**UNIVERSITATEA BABEŞ-BOLYAI**

**Facultatea de Ştiinţe Economice şi Gestiunea Afacerilor**

**Informatică Economică licență, învățământ la distanță**

**Lucrare de licenţă**

Absolvent,

Liviu Flavius **BRÎNZĂ**

Coordonator ştiinţific,

Conf. univ. dr. Monica **CIACA**

**2019**

**UNIVERSITATEA BABEŞ-BOLYAI**

**Facultatea de Ştiinţe Economice şi Gestiunea Afacerilor**

**Informatică Economică licență, învățământ la distanță**

**Lucrare de licenţă**

Automatizarea casei cu echipamente interconectate prin Z-Wave, folosind Inteligență Artificială pentru interfațare

Absolvent,

Liviu Flavius **BRÎNZĂ**

Coordonator ştiinţific,

Conf. univ. dr. Monica **CIACA**

**2019**

**Rezumat**

Prezenta lucrare reprezintă soluția propusă pentru realizarea unui sistem de casă inteligentă offline, implementat pe echipamente simple și accesibile publicului larg, capabil să interacționeze cu utilizatorii utilizând limbajul natural ca date de intrare, interpretarea acestora folosind o rețea neuronală pentru clasificarea comenzilor și executarea lor cu ajutorul unor componente capabile să comunice prin protocolul Z-Wave. Totodată, se observă avantajele și limitările rețelelor neuronale de tip “feed forward”, precum și starea curentă de maturitate a modulelor integrabile de control Z-Wave. La nivel de interfață s-a folosit mediul de programare vizuală Node-RED și interconectarea dintre nivelul de prezentare (Front End) și cel de logică (Back End) s-a realizat folosind protocolul standard MQTT, ambele menite să faciliteze integrarea echipamentelor fizice.

# Cuprins

[Abrevieri <TO BE ADDED> v](#_Toc536483610)

[Lista figurilor vi](#_Toc536483611)

[Introducere 1](#_Toc536483612)

[1. Internetul lucrurilor și automatizare 3](#_Toc536483613)

[1.1 Casă inteligentă 3](#_Toc536483614)

[1.2 Protocolul Z-Wave 4](#_Toc536483615)

[1.3 Node-RED 7](#_Toc536483616)

[1.4 Protocolul MQTT 8](#_Toc536483617)

[2. Inteligență Artificială 10](#_Toc536483618)

[2.1 Rețea neuronală pentru clasificare 12](#_Toc536483619)

[*2.1.1 Perceptronul* 14](#_Toc536483620)

[*2.1.2 Antrenarea rețelei* 18](#_Toc536483621)

[*2.1.3 Backpropagation* 21](#_Toc536483622)

[2.2 Procesarea limbajului natural 22](#_Toc536483623)

[*2.2.1 Codarea „one hot”* 23](#_Toc536483624)

[*2.2.2 Identificarea formei de bază și a rădăcinii cuvintelor* 25](#_Toc536483625)

[3. Proiectarea conceptuală 26](#_Toc536483626)

[3.1 Cerințele sistemului 26](#_Toc536483627)

[3.2 Proiectarea logică 27](#_Toc536483628)

[3.3 Arhitectura componentelor hardware 28](#_Toc536483629)

[3.4 Arhitectura componentelor software 29](#_Toc536483630)

[3.5 Baza informațională 32](#_Toc536483631)

[*3.5.1 Datele de antrenare* 32](#_Toc536483632)

[*3.5.2 Datele de intrare* 33](#_Toc536483633)

[4. Proiectarea Tehnică 35](#_Toc536483634)

[4.1 Interfața 38](#_Toc536483635)

[*4.1.1 Panoul Input* 40](#_Toc536483636)

[*4.1.2 Panoul Lighting* 42](#_Toc536483637)

[*4.1.3 Panoul Heating* 44](#_Toc536483638)

[4.2 Prelucrarea comenzilor 45](#_Toc536483639)

[4.3 Implementarea modulului client MQTT 47](#_Toc536483640)

[4.4 Implementarea rețelei neuronale pentru clasificare 48](#_Toc536483641)

[4.5 Implementarea modulului de control Z-Wave 51](#_Toc536483642)

[5. Limitări și posibile îmbunătățiri viitoare 53](#_Toc536483643)

[5.1 Problema minimei locale 53](#_Toc536483644)

[5.2 Structura rețelei neuronale feed forward 54](#_Toc536483645)

[5.3 Limitările librăriei python-openzwave 56](#_Toc536483646)

[5.4 Metode alternative de interacțiune 57](#_Toc536483647)

[6. Concluzii 59](#_Toc536483648)

[Bibliografie 60](#_Toc536483649)

[Anexe 61](#_Toc536483650)

# Abrevieri <TO BE ADDED>

*IP,* Internet Protocol

*GPIO,* General Purpose Input Output

*MQTT,* Message Queuing Telemetry Transport

*ReLU,* Rectified Linear Unit

*JSON,* JavaScript Object Notation

*FFNN*, Feed Forward Neural Network

RNN, Recurrent Neural Network

LSTM, Long Short Term Memory

DRNN, Deep Recurrent Neural Network

BRNN, Bidirectional Recurrent Neural Network

# Lista figurilor

**Figuri:**

[**Figura 1**. Logourile specifice echipamentelor certificate Z-Wave și Z-Wave plus 4](file:///E:\GitBase\alice\documentation\Documentatie_licenta.docx#_Toc536379787)

[**Figura 2**. Vizualizarea rețelei de tip plasă 5](#_Toc536379788)

[**Figura 3**. Exemplu de implementarea unui flow in Node-RED 8](#_Toc536379789)

[**Figura 4**. Componente de interfață specifice Node-RED 9](#_Toc536379790)

[**Figura 5.** Fluxul de date într-o rețea MQTT 10](#_Toc536379791)

[**Figura 6**. Structura și funcțiile unui neuron biologic 13](#_Toc536379792)

[**Figura 7.** Componentele preceptronului 14](#_Toc536379793)

[**Figura 8**. Exemple de funcții de activare 15](#_Toc536379794)

[**Figura 9.** Structura funcțională a reșelelor neuronale 16](#_Toc536379795)

[**Figura 10**. Recunoașterea caracteristicilor într-o rețea neuronală 17](#_Toc536379796)

[**Figura 11**. Rețea neuronală pentru însumarea a două cifre 22](#_Toc536379797)

[**Figura 12.** Codare și decodare "one hot" 23](#_Toc536379798)

[**Figura 13.** Diagrama fluxului de date 26](#_Toc536379799)

[**Figura 14.** Componentele hardware utilizate 27](#_Toc536379800)

[**Figura 15.** Componentele software utilizate 29](#_Toc536379801)

[**Figura 16.** Conținutul trainingCorpus.txt 31](#_Toc536379802)

[**Figura 17.** Diagrama Use-Case 33](#_Toc536379803)

[**Figura 18.** Diagrama de secvență a inițializării sistemului 34](#_Toc536379804)

[**Figura 19**. Diagrama de secvență a prelucrării comenzilor 35](#_Toc536379805)

[**Figura 20**. Diagrama de clase 35](#_Toc536379806)

[**Figura 21**. Structura internă a claselor 36](#_Toc536379807)

[**Figura 22.** Interfața aplicației 37](#_Toc536379808)

[**Figura 23.** Elemenetele panoului "Input" 38](#_Toc536379809)

[**Figura 24.** Elementele panoului "Lighting" 40](#_Toc536379810)

[**Figura 25.** Elementele panoului "Heating" 42](#_Toc536379811)

[**Figura 26**. Pașii generării lexicului 44](#_Toc536379812)

[**Figura 27**. Codarea datelor de antrenare 44](#_Toc536379813)

[**Figura 28**. Vizualizarea optimizării Adam 47](#_Toc536379814)

[**Figura 29**. Viteze de convergență a funcțiilor de optimizare 48](#_Toc536379815)

[**Figura 30**. Optimizarea funcției cost și a acurateții 48](file:///E:\GitBase\alice\documentation\Documentatie_licenta.docx#_Toc536379816)

[**Figura 31**. Minimele locale ale unei funcții 51](#_Toc536379817)

# Introducere

Automatizarea de case inteligente și Inteligență Artificială reprezintă două noțiuni care au cunoscut un avânt remarcabil în industria IT din ultimii ani, tot mai mulți producători lansând pe piață propriile soluții pentru îmbinarea lor, reorientându-și clientela țiintă de la firme de producție, către persoane fizice. Ele oferă posibilitatea integrării unui număr mare de echipamente inteligente într-o manieră agnostică de producător, precum și imbunătățiri continue prin actualizări recurente de software. Totuși, majoritatea soluțiilor cunoscute au două dezavantaje semnificative pentru utilizatorul de rând: prețul unui sistem complet, precum și necesitatea accesului continuu la internet; mai concret, la serverele producătorului, unde se realizează majoritatea procesărilor.

Proiectul descris reprezintă rezultatul încercării de a evita dezavantajele menționate mai sus, prin crearea unei soluții complet offline, izolat la un singur echipament de control și la un număr de echipamente controlate, toate accesibile ca preț pentru potențiali cumpărători, care totodată oferă destule funcționalități pentru a fi utilizate într-un mediu inteligent. Tehnologiile utilizate sunt, de asemenea, disponibile publicului, astfel încât modificările sau extinderile ulterioare sunt și ele, posibile, fără un efort semnificativ.

Deoarece utilizabilitatea a fost unul dintre cele mai importante aspecte în dezvoltarea aplicației, am optat pentru utilizarea și interpretarea limnajului natural ca mijloc de interacțiune, în defavoarea comenzilor specifice de control de forma butanelor, sliderelor, sau a pictogramelor interactive, tipice pentru astfel de soluții de automatizare. Dat fiind faptul că formarea propozițiilor poate varia semnificativ de la un utilizator la altul, hardcodarea tuturor comenzilor specifice ar fi fost o limitare nedorită a soluției propuse, tocmai din acest motiv am optat pentru utilizarea unei rețele neuronale de clasificare, antrenat specific pe posibile comenzi dedicate interacțiunii cu proiectul, facilitând interpretretarea cu succes a comenzilor similare semantic, dar care diferă sintactic.

Interacțiunea cu echipamentele dedicate (în cazul de față, un bec și un termostat) s-a realizat folosind protocolul de comunicare Z-Wave, acesta asigurând prin natura sa interoperabilitate între echipamente și unitatea de control, toate de la diferiți producători,chiar și la distanțe mari. Totodată, se asigură și menținerea rețelei tip plasă creată între nodurile sale, chiar și în cazul în care vreunul devine indisponibil, sau inoperabil, din diferite motive.

Structura lucrării va include în Capitolul 1 concepte generale și noțiuni necesare înțelegerii diferitelor tehnologii utilizate din Internetul Lucrurilor (IoT), Capitolul 2 va prezenta o scurtă istorie cât și noțiunile de Inteligență Artificială folosite, fragmentând subcapitolele astfel încât să urmeze fluxul logic al prelucrării informațiilor de la *input* la *output*, Capitolul 3 va prezenta modul în care noțiunile și tehnologiile abordate în capitolele precedente vor fi utilizate în descrierea logică al proiectului, în Capitolul vom insista asupra proiectării tehnice, iar în Capitolul 5 vom analiza limitările curente observate de-a lungul procesului de implementare, oferind posibile soluții de îmbunătățire.

# Internetul lucrurilor și automatizare

În cadrul acestui capitol, voi aborda principalele concepte și tehnologii de IoT și care stau la baza proiectului descris, într-un context generalizat.

Internet of Things (IoT) și Industrial Internet of Things (IIoT) sunt două noțiuni care au cunoscut un avânt remarcabil în industria IT din ultimii ani. Majoritatea companiilor se orientează spre crearea propriilor soluții din acest domeniu, sau adaptarea unora deja existente la propriile culturi și politici, în scopuri comerciale. IoT presupune crearea unor rețele interconectate de dispozitive, servicii și sisteme automate sau automatizabile, accesibile și controlabile prin internet. În domeniu industrial, aceasta se transpune în posibilitatea supravegherii, controlului și prezicerii de activităților economice, oferind un timp de reacție mult mai prompt în fața unor situații nou apărute. Aceste beneficii, la rândul lor, se traduc în cheltuieli mai mici și câștiguri mai mari. În domeniul casnic, interconectivitatea dispozitivelor electrice și controlul acestora permite utilizatorului să monitorizeze și să fie notificat în timp util de orice eveniment petrecut la nivelul spațiului automatizat, eficientizând, astfel, costurile de trai un grad de confort sporit. În ambele cazuri de implementare, avantajele IoT-ului sunt clar definite: oferă posibilitatea monitorizării și controlării mediului.

Subramura acestui domeniu cel mai des exploatat de persoanele fizice o constituie automatizarea caselor.

## Casă inteligentă

Noțiunea de casă inteligentă exista deja în anii 1980, când implementarea lor semăna mai mult cu o mașinărie Rude Goldberg. Chiar dacă acest concept nu mai reprezintă o noutate în ultimii ani, este totuși unul destul de puțin cunoscut pentru cei mai mulți dintre locuitori și cu atât mai puțin pus în practică. Principalele argumente împotriva implementării unor astfel de soluții o reprezintă cele de natură financiară, în contextul în care realizarea acestor sisteme implică achiziționarea unui număr semnificativ de echipamente specializate, precum și viabilitatea soluțiilor care, până nu demult, întâmpinau dificultăți în funcționalitate, accesibilitate și securitate. În ultimii ani, însă, odată cu focalizarea atenției producătorilor pe această arie și creșterea concurenței pe piață, prețurile de achiziționarea componentelor au scăzut la un nivel acceptabil, iar utilizabilitatea lor a crescut semnificativ, aceste schimbări explicând creșterea atenției acordate de consumatori, pentru astfel de produse.

Realizarea unui astfel de proiect înseamnă echiparea locuinței cu diferiți senzori, monitorizarea acestora și controlarea unor aparate fie manual, fie printr-un sistem automatizat, astfel încât condițiile monitorizate să se încadreze permanent în parametrii prestabiliți, sau pentru a iniția unele rutine dorite.

Cele mai des automatizate aspecte se referă la: temperatură, luminozitate, supraveghere și acționarea unor aparate electrice pentru controlarea ușilor, sistemelor audio – video, aparatelor casnice, etc.

Cea mai des întâlnită soluție o reprezintă montarea unor senzori și / sau relee pentru controlarea echipamentelor electrice și conectarea acestora la o rețea pentru a putea fi monitorizate și accesate de către un sistem central (de obicei, un computer). Pentru realizarea unor astfel de rețele există diferite soluții la ora actuală, fiecare având avantaje și dezavantaje. Interconectarea echipamentelor se poate realiza prin mai multe tehnologii: conectivitatea prin cablu, infraroșu, bluetooth, GSM, rețea wireless simplă, sau prin protocoale radio dedicate: NFC, Zigbee, LoRa, Z-Wave.

## Protocolul Z-Wave

Reprezintă una dintre cele mai avantajoase protocoale de comunicare dedicată IoT-ului în general, dar axat mai ales pe echipamente pentru automatizare de case.

