**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ ,,FERDINAND I”**

**FACULTATEA DE SISTEME ELECTRONICE ŞI INFORMATICE MILITARE**

**Specializare: Comunicații pentru apărare și securitate**



**SISTEM AUTONOM DE DEPLASARE PE BAZA RECUNOAȘTERII SEMNELOR DE CIRCULAȚIE CU RASPBERRY PI**

CONDUCĂTOR ȘTIINȚIFIC:

**Conf. univ. dr. ing Florin Roman ENACHE**

ABSOLVENT:

**Sd. sg. maj. Adriana-Victorița MIU**

Conţine \_\_\_\_\_\_\_\_\_\_\_ file

Inventariat sub nr. \_\_\_\_\_\_

Poziţia din indicator: \_\_\_\_

Termen de păstrare: \_\_\_\_\_

**BUCUREŞTI**

**2022**

# ABSTRACT

Traditional methods in machine learning for detecting traffic signs and classification are replaced by the recent enhancements of deep learning object detection methods by success of building convolutional neural networks (CNN), which is a component of deep learning.

Edge AI is a growing area. The use of deep learning on low cost machines, such as the Raspberry Pi, may be used more than ever due to the easy use, availability and high performance.

This paper presents a deep learning approach for detection and recognition of traffic signs by implementing one object detection model and by evaluating the flexibility of the TensorFlow Object Detection Framework to solve the real-time problems. The model is Single Shot Multibox Detector (SSD)MobileNet V2.

Also, this paper describes a method for detecting and recognizing road signs in videostreaming acquired by a single raspicam mounted on an handmade moving vehicle in miniature using Raspberry Pi for controlling it.

The mean purpose of the project is to perform a real-time mobile platform which is capable to take quick decisons based on what it sees around. So, for developing and performing real-time operation capability, I will approach two diferent methods: first one is based on quantization and using of Coral USB Accelerator and the second one involves a client-server mechanism between Raspberry pi and my laptop in a local network.

As regards the first method, the inference runs on Raspberry Pi and for the model to be compatible and easy to run it in real-time on Raspberry Pi, it has to be quantized. So, a quantized pretrained SSD object detection model was deployed to our machine, Raspberry Pi 4 B, to evaluate if the throughput is sufficient for doing real-time object recognition. With input size of 320x320, an inference frame rate of 1.5-2 fps was obtained, which is still not enough for real-time. Therefore, for accelerating the inference, it will be used an Edge TPU coprocessor to the system, such as the Coral USB Accelerator from Google, enabling high-speed machine learning inferencing

Using a lightweight model is for the benefit of higher throughput as a trade-off for lower accuracy. To compensate for the loss of accuracy, using transfer learning and tensorflow, a custom object detection model has been trained by fine-tuning a pretrained SSD model. The fine-tuned model was trained on images scraped from the web with a few specific traffic signs images, such as: straight ahead sign, stop sign, turn right sign snd speed limit sign. The pretrained model was trained on COCO dataset, an huge dataset of 328k images, to detect different objects. Predictions shows that the custom model performs significantly better doing detections on frames with traffic signs. However, some images are of bad quality and therefore it is important to thoroughly clean and select which images that is suitable to keep, given a specific application.

In terms of the second method, the inference runs on my laptop with GPU capabilities in the same local network with the Raspberry Pi. So, the video stream aquired by raspicam will be transmitted using sockets to my laptop, where with the help of OpenCV and of the custom model, the detection will be made. The detection output will be transmitted through sockets back to the Raspberry Pi and processed in the motor gear control script.

The both methods show good results, as example of 12-15 fps for inference, which is a visbile improvement compared to the previous result of 1.5-2 fps inference.

# REZUMAT

Acest proiect de diplomă își propune realizarea unui sistem autonom care să se deplaseze pe un traseu bine delimitat pe baza semnelor de circulație și demonstrarea posibilității de concretizare a unui software capabil să detecteze și să realizeze recunoașterea acestora în timp util. Software-ul este apt în a fi rulat pe sisteme embedded datorită dimensiunii mici și a vitezei mari, menținând în parametri optimi precizia detecției si a prelucrării imaginilor în timp real.

De-a lungul acestei lucrări sunt explicate bazele realizării proiectului, cât și procesele ce au condus la implementarea și configurarea software-ului și a compoziției hardware .

Limbajul de programare majoritar utilizat în cadrul proiectului este Python, un limbaj de nivel înalt, datorită dinamicității si a capabilității sale de a dezvolta aplicații de tip *Machine Learning*, sisteme capabile să învețe prin experiență în urma aplicării unor algoritmi.

Primul capitol are un rol introductiv și prezintă tema și scopul proiectului alături de noțiunile generale despre inteligența artificială și subdomeniul acesteia utilizat în cadrul proiectului, Deep Learning, precum și impactul atins de AI în dezvoltarea mașinilor autonome.

Al doilea capitol aprofundează domeniul Deep Learning-ului, în speță a învățarii supravegheate. Astfel, acest capitol detaliază teoretic algoritmii, arhitectura rețelei neuronale și procesul de obținere a unui model apt sa realizeze detecția și recunoașterea semnelor rutiere. Sunt discutate modul de funcționare al rețelelor neuronale convoluționale și procesele care sunt vizate în vederea antrenării unui model de detecție a obiectelor.

Al treielea capitol are rolul de a prezenta implementarea software și implementarea hardware ale proiectului. Implementarea software implică mediul de lucru, framework-ul de DL și structuralizarea arhitecturii rețelei neuronale utilizate și limitările impuse de factorii de mai sus. Implementarea hardware urmărește procesul de realizare al platformei, componența acesteia, legăturile fizice între modulele externe și platforma de dezvoltare și limitările la nivel hardware. Sunt prezentate, de asemenea , și detalii despre senzorii de distanță, senzorii cu infraroșu, modul de lucru și schemele lor electrice

Al patrulea capitol are un rol experimental, prezentând rezultatele practice obținute și concluziile trase în urma acestora. Totodată, în acest capitol se realizează un debug general asupra întregului sistem realizat și se evidențiază limitările și problemele întălnite și posibilele viitoare îmbunătățiri ale acestora.

# CUPRINS

[ABSTRACT 5](#_Toc103874605)

[REZUMAT 6](#_Toc103874606)

[CUPRINS 7](#_Toc103874607)

[LISTĂ DE ABREVIERI 9](#_Toc103874608)

[TABELĂ FIGURI 10](#_Toc103874609)

[1. OBIECTIVE ȘI NOȚIUNI GENERALE 11](#_Toc103874610)

[1.1 Obiective,motivația alegerii temei și domeniile de aplicație 11](#_Toc103874611)

[1.2 Noțiuni generale 12](#_Toc103874612)

[1.2.1 Deep Supervised Learning (DSL) 14](#_Toc103874613)

[1.2.2 Deep Unsupervised Learning (DUL) 15](#_Toc103874614)

[1.2.3 Deep Semi-Supervised Learning (DSSL) 16](#_Toc103874615)

[1.3 Abordarea Deep Learning 16](#_Toc103874616)

[1.4 Provocări impuse de Deep Learning 17](#_Toc103874617)

[1.5 Deep Learning și Machine Learning. Diferențe. 17](#_Toc103874618)

[1.6 Deep Learning în contextul automatizării autovehiculelor 19](#_Toc103874619)

[1.6.1 Percepția 20](#_Toc103874620)

[1.6.2 Localizarea 21](#_Toc103874621)

[1.6.3 Predicția 21](#_Toc103874622)

[1.6.4 Luarea deciziilor 21](#_Toc103874623)

[2 NOȚIUNI TEORETICE 23](#_Toc103874624)

[Definiții 23](#_Toc103874625)

[2.1 Procesul de învățare 28](#_Toc103874626)

