $\mathbb{E}_r [L_D(x_r)] - \mathbb{E}_g [L_D(G(z))]$).
 - ② Train the generator $\rightarrow$ the aim is to update the weights of the generator in order to maximize its ability to trick the discriminator (minimize $\mathbb{E}_z [L_D(G(z))]$).
- Conditional GANs (cGANs) $\rightarrow$ it's a particular type of GANs that adds a condition to the generation process (ex generate human faces of 'male'). In this case the architecture is slightly different:



Evolutionary Algorithms

These techniques are inspired by Darwin's evolutionary theory: they simulate natural selection to solve complex problems in a non-deterministic way, exploring the solution domain in a creative way.

• Natural Evolution principles:

- Selection → some individuals have higher probability to survive and reproduce, transmitting their features to their descendents.
- Heredity → parents features are inherited from children, allowing the transmission of good characteristics.
- Variation → mutations and recombination introduce diversity, making easier for the 'population' to adapt to new environments.

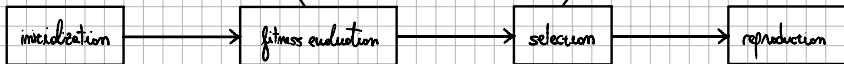
Evolutionary algorithms apply these principles to solve computational creative problems.

• Genetic Algorithms → they are a sub-class of evolutionary algorithms that works on solution populations. Their generic structure is:

Each individual is evaluated using the fitness function. Individuals having higher fitness have higher probability to be selected for reproduction.

The "most adapted" individuals are selected for reproduction. There are different approaches:

- elitist → we select the best individuals in a deterministic way.
- probabilistic → individuals are selected probabilistically, proportionally to their fitness.



A population of random solutions is generated.

Each solution (individual) is characterized by:

• Genotype = computational representation of the individual, compared by:

• Chromosome = it is the 'genetic representation' of the individual (sequence of genes)

• Gene = it is a certain position in the chromosome.

• Allele = it is the value that the position assumes

Phenotype = real manifestation of the individual (solution)

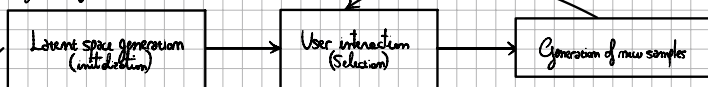
Reproduction generates new individuals combining genetic material of the two parents. There are two mechanisms:

- Crossover → it combines parents' genes.
- Mutation → it introduces random variations in order to maintain genetic variability.

Deep Interactive Evolution → it is a hybrid technique that combines Deep Generative Models with Evolutionary Computation. There are two main reasons to use this kind of algorithm:

- Subjective fitness function: instead of mathematically define objectives, user preference guides the evolution of the process.
- Reduce data complexity: instead of working with raw data (e.g. pixels) the model finds a latent space, which captures structure and meaningful data representation.

The process is the following:



A deep generative algorithm, such as VAE or GAN, is pre-trained on a dataset of relevant samples.

data → encoder → latent space → decoder → reconstructed data

These samples serve as the initial population in the evolutionary system.

The generated samples are presented to the user (decoder is used). Users evaluate and select the samples that align with their preference or desired qualities. This is important because in order to be able to mathematically define a fitness function on objective aim.

The algorithm generates a new population of samples by applying genetic operations (mutation and crossover) to the latent vectors of the selected samples.