Introdus în anul 2001 de către firma daneză Zensys, acesta era destinat utilizării în sisteme de auntomatizări de case, însă era necunoscut publicului până în 2005, când s-a format consorțiul “Z-Wave Alliance”, scopul acestuia fiind de a injecta protocolul Z-Wave pe piață, prin crearea unui cadru unanim de utilizare și implementare în cât mai multe echipamente hardware. Această inițiativă și-a atins scopul, numărul echipamentelor echipate cu protocolul Z-wave crescând de la 6 în 2005, la peste 2.400 în Aprilie 2018.

**Figura 1**. Logourile specifice echipamentelor certificate Z-Wave și Z-Wave plus



În anul 2013 s-a introdus chipul dedicate seria 500, numit și Z-Wave Plus, care oferă o creștere semnificativă în capacitatea internă a memoriei, a distanței de comunicare wireless și a utilizării bateriei, acestă formă de protocol îmbunătățită fiind compatibil și cu cel Z-Wave standard, permițând interoperabilitatea lor în aceași rețea.

Caracteristici tehnice de menționat:

* Topologia rețelei de tip plasă: Spre deosebire de comunicarea tip wireless clasică, în care două componente trebuie să fie în aceași arie de acoperire pentru realizarea unei comunicări, Z-Wave crează o rețea de tip plasă între toate echipamentele compatibile, în scopul extinderii ariei de acoperire și oferă posibilitatea transmiterii informației printre toate componentele conectate, până ce aceasta ajunge la destinație (information bouncing). Totadă, are capacitatea de a monitoriza buna funcționalitate a componentelor din rețea și în cazul unei defecțiuni, își poate reconfigura structura rețelei pentru a genera noi trasee de comunicare.



**Figura 2**. Vizualizarea rețelei de tip plasă

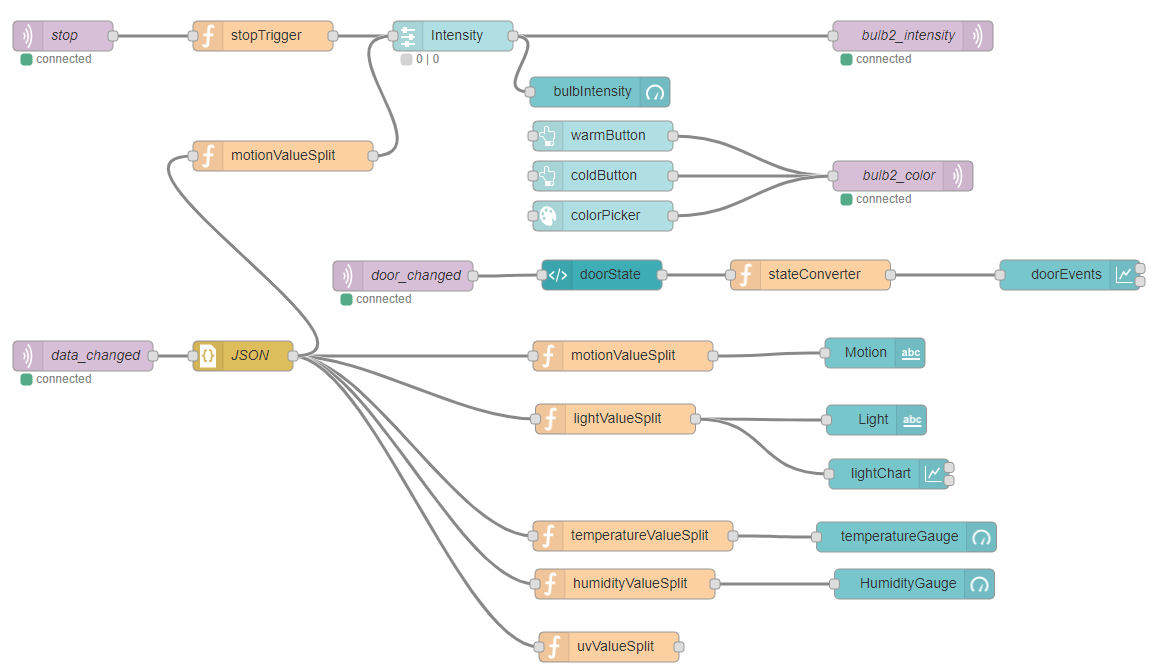
În figura 2 se observă configurația unei rețele de tip plasă, în care nodul de control, deși este situat în afara ariei de acoperire al uni nod din extremitatea rețelei, acestea pot totuși transmite informații datorită posibilității information bouncing-ului.

* Bandă de frecvență joasă: Datorită frecvenței de operare între 800 – 900 MHz (aceasta variind în funcție de țară), Z-Wave permite o transmitere sigură și rapidă de pachete mici de date, cu rate de transfer de până la 100 kb/s. Din însăși natura comunicării dintre echipamente, nu este necesară transmiterea unui număr mare de pachete de date și nici mărimile acestore nu trebuie să fie foarte mari. Banda de frecvență joasă permite o distanță maximă de comunicare între noduri independente de până la 40 m, în cazul protocolului Z-Wave standard și până la 100 m, în cazul protocolului Z-Wave Plus.
* Identitatea unică a echipamentelor în cadrul unei rețele: Fiecare rețea de Z-Wave se identifică unic printr-un ID propriu, generat la nivel de controller. Odată generat, acest identificator se propagă prin toate nodurile rețelei, permițând astfel izolarea echipamentelor print apartenența lor la aceași rețea. Înainte de a putea fi utilizat, fiecare echipament trebuie inclus manual într-o rețea Z-Wave, cu ajutorul nodului de control. Acesta poate include sau exclude (pairing / unpairing) echipamente din rețeaua controlată, printr-o rutină specifică. În momentul includerii unui nou nod, controllerul îî trimite acestuia identificatorul rețelei în care se include, un identificator nou generat al nodului nou incluls, cât și informații adiționale de criptare a mesajelor transmise, dacă acesta au fost configurate în prealabil. Datoriă acestei metodologii de apartenență, este posibil ca în același perimetru să funcționeze un număr nelimitat de rețele independente, dără interferență, Nodurile unei rețele fiind incapabile de a intercepta sau a transmite ale informații dintr-o rețea alăturată.
* Interoperabilitate garantată: Protocolul Z-Wave se bazează pe clase de comandă unice, trimise între echipamente. Există trei grupe mari de comenzi cu care se operează: comenzi de bază, comenzi generice și comenzi specifice.
  + **Comenzi de bază**: acestea sunt definite la nivel de protocol Z-Wave și sunt universal valabile pentru toate tipurile de echipamente, indiferent de tipul lor, sau de producător. Acestea definesc funcționalitățile de bază ale unui nod și rolul acestuia în interiorul rețelei.
  + **Comenzi generice**: definesc la nivel abstract, funționalitățile nodului. Acestea sunt necesare pentru a putea identifica tipul echipamentului (ex. de iluminat, de termoficat, de securitate, utilitară, sau de confort)
  + **Comenzi specifice**: sunt definite generic, dar interpretarea lor depinde strict de nodul care le utilizează (ex. Setare de valoare, interogare de câmp, etc.)

Combinând cele 3 tipuri de comenzi, se realizează comunicarea în interiorul unei rețele, între toate nodurile sale, indeiferent de capacitățile sale, ori de producător

## Node-RED

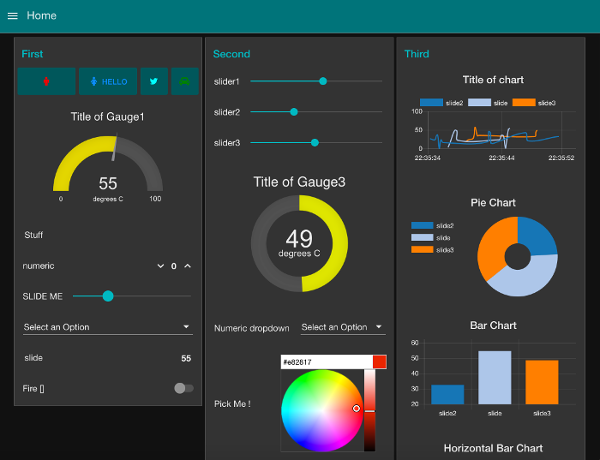
Este un mediu de programare vizuală, open source, dedicată integrării echipamentelor hardware în proiecte de IoT. Acesta permite utilizarea unui număr vast de protocoale de comunicare, API-uri și servicii online, cât și prelucrarea mai multor structuri de date. Permite dezvoltarea pe bază fluxurilor de date, descriind modul de operare și interconectarea diferitlor noduri de lucru, palete și elemente vizuale, printr-un editor accesibil direct din browserul web. Aplicațiile vizuale astfel create, pot fi deployate în serverul web proprietar Node-RED, ca serviciu, acestea fiind accesibile folosind adresa IP al echipamentului pe care acestea rulează.



**Figura 3**. Exemplu de implementarea unui flow in Node-RED

Node-RED oferă o gamă largă de elemente (noduri) de lucru, variind de la tipuri generice (input de date prin diferite protocoale) până la unele specializate pe echipamente de lucru (acces la pini GPIO al unui Raspberry PI), dar totodată oferă posibilitatea dezvoltării propriilor noduri, folosind limbajul Node.js și extensiile sale pentru implementare.

Elementele vizuale disponibile sunt ușor configurabile și utilizabile, permițând dezvoltarea rapidă a unor interfețe de lucru, specifice nevoilor fiecărui proiect în parte. Acestea se adaptează automat mediului de prezentare, restructurându-se automat în funcție de rezoluția ecranului.



**Figura 4**. Componente de interfață specifice Node-RED

Detalii specifice de implementare și de utilizare se găsesc pe pagina web oficială <https://nodered.org/>

## Protocolul MQTT

Message Queuing Telemetry Transport (abreviat MQTT) este un protocol simplu de transmiterea mesajelor pe principiul publicării / subscrierii dezvoltat pentru interconectarea dispozitivelor într-un mediu cu resurse limitate (lățime de bandă redusă, conectivitate sporadică), dezvoltat de IBM în colaborare cu Arcom, în anul 1990.

Într-un sistem MQTT, mai mulți clienți comunică cu un server, denumit și broker. Un client poate fi subscris pentru a primi anumite informații și poate, de asemenea, publica informații. Subscrierea și publicarea de infiormații se grupează pe ierarhie topicuri. În momentul în care un client publică un mesaj, acesta transmite numele topicului pe care dorește să transmită informația, cât și pachetul mesaj, către broker. Acesta, la rândul său, retransmite informația tuturor clienților care s-au subscris la topicul de transmisie.



**Figura 5.** Fluxul de date într-o rețea MQTT

Conexiunea la nivel de rețea se bazează pe protocolul TCP, controlul fluxului de date cât și securitatea la nivelul de transport fiind realizat de acesta. Singura măsură configurabilă de transmiterea datelor la nivelul MQTT se referă strict la Calitatea serviciului (QoS – Quality of Service). În acest context, conexiunea dintre client și broker se poate configura în 3 variante:

* Cel mult odată: în acest fel, mesajul se transmite fără așteptarea vreunei confirmări din partea partenerului de conversație, pe principiul “fire and forget”
* Cel puțin odată: transmiterea mesajul este reîncercată până se primește o confirmare din partea partenerului de conversație, pe principiul “acknowledgement delivery”
* Exact odată: expeditorul și receptorul realizează un “handshake” bidirectional pentru asigurarea transmiterii mesajului o singură dată, în condiții de siguranță, pe principiul “assured delivery”.

Specificațiile protocolului MQTT sunt documentate extensiv pe pagina web oficială <http://mqtt.org/documentation>

# Inteligență Artificială

În acest capitol vom încerca să definim noțiunea de Inteligență Artificială, cât și evoluția istorică a acesteia, până la aplicabilitatea sa în zilele noastre. Ne vom axa pe rețele neuronale, tipurile acestora și utilitatea lor în contextul automatizării de case inteligente.

Inteligența este un termen foarte complex și ambiguu. Se poate defini ca logică, înțelegere și învățare, conștiință, planificare, creativitate și rezolvare de probleme. Noi, oamenii, ne considerăm inteligenți pentru că utilizăm toate noțiunile enumerate, percepem mediun înconjurător, învățăm și ne planificăm acțiunile în funcție de experiențe și inputuri senzoriale. În acest context putem folosi sintagma de Inteligență Naturală.

În cazul Inteligenței Artificiale, ne referim la o subramură a informaticii, în contextul mașinilor, computerelor și a programelor care le ghidează. Astfel de mașinării mimică funcțiile cognitive ale entităților vii analizând mediul înconjurător, procesând informații, și deducând concluzii logice bazate pe recompense și experiențe dobândite prin procese de învățare.

În 1950 Alan Turing publica în revista Mind o lucrare bazată pe conceptual inteligenței artificiale. Conform lui, o mașină care ar fi capabil să converseze cu o ființă umană, fără ca acesta să poată realiza că partenerul său de conversație nu este om, ar câștiga “jocul imitației” și ar putea fi considerat intelligent. Acest test urma să devină arhicunoscutul Test Turing, un etalon în evaluarea inteligenței artificiale și în zilele noastre.

Paralel cu cercetările filozofice ale lui Alan Turing, mai mulți ingineri încercau implementarea unor mecanizme bazate pe teoria jocurilor și pe lucrările lui John von Neumann, capabile să analizeze risuri și recompense pentru a planifica acțiuni viitoare. În acest context, în anul 1951, un aparat numit Ferranti Mark 1, reușește în premieră șă câștige un joc de șah împotriva unui jucător uman, bazându-se pe algoritmi statistici. Chiar dacă la momentul respectiv aceasta era o realizare semnificativă, faptul că procesarea tuturor posibilităților de mutare și alegerea celei optime dura între 15 – 20 de minute, făcea ca soluția să nu fie una utilizabilă. La scurt timp după, în 1957, Allen Newell și Herbert Simon reușeau dezvoltarea unei aplicații capabile să resolve probleme generale de matematică.

În 1956, un tânăr profesor de matematică numit John McCarthy a organizat conferința numită “Dartmouth Summer Research Project on Artificial Intelligence”, în esență o sesiune de brainstorming despre viitoarele prospecte ale domeniului de Inteligență Artificială, locul unde această sintagmă s-a folosit pentru prima oară.

Anii ’60 s-au axat mai ales pe dezvoltarea algoritmilor pentru rezolvarea problemelor matemaice și a roboticii combinate cu învățare prin Machine Vision. Astfel, în 1972 a fost prezentat primul robot umanoid considerat inteligent, în Japonia, numit WABOT-1.