[2.1.1 Funcția de activare 29](#_Toc103874627)

[2.1.2 Funcția de pierdere 33](#_Toc103874628)

[2.1.3 Gradienții de calcul 33](#_Toc103874629)

[2.1.4 Funcția de optimizare 33](#_Toc103874630)

[2.1.5 Parametrii și hiperparametrii rețelelor neuronale convoluționale. Probleme impuse în urma modificării acestora 33](#_Toc103874631)

[2.2 Rețelele neuronale convoluționale (CNN) 34](#_Toc103874632)

[2.2.1 Arhitectura 34](#_Toc103874633)

[2.2.2 Convoluția 34](#_Toc103874634)

[2.2.3 Pooling 34](#_Toc103874635)

[2.2.4 Fully-Connected 34](#_Toc103874636)

[2.3 Modelul de antrenare ales 34](#_Toc103874637)

[2.4 Modalități de antrenare 34](#_Toc103874638)

[2.4.1 Fine Tuning 34](#_Toc103874639)

[2.4.2 Transfer Learning 34](#_Toc103874640)

[2.4.3 Learning from scratch 34](#_Toc103874641)

[2.5 Procesarea imaginilor în DL 34](#_Toc103874642)

[2.5.1 Adnotarea 34](#_Toc103874643)

[2.5.2 Augumentarea 34](#_Toc103874644)

[3 METODE DE IMPLEMENTAREA ȘI VERIFICARE 34](#_Toc103874645)

[3.1 Implementarea software 34](#_Toc103874646)

[3.1.1 Mediul de dezvoltare, limbajul de programare și librării utilizate 34](#_Toc103874647)

[3.1.2 Construirea setului de date 34](#_Toc103874648)

[3.1.3 Implementarea CNN pe baza principiului de Transfer Learning 34](#_Toc103874649)

[3.1.4 Modelul SSD MobileNet v2 320x320 34](#_Toc103874650)

[3.1.5 Procesorul grafic Nvidia 34](#_Toc103874651)

[3.1.6 CPU și GPU. Limitări impuse de placa grafică. 34](#_Toc103874652)

[3.1.7 Coral USB Accelerator (TPU Edge) 34](#_Toc103874653)

[3.1.8 Tensorflow. Framework-ul TFLite 34](#_Toc103874654)

[3.1.9 Tensorboard 34](#_Toc103874655)

[3.1.10 OpenCV 34](#_Toc103874656)

[3.1.11 Probleme de implementare și soluții 34](#_Toc103874657)

[3.2 Implementare hardware 34](#_Toc103874658)

[3.2.1 Platforma computațională Raspberry Pi 4B 34](#_Toc103874659)

[3.2.2 Driver-ul L298N Dual Motor 34](#_Toc103874660)

[3.2.3 Modulul cameră v2 Raspberry Pi 34](#_Toc103874661)

[3.2.4 Probleme hardware de implementare și soluții 34](#_Toc103874662)

[4 REZULTATE EXPERIMENTALE 34](#_Toc103874663)

[4.1 Prezentarea evoluției sistemului autonom analizat 34](#_Toc103874664)

[4.2 Rezultate obținute 35](#_Toc103874665)

[4.3 Analiza performanțelor sistemului obținut 35](#_Toc103874666)

[5 CONCLUZII ȘI PERSPECTIVE DE VIITOR 35](#_Toc103874667)

[6 BIBLIOGRAFIE 35](#_Toc103874668)

[7 Bibliography 35](#_Toc103874669)

[8 ANEXE 36](#_Toc103874670)

# LISTĂ DE ABREVIERI

|  |  |  |
| --- | --- | --- |
|  | AV | Autonom Vechicle |
|  | AI | Artificial Intelligence |
|  | ADAS | Advanced Driver Assitance Systems |
|  | SVM | Support Vector Machine |
|  | ANN | Artificial Neural Network |
|  | DNN | Deep Neural Network |
|  | DL | Deep Learning |
|  | CV | Computer Vision |
|  | LiDAR | Light Detection And Ranging |
|  | RADAR | Radio Detection and Ranging |
|  | DRL | Deep Reinforcement Learning |
|  | MSE | Mean Square Error |
|  | MAE | Mean Basolute Error |
|  | PDF | Probability Density Function |
|  | ADAM | Adaptive Moment Estimation |
|  | SGD | Stochastic Gradient Descent |
|  | BGD | Batch Gradient Descent |
|  | GPU | Graphic Processing Unit |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# TABELĂ FIGURI

# 1. OBIECTIVE ȘI NOȚIUNI GENERALE

## Obiective,motivația alegerii temei și domeniile de aplicație

Prezenta lucrare are ca obiectiv principal realizarea cu ajutorul Raspberry Pi a unui dispozitiv autonom ce se deplasează pe baza semnelor de circulație pe o machetă ce simulează un drum real.

Motivația pentru alegerea acestui proiect de diplomă îsi are originea în aspirația de a studia și de a aprofunda domeniul inteligenței artificiale, în speță aria rețelelor neuronale, cât și evoluția exponențială a cercetărilor în zona self-drivingului.

Conducerea autonomă este unul dintre vastele domenii de aplicare ale inteligenței artificiale și unul dintre cele mai complexe proiecte derulate la momentul actual. Companiile care dezvoltă sisteme AV se bazează majoritar pe AI, sub forma de deep learning și machine learning, pentru a procesa o cantitate mare de date în mod eficient și pentru a antrena și valida sistemele de conducere autonomă. Companii precum: Telsa, Cruise, Alphabet Inc’s, Waymo sau Aurora Innovation Inc dezvoltă tehnologii ce permit automatizarea completă a vehiculelor.

Totodată, lucrarea de față vizează modalități de aplicare a modeului de Deep Learning (“învățare profundă”) pe dispozitive cu resurse și costuri reduse, precum placa de dezvoltare Raspberry Pi, demonstrând astfel utilizarea facilă, disponibilitatea de lucru și o performanță bună spre foarte bună.

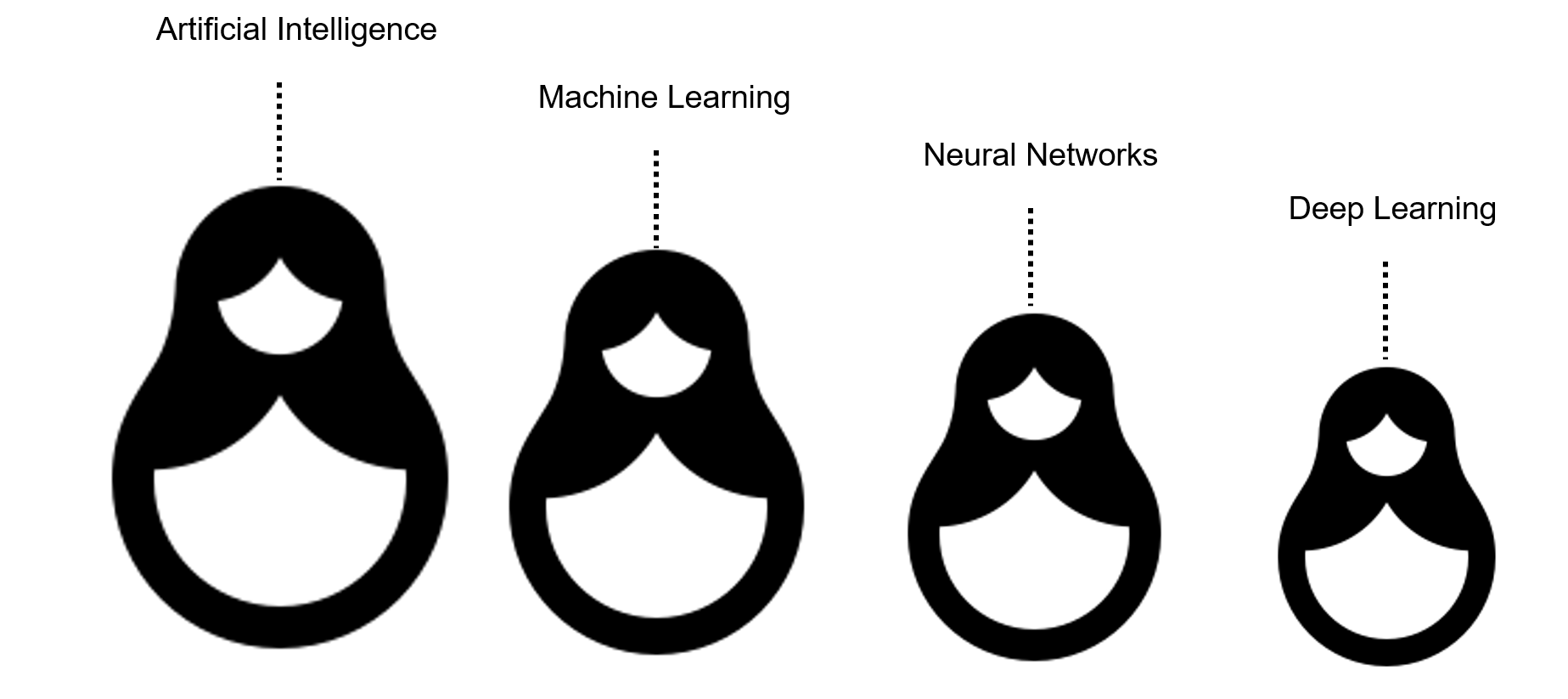
## Noțiuni generale

Siguranța este un aspect important al conducerii autonome. Factorul uman rămâne cea mai frecventă cauză a accidentelor rutiere. Astfel introducerea vechiculelor autonome va reduce considerabil o mare parte din aceste probleme, desigur cu un compromis reprezentat de o marjă infimă de risc. În cadrul dezvoltării vehiculelor autonome, în speță a sistemelor de asistență avansată în conducere (ADAS), inteligența artificială a câștigat mult teren, astfel cercetările s-au orientat în direcția exploatării informațiilor vizuale datorită utilității acesteia pentru detectarea semnelor rutiere, a drumurilor, a celorlalte vechicule și a pietonilor. Un mare interes îl prezintă sistemele de recunoaștere a semnelor rutiere care vizează detecția semnelor, interpretarea acestora și transmisia informațiilor către șofer sau în cazul dorit, transmisia informațiilor către vechicul care efectuează execuția fără să aibe nevoie de o decizie umană. Odată cu dezvoltarea domeniului Deep Learning, utlimii algoritmi și arhitecturi utilizate în dezvoltarea sistemelor de recunoaștere a semnelor rutiere vizează rețelele artificiale neuronale, precum SSD MobileNet V2 și Faster-RCNN abordate și în cadrul acestui proiect, acestea din urmă utilizează principiul rețelelor neuronale convoluționare.

În această lucrare, întregul sistem de recunoașterea a semnelor rutiere va fi împărțit într-o arii de lucru: dezvoltarea software, în speță zona ce presupune dezvoltarea modelului de recunoaștere a semnelor rutiere și implementarea hardware a sistemului mobil ce presupune asamblarea și programarea motoarelor, a senzorilor și a camerei video. Modelul de recunoaștere a semnelor rutiere este, de altfel, abordat în 2 etape, prima etapă o reprezintă detecția semnelor dintr-o imagine sau un frame al fluxului video, iar etapa a doua reprezintă procesul de clasificare în urma căruia semnul detectat în prima etapă este clasificat într-una din clasele de referință definite în dataset.

Deși inteligența artificială(AI), învățarea profundă (DL) și machine learning sunt folosite concomitent, acestea prezintă concepte și arii de interes diferite înrudite între ele. În cei mai simpli termeni, AI este o ramură a informaticii care acoperă procedeele de dezvoltare a dispozitivelor inteligente.

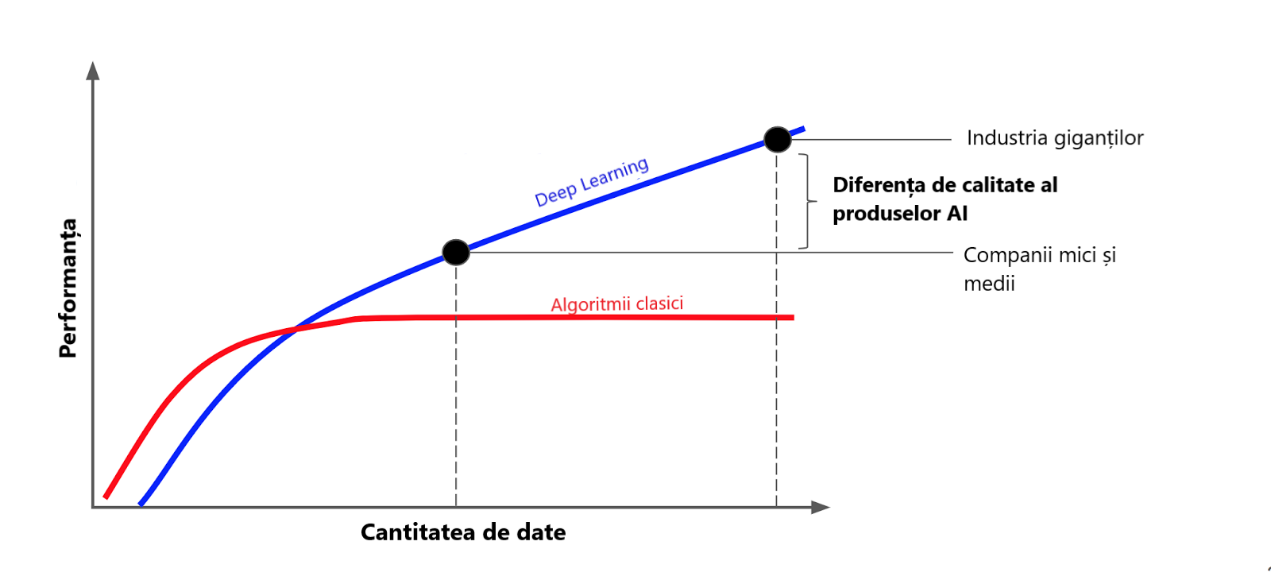
Probabil cel mai simplu mod de a ne gândi la inteligența artificială, Deep Learning și Machine Learning este, conform unui articol postat de IBM, vizualizarea ierarhică a acestora ca în cazul păpușilor rusești Matryoshka, fiecare fiind în esență o componentă a domeniului anterior.



Structura ierarhică a subdoemeniilor inteligenței artificiale [1]

Machine Learning este un subdomeniu al AI care oferă sistemelor capacitatea de a învăța și de a se îmbunătăți automat din experiență, fără a fi programate în mod explicit. Mașina este antrenată folosind cantități mari de date și algoritmi care îi oferă capacitatea de a învăța cum să îndeplinească sarcina. Abordările în domeniul Machine Learning includ învățarea arborelui de decizie, programarea logică inductivă, clustering, învățarea prin consolidare și rețelele bayesiene.

Deep Learning introduce o abordare extrem de sofisticată a învățării automate, fiind modelat pe baza rețelelor neuronale profunde. Acestea din urmă sunt inspirate din înțelegerea noastră a biologiei creierului uman cu toate interconexiunile între neuroni, dar spre desosebire de creierul uman în care orice neuron este conectat cu alt neuron, rețelele neuronale au straturi, conexiuni și direcții de propagare a datelor discrete. Astfel, DL a reprezentat un pas important în evoluția sistemelor automate și în general în domeniul inteligenței artificiale.