Din nefericire, puterea de procesare a calculatoarelor anilor ‘70 – ’90 era mult sub necesarul procesării volumului colosal de date cu care se lucra în domeniul inteligenței artificiale. Din acest motiv, atât investițiile cât și interesul în domeniu a scăzut dramatic, până aproape de stagnare. Această perioadă a fost denumită “Iarna Inteligenței Artificiale” și a ținut până la finele anilor ’90, când tot Japonia a lansat conceptul de a 5.a generașie de calculatoare, menite să promoveze și să dezvolte aria Inteligenței Artificiale. Chiar dacă această tentativă a fost mai degrabă o mișcare de marketing al guvernului Japonez, efectele sale s-au resimțit la nivel mondial, tot mai mulți ingineri reaprinzând flacăra IA-ului, iar în 1997, IBM prezenta primul sistem, numit Deep Blue, capabil să câștige un joc de șah împotriva campionul mondial Garry Kasparov, folosind Inteligență Artificială și nu algoritmi statistici. Arhitectura Deep Blue a facilitat dezvoltarea de sisteme care, bazate pe reguli de operare și experiență antrenată pe principiul cauză – efect, s-au reprofilat pe domenii specifice de activitate. Asemenea implementări au fost denumite sisteme expert. Există două tipuri de sisteme expert utilizate în diferite domenii economice:

* Motoare de cunoștiințe: acestea derivă rezultatele unei probleme din reguli de aplicarea a datelor
* Motoare de deducție: care generează rezultatele problemei utilizând date situaționale (stare curentă a sistemlui asupra căruia se operează), cât și reguli de aplicare a datelor

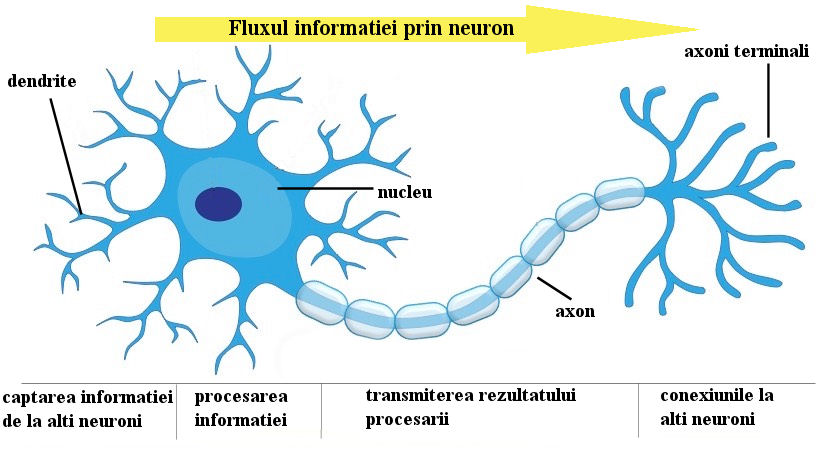
Viteza de dezvoltare exponențială a puterii computaționale cât și capacitățile de stochare a unui număr din ce în ce mai mare de date din ultimii 15 de ani, a stimulat companii mari ca Amazon, sau Google să profite de învățare automată (machine learning) pentru câștiguri comerciale. În afară de procesarea datelor utilizatorilor pentru înțelegerea clientelei, aceste companii și-au continuat progresele tehnologice în procesarea limbajului natural, recunoaștere de imagini, sisteme de clasificare și generatoare de date, precum și multe aplicații care utilizează Inteligență Artificială în operațiile sale zilnice. Utilizarea rețelelor neuronale în operații complicate de procesare, analiză și generare a facilitat explorarea unor noi soluții de implementare și această tendință va continua să se perfecționeze și în următorii ani.

## Rețea neuronală pentru clasificare

Una dintre subcategoriile ale “Inteligenței Artificiale”, recente apărute în aria de interes a cercetătorilor o reprezintă cea a rețelelor neuronale. Interesul acordat acestor structuri a cunoscut un avânt semnificativ în ultimii ani, proporțional corelat cu progresul tehnologic, datorită cantității mari de date utilizate și a vitezei de procesare cu care acestea pot fi prelucrate.

În lucrările de laborator ale Conf. dr. Radu Ionescu, membrul departamentului de informatică din cadrul Universității din București, rețelele neuronale sunt defininte ca “rețele de elemente simple puternic interconectate prin intermediul unor legături numite interconexiuni prin care se propagă informație numerică”[[1]](#footnote-1). Această definiție este una general acceptată și agreată de comunitatea cercetătorilor din domeniu. Așa cum se poate infera din numele lor, rețele neuronale încearcă să replice funționalitatea creierului uman, imitând atât structura neuronală biologică, cât și funcționalitatea sa.

Componenta de bază al creierului uman o reprezintă neuronul. Acesta are rolul de a primi stimuli electrici (informații), a procesa și interpreta semnalele primite și de a tranzmite mai departe rezulatul procesării către alți neuroni conectați care, la rândul lor, vor face același lucru. Deși capacitatea de procesarea datelor al unui singur neuron pare nesemnificativ, cum creierul uman are în componența sa aproximativ 86 miliarde de astfel de neuroni interconectați, aceasta este capabilă de procesarea unui volum extraordinar de date.



**Figura 6**. Structura și funcțiile unui neuron biologic[[2]](#footnote-2)

Probabil cea mai uluitoare capacitate al unui creier biologic este puterea de a învăța, folosind experiențe anterioare pentru a-și îmbunătății performanțele. Datorită acesteia, ființele umane se dezvoltă de la un bebeluș neajutorat, fără control asupra propriului corp, la un matur capabil să analizeze mediul înconjurător, să interpreteze și să înțeleagă stimulii senzoriali bazat pe evenimente anterioare și, la nevoie, să se adapteze. În lumea programării, o astfel de capacitatea de adaptare era, până de curând, imposibili de atins. În cea mai bună variantă, necesita implementarea unui număr exorbitant de condiții, precum și efectul lor asupra sistemului. Deși au existat de-a lungul anilor diferite metode de implementare de la simple cauze “if” la analiză statistică a unor date, niciuna dintre variante nu a avut adaptabilitatea unui sistem biologic.

Rețelele neuronale artificiale încearcă să replice exact această capacitate al echivalentului lor biologic, reproducând structura, fluxul de informații, precum și capacitatea de a învăța din exemple și experiență, cu rezultate remarcabile în diverse domenii de aplicabilitate cum ar fi clasificare, procesare de imagine, robotică și control, generare de date, previziuni statistice, securitate sau medicină.

### Perceptronul

Așa cum unitatea de bază al unui creier este o singură celulă, neuronul, la fel și unitatea de bază a unei rețele neuronale este o singură structură, perceptronul. Inițial introdus ca o modalitate de implementarea porților logice, capacitățile sale de adaptare s-au dovedit a fi esențiale în rețele neuronale. Rolul acestuia este unul decizional. În funcție de parametri de intrare și o regulă de procesare, perceptornul generează diferite date de ieșire[[3]](#footnote-3).



**Figura 7.** Componentele preceptronului

În figura 7 este prezentat un perceptron împreună cu principalele sale componente, unde:

* **i0 ... in**: valorile numerice de intrare ale perceptronului
* **w0 ... wn**: factorul de importanță alocat fiecărui input
* ∑**:** operația de însumarea valorilor de input, combinate cu factorul lor de importanță
* **f(x):** funcția de interpretarea rezultatului însumării, numită și funcție de activare sau de transfer
* **o:** rezultatul funcției de activare, valoarea rezultată al perceptronului

În funcționalitatea unui perceptron, fiecărei valori de intrare i se atribuie un factor de importanță, care stabilește relevanța unei valori în calculul rezultatului final. Această pereche de valori se însumează și, după caz, i se adaugă o valoare constantă, numită “bias” care permite ajustarea activării perceptronului în funcție de necesități.

Un exemplu simplu de utilizarea unei valori “bias” ar fi cazul în care dorim ca perceptronul să se activeze doar dacă valoarea calculată este mai mare de 2. În acest caz, la rezultatul însumării adaugăm un bias de -2.

Funcția de activare, sau funcția de transfer, are rolul de a mapa rezultatul operației precedente la o mulțime de valori iarăși, în funcție de cerințele sistemului. Printre cele mai des utilizate merită meționate:

* hard-limit: limitează ieșirea unui neuron la două valori 0 și 1
* liniară: folosiți ca aproximatori liniari în cadrul filtrelor liniare
* sigmoidală: transformă argumentul într-un număr cu valoare cuprinsă între 0 și 1
* ReLU (Rectified Linear Unit): limitează rezultatul operației la valori mai mari de 1



**Figura 8**. Exemple de funcții de activare

Dacă aplicăm funcția de activare pentru valoare deja însumată, obținem următoare formulă de prelucrarea informațiilor de către un perceptron:

Combinând mai mulți perceptroni într-o structură origanizată pe nivele, se obțin rețelele neuronale, consistând din trei straturi principale:

* **Stratul de intrare**: numărul perceptronilor de pe acest nivel este egal cu numărul valorilor de intrare individuale primite în același moment.
* **Straturi ascunse**: în funcție de scopul rețelei, numărul acestora poate varia între [1, n], de cele mai multe ori n fiind un număr ales după metode euristice, uneori poate chiar alese la întâmplare, iar numărul perceptronilor din fiecare strat, de asemenea, poate fi ales cu metode similare.
* **Stratul de ieșire**: în cazul rețelelor neuronale de clasificare, numărul perceptronilor de pe acest ultim nivel este egal cu numărul răspunsurilor posibile în problema abordată.

În funcție de numărul straturilor ascunse putem vorbi ori de rețele neuronale clasice, în cazul în care acesta are în componența sa doar 1 – 2 straturi ascunse, sau de rețele profunde, acestea având mai multe straturi.

Depinzând de fluxul informației în interiorul structurii, se diferențiază:

* Rețele neuronale feed forward: unde conexiunile dintre neuroni nu formează cicluri, datele propagându-se numai înainte
* Rețele neuronale recurente: unde conexiunile dintre neuroni formează un graf orientat, bucați de date deja, sau parțial, prelucrate putând influența fluxul sau valoare altor bucăți.



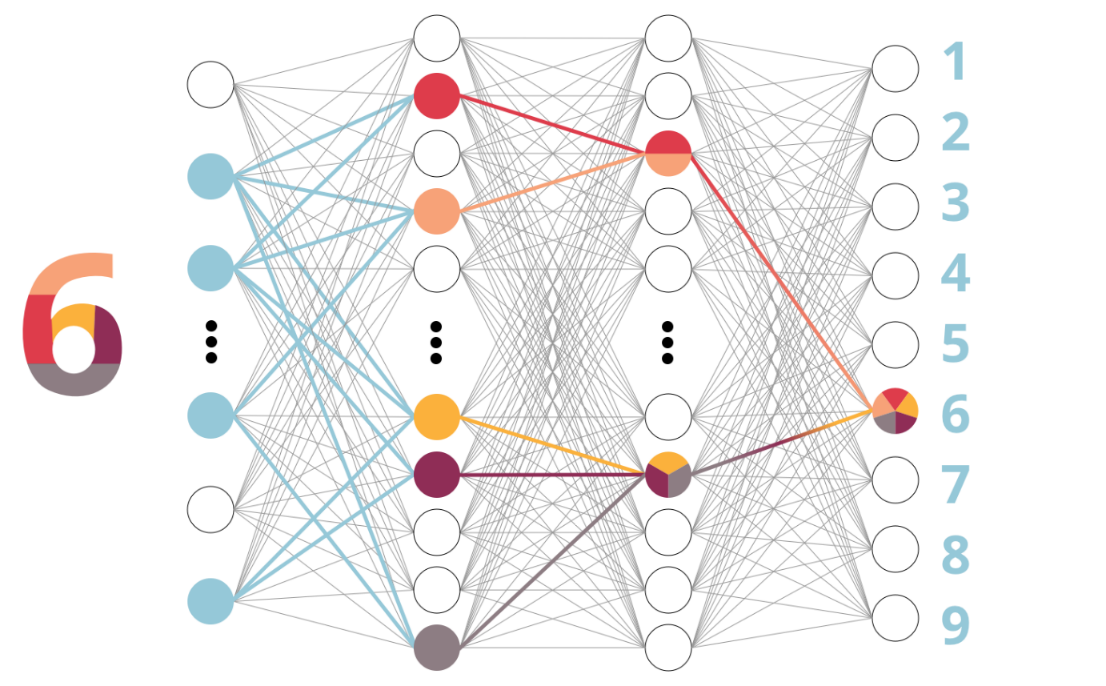
**Figura 9.** Structura funcțională a rețelelor neuronale feed forward

Datorită faptului că doi perceptroni de pe același nivel, primesc aceleași date de intrare, straturile fiind puternic interconectate și cunoscând formula de prelucrarea datelor de intrare al unui singur perceptron, putem extrapola formula de prelucrarea tuturor datelor de pe un strat, în formă matriceală, cât și în forma sa simplificată:

Aplicând această formulă fiecărui start, putem afla rezulatul procesării datelor de intrare. Desigur, fără a optimiza valorile implicate în formulele de calcul ale fiecărui perceptron, acest rezultat va fi unul aleator, fără valoare. Pentru a înțelege felul în care ar trebui ajustate aceste valori, mai întâi trebuie înțeles felul în care ar trebui să funcționeze respectiva rețea.

Teoretic, fiecare perceptron dintr-o rețea neuronală de clasificare are rolul de a identifica tipare specifice din datele de intrare, iar combinând rezultatele fiecărui perceptron, de pe un nivel, ar trebui sa permită identificarea unor tipare mai complexe pe următorul nivel. Această compunere se continuă pe fiecare strat, până la cel de ieșire, moment în care ar trebui să se poată clasifica inputul.

Pentru a putea exemplifica această funcționalitatea teoretică, vom considera exemplul clasic de introducere în arhitectura rețelelor neuronale, cel al identificării imaginii unui număr scris de mână. Scopul acestui exercițiu este a da valorile tuturor pixelilor care compun o imagine ca date de intrare unei rețele neuronale, iar acesta să poată identifica cifra scrisă în imagine.



**Figura 10**. Recunoașterea caracteristicilor într-o rețea neuronală

În figura 10 se poate vedea rolul straturilor unei rețele în identificarea și combinarea carecteristicilor datelor de intrare pentru aflarea rezultatului clasificării. Așa cum este evidențiat, atât numărul de perceptroni în fiecare strat, cât și numărul straturilor poate impacta semnificativ performanțele sistemului astfel:

* Numărul perceptonilor: impactează direct numărul de caracteristici diferite identificabile
* Numărul straturilor: impactează direct complexitatea caracteristicilor diferite identificabile

Pornind de la această schemă de funcționare, ar fi justificabil să concluzionăm că o rețea neuronală cu cât are mai mulți perceptroni și mai multe straturi, cu atât poate fi mai performant. Realitatea, însă, nu este atât de simplă. Depinzând de complexitatea și de varietatea posibilelor date de intrare, această performanță poate varia, însă acest aspect îl vom aborda în Capitolul 4.