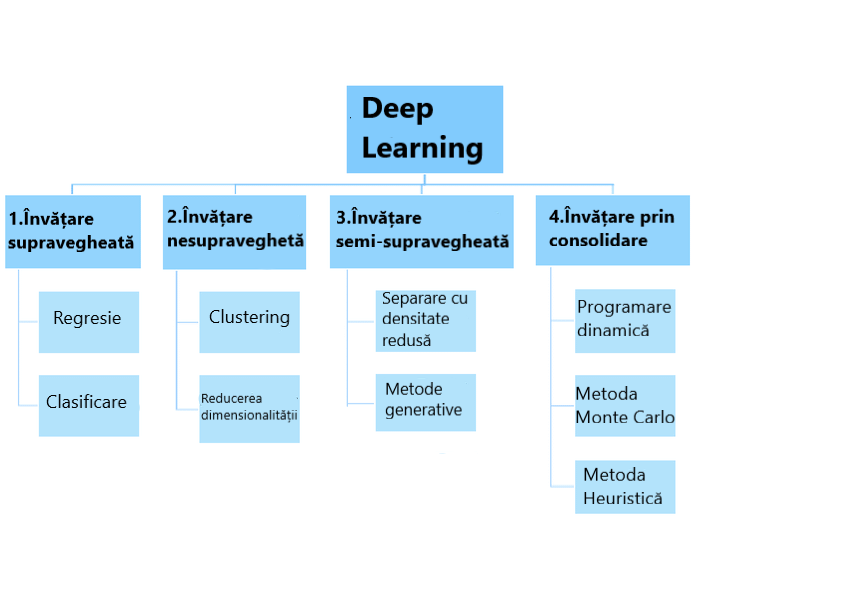
Performanțele DL în comparație cu abordările clasice ale ML

Graficul din figura 1.3 a fost realizat în urma unui studiu asupra comparației între metodele clasice de învățare în machine learning și domeniul deep learning.Se poate observa că pentru o cantitate mică de date algoritmii clasici prezintă o mai bună perfomanță în detrimentul deep learning, dar cum tehnologia evoluează exponențial și cantitatea de date utilizată crește proporțional, astfel domeniul de deep learning a câștigat mult teren în favoarea performanțelor pe care le acesta le oferă.

Deep Learning este o tehnologie cheie din spatele mașinilor fără șofer, permițându-le să recunoască un semn de oprire sau să distingă un pieton de un stâlp de iluminat. Reprezintă factorul esențial al controlului vocal în dispozitivele de consum, precum tabletele, telefoanele, televizoarele și difuzoarele hands-free.

În cadrul învățării profunde, un model de calculator învață să efectueze sarcini de clasificare direct din imagini, text sau sunet. Modelele de învățare profundă pot atinge o precizie uimitoare, depășind uneori performanța la nivel uman. Modelele sunt antrenate prin utilizarea unui set cât se poate de mare de date etichetate (dataset) și a unor arhitecturi de rețele neuronale care conțin mai multe straturi (layers) fără a fi necesară extragerea manuală a atributelor.[1]

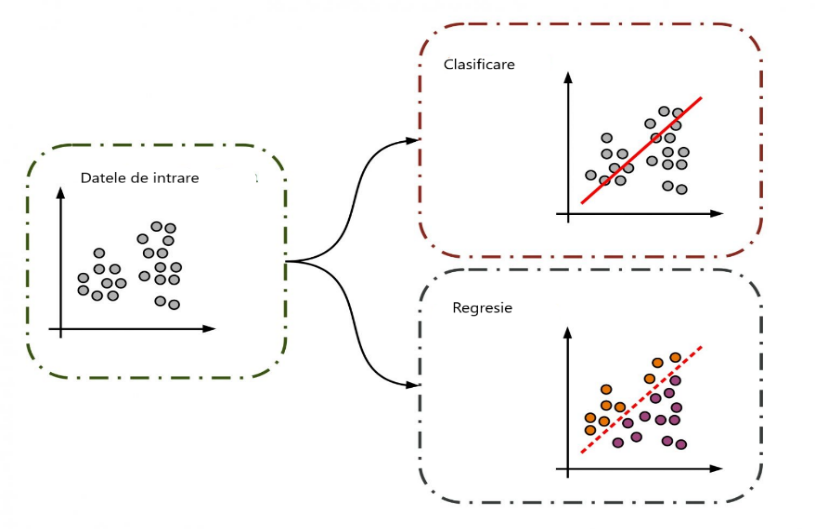
La momentul actual sunt cunoscute patru mari tehnici ale DL: învățare supravegheată (deep supervised learning), învățare nesupravegheată (deep unsupervised learning), învățarea semi-supraveghetă (deep semi-supervised learning) și învățare prin consolidare (deep reinforcement learning).



Structura tehnică a DL

### Deep Supervised Learning (DSL)

Învățarea supervizată este prezentă atunci când învățăm sau antrenăm mașina cu ajutorul unor date bine etichetate (labeled data), ceea ce înseamnă că anumite date sunt deja etichetate în mod corect. În învățarea supravegheată sunt variabilele de intrare (x) și variabila de ieșire (Y) și se utilizează un algoritm de învățare a funcției de mapare (Y=f(x)). Scopul final este aproximația cât mai bună a funcției de mapare, astfel încât la introducere de date noi de intrare (x), să se poată realiza prezicerea datei de ieșire (Y). Modelul este elaborat prin deducerea anumitor structuri existente în datele de intrare.



Metode de învățare supervizată

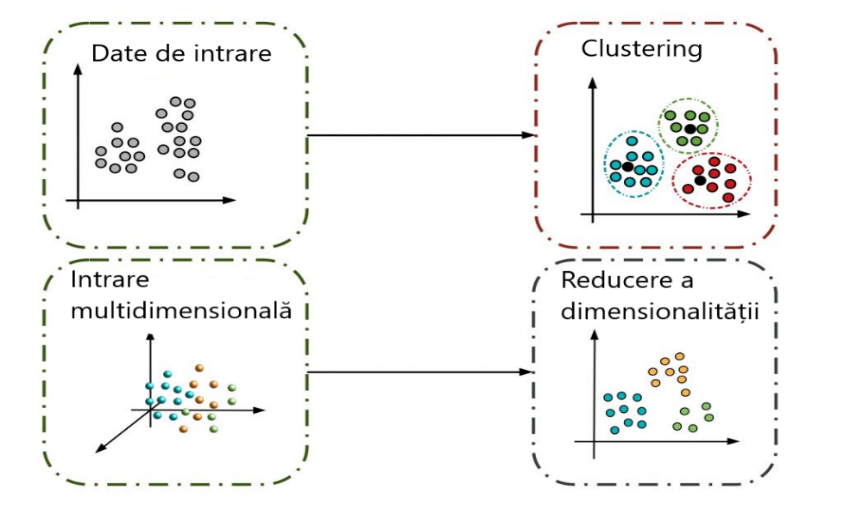
**Clasificarea** folosește un algoritm pentru a atribui cu precizie datele de testare în categorii specifice. Recunoaște anumite entități din setul de date și încearcă să tragă câteva concluzii cu privire la modul în care aceste entități ar trebui să fie etichetate sau definite.Exemple de metode de clasificare: clasificatorii liniari, mașina cu suport vectorial (SVM), arbori decizionali și pădure de arbori decizionali (random forest).

**Regresia** utilizează un algoritm pentru a înțelege relația dintre variabilele dependente și cele independente. Modelele de regresie sunt utile pentru prezicerea valorilor numerice pe baza diferitelor puncte de date. Unii algoritmi de regresie populari sunt regresia liniară, regresia logistică și regresia polinomială.