### Antrenarea rețelei

Așa cum am menționat la începutul capitolului, caracteristica principală a unei rețele neuronale constă în capacitatea sa de a învăța din experiențe trecute pentru a-și îmbunătății performanțele. În contextul unui claisificator, această învățare se reflectă în ajustarea parametrilor rețelei (weights și bias-uri) astfel încât fiecare perceptron să poată utiliza și interpreta cât mai eficient posibil datele sale de intrare. Automatizarea acestei repetate ajustări de parametri, pentru a putea identifica orice set de date de intrare definește antrenarea rețelei. Pentru a fi cât mai eficient, această antrenare trebuie făcută pe un volum cât mai mare, dar omogen, de date și se împarte în următorii pași:

1. Inițializarea parametrilor cu valori aleatoare: se asignează câte o valoare fiecărui factor de importanță (și în unele cazuri fiecărui bias), de pe toate conexiunile
2. Parcurgerea rețelei: se introduce un set de date în sistem și se reține rezultatul procesării
3. Analiza rezultatului: Se compară rezultatul primit cu cel dorit, se calculează diferența și se analizează felul în care această diferență s-ar putea reduce
4. Ajustarea parametrilor: odată cunoscută metoda de reducerea diferenței anterioare, această informație se propagă înapoi prin straturile rețelei, modificând proporțional fiecare parametru
5. Se repetă de la punctul 2 până ajungem la un rezultat satisfăcător.

Analiza rezultatului unui pas de antrenare se reduce la compararea rezultatului calculat, cu rezultatul dorit și calcularea diferenței sale. Acesta se numește **funcția de cost**. În contextul rezultatelor matriceale folosite, funcția cost reprezintă suma diferențelor rezultatelor fiecărui perceptron de pe stratul de ieșire.



Ecuația de mai sus prezintă rezultatul funcției cost a unei rețele neantrenate, în care valoarea de intrare era cifra 3, iar rezultatul calculului a fost cifra 8. O rețea bine antrenată va avea funcția cost cât mai aprope de 0. Dacă interpretăm întreaga rețea neuronală ca o funcție foarte complexă, atunci antrenarea sa se poate vedea procesul de găsirea valorii minime a funcției respective, într-un spațiu n-dimensional, n fiind numărul de perceptroni din stratul de intrare.

Pentru găsirea acestei minime, prima dată trebuie să calculăm **gradientul funcției cost**. Concret, este necesar să aflăm în ce proporție ar trebuie să difere factorul de importanță a fiecărei valori transmise de la stratul precedent fiecărui perceptron, astfel încât sa obținem rezultatul dorit. Gradientul unei funcții cost reprezintă proporția cu care variația fiecărei valori de intrare afectează variația valorii de ieșire și se poate reprezenta matematic sub forma:

Utilizând valoarea negativă al gradientului de cost, putem afla exact în ce sens trebuie să modificăm valorile parametrilor rețelei neuronale, cât și impactul fiecărei modificări asupra stării curente, astfel încât să minimizăm funcția cost și să ne apropiem de rezultatul dorit.

În exemplul de mai sus, având vectorul tuturor factorilor de importanță, negativul rezultatului gradientului funției cost indică valoarea și sensul în care fiecare valoare de W trebuie schimbat astfel încât să ajungem la rezultatul dorit. Valorile care trebuie să crească sunt marcate cu albastru, iar valorile care trebuie scăzute sunt cu roșu. Cu cât o valoare este mai mare, cu atât o schimbare a sa va impacta rezultatul final al operației.

În esență, calculul gradientului funcției cost o să ne indice locul curent într-un spațiu n-dimensional și direcțiile de urmat către minima spațiului respectiv.

### Backpropagation

În explicarea felului în care se calculează gradientul funcției cost s-a menționat faptul că toți parametrii rețelei trebuie calculați și ajustați însă, în realitate, acest lucru nu se poate realiza direct tocmai din cauza structurii de rețea, în care schimbarea unei valori într-un nivel, afectează direct toți parametrii de pe nivelul următor. Din acest motiv, parcurgerea și adaptarea valorilor trebuie să se facă prin propagarea efectelor dinspre stratul de ieșire, până la stratul de intrare. Acest proces se numește backpropagation.

Pentru a facilita vizualizarea acestei propagări de valori, vom consideră o rețea constând în 4 straturi, cu câate un singur neuron pe fiecare nivel. Formula obținută prin analiza acestei structuri se va putea extrapola în cele ce urmează pentru rețele mai complicate. Notațiie folosite în exemplu vor fi următoarele:

* as valoare neuronului aflat pe stratul s.
* ws factorul de importanță a valorii as-1 în calculul valorii as.
* bs biasul conexiunii între as-1 și as.
* C valoare funcției cost
* f funcția de activare
* ∑s valoare funcției de însumare al perceptronului pe stratul s
* y rezultatul dorit

Urmărind ordinea de înlănțuire a operaților, efectul variației valorii ws asupra lui C se poate reprezenta astfel:

Cunoscând această formulă, putem înlănțui propagarea valorilor de la rezultatul calculat, până la stratul de intrare, ajustând fiecare parametru pe parcurs. În același fel putem utiliza această formulă și pentru rețele cu mai mulți neuroni pe fiecare strat, trebuie ținut cont doar de faptul că valoare de ieșire al unui perceptron afectează valoarea tuturor de pe următorul strat, astfel, trebuie însumate efectele:

Odată cunoscută, această formulă se poate aplica pe orice valoare de intrare și folosită pentru a verifica și a corecta funcționalitatea rețelei neuronale. Repetând această verificare cu un număr mare de date de intrare este, în esență, procesul de antrenare a sistemului.

## Procesarea limbajului natural

Interacțiunea dintre om și echipamente digitale este considerat ceva absolut natural în zilele noastre, tot mai mulți oameni utilizând telefoane mobile, internetul și servicii oferite de acestea într-un mod intuitiv. Acest lucru acum 50 de ani era încă de domeniul ficțiunii, din cauza cunoștiințelor tehnice specifice necesare a fi dobândite pentru a putea interacționa cu un calculator. Odată cu digitalizarea utilajelor de calcul, s-a realizat tranziția de la acționarea mecanică a acestora la implementarea unor șiruri de comenzi de executat. Primele astfel de comenzi au fost elaborate pe cartoane performate într-o ordine interpretabilă și executabilă de calculator, urmat la puțin timp de apariția limbajelor de programare rudimentare și specifice fiecărui aparat. Aceste metode au continuat să evolueze, devenind tot mai intuitive și totodată facilitând accesul unui număr din ce în ce mai mare de utilizatori. Următorul pas logic în această etapă de evoluție este interacțiunea dintre om și calculator, folosind limbajul uman.

Chair dacă studiul aceastei teme și-a găsit începuturile deja în ani 1950, dezvoltarea sa a fost unul constant și relativ lent, din cauza limitărilor tehnologice. La fel ca și alte subdomenii ale Inteligenței Artificiale, procesarea limbajului natural a cunoscut un avânt în dezvoltarea sa în ultimii 20 de ani, proporțional cu avansarea vitezelor de procesare și a capacității de stocarea datelor. Dezvoltarea rețelelor neuronale a fost un factor important în dezvoltarea acestei arii de cercetarii, din cauza capacității acestora de interpretare, clasificare și chiar generare a datelor.

Datorită complexității limbajului uman, interpretarea sa din punct de vedere structural, sintactic și semantic poate varia atât de mult, încât ar fi practic imposibil de implementat un program care să acopere toate variațiile posibile, deși această formă de implementare era specifică primelor tentative de prelucrarea limbajului natural, incluzând arborii decizionali.

Cercetările lui Noam Chomsky, considerat părintele lingvisticii moderne, au favorizat abordarea limbajului natural într-o manieră revoluțională, marcând un punct de cotitură în evoluția științei. Acesta a psotulat ideea de structurarea limbajului pe nivele, începând de la elemente generice prezente în orice limbaj, la construcții specifice, toate interconectabile prin reguli definite de formare și interpretare. Utilizând aceste baze de cunoștiințe și reguli, ș-au dezvoltat modele statistice de procesarea limbajului, variațiile cărora sunt utilizate și în zilele noastre într-o varietate de domenii, de la analize economice, până la dezvoltarea compilatoarelor.

În contextul rețelelor neuronale, analiza limbajului natural a întâmpinat dificultăți din cauza structurii riguroase în care astfel de implementări procesează datele, datele de intrare necesitând a avea o structură bine definită șu utilizabilă de către calculator, lucru incompatibil cu versatiliatea limbajului omenesc. Tocmai din acest motiv, s-au dezvoltat metode de procesare și transformare a datelor într-o formă utilizabilă.

### Codarea „one hot”

În cazul utilizării datelor cu un număr limitat și cunoscut de posibile variații, cea mai utilizată metodă de transformarea acestora în formă utilizabilă de rețele neuronale o reprezintă codarea one hot. Aceasta implică transformarea datelor de intrare într-o formă binară de tip ordinal.

Pentru înțelegerea proceurii de codare vom considera un exemplu de rețea neuronală care însumează două cifre.



**Figura 11**. Rețea neuronală pentru însumarea a două cifre

Chiar dacă primul insinct ar fi să creăm nivelul de intrarea din 2 perceptroni, unul pentru fiecare cifră în parte, felul în care rețelele neuronale procesează datele ar produce rezultate incorecte. Din acest motiv, datele de intrare vor fi transformate într-o listă de 10 elemente populate cu valoarea 0, mai puțin elementul de pe poziția valorii cifrei de procesat.

Numărul total de cifre: 10

Cifrele se vor transforma astfel în:

0: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

1: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

....

8: [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

9: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

Datele de intrarea astfel transormate, permit crearea unei rețele neuronale cu 20 de perceptroni în stratul de intrare, reprezentând concatenarea a celor două cifre care se doresc a fi însumate. Rezultatul operației va fi dat în aceeași formă codată, iar decodarea sa folosind inversul procedurii one hot, obținem cifra rezultat în formă decimală.



**Figura 12.** Codare și decodare "one hot"

Această formă de codare funcționează optim în cazul utilizării unor date cu formă fixă și bine cunoscută. Deoarece vocabularul unei limbi nu respectă aceste constrângeri, este necesară fie utilizarea unei forme alternative de codare compatibil cu rețelele neuronale, fie prelucrarea prealabilă a datelor utilizate pentru a satisface necesitățile codării one hot.

### Identificarea formei de bază și a rădăcinii cuvintelor

O caracteristică importantă a limbajului natural o reprezintă îmbogătățirea vocabularului prin derivare. Aceasta se defineşte ca procedeul de formare a unui cuvânt nou având ca bază un alt cuvânt existent în limbă (Marouzeau, 1933: 63 ş.u.). Acesta are două subtipuri: derivarea progresivă sau propriu-zisă şi derivarea regresivă. Cea dintâi constă din adăugarea unor afixe (sufixe sau prefixe) la un cuvânt bază. În funcţie de poziţia din cuvântul bază în care se ataşează afixul, vorbim despre prefixare sau sufixare. Derivarea regresivă constă în suprimarea unor afixe de la un cuvânt bază[[4]](#footnote-4).

Pentru a putea utiliza cuvinte derivate în contextul codării one hot, este necesar ca acestea să fie reduse în prealabil la cuvintele sale de bază. Pentru aceasta există două tehnici frecvent utilizate: lematizare și identificarea rădăcinii cuvintelor.

Lematizarea reprezintă identificarea formei de bază a cuvintelor, prin eliminarea tuturor afixelor (atât prefixe cât și sufixe) pentru a găsi forma de dicționar al unui cuvânt lematizat. Folosind acest proces, din cuvântul *sfințeniei* obținem cuvântul de bază, *sfințenie.* Acest proces reduce cuvântul procesat la forma sa de bază, fără a schimba cuvântul rădăcină.

Identificarea rădăcinii cuvintelor (eng. *Stemming*), pe lângă reducerea tuturor afixelor, încearcă să identifice rădăcina cuvântului prelucrat, obținându-se, astfel, chiar și alte cuvinte. Folosind același cuvânt ca în exemplul precedent, *sfințeniei*, identificând rădăcina cuvântului obținem cuvântul *sfânt*.

Aplicând ambele metode de procesare, obținem un lexic compus doar din cuvinte cu formă fixă, deci utilizabile în procesul codării one hot. Acesta trebuie efectuat pentru toate propozițiile menite a fi utilizate ca date de intrare pentru rețele neuronale.

**Propoziția originală**: Mașinuțele copilului au culori diferite.

**Propoziția procesată:** Mașina copil avea culoare diferit.

Din exemplul precedent este ușor de observat că dacă dorim a reține diferențe subtile de context și nuanță, cum ar fi faptul că este vorba de mai multe mașini, rezultatul s-ar putea să nu fie optim scopului și ar fi indicată utilizarea unei alte forme de procesare.

# Proiectarea conceptuală

Acest capitol tratează detaliile procesului elaborării sistemului la nivel de concept, pornind de la ideea de bază, planificarea și proiectarea logică. Se vor elabora cerintțele fundamentale care au fost urmărite și se vor analiza pașii realizați pentru îndeplinirea acestora. Totodată, vom evidenția cursul logic a datelor precum și prelucrarea acestora de la propoziții simple folosite ca date de intrare, până la efectul executării comenzilor interpretate din acestea. Structura proiectului este relativ simplă însă, realizată în așa fel încât să faciliteze orice schimbare viitoare precum și posibile extinderi ale logicii de implementare și utilizare.

## Cerințele sistemului

Cerința principală care a stat la baza realizării proiectului A.L.I.C.E. (Automation of Locally Integrated Control Equipment) a fost dorința de a avea un sistem capabil să controleze echipamentele integilente dintr-o încăpere, utilizând limbajul natural pentru interacționare. Acesta trebuia să satisfacă următoarele cerințe:

* Disponibilitate: Mediul pe care rulează trebuie să fie activ în permanență, pentru a putea interacționa cu echipamentele inteligente conectate.
* Eficiență: costurile de utilizare și mentenanță trebuie să fie destul de mici încât sistemul să fie accesibil unui număr cât mai mare de utilizatori potențiali
* Utilizabilitate: interacțiunea cu sistemul trebuie să fie cât mai simplă și intuitivă, fără a necesita cunoștiințe de programare sau de funcționare pentru utilizarea cotidiană. Totodată, acesta trebuie să se poată iniția și configura în mod autonom, în cazul unei reporniri (ex. Întrerupere de curent)
* Precizie: fiind un sistem cu o utilitate bine definită, acesta trebuie să cunoască un număr limitat dar precis, de comenzi și să poată interpreta și variațiile lor. În cazul în care întâmpină o comandă necunoscută, sau destul de ambiguă încât să nu poată interpreta, acesta trebuie să informeze utilizatorul de nevoia clarificării comenzii.
* Monitorizare: în afară de executare de comenzi, sistemul trebuie să poată monitoriza și înregistra condițiile de mediu, și, în unele cazuri, să poată reacționa la schimbări fără intervenție externă (ex. intervenția utilizatorului)

## Proiectarea logică

Principala formă de interacțiune cu A.L.I.C.E. va fi prin limbajul natural sau, mai precis, propoziții de comandă ce se vor furniza ca date de intrare. Acestea vor fi prelucrate în așa fel încât să poată fi interpretat de către sistem și vor servi drept date de intrare pentru modulul decizional al sistemului, o rețea neuronală pentru clasificare, care va decide asupra tipului de comandă furnizată. Rezultatul clasificării va fi transmis modului central unde, în funcție de clasa de comandă, se va executa rutina adecvată și se va transmite un răspuns adecvat utilizatorului.