### Deep Unsupervised Learning (DUL)

Învățarea nesupervizată utilizează algoritmi de învățare automată pentru a analiza și a grupa seturi de date neetichetate. Acești algoritmi descoperă modele ascunse în date fără a fi nevoie de intervenția umană. Modelele de învățare nesupervizată sunt utilizate pentru trei scopuri esențiale: grupare(clustering), reducerea dimensionalității și învățarea regulilor de asociere. Deci sarcina mașinii este să grupeze informațiile nesortate în funcție de asemănări, modele și discrepanțe, fără nicio pregătire anterioară a datelor.

Exemple de algoritmi: algoritmul Apriori și k-Means.



Metode de învățare nesupraveghetă

**Clustering** reprezintă o modalitate de a extrage datele în scopul grupării acestora (neetichetate) pe baza asemănărilor sau diferențelor lor.

**Asocierea** utilizează reguli diferite pentru a găsi relații între variabilele dintr-un set de date prestabilit.

**Reducerea dimensionalitățiilor** este o tehnică de învățare atunci când numarul de dimensiuni dintr-un anumit set de date este prea mare. Reduce numărul de intrări de date la o dimensiune rezonabilă, păstrând integritatea lor.

### Deep Semi-Supervised Learning (DSSL)

În cazul învățării semi-supervizate datele de intrare reprezintă o mixare de date etichetate și neetichetate. În cazul acestui tip de învățare, există o anumită deficiență în ce privește predicția, însă modelul trebuie să învețe structura existentă în date pentru a le putea grupa și a realiza predicții.

## Abordarea Deep Learning

Deși învățarea profundă a fost teoretizată pentru prima dată în anii 1980, există două motive principale pentru care a devenit utilă abia în ultimii ani și anume:

* Învățarea profundă necesită cantități mari de date etichetate. De exemplu, progesul dezvoltării mașinilor autonome implică milioane de imagini și mii de ore de înregistrări video a traficului, a semnelor de circulație și tot ce implică interacțiunea umană cu circumstanțele din timpul șofatului.
* Învățarea profundă necesită o putere de calcul substanțială. Astfel, GPU-urile de înaltă performanță reprezintă o soluție fezabilă, acestea având o arhitectură paralelă ce poate realiza un număr considerabil de calcule simultan, fiind eficientă pentru învățarea profundă. Atunci când este combinată cu clusterele sau cu cloud computing-ul, aceasta permite echipelor de dezvoltare să reducă timpul de antrenare pentru o rețea de învățare profundă de la săptămâni la ore sau mai puțin, optimizând astfel sarcinile urmărite în acest domeniu de actualitate.

## Provocări impuse de Deep Learning

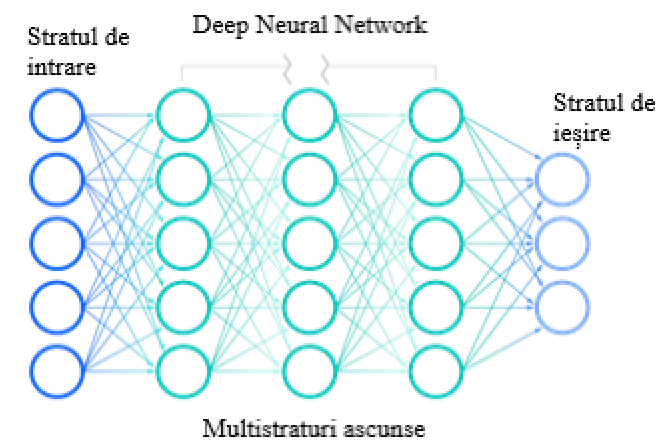
Rețelele de DL pot fi aplicate cu succes unor cantități mari de date în scopul descoperirii de noi informații, aplicarea informațiilor deja cunoscute cât și predicția bazată pe informații (pentru a dezvolta noi structuri). Altfel spus, DL poate fi un motor puternic pentru obținerea unor rezultate care pot fi puse în practică.[2]

Principalele avantaje ale utilizării DL:

* Utilizarea optimă a datelor nestructurate: procente uriașe din cantitatea de date ale unei organizații sunt nestructurate, deoarece o bună parte din acestea există în diferite tipuri de formate, cum ar fi texte, imagini, audio, video etc.
* Eliminarea nevoii de inginerie a caracteristicilor (feature engineering): ingineria caracteristicilor reprezintă procesul de utilizare a informațiilor deja cunoscute dintr-un domeniu pentru a selecta și transforma cele mai relevante variabile din datele brute.
* Capacitatea de a furniza rezultatele cu o acuratețe mare: un model de DL este capabil să îndeplinească mii de sarcini de rutină și repetitive într-o perioadă de timp mai scurtă.
* Reducerea costurilor inutile.
* Eliminarea necesității de etichetare a datelor: nevoia de date etichetate corespunzatoare se reduce, deoarece algoritmii excelează la învățarea fără supervizare.[2]
* Arhitectura DL este flexibilă, astfel încât sa poată fi adaptată la noi probleme în viitor.

## Deep Learning și Machine Learning. Diferențe.

Termenul “deep”din Deep Learning se referă la adâncimea straturilor dintr-o rețea neuronală. O rețea neuronală care constă din mai mult de trei straturi (care ar include intrările și ieșirile) poate fi considerată un algoritm DL. Acest lucru este reprezentat, cu ajutorul următoarei diagrame:

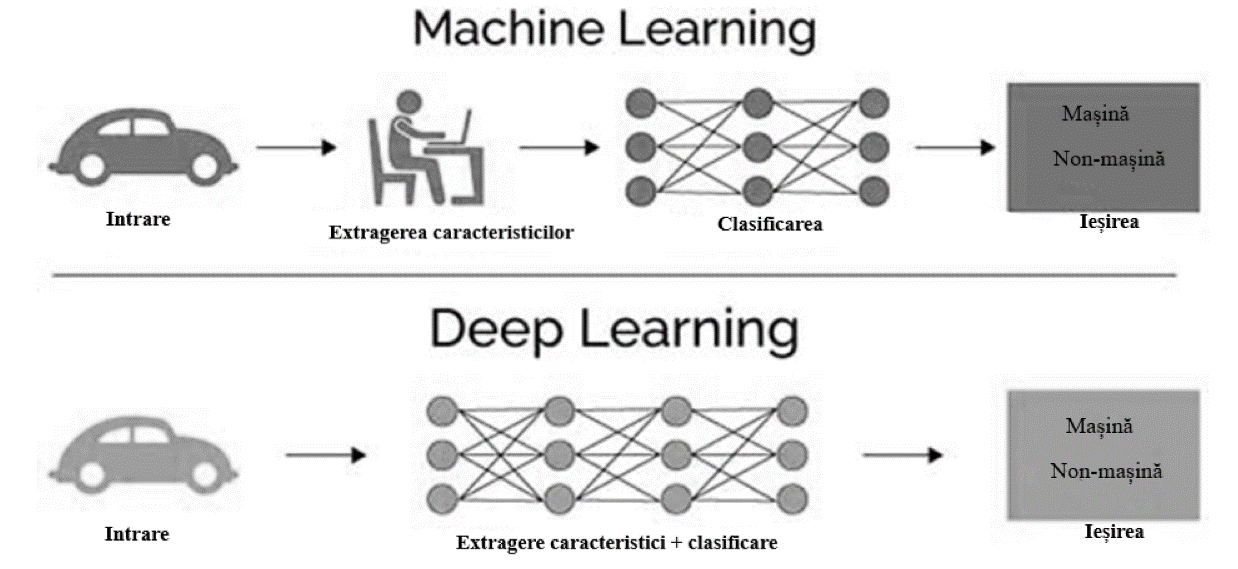


Rețea neuronală cu straturi multiple

Din figura se observă că fiecare componentă a unui strat este interconectată cu toate componentele stratului ulterior.