Comenzile posibile vor fi pentru aprinderea și stingerea unui bec, precum și modificarea temperaturii setate al unui termostat. Echipamentele menționate vor fi interconectate în aceeași rețea cu unitatea de comandă prin care sistemul nostru va putea interacționa cu ele. Deși nu oferă vreun avantaj funcțional, se va trata și cazul în care utilizatorul salută sistemul, doar pentru îmbunătățirea calității interacțiunii. În transformarea comenzilor într-un format utilizabil de către rețeaua neuronală pentru clasificare se va ține cont de structura acestuia, în special de tipul de date așteptat și de mărimile datelor de intrare.

Interfața cu utilizator va oferi posibilitatea introducerii comenzilor de intrare și a vizualizării răspunsurilor primite în limbaj natural. Pe lângă acestea, va putea furniza informații despre starea curentă a sistemului precum și o modalitate de vizualizarea unui istoric de evenimente raportate de către echipamentele inteligente.

Înainte de a putea fi utilizat, rețeaua neuronală va trebui antrenată adecvat, folosind o suită de date de antrenare și de verificare special realizate pentru acest sistem. Tototadă, implementarea va trebui să permită și reantrenarea rețelei, indepent de restul sistemului, precum și posibilitatea schimbării datelor cu care lucrează.

Controlul echipamentelor inteligente se va realiza folosind protocolul Z-Wave. În acest scop, sa va implementa un modul dedicat acestei interacțiuni, care va putea genera pachetele specifice de comandă și va fi capabil să monitorizeze schimbările raportate de către nodurile membre ale rețelei.

Nu în ultimul rând, se vor considera metode de tratarea cazurilor de închidere comandată specifice sistemului, pentru a nu periclita integritatea componentelor și a echipamentelor utilizate.



**Figura 13.** Diagrama fluxului de date

## Arhitectura componentelor hardware



**Figura 14.** Componentele hardware utilizate

Figura 14 prezintă componentele hardware utilizate în realizarea acestui proiect. Conform numerotării prezentate acestea sunt:

1. Terminal de I/O pentru acces la interfața aplicației. Acesta poate fi orice echipament capabil să interpreteze și să afișeze pagini web cu conținut javascript.
2. Raspberry Pi 3 model B. Unitatea principală de rulare al proiectului bazat pe sistemul de operare Raspbian care oferă posibilitatea integrării tuturor tehnologiilor componente sistemului.
3. Stick USB controller **Aeotec Z-Stick Gen5**. Acesta permite configurarea și interacțiunea cu rețeaua Z-Wave, precum și interacțiunea directă cu toate nodurile membre rețelei. Interfațarea se realizează facil, prin comunicare serială directă.
4. Echipamente inteligente compatibile Z-Wave. În cazul de față vom utiliza:
   * Bec inteligent **Zipato Bulb 2**, cu posibilitatea controlării intensității luminii
   * Valvă inteligentă **Popp TRV**, cu termostat integrat. Acesta permite monitorizarea temperaturii ambientale și setării a unor praguri de acționare.

## Arhitectura componentelor software

Din punct de vedere funcțional, proiectul se împarte în 2 module principale:

* Front End: realizat în **Node-RED**
* Back End: implementat în **Python**, folosind module dedicate

Interacțiune dintre utilizator și aplicație se va realiza prin interfața web generată de Node-RED. Principalele motiev pentru alegerea acesteia au fost ușurința de implementare a unor interfețe compatibile cu orice tip de echipament capabil să interpreteze HTML și Node.js (ex.PC, telefon mobil, etc. ) și ușurința integrării acesttuia cu protocoale specifice utilizate în industria IoT.

Comunicarea dintre cele două module principale se va realiza prin protocolul MQTT, folosind brokerul dedicat **mosquitto-mqtt**. Atât Front End-ul cât și Back End-ul vor expune API-uri specifice acestui protocol, pentru interschimbare de date.

Ca date persistente se va stoca o suită de exemple de posibile comenzi în utilizarea normală. Acestea vor fi folosite atât de rețeaua neuronală pentru clasificare, în procesul de antrenare, cât și de procesorul de text, pentru codarea datelor de intrare.

Tot persistent se vor stoca și caracteristicile rețelei neuronale de după antrenare. Acestea se vor reîncărca la fiecare repornire de sistem, eliminând astfel nevoia de reantrenarea rețelei la fiecare rulare.

Intercațiunea cu controllerul de Z-Wave USB se va realiza prin modulul dedicat acestui scop, acesta folosindu-se de librăria **python-openzwave**, cu ajutorul căruia se pot interoga si configura toate elementele rețelei Z-Wave, în limita capabilităților fiecărui nod.

Modulul de Back End se va reliza integral în limbajul de programare Python, deoarece oferă suport nativ pentru o multitudine de module utilizate. Printre aceste merită meționate:

* **paho-mqtt**: librăria dedicată comunicării prin protocolul MQTT
* **numpy**: librăria standard utilizată pentru operații cu matrici multidimensionali precum și o colecția amplă de funcții matematice
* **tensorflow**: librăria de manipulare grafică de date, folosind adesea în aplicații de inteligență artificială, în special pentru definirea structurilor de rețele neuronale
* **json**: librărie folosită pentru convertirea datelor în format JSON, atât la transmisia datelro prin MQTT, cât și serializarea și deserializarea datelor relevante prelucrării limbajului natural
* **nltk**: acronim provenit din Natural Language Toolkit; oferă o gamă largă de operații utile în prelucrarea limbajului natural.
* **pickle:** librărie specializată pe serializare și deserializare de date și stocarea persistentă a acestora.



**Figura 15.** Componentele software utilizate

## Baza informațională

Utilizarea aplicației necesită doar două surse de date: date de antrenare și date de intrare. Cele două sunt dependente atât structural cât și semantic deoarece interpretarea datelor de intrare se bazează pe datele cu care sistemul a fost antrenat în prealabil.

### Datele de antrenare

Deoarece atât codarea datelor de intrare, cât și clasificarea comenzilor se bazează pe limbaj natural, o singură sursă de date de antrenare se poate folosi pentru a reconfigura întregul proiect, în orice moment. Toate datele persistente se generează pe baza acestor surse de antrenare. Singura sursă de date persistente de care este nevoie înaintea rulării este fișierul **/persistent/trainingCorpus.txt**. Acesta conține toate datele de antrenare cu care va lucra atât modulul de codare a datelor, cât și rețeaua neuronală în timpul antrenării sale.

Formatul datelor este de tipul unei liste de perechi, în care totalitatea elementelor listei reprezintă totalitatea datelor de antrenare, iar fiecare pereche este de forma (categorie, propoziție), în care categoria reprezintă categoria de comandă, iar propoziția reprezintă însăși comanda. Deși inversarea perechii ar fi de bun simț, posibilitatea grupării elementelor pe categorii a stat la baza alegerii ordinii.

Exemplu de conținut fișierului datelor de antrenare:

greeting : greetings

greeting : hello there

greeting : hi

greeting : hey there alice

........

lights on : turn on the lights

lights on : make it brighter

lights on : lights on

.......

lights off : please turn off the lights

lights off : lights off please

......

Prelucrarea datelor de antrenare se efectuează în două module distincte. Primul modul, de codare, încarcă toate datele, le separă funcțional, extrage informațiile necesare și salvează persistent urmaătoarele fișiere:

* **refinedTrainingCorpus.txt**: conține datele identice din fișierul sursă, dar prelucrate prin eliminarea cuvintelor nefolositoare și transformarea restul cuvintelor în rădăcinile sale
* **lexicon.pickle**: conține totalitate cuvintelor individuale
* **vocabulary\_data**.**pickle**: conține toate structurile de date cu care lucrează sistemul, incluzând categoriile de comenzi ordonate, perechile de (categorie, comenzi) codate și lexiconul.

### Datele de intrare

Așa cum s-a menționat la începutul capitolului, interpretarea datelor de intrare se bazează pe cele cu care s-a făcut antrenarea prealabilă al sistemului. Din acest motiv, nu orice tip de comandă inițiată de utilizator poate fi corect interpretată. Atâta timp cât se respectă cadrul operațional, aplicația ar trebui să fie capabilă să interpreteze și să execute comenzile furnizate însă și acest lucru depinde de gradul de abstractizare semantică.

Atâta timp cât se încadrează în una din categoriile de comenzi cu care a fost antrenat sistemul, orice propoziție furnizată de către utilizator va putea fi interpretată cu un oarecare grad de prezicie. Dacă respectiva precizie este sub o valoare setată, aplicația va întoarce un mesaj informând utilizatorul de imposibilitatea interpretării corecte.

Lungimea și structura datelor de intrare nu sunt relevante în contextul interpretării deoarece, din însăși natura codării și clasificării inputurilor, orice cuvânt necunoscut va fi eliminat, cât și cuvintele cunoscute a nu fi purtătoare de informație relevantă, detaliile cărora vor fi elaborate în subcapitolul4.2. Prelucrarea comenzilor. Singura constrângere importantă este ca propozițiile furnizate de către utilizator să conțină suficiente cuvinte utilizate în timpul antrenării, pentru a le putea clasifica îndeajuns de corect.

Considerând propoziția *“Turn on the lights”*, aceasta conține suficiente cuvinte folosite în timpul antrenării pentru a putea fi interpretată cât și corect clasificată. Cuvintele *“turn”, “on”* și *“light”* vor fi suficiente pentru ca aplicația să le clasifice ca făcând parte din clasa de comandă “lights on” și, în consecință, va lansa comanda de aprinderea becului. Pe de altă parte, propoziția *“I would like to eat something”* conține un singur cuvânt întâmpinat în timpul antrenării, mai exact cuvântul *“I”*, iar acesta apare în propoziții făcând parte din 3 categorii separate de comenzi: “lights on”, “heat up” și “heat down”. În funcție de eficiența de antrenare făcută, deși aplicația va recunoaște respectivul cuvânt, clasificarea acestuia va rezulta într-un indice de certitudine sub limita permisă și va informa utilizator că nu înțelege comanda.

# Proiectarea Tehnică

Procesul de implementarea a ținut cont de separarea principială dintre Front End și Back End, cât și de interacțiunea dintre cele două. Ca responsabilități, subsistemul de interfață este responsabilă pentru prezentarea informațiilor de stare, cât și de preluarea comenzilor și transmiterii acestora. În contrast, subsistemul de Back End, este responsabil pentru prelucrarea și clasificarea comenzilor, cât și executarea acestora.



**Figura 17.** Diagrama Use-Case

Figura 17prezintă scenariile de utilizare ale aplicației, cât și interacțiunile dintre modulele logice. Fundamental, utilizatorul are două posibilități de interacționare cu sistemul, fie introduce o comandă în limbaj natural, fie alterează direct una dintre valorile posibile. Aceste comenzi sunt transmise print protocolul MQTT (fiind ghidate prin broker) la subsistemul de Back End. În cazul comenzilor furnizate în limbaj natural, acesta utilizează o rețea neuronală de clasificare pentru identificarea clasei de comandă ce se dorește a fi executat, în funcție de rezultatul clasificării poate extrage date adiționale (valori procentuale sau absolute) din comandă, urmând ca acetstea să fie executate de submodulul specific operațiilor pe rețeaua de control Z-Wave.

Pentru a facilita ușurința utilizării, întreaga aplicație a fost elaborată astfel încât să își poată genera toate datele persistente necesare, în cazul lipsei lor. Submodulul de codare poate regenera întreaga suită de date necesare pentru utilizare, pornind doar de la lista de date de antrenare, iar submodulul rețelei neuronale încearcă să își încarce modelul de tensorflow preantrenat. În cazul în care acesta nu se găsește, se reexecută întregul proces de antrenarea și persistarea modelului, după care această nouă instanță se va utiliza.



**Figura 18.** Diagrama de secvență a inițializării sistemului

Pentru a elimina potențiale efecte secundare cauzate de închiderea bruscă al sistemului, modulul principal implementează un o rutină de închidere controlată și secvențială a componentelor, inițiată în momentul întâmpinării semnalului de INTERRUPT, lansat în momentul închiderii aplicației folosind combinația de taste CTR-C.

Întregul sistem fiind bazat pe evenimente, în timpul secvenței de inițializare se configurează toate metodele de callback dintre module. Atât MqttController cât și ZwaveController se bazează pe metode setate în prealabil, în procesul prelucrării evenimentelor. Tocmai din acest motiv, toate comenzile se tratează uniform, indiferent de natura lor, metodele de prelucrare fiind abstracte la momentul apelurilor acestora,



**Figura 19**. Diagrama de secvență a prelucrării comenzilor

În figura 19se observă această abstractizare a metodelor de prelucrarea comenzilor în momentul trimiterii acestora la modulul de ZwaveController, pentru a fi setate la nivel de rețea.



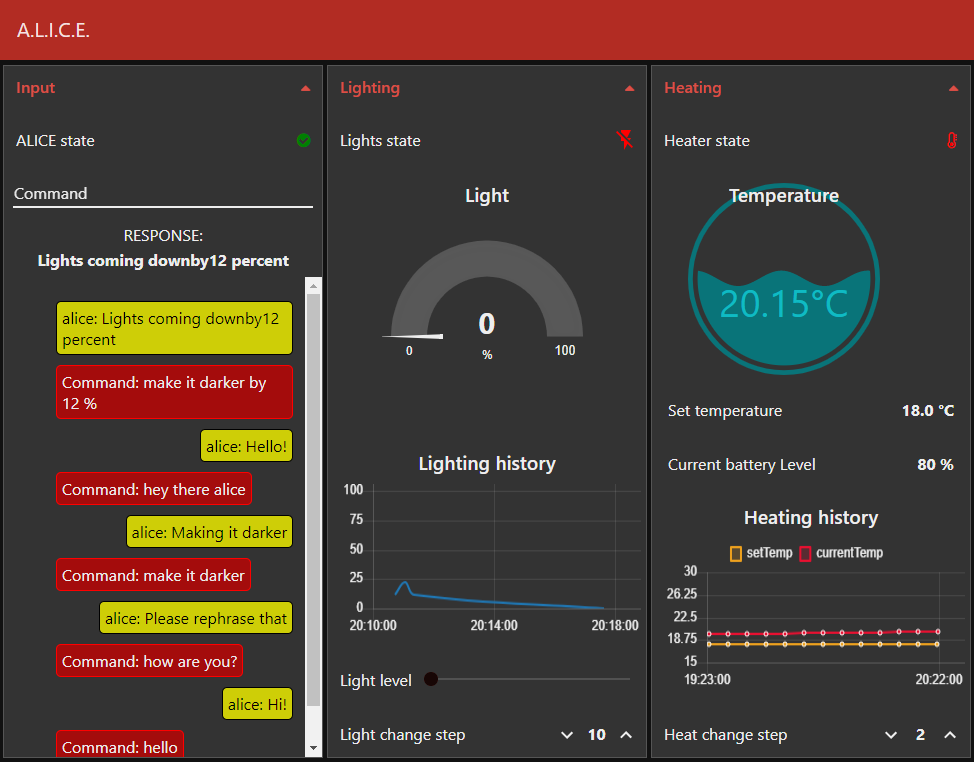
**Figura 20**. Diagrama de clase

Datorită limitărilor de spațiu, elementele componente ale claselor se vor elabora în Anexa 3. Structura internă ale claselor componente din Back End.