Machine Learning sau învățarea automată a fost definită în anii 1950 de către pionierul AI Arthur Samuel ca fiind “domeniul de studiu care oferă computerelor capacitatea de a învăța fără a fi programate în mod explicit.”.

* + Principala diferență dintre DL și ML constă în modul în care fiecare algoritm al acestor domenii învață, dupa cum este exemplificat și în figura, și în cantitatea de date pe care fiecare tip de algoritm o utilizează:
* DL automatizează o mare parte din procesul de extragere a caracteristicilor, reducând intervenția umană manuală necesară. De asemenea, permite utilizarea unor seturi mari de date, ceea ce i-a adus titlul de "învățare mecanică scalabilă".
* ML sau învățarea "non-profundă" depinde mai mult de intervenția umană pentru a învăța. Experții umani determină ierarhia caracteristicilor pentru a înțelege diferențele dintre intrările de date, fiind nevoie de mai multe date structurate pentru a învăța.[3]



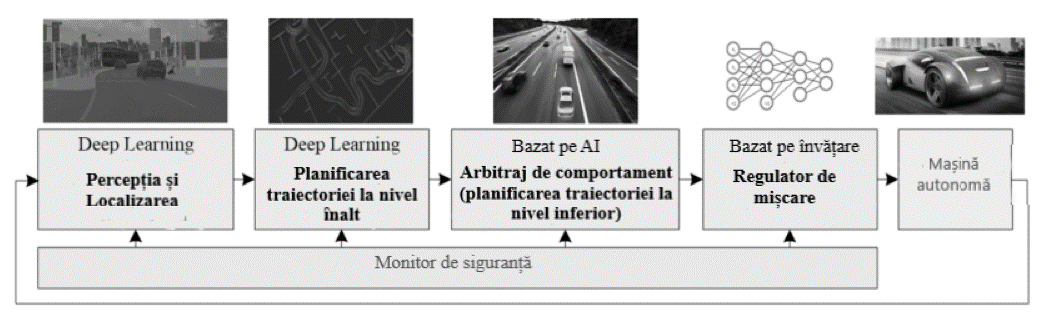
Deep Learning/Macine Learning

* + "Deep" Machine Learning poate utiliza seturile de date etichetate pentru a-și informa algoritmul, dar nu necesită neapărat un set de date etichetate. Aceasta poate asimila date nestructurate în forma lor brută (de exemplu: text, imagini) și poate determina automat setul de caracteristici care disting diverse clase.[4]
  + Prin observarea modelelor din date, un model de Deep Learning poate grupa intrările în mod corespunzător.Astfel, un model DL are nevoie de mai multe date pentru o îmbunătățire a acurateței, în timp ce un model ML se bazează pe mai puține date, având în vedere structura de bază a datelor.[3]
  + Algoritmii ML tind să fie mai puțin complecși decât algoritmii DL și pot fi rulați adesea pe computere convenționale, însă sistemele DL necesită resurse hardware mult mai puternice. Această cerere de putere a determinat o utilizare sporită a unităților de procesare grafică. GPU-urile sunt utile în cadrul aplicațiilor de AI, deoarece au lățime de bandă mare și capacitatea de a reduce latența (întârzierile) în transferul de memorie datorită paralelismului de tip thread (capacitatea mai multor operații de a rula eficient simultan).
  + ML este deja utilizată în serviciul de e-mail, în cadrul băncilor și în medicină. Tehnologia DL permite programe mai complexe și urmărește automatizarea acestora, cum ar fi mașinile autonome sau roboții care efectuează operații chirurgicale avansate.[5]

## Deep Learning în contextul automatizării autovehiculelor

Prima mașină parțial autonomă a fost inventată în 1989, fiind vorba despre vehiculul terestru automat în rețea neuronală (ALVINN). Acesta folosea rețele neuronale pentru a detecta liniile, a segmenta mediul, a se autodetecta și a conduce. A funcționat bine, dar a fost limitată de puterea de procesare lentă și de datele insuficiente la vremea respectivă.

Mașinile fără șofer sunt sisteme autonome de luare a deciziilor. Acestea pot procesa fluxuri de date provenite de la diverși senzori, cum ar fi camere, LiDAR, RADAR, GPS sau senzori de inerție. Aceste date sunt apoi modelate cu ajutorul unor algoritmi DL, care iau apoi decizii relevante pentru mediul în care se află mașina. În figura este ilustrată structura evenimentelor care se urmăresc în realizarea unor sisteme de deplasare complet autonome.



un pipeline modular de percepție-planificare-acțiune de funcționare a mașinilor autonome[6]

Pentru a înțelege cum funcționează mașinile autonome, trebuie să examinăm următoarele direcții:

* Percepția
  + Localizarea
  + Predicția
  + Luarea deciziilor
* Planificarea traiectoriei la nivel înalt
* Arbitrajul de comportament
* Regulator de mișcare

Detecția obiectelor este o problemă bine cunoscută în domeniul Computer Vision (CV) și al Deep Learning-ului. Există două componente într-un model de detecție a obiectelor, și anume, rețeaua neuronală de bază și rețeaua neuronală de detectare.

### Percepția

Percepția ajută mașina “să observe” lumea înconjurătoare, precum și să recunoască și să clasifice lucrurile pe care le vede. Pentru a lua decizii bune, mașina trebuie să recunoască instantaneu obiectele, un aspect ce impune provocăroi uriașe în asigurarea detecției real-time..

Detecția obiectelor a fost și este o problemă bine cunoscută în domeniul computer vision (CV) și al Deep Learning-ului. Există două componente într-un model de detectare a obiectelor, și anume, rețeaua neuronală de bază și rețeaua neuronală de detectare.

Astfel, mașina trebuie să vadă și să clasifice semafoarele, pietonii, semnele de circulație, trotuarele, locurile de parcare, benzile de circulație și multe altele. Totodată, trebuie să știe și distanța exactă dintre ea și obiectele din jur. Percepția este mai mult decât simpla vedere și clasificare, ea permite sistemului să evalueze distanța și să decidă fie dacă să încetinească, fie să frâneze.

Pentru a obține un nivel înalt al percepției, o mașină autonomă trebuie sa aibe în dotare următorii senzori:

* *Camera*: permite mai multe sarcini precum clasificarea, segmentarea și localizarea
* *LiDAR*: o metodă de măsurare a distanței dintre obiecte prin emiterea unui fascicul laser pentru a fi reflectat de obstacol
* *RADAR*: calculează distanța folosind unde radio, care pot funcționa în orice condiții față de fasciculele laser.[7]

### Localizarea

Algoritmii de localizare folosiți pentru mașinile autonome calculează poziția și orientarea vehiculului în timp ce acesta navighează (o știință cunoscută sub numele de odometrie vizuală, VO).

DL este utilizată în general pentru a îmbunătăți performanța VO și pentru a clasifica diferite obiecte. Rețelele neuronale, cum ar fi PoseNet și VLocNet++, sunt unele dintre modelele care utilizează date punctuale pentru a estima poziția și orientarea 3D.

### Predicția

Înțelegerea conducătorilor auto umani este o sarcină foarte complexă. Aceasta implică mai degrabă emoții decât logică, iar acestea sunt alimentate de reacții. Devine foarte nesigur care va fi următoarea acțiune a șoferilor sau a pietonilor din apropiere, astfel încât un sistem care poate prezice acțiunile altor utilizatori ai drumurilor poate fi foarte important pentru siguranța rutieră.