## Interfața

Pornind de la scopul principal al proiectului, interacțiune prin limbaj natural, putem deriva principalele cerințe pentru elaborarea interfeței. Totodată, implementarea în Node-RED implică nevoia realizării unei căi de comunicare între Front End și Back End, precum și al unui semnal de confirmarea existenței a acestei conexiuni, util în cazul în care modulul python se închide brusc din orice motiv. Astfel, în procesul elaborării interfeței, s-au ținut cont the următoarele cerințte:

1. Posibilitatea funrizării comenzilor și a primirii răspunsurilor în limbaj natural.
2. Vizualizarea realizării comenzilor.
3. Realizarea comunicării bidirecționale între Front End și Back End folosind protocolul MQTT.
4. Semnalizarea întreruperii comunicării.



**Figura 22.** Interfața aplicației

Implementarea fluxului elementelor de interfață se regăsește integral în Anexa 2.

Interfața se împarte în trei module separate, fiecare înglobând informațiile asociate. Astfel avem panoul *Input*, folosit pentru interacțiunea cu ALICE, panoul *Lighting*, unde se regăsesc toate componentele legate de iluminare și panoul *Heating,* care conține informațiile de temperatură.

### Panoul Input



**Figura 23.** Elemenetele panoului "Input"

Primul panou a interfeței grafice conține date despre interacțiunea utilizatorului cu aplicația. Acesta conține 4 elemente grafice distincte, după cum urmează:

**[Pt. 1]** Pictograma prezintă starea conexiunii dintre Front End (Node-RED) și Back End (Python). În cazul unei conexiuni active, pictograma este una verde, iar în cazul unei întreruperi, aceasta devine roșie. Starea conexiunii este stabilită prin primirea periodică al unei notificări din partea Back Endului, la intervale de 4 secunde. Cât timp aceste semnale sunt primite, pictograma se menține verde, iar în cazul lipsei semnalului pe o perioadă de minim 5 secunde, pictograma devine roșie.

**[Pt. 2]** Câmpul de input este folosit la furnizarea comenzilor în limbaj natural, pentru a fi prelucrate de aplicație. După apăsarea tastei “*Enter*”*,* câmpul este golit, folosind o rutină dedicată de Node-RED.

**[Pt. 3]** Câmpul de răspuns reține răspunsul aplicației la ultima comandă furnizată.

**[Pt. 4]** Repezintă zona de istoric al interacțiunilor. Aceasta este o implementare hibridă între Node-RED și AngularJS, de forma unui câmp de chat de forma unei cozi de lungime 10, în care comenzile transmise de utilizator sunt afișate din stânga spre dreapta, având culoarea roșie de fundal, iar răspunsurile primite de la aplicație sunt afișsate din dreapta spre stânga, având culoare galbenă de fundal.

### Panoul Lighting



**Figura 24.** Elementele panoului "Lighting"

Al doilea panou a interfeței grafice conține date despre starea curentă, cât și trecută, al becului, cât și posibilități de configurare manuală.

**[Pt. 1]** Pictograma prezintă starea curentă al becului. Starea de *aprins*este ilustrată prin fulgerul galben, iar în cazul în care becul se stinge, acesta se transformă într-un fulger roșu barat.

**[Pt. 2]** Indicator al intensității luminii, în procente.

**[Pt. 3]** Grafic prezentând variațiile intensității luminii în ultima oră.

**[Pt. 4]** Potențiometru ce permite setarea manuală a intensității luminii. Acesta suprascrie datele setate prin comandă în limbaj natural și este, la rândul său, suprascris de o nouă comandă de schimbarea intensității.

**[Pt. 5]** Câmp de configurare ce permite schimbarea indicelui de variație a intensității luminoase. În cazul în care în comanda lansată nu se specifică variația procentuală de intensitate, acesta va crește sau, respectiv, descrește, cu procentajul arătat de acest câmp de configurare. În cazul în care acesta eset setat la valoare 10, iar aplicație primește comanda *“*make it brighter*”*, intensitatea luminii va crește cu 10 procente. Dacă însă comanda specifică exact procentajul variației, cum ar fi comanda *“make it brighter by 20 percent”* intensitatea se va mări cu 20 de procente, ignorând valoarea setată.

### Panoul Heating



**Figura 25.** Elementele panoului "Heating"

Al treilea panou a interfeței grafice conține date despre starea curentă, cât și trecută, a temperaturii și al sistemului de încălzire, cât și posibilități de configurare manuală.

**[Pt. 1]** Pictograma prezintă starea curentă al termostatului. Starea de *deschis*este ilustrată prin termometrul galben, indicând faptul că valva de apă al termostatului permite trecerea apei, încăperea fiind astfel în starea de încălzire, iar în cazul în care valva termostatului se închide, acesta se transformă într-un termometru roșu, indicând oprirea încălzirii.

**[Pt. 2]** Indicator a temperaturii ambientale curente.

**[Pt. 3]** Temperatura prag setată. Dacă temperatura ambientală scade sub acest prag, valva termostatului se deschide, înițiind încălzirea încăperii.

**[Pt. 4]** Indicatorul nivelului de încărcare a bateriei din termostat. În cazul în care echipamentul s-ar înlocui cu unul aliment de la sursă permanente de curent, acest indicator ar fi gol.

**[Pt. 5]** Grafic prezentând variațiile temperaturii setate și a celei ambientale, în ultima oră.

**[Pt. 6]** Câmp de configurare ce permite schimbarea indicelui de variație a temperaturii setate. În cazul în care în comanda lansată nu se specifică variația temperaturii, acesta va crește sau, respectiv, descrește, cu numărul de grade Celsius arătat de acest câmp. În cazul în care acesta este setat la valoare 10, iar aplicația primește comanda *“*make it warmer*”*, temperatura setată va crește cu 10 grade. Dacă însă comanda specifică exact valoarea variației, cum ar fi comanda *“make it warmer by 5 degrees”* valoare temperaturii setate va crește cu 5 grade, ignorând valoarea setată.

## Prelucrarea comenzilor

Structura statică a rețelelor neuronale a fost principalul factor decisiv în alegerea metodologiei de prelucrare a comenzilor în limbaj natural. Datorită faptului că o astfel de rețea neuronală are în componența stratului de input un număr fix de neuroni, la fel și numărul datelor de intrare furnizate la un moment dat trebuie să fie fix. Această limitare contravine structurii de lungime variabilă a propozițiilor folosite în limbajul natural. Codarea *one hot* permite transformarea datelor de intrare de lungime variabilă, într-o structură de date de mărime fixă, permițând astfel utilizarea propozițiilor ca date de intrarea pentru o rețea feed forward.

Pentru a putea utiliza codarea one hot, în primă instanță este necesară determinarea lungimii rezultatului codării, care este, în fapt, numărul total de cuvinte diferite utilizabile pentru formarea propozițiilor de comandă. Deoarece acest număr este practic incuantificabil, alegerea logică a fost utilizarea doar a acelor cuvinte care se regăsesc în lista de date folosite pentru antrenarea rețelei. Astfel, a apărut noțiunea de *lexic*.

În procesul generării lexicului aplicației, se iterează prin toate elementele listei de antrenare și se izolează propoziția exemplu de categoria. Propoziția se imparte în cuvinte individuale, iar cele care încă nu se regăsesc în componența lexicului, se adaugă la acesta.

Paralel cu prelucrarea propozițiilor, categoriile individuale întâmpinate se salvează într-un vector separat.



**Figura 26**. Pașii generării lexicului

Utilizând cei doi vectori creați, lexic și categorii, toate datele de antrenare se pot coda folosind tehnica *one hot*, obținându-se astfel date interpretabile de rețeaua neuronală.



**Figura 27**. Codarea datelor de antrenare

Odată generate, lexicul și lista categoriilor se vor utiliza pentru codarea tuturor comenzilor de-a lungul rulării, urmând identic pașii de la codarea datelor de antrenare.

## Implementarea modulului client MQTT

În realizarea modulului de interacțiune cu subsistemul de Front End, s-a utilizat librăria python Eclipse Paho MQTT care permite crearea unui client de comunicare prin protocolul MQTT. Acesta poate publica mesaje, se poate subscrie la topicuri și poate intercepta mesajele trimise pe acestea. Totodată, oferă metode ajutătoare pentru a facilita implementarea cât mai simplistă.[[5]](#footnote-5)

În momentul inițializării clasei de MQTTController, se crează un client de mqtt. Acestuia i se setează metoda de callback apelabilă în momentul interceptrării mesajelor, se configurează conexiunea la broker print setarea IP-ului acestuia și, la nevoie, portul de comunicare, iar în ultima instanță se apelează metoda de pornirea ciclului infinit de ascultare. Din acest moment, clientul de MQTT este funcțional și capabil de publicare și interceptare de mesaje.

Pentru fiecare topic relevant de primirea mesajelor, clasa de controller expune câte o metodă de setarea a unor callbackuri, astfel realizându-se legătura dintre comunicare MQTT și restul sistemului. Aceste metode de callback se pot seta o singură dată, pentru evitarea unor suprascrieri accidentale. De altfel, pentru fiecare răspuns ce se trimite către brokerul de MQTT, clasa expune câte o metodă de publicarea lor, acestea fiind responsabile de publicarea mesajelor primite pe câte un topic unic.

Formatul mesajelor cu care se lucrează variază în funcție de scopul acestora. Preponderent se folosesc valori de tip intreg, pentru setare de valori, sau șiruri de caractere pentru primirea comenzilor și trimiterea răspunsurilor. Valorile booleene utilizate în setarea stărilor de *pornit* și *oprit* sunt de asemenea șiruri de caractere (*“on”* și *“off”*), această implementare fiind necesară din cauză limitărilor cauzate de serializarea datelor înainte transmiterii lor. Toate mesajele au o structură simplă, cu un singur tip de date, pentru izloarea logică și funcțională a categoriilor de evenimente, cu o singură excepție. După rutina de inițializare, primul mesaj publicat prin MQTT este o structură complexă JSON, care conține toate datele implicite de inițializare ale sistemului. Acestea sunt necesare pentru a îniția procedura de resetare, sau, mai propriu spus, curățare a interfeței, în cazul unei reporniri.

Din contextul fluxului de date și a metodelor de callback setate, doar comenzile primite prin limbaj natural de la utilizator se trimit pentru o prelucrare prealabilă, toate celelalte mesaje interceptate sunt trimise direct modului de control a rețelei Z-Wave pentru prelucrare.

## Implementarea rețelei neuronale pentru clasificare

Implementarea modulului urmează arhitectura standard a unei rețele neuronale feed forward, bazându-se pe datele de antrenare pentru generarea structurii. Concret, numărul de neuroni atât în stratul de intrare, cât și în stratul de ieșire se redefinesc dinamic înainte procesului de antrenare, acestea fiind stabilite de numărul de elemente din lexicul aplicației respectiv, de numărul de categorii cunoscute. Structura stratului ascuns a fost ales în mod empiric, acesta având 3 straturi, fiecare conținând câte 500 de neuroni. Datorită multitudinii de feluri în care utilizatorul poate cere exeuția aceleiași comenzi, este practic imposibil de aproximat, cu atât mai puțin, dedus numărul de caracteristici individuale posibile ce trebuie identificate, pentru a putea cunatifica numărul optim de neuroni necesari în fiecare strat.

Deoarece utilizăm codarea *one hot* pentru procesarea datelor de intrare, rețeaua noastră va primi doar valorile 0 sau 1 în fiecare neuron din stratul de intrare. Stiind că valorile noastre nu pot fi negative, fiecare neuron din straturile ascunse va utiliza funcția de activa ReLU în procesare. Folosind această funcție putem evita problema de *vanishing gradient,* o dificultate întâlnită la antrenarea bazată pe backpropagation, când derivatele valorilor tind spre 0 și se apropie atât de mult de aceasta încât devin irelevante în calcule.

Spre deosebire de neuronii din straturile ascunse, cei din stratul de ieșire vor folosi funcția de activare *Softmax* datorită capacității acesteia de a returna încrederea în corectitudinea rezultatului returnat, în format procentual. Totodată, acesta folosește rezultatele de la toți neuronii din stratul de ieșire, pentru ca suma încrederilor să fie 100%. Astfel, un rezultat returnat de forma vectorului [0.2, 0.7, 0.1] ne rată faptul că din cele trei variante posibile de clase, datele de intrare au o probabilitate de 20% să aparțină primei clase, 70% să aparțină celei de a doua clasă și 10% să aparțină clasei numărul trei.

Pentru evitarea luării unei decizii în condiții de incertitudine, s-a introdus o variabilă prag de decizie setată la 0.85, reprezentând certitudinea minimă a funcției de activare Softmax ce se poate considera viabilă. Altfel spus, orice rezultat care nu a fost calculat cu o precizie de cel puțin 85% va fi evaluat ca imprecis. În acest caz, rețeaua neuronală va returna clasa -1, semnificând imposibilitatea clasificării datelor de intrare, iar utilizatorul va fi rugat să își reformuleze comanda.

Antrenarea rețelei neuronale utilizează lista de date de antrenare preprocesate și codate prin tehnica *one hot*. Din cauza numărului mic de date de antrenare, nu putem folosi antrenarea pe *batch*-uri, dar vom putea organiza datele astfel încât să le putem reutiliza de mai multe ori, de-a lungul antrenării. În primă instanță, se amestecă aleator elementele listei de date, pentru evitarea grupării secvențiale ale categoriilor. Din totalul datelor se izolează ultimele 10%, cu care se vor executa doar operații de verificare, iar antrenarea propriu zisă se va realiza cu primele 90% de date. Din acest punct se inițializează procesul de antrenare, care constă în prelucrarea datelor de intrare, compararea rezultatului dorit cu cel obținut și efectuarea pașilor de *backpropagation*. Acest trio de pași se repetă pentru 100 de iterații, sau epoci. Pentru a putea eficientiza la maxim procesul de învățare, după terminarea iterațiilor, datele de antrenare se vor reamesteca, ultimele 10% se vor izola pentru verificarea acurateții și procesul se repetă, reducând astfel șansele ca unele date de antrenare relevante să nu se ia în calcul.

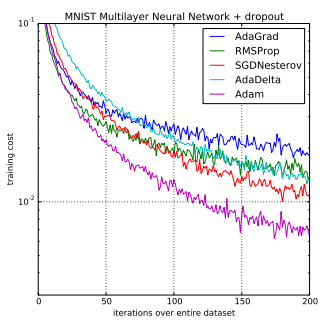
În optimizarea procedurii de *backpropagation* am utilizat algoritmul de optimizare Adam care, la fiecare iterație de calcul al gradientului funcției cost, poate ajusta amploare optimizării valorilor, țînând cont de înclinația pantei funcției. Cu cât această pantă este mai abruptă, cu atât rezultatul optim se află la o distanță mai mare, iar pașii de optimizare pot fi mai mari. Cu cât rezultatul calculat se apropie mai mult de cel dorit, cu atât panta funcției va fi mai puțin abruptă, și pașii de optimizare mai mici.