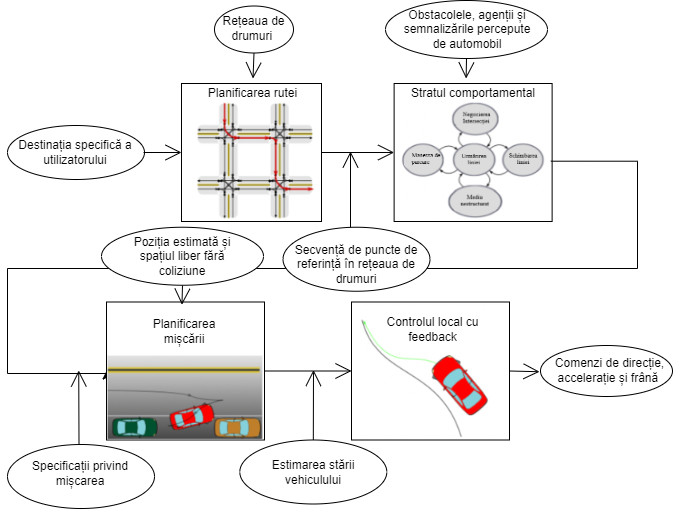
Automobilul are o vedere de 360 de grade asupra mediului înconjurător care îi permite să perceapă și să capteze toate informațiile și să le proceseze. Odată introdus în algoritmul DL, acesta poate să prezinte toate mișcările posibile pe care le-ar putea face ceilalți participanți la trafic.

Rolul DL este de a interpreta sarcinile complexe de viziune, de a se localiza în mediul înconjurător, de a îmbunătăți percepția și de a acționa manevrele cinematice în mașinile care se conduc singure.

### Luarea deciziilor

Procesul decizional este vital în cazul mașinilor autonome. Acestea au nevoie de un sistem care să fie dinamic și precis într-un mediu incert. Acesta trebuie să țină cont de faptul că nu toate datele preluate de la senzori vor fi adevărate și că oamenii pot face alegeri imprevizibile în timpul conducerii.[8]

În cazul luării deciziilor, se utilizează Deep Reinforcement Learning (DRL). Mai concret, în centrul DRL se află un algoritm de luare a deciziilor numit proces decizional Markov (MDP), utilizat pentru a prezice comportamentul viitor al șoferilor.



*Etapele parcurse de vehicul în luarea unei decizii*

Conform figurii , în luarea deciziilor se țin cont de următoarele:

* *Planificarea traseului sau a rutei*: automobilul trebuie să planifice cel mai bun traseu posibil de la poziția sa curentă la destinația solicitată.
* *Arbitrajul comportamentului*: incertitudine în ceea ce privește acțiunile celorlalți șoferi este rezolvată prin utilizarea algoritmilor de planificare probabilistică, cum ar fi MDP.
* *Planificarea mișcării*: include viteza vehiculului, schimbarea benzii de rulare și multe altele.
* *Controlul vehiculului*: este utilizat pentru a realiza traiectoria de referință din sistemul de planificare a mișcării.

# NOȚIUNI TEORETICE

## Definiții

*Rețelele neuronale artificiale (ANN)* sunt cuprinse în Deep Learning cu o structură asemănătoare organizării creierului uman. Rețelele neuronale sunt concepute pentru a recunoaște modele în date complexe și, adesea, funcționează cel mai bine atunci când recunosc tipare în audio, imagini sau video

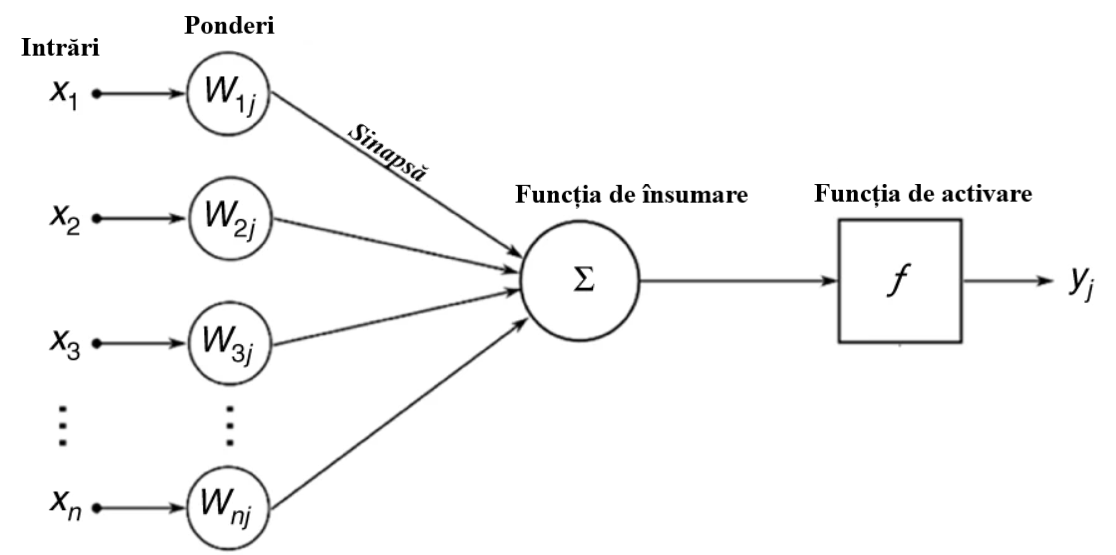
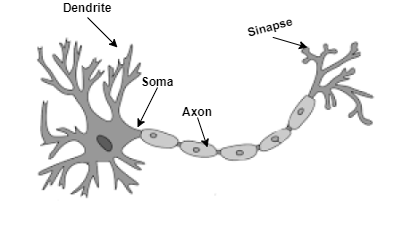
ANN sunt caracterizate astfel:

* sunt compuse din straturi de noduri;
* fiecare nod este conceput în a se comporta similar cu neuronii prezenți în creierul uman;
* primul start al rețelei neuronale este numit stratul de intrare (“input layer”), urmat de straturile ascunse (“hidden layers”) și în cel din urmă stratule de ieșire;
* fiecare nod al rețelei efectuează un anumit calcul, urmând să fie pasat către celelalte noduri mai profund în rețea.

Comparația tehnicilor de ML cu structura creierului uman a fost abordată prima dată de Geoffrey Hinton in anul 1980. Abordarea acestuia a plecat de la conceptul de “rețea neuronală”, deoarece conform viziunii acestuia creierul uman este cel mai puternic calculator computațional cunoscut pâna la vremea respectivă, chiar până la momentul actual.[9]

Elementul fundamental de procesare al unei rețele neuronale îl reprezintă *neuronul*. Acest element constructiv al rețelei neuronale umane cuprinde câteva capacități generale. Practic, un neuron biologic primește intrări din alte surse, le combină conform unui algoritm variabil, adaptat situației și efectuează o operație în general neliniară asupra rezultatului și în cele din urmă emite rezultatul final.

Neuronul artificial poartă denumirea de *nod*, iar structura acestuia este reprezentată în figura ??? ,fiind inspirată de structura neuronilor din creierul uman.



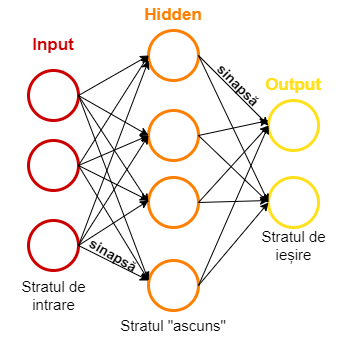
*Neuron biologic Neuron artificial*

Întrebarea ridicată de Georffrey Hinton de-a lungul seminarului său de cercetare în zona rețelelor neuronale a vizat capcitatea de construire a unor algoritmi de calcul care să se comporte similar cu neuronii biologici. Astfel, cercetătorii au studiat modul în care neuronii biologici acționează și principalul aspect interpretat din cercetări susține că neuronul de unul singur nu prezintă nici o funcționalitate si doar angrenat într-o rețea acesta poate genera capabilități uimitoare. Acest lucru se întâmplă deoarece neuronii funcționează prin emisa și recepția de semnale, mai specific conform figurii **???** neuronii primesc semnale prin dendrite si le transmit mai departe prin intermediul axonului, iar dendritele fiecărui neuron sunt conectate la axonul altuia, aceste conexiuni numindu-se sinapse, un termen utilizat adesea în câmpul DL.