**Figura 28**. Vizualizarea optimizării Adam

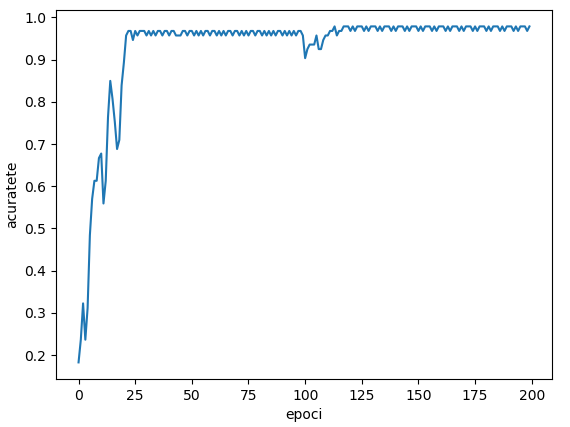
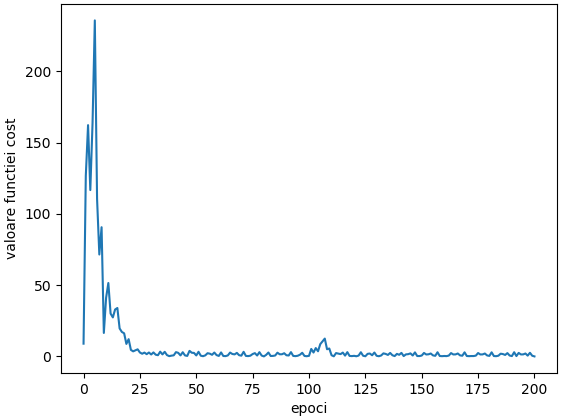
În figura 28este prezentată schimbarea vitezei de optimizare în cazul utilizării funcției Adam, pe graficul unei funcții. Punctul de pornire este marcat prin cercul cu numărul 1, fiecare pas de optimizare urmând panta până în punctul rezultat, marcat prin cercul verde. Înclinațiile pantei în punctele calculate sunt marcate cu galben, iar mărimile pașilor de optimizare prin săgețile negre.

Folosindu-se de capacitatea sa adaptivă, funcția de optimizare Adam are cea mai mare viteză de convergență dintre funcțiile de optimizare utilizate pentru antrenarea rețelelor neuronale, fapt prezentat și în figura urmăatoare.



**Figura 29**. Viteze de convergență a funcțiilor de optimizare[[6]](#footnote-6)

Rezultatele procesului de antrenare se oversă dion reducerea progresivă a funcției cost, precum și creșterea acurateții generale ale clasificărilor. Ambele variabile se memorează la sfârșitul fiecărei epoci de antrenare, înaintea efectuării pașilor de *backpropagation.*



**Figura 30**. Optimizarea funcției cost și a acurateții

În urma ciclului de antrenare, aplicația a ajuns la un procentaj de acuratețe de 97.84%, iar minima valorii cost a fost de 0.068.

După finalizarea etapei de antrenare, modelul rețelei neuronale se persistă pentru utilizări ulterioare.

## Implementarea modulului de control Z-Wave

Pentru a putea interacționa cu echipamentele interconectate în rețeaua Z-Wave s-a folosit librăria python-openzwave[[7]](#footnote-7), care expune un API în limbajul Python și permite interacționare prin clase predefinite, elemente componente ale structurii librăriei.

Clasa *ZwaveNetwork* este elementul principal în ierarhia de interacțiune cu controller-ul, acesta fiind configurat la momentul inițializării modulului. Rolul acestuia este de a identifica elementele componente ale rețelei, de a interoga capacitățile fiecărui element și de a seta configurările generale ale rețelei. După inițializarea acestuia, se pot interoga nodurile conectate folosind clasa *ZWaveNode*. Fiecare echipament conectat se consideră un astfel de nod în contextul rețelei, fiecare având capabilități diferite în funcție de tipul echipamentului. Astfel, avem elemente de bază (ZWaveNode*Basic*), întrerupătoare (*ZWaveNodeSwitch*), senzori (*ZWaveNodeSensor*), control de temperatură (*ZWaveNodeThermostat*), elemente de securitate (*ZWaveNodeSecurity*) și încuietoare (*ZWaveNodeDoorLock*). Folosind combinații de aceste noduri, se obțin capacitățile de manipulare a tuturor echipamentelor conectate la rețea. Pentru a putea interacționa cu aceste noduri, există clasa *ZwaveCommand* cu ajutorul căruia se pot interoga și seta toate clasele de comandă specifice protocolului. Fiecare clasă de comandă se identifică printr-un identificator valoric unic (ex. 72057594076299265). La rândul său, această clasă se descompune în mai multe clase de valori ZwaveValue cu ajutorul cărora se pot interoga și seta valorile disponibile din structura unei comenzi. Aceste clase se identifică print identificatori de tip șir de caractere (ex. *“current\_temperature”*). Mai multe detaliile despre implementarea curentă a librăriei python-openzwave se pot găsi în manualul său de utilizare[[8]](#footnote-8).

După inițializarea cu succes a rețelei, putem interoga toate nodurile conectate identificate, iar în funcție de clasele de comandă cunoscute, putem identifica tipul fiecărui nod. În cazul aplicației curente, se identifică trei noduri diferite ale rețelei, controllerul USB și alte două echipamete active. Stabilirea identității fiecărui nod se face prin interogarea capacităților sale de a prelucra anumite valori din structura comenzilor suportate. Astfel, nodul care conține valoarea “*Level*”, reprezentând intensitatea de iluminare, se consideră becul din sistemul nostru, iar nodul care conține în structura comenzilor suportate valoarea “*Temperature*”*,* se consideră valva termostat. Iterând peste toate elementele componente ale rețelei, se obțion toți identificatorii cu ajutorul cărora aplicația noastră va putea interacționa cu rețeaua Z-Wave. Totodată, datorită faptului că în momentul alterării structurii rețelei prin adăugare sau eliminare de echipamente, identificatorii acestora se schimba, acest proces de identificare dinamică în momentul inițializării permite menținera integrității sistemului, fără a necesita schimbarea manuală a identificatorilor.

Interacțiunea cu acest modul se realizează print intermediul metodelor de *callback* setate în parte pentru fiecare tip de valoare urmărită. În cazul becului se urmărește nivelul de iluminare, iar în cazul termostatului se urmăresc valorile pentru temperatura ambientală, temperatura prag de activare, cât și nivelul curent a bateriei. Pentru a intercepta schimbările acestor valori, a fost necesară implementarea adiționalăp a unei metode de setarea unui *callback* în interiorul clasei *ZWaveNetwork*, care va fi apelată în cazul oricărei modificări din cadrul rețelei Z-Wave.

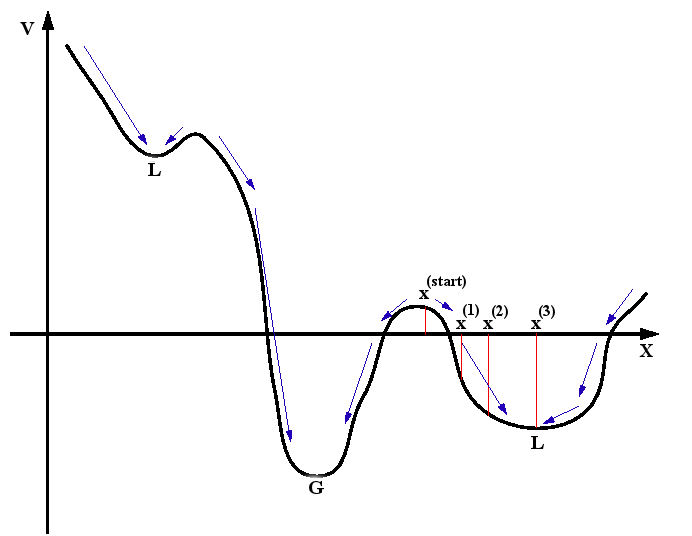
În momentul închiderii sistemului, toate valorile de control se setează la valorile lor implicite și se apelează metoda închiderii rețelei care, la rândul său, va închide comunicarea cu controllerul USB și va elibera comincarea serială către acesta. Fără apelul explicit a acestei rutine de închidere, există riscul menținerii controlului asupra comunicării seriale cu controllerul, eliberarea sa ulteriară fiind posibilă doar prin repornirea sistemului.

# Limitări și posibile îmbunătățiri viitoare

În starea curentă, chiar dacă aplicația dezvoltată este funcțională, suferă de unele limitări funcționale datorate atât tehnicilor, cât și tehnologiilor utilizate în procesul de implementare. În acest capitol vom analiza principalele impedimente ale aplicației și vom considera posibile variante de îmbunătățire.

## Problema minimei locale

Tehnicile de învățare supervizată aplicate rețelelor neuronale multistrat folosind algoritmi convențioanli de învățare suferă adesea de problema minimelor locale. Algoritmii bazați pe *gradient descent*, inclusiv algoritmul de *backrpopagation* modifică parametrii rețelei neuronale utilizând o suită de date de antrenare fără cunoștiințe prealabile despre natura acestora. Utilizarea metodei *gradient descent* pentru ajustarea parametrilor implică urmărirea înclinării pantei suprafeței de eroare, care poate duce la puncte nedorite, minime locale[[9]](#footnote-9). Pentru o funcție cost aplicată în procesul de antrenare, minimele locale reprezintă un punct în codomeniul funcției în care soluția este optimă pentru toate valorile din imediata sa vecinătate, dar nu și pentru întregul domeniu de valori. În schimb, minima globală reprezintă punctul în codomeniul funcției în care soluția este optimă pentru întregul domeniu de valori, funcția neavând o soluție mai bună în niciun punct al său.



**Figura 31**. Minimele locale ale unei funcții[[10]](#footnote-10)

Figura 31 prezintă graficul unei funcții oarecare, în planul compus de axele **X** și **V**. Pe lângă minima globală marcată cu **G**, acesta prezintă două minime locale, ambele marcate cu **L** și sensurile de convergență către acestea marcate prin săgețile albastre. Folosind o astfel de funcție cost în procesul de antrenare, poate exista situația în care antrenarea începe în punctul **x(start)**, iar primul pas de optimizare se face spre dreapta în punctul **x(1)**. Din acest moment este practic imposibil ca antrenarea să rezulte în punctul optim G, cel mai bun rezultat returnabil fiind minimia locală în punctul **x(3)**, iar acest lucru ar rezulta în blocarea acurateții antrenării sub potențialul său optim.

Pentru evitarea acestor blocaje, există diferite tehnici de optimizare ale procesului de antrenare sau de preanaliza datelor de antrenare. În cazul tehnicilor de optimizare, variantele cel mai des aplicate sunt, fie utilizarea unor algoritmi care nu depind de analiza pantei funcției la un moment dat, fie introducerea unor *sărituri* aleatoare în cazul în care se observă încetinirea vitezei de învățare. În cazul preanalizei detelor de antrenare, cel mai adesea se verifică omogenitatea acestora în planul de cunoștiințe. Concret, se încearcă vizualizarea planului de cunoștiințe format din totalitatea datelor, iar în cazul întâmpinării unor minime locale, se încearcă fie înlocuirea acelor date care corespund zonei respective, fie eliminarea lor completă, pentru a forma un plan uniform și convergent către un singur punct optim.

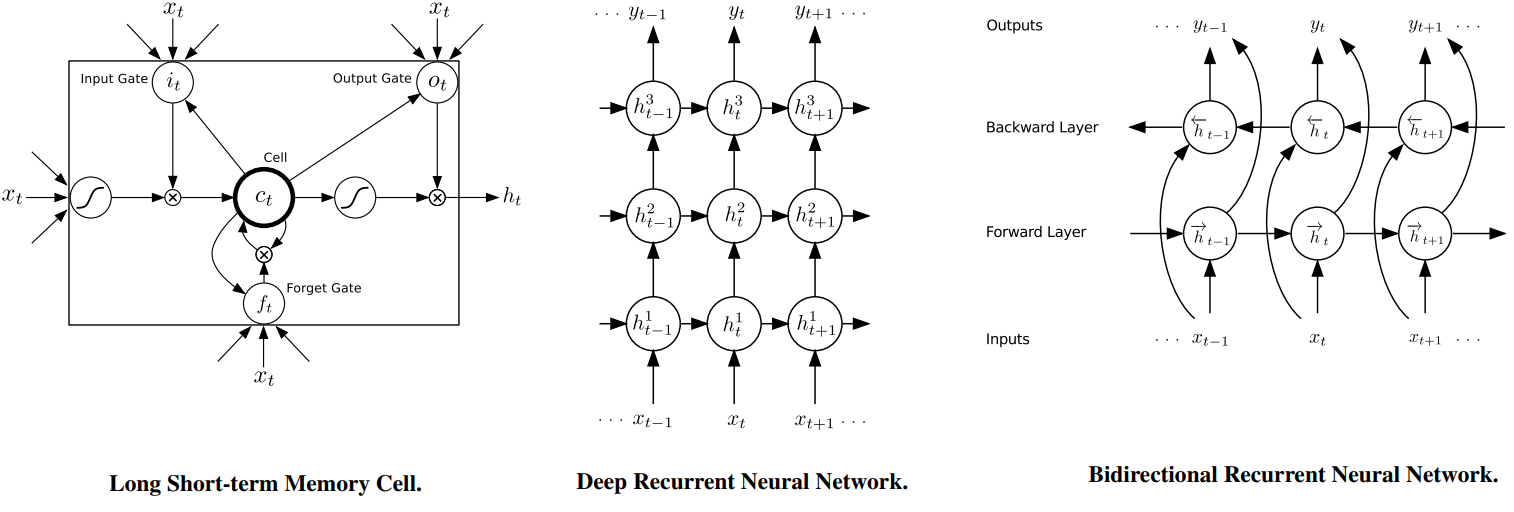
## Structura rețelei neuronale feed forward

Aplicația noastră a fost dezvoltată în jurul conceptului de clasificare folosind rețele neuronale, motiv pentru care alegerea unei structuri de rețea *feed forward* pare justificată însă, utilizând texte în limbaj natural ca date de intrare a prezentat unele impedimente întâlnite și de alte aplicații bazate pe procesarea limbajului natural.