În aria DL neuronii sunt noduri prin care trec datele și operațiile computaționale, astfel fiecare nod înglobeaza o anumită valoare, iar fiecare conexiune (sinapsă) o anumită *pondere.*

*Neuronii artificiali* funcționează în felul următor conform figurii ???:

* primesc unul sau mai multe semnale . Aceste semnale provin din datasetul de intrare sau de la neuronii poziționați pe stratul anterior din rețea;
* realizează anumite calcule computaționale;
* trimit semnalele de ieșire către nueronii aflați mai adânc în rețea prin intermediul sinapselor;
* fiecare sinapsă are asociată o *pondere*, care determină gradul de importanță al neuronului precedent în rețea;
* odată ce neuronul primește datele de la neuronii din stratul precedent, adună fiecare semnal multiplicat cu ponderea specifică și le trece prin *funcția de activare*;
* funcția de activare calculează valoarea de ieșire a neuronului, această valoare urmând să fie trecută în stratul următor prin alte sinapse.



Arhitectura rețelei neuronale prevede în compunerea acesteia 3 tipuri de straturi de noduri: *input layer (stratul de intrare), output layer (stratul de ieșire) și hidden layer (stratul “ascuns”).*.[10]

*Stratul de intrare* (input layer) are scopul de a prelua datele de intrare. Astfel, numărul nodurilor de intrare este egal cu numărul de variabile explicative, care sunt transmise către unul sau mai multe straturi “ascunse”. Nodurile stratului de intrare sunt pasive , însemnând că acestea nu vor modifica datele, le vor primi și le vor duplica către către straturile “ascunse”. Astfel, nodurile din stratul de intrare funționează precum *un hub*, echipament de repetare a semnalului de intrare pe mai multe ieșiri, din domeniul comunicațiilor.

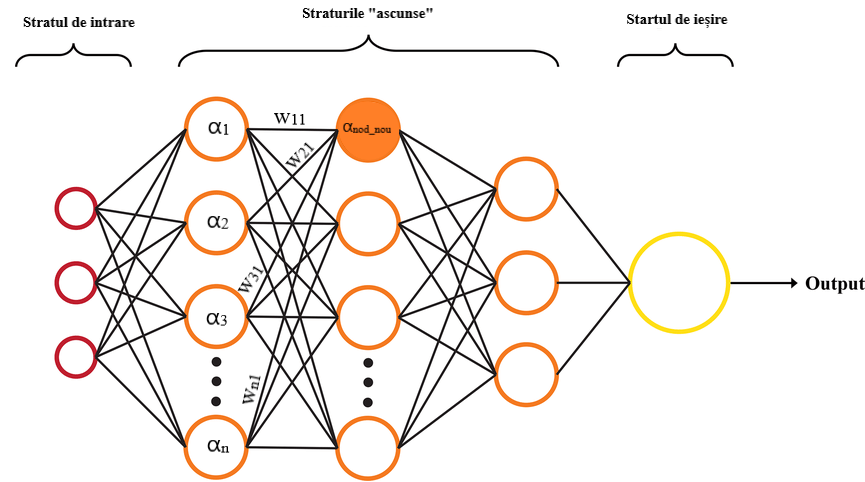
*Stratul “ascuns”*(hidden layer) aplică trasnformări datelor de intrare în interiorul rețelei. În stratul “ascuns” procesarea efectivă se realizează prin intermediul unui sistem de “conexiuni” ponderate, numite *sinapse*. Într-o rețea neuronală pot exista unul sau mai multe starturi “ascunse”. Valorile care intră într-un strat “ascuns” sunt multiplicate cu ponderile, care vor fi adunate pentru a produce un singur număr.

*Stratul de ieșire* (output layer) primește conexiunile de la stratul ”ascuns” sau de la stratul de intrare și returnează o valoare de ieșire care corespunde predicției variabilei de răspuns. De exemplu, în clasificare este doar un singur astfel de nod de ieșire. Nodurile active ale stratului de ieșire combină și modifică datele pentru a putea determina valorile de ieșire.

În ceea ce privește arhitectura rețelelor neuronale, acestea sunt împărțite în două mari categorii: rețele neuronale exclusiv feed-forward (propagare directă) și rețelele neuronale backward (cu propagare inversă a erorii).

O rețea neuronală de tip *feedforward* (doar cu propagare directă a informației) este o rețea neuronală artificială în care nodurile nu formează niciodată un ciclu. Acest tip de rețea neuronală are un strat de intrare, straturi “ascunse” și un strat de ieșire. Este primul și cel mai simplu tip de rețea neuronală artificială.[11]

Fiecare nod are o valoare de activare α cu valori între 0 și 1 și fiecare conexiune care leagă două noduri are o anumită pondere w. Se multiplică valorile de activare cu ponderile și astfel se obține un nod nou în stratul următor, precum este reprezentat și în figura ????



Rețea neuronală cu propagare directă

Conform figurii ??? , pentru exemplificare, valoarea noului nod într-o rețea de tip feedforward este dată de formula :

|  |  |  |
| --- | --- | --- |
|  |  |  |

Generalizând relația de mai sus, pentru a înainta în rețea se va folosi formula ??? pentu a calcula fiecare neuron al stratului următor:

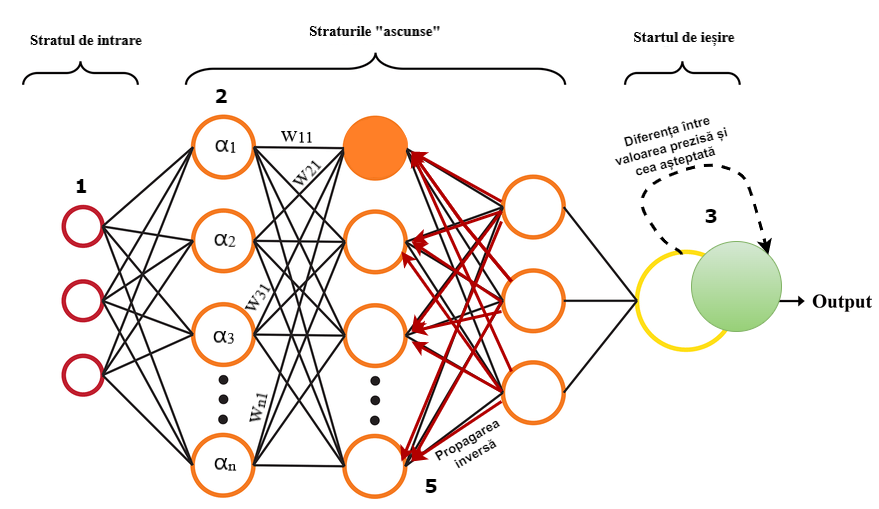
|  |  |  |
| --- | --- | --- |
|  |  |  |

,unde 

Scopul unei rețele neuronale artificiale este de minimiza valoarea funcției de cost, astfel încât valoarea prezisă de algoritm să fie cât mai apropiată de valoarea așteptată.

Pentru a crește acuratețea predicției este necesară o ajustare a ponderilor, astfel încât să optimizăm funcția de cost, iar de acest aspect se ocupă *rețelele cu propagare inversă*.

*Rețelele cu propagare