Principalele dificultăți în utilizarea textelor în limbaj natural provin din complexitatea structurală și diversitatea semantică a diferitol construcții. Este foarte grea prelucrarea acestor structuri într-o formă utilizabilă de o rețea neuronală astfel încât structura lor să nu fie alterată. În implementarea noastră am utilizat codare *one hot* cu ajutorul căreia datele de intrare erau transformate în vectori de lungimea lexicului aplicației, o valoare fixă, iar fiecare element al vectorului marca existența sau inexistența unui cuvânt din lexic în proziția noastră inițială. O astfel de abordare prezintă trei impedimente importante:

1. Indiferent de lungimea datelor de intrare, forma lor procesată va avea lungime fixă și egală cu numărul cuvintelor din care se compune lexicul aplicației. În situația dată, există riscul ca informații vitale din propoziția dată ca date de intrare să fie pierdută. Desigur, astfel de cazuri se pot evita prin utilizarea unui număr mare de cuvinte individuale în datele utilizate pentru antrenare și din care se formează lexicul aplicației. Această contrângere apare din cauza structurii rigide ale rețelelor neuronale de tipul *feed forward*, numărul neuronilor în stratul de intrare fiind fixe.
2. Numai cuvintele utilizate în procesul de antrenare vor putea fi interpretate de către aplicație. Din acest motiv, orice cuvânt care nu este prezent în lexic, în momentul codării, va fi ignorat și marcat ca inexistent. Drept exemplu putem prezenta o situație în care lexicul este compus din cuvintele *hello, I, am* și*, cold.* Dacă încercăm să procesăm propoziția dată *I cannot see*, singurul cuvânt codat va fi *I,* pierzând astfel o treime din informațiile utile, iar aplicația va întâmpina dificultăți încercând să clasifice datele de intrare.
3. O structură neuronală care utilizează totalitatea datelor de intrare în același timp, nu va cunoaște noțiunea temporală frecvent prezentă în semantica propozițiilor. Transformarea cuvintelor în radicalii lor de bază elimină orice indiciu de cadru temporal, toate propozițiile fiind codate în forma lor prezentă. O propoziție de forma *I was cold* (mi-a fost frig) este echivalent cu *I am cold (îmi este frig).* Datorită lipsei de inferare contextuală, complexitatea interacțiunilor cu sistemul se limitează la comenzi simple în forma prezentă.

Datorită complexității lor semantice și sintactice, datele de intrare în limbaj natural au constitui mereu o provocare aparte. Pentru intensificare capacității de interpretarea acestora, s-au dezvoltat structurile rețelelor neuronale de tip recurente*.* Spre deosebire de *FFNN,* strucura unui *RNN* permite prelucrarea secvențială a datelor de intrare cu posibilitatea inferării semantice ale datelor, raportat la cele prelucrate în prealabil. Concret, structurile neuronale recurente permit procesarea unui singur cuvânt, reținera semanticii sale și utilizând această informație în prelucrarea următorului cuvânt. Astfel, se pot prelucra propoziții de lungime aleatoare și cu sensuri multiple, eliminând cele trei limitări principale sus menționate.



**Figura 32**. Diferite structuri ale rețelelor neuronale recurente[[11]](#footnote-11)

Rețelele neuronale recurente diferă de cele *feed forward* atât în legăturile dintre neuronii să, cât și în structura și funcționalitatea neuronilor. În variantele LSTM, neuronii pot reține unele informații procesate pentru un interval definit de timp, utilizându-le în procesările ulterioare, oferind astfel posibilitatea deducerii informațiilor prin compunerea mai multor date de intrare. Formele de DRNN conțin în structura lor legături între neuronii din același strat, oferind o complexitate mai mare în procesarea caracteristicilor individuale. Ultima variantă prezentată în imagine, BRNN, oferă posibilitatea transmiterii informațiilor procesate de neuroni atât în față cât și în spate, în cadrul procesării secvențiale de date, influențând astfel interpretarea datelor furnizate atât înaintea cât și în urma procesării curente.

## Limitările librăriei python-openzwave

Protocolul Z-Wave este relativ des întâlnit în domeniile industriale de automatizare dar, din cauza unei strategii slabe de promovare și a popularității rivalului său direct, ZigBee, nu s-a format o comunitate de suporteri pentru acesta, motiv pentru care integrarea sa cu diverse limbaje și tehnologii de programare este foarte limitată. Librăria python-openzwave a fost dezvoltat ca *software open source*, tocmai pentru a ușura integrarea protocolului cu mediile de dezvoltare care utilizează limbajul de programare Python. Fiind un modul în continuă dezvoltare datorită membrilor contribuitori, există unele buguri raportate care încă nu au fost tratate, precum și unele funcționalități lipsă. Din cauza schimbărilor continue în codul sursă, unele funcționalități pot suferi modificări de la o versiune la alta.

La momentul realizării acestei documentației, versiunea 0.4.0.27 este cea mai recentă variantă utilizabilă și este versiunea pe care se bazează aplicația noastră. Acesta suferă de unele probleme de consistență la inițializare și la dealocări de resurse. De-a lungul implementării am întâmpinat probleme aparent aleatoare de inițializarea rețelei, în special probleme de citirea configurațiilor de sistemului. În aceeași categorie au existat și unele probleme de dealocare de resurse în momentul închiderii sistemului, deși acestea au fost mult mai rare. O altă problemă importantă de remarcat a fost lipsa posibilității de setare a culorii unor becuri cu capacități de RGB, deși în versiunea anterioară al modului acest lucru era posibil, motiv pentru care prezenta lucrare nu a fost extinsă cu posibilitatea interpretării comenzilor de schimbarea culorii becului, deși acestă opțiune era una dorită. Totuși, codul de tratarea schimbării culorilor becului există în sursele aplicației, în speranța ca o nouă versiune al modului să suporte acest lucru.

## Metode alternative de interacțiune

Pe lângă posibilitatea interacțiunii prin mesaje text, cerând executarea unei comenzi și confirmării acesteia, o funcționalitate planificată pentru viitoarele versiuni ale aplicației ar fi capacitatea interpretării vocii utilizatorului și a generării răspunsurilor tot în format de voce. Majoritatea aplicațiilor curente de tipul asistenților virtuali oferă aceste funcționalități, succesul implementării lor depinzând de la un produs la altul. Pentru a putea sintetiza voce atât în direcția de intrare (Voice tot text) cât și de ieșire (Text to voice), cele mai multe aplicații se bazează pe servicii externe, acceesibile din internet. Ca si flux de lucru, în momentul primirii unei înregistrări, aceasta este trimisă servicului extern pentru sintetizare, iar reztultatul operației este primit înapoi. Operațiunea inversă este efectuată în cazul răspunsurilor generate de aplicație. Externalizarea acestei procesări contravine, însă, principiului fundamental de care s-a ținut cont în momentul dezvoltării aplicației curente, cea de a oferi o soluție independentă, fără necesitatea unei conexiuni la internet. O alternativă ar consta în implementarea propriului sintetizator de voce, dar cum procesările complexe de semnal pot fi costisitoare în resurse și timp, este necesară o analiză prealabilă a viabilității soluțiilor posibile.

Un alt aspect ce se dorește a se dezvolta este posibilitatea accesării aplicației din afara rețelei locale. Sunt considerate două posibilități în momentul de față, fie accesarea interfeței printr-o adresă publică de internet, fie folosind o aplicație de mesagerie instantă de genul Facebool Messenger sau WhatsApp, folosind platforme de integrare de genul ChatFuel[[12]](#footnote-12) sau Xenioo[[13]](#footnote-13), caz în care interacțiunea dintre utilizator și aplicație ar rămâne la nivelul mesajelor text.

# Concluzii

În această lucrare am abordat detaliile implementării aplicației A.L.I.C.E. (Automation of Locally Integrated Control Equipment) și am analizat viabilitatea soluției curente. Consider că aplicația și-a atins scopul principal, acela de a integra cu succes două dintre domeniile curente foarte des tratate, Inteligența Artificială și Internetul Lucrurilor (IoT) și am demonstrat faptul că este posibilă implementarea unor astfel de aplicații folosind doar resurse tehnologiice accesibile publicului în masă.

În urma experiențelor cumulate de-a lungul perioadei de cercetare și implementare, pot conclude că atât domeniul Inteligenței Artificiale, cât și cel al Internetului Lucrurilor se află încă în etapele inițiale de formare și maturizare, cu șanse reale de dezvoltare semnificativă în viitorul apropiat, deja observându-se o tendință accentuată de extinderea ariei de interes din domeniul academic și cel al cercetării către sfera utilizatorilor de rând.

Rezultatele empirice arată că procesarea limbajului natural de către rețele neuronale încă întâmpină dificultăți, mai ales din cauza limitărilor tehnologice ale implementării, un foarte mare impact în calitatea rezultatelor fiind analiza prealabilă și alegerea datelor cu care se va realiza antrenarea rețelei. În mod ideal, acestea ar trebui să fie cât mai omogene dar și cuprinzătoare, astfel încât să includă cât mai mult posibil din posibilele variantele de utilizare, distribuite cât mai uniform în aria de cunoștiințe. Totodată, combinând funcționalitățile de interpretarea limbajului natural și a operațiilor de clasificare s-au dovedit rezultatele cercetărilor vizând utilitatea diferitelor structuri ale rețelelor neuronale, în preponderent dificultățile unei rețele de tipul *feed forward* în utilizarea limbajului natural ca date de intrare. Consider că proiectul curent se situează la limita superioară a capacităților unui FFNN, producând încă rezultate acceptabile. Dacă acestea s-ar dori îmbunătățite, va fi necesară utilizarea unei alte structuri din categoria rețelelor neuronale recurente.

La nivel comunicării prin protocolul Z-Wave, rezultatele sunt limitate doar din cauza maturității curente a modulelor de control dedicate. Nu am întâmpinat dificultăți în integrarea sau utilizarea echipamentelor inteligente, concluzionând că protocolul Z-Wave este o variantă viabilă pentru proiecte de case inteligente și în egală măsură pentru orice tip de proiect din domeniul Internetului Lucrurilor.

În stadiul curent, aplicația se poate considera utilizabilă în limite acceptabile, având posibilitatea extinderii funcționale cu tehnologii și module adiționale cu scopul îmbunătățirii experienței utilizatorului.

# Bibliografie

# Anexe

Anexa 1. Lista datelor de antrenare a rețelei neuronale

greeting : greetings

greeting : hello

greeting : hello there

greeting : hi

greeting : hi there

greeting : greetings alice

greeting : greetings to you

greeting : hello alice

greeting : hi alice

greeting : hey there alice

greeting : hello there alice

greeting : alice

greeting : good morning

greeting : good day

greeting : good afternoon

greeting : good evening

greeting : good night

greeting : hey

greeting : howdy

lights on : turn on the lights

lights on : could you turn the light on?

lights on : please turn on the lights

lights on : could you turn the lights on please

lights on : would you please turn on the lights

lights on : turn the light on

lights on : it's dark in here

lights on : make it brighter

lights on : brighter please

lights on : i want the room brighter

lights on : i can't see

lights on : i want to see

lights on : i need to see

lights on : i need light

lights on : lights on

lights on : switch on the light

lights on : switch the lights on

lights on : please switch on the light

lights on : could you switch the lights on

lights on : would you switch the lights on

lights on : let there be light

lights on : illuminate

lights on : could you please turn on the lights?

lights on : turn up the light

lights off : turn off the lights

lights off : could you turn the light off

lights off : please turn off the lights

lights off : turn the light off

lights off : it's too bright in here

lights off : there is too much light

lights off : make it darker

lights off : would you turn the light off please

lights off : lights off

lights off : switch off the light

lights off : switch the lights off

lights off : could you switch the lights off

lights off : would you switch the lights off

lights off : deluminate

lights off : could you please turn off the lights?

lights off : lights off please

lights off : turn down the light

lights off : dim the lights

lights off : dim the room

lights off : make it dark

lights off : make dark

heat up : make it warmer

heat up : it's cold

heat up : it's too cold in here

heat up : it's too cold

heat up : i am freezing

heat up : would you make it warmer please

heat up : please make it warmer in here

heat up : could you please make it hotter

heat up : make it warmer please

heat up : make it hotter please

heat up : turn up the heat

heat up : heat up

heat up : i am cold

heat up : it's chilly

heat up : switch on the heating

heat up : heat up

heat up : I'm so cold

heat up : I'm cold

heat up : me so cold

heat up : me is cold

heat up : increase temperature

heat up : raise the temperature

heat up : raise the heat

heat up : it's freezing

heat up : it's freezing in here

heat up : it's cold in here

heat up : warm me up

heat up : increase heat

heat up : temperature up

heat up : temperature increase

heat down : make it colder

heat down : make it cooler

heat down : it's hot

heat down : it's warm

heat down : it's too hot

heat down : it's too warm

heat down : it's too warm in here

heat down : it's too hot in here

heat down : please make it colder

heat down : please make it cooler

heat down : make it colder please

heat down : make it cooler please

heat down : please make it cooler

heat down : could you please make it cooler

heat down : would you please make it cooler

heat down : could you please make it colder

heat down : would you please make it colder

heat down : make it cooler please

heat down : make it colder please

heat down : heat down

heat down : turn down the heat

heat down : me so hot

heat down : it's getting hot in here

heat down : it's getting warm in here

heat down : decrease temperature

heat down : temperature decrease

heat down : decrease the temperature

heat down : decrease heat

heat down : decrease the heat

heat down : heat down

heat down : heat decrease

heat down : lower the temperature

Anexa 2.Implementarea fluxului de elemente ale interfeței Node-RED



Anexa 3. Structura internă ale claselor componente din Back End

1. Radu Ionescu (2013), Note de laborator, Disponibil pe [http://raduionescu.herokuapp.com](http://raduionescu.herokuapp.com/ia%20/) [Data accesării 10 Ianuarie 2019] [↑](#footnote-ref-1)
2. Arizona State University, (2011), ***Neuron anatomy.*** [Online] Disponibil pe: <https://askabiologist.asu.edu/neuron-anatomy> [Data accesării 10 Ianuarie 2019] [↑](#footnote-ref-2)
3. http://www.aut.upt.ro/~andreea.robu/ReteleNeuronale2.pdf [↑](#footnote-ref-3)
4. Verginica Barbu Mititelu (2011), Identificarea automată a afixelor româneşti. Studiu de caz: identificarea sufixelor, *România de Interacțiune Om-Calculator*, 4, p 110 - 112 [↑](#footnote-ref-4)
5. https://pypi.org/project/paho-mqtt/ [↑](#footnote-ref-5)
6. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ [↑](#footnote-ref-6)
7. https://github.com/OpenZWave/python-openzwave [↑](#footnote-ref-7)
8. https://media.readthedocs.org/pdf/pyozw/latest/pyozw.pdf [↑](#footnote-ref-8)
9. A. Atakulreka și D. Sutivong (2007), “Avoiding Local Minima in Feedforward Neural Networks by Simultaneous Learning” (M. A. Orgun and J. Thornton Eds.), pag.2, Note de curs despre Inteligență Artificială 4830, Springer-Berlin. [↑](#footnote-ref-9)
10. http://www.ccl.net/cca/documents/molecular-modeling/node8.html [↑](#footnote-ref-10)
11. Graves, A. și Jaitly, N. (2014). *Towards end-to-end speech recognition with recurrent neural networks*, 31st International conference on machine learning, Google scholar [↑](#footnote-ref-11)
12. https://chatfuel.com/ [↑](#footnote-ref-12)
13. https://www.xenioo.com/creating-first-whatsapp-bot/ [↑](#footnote-ref-13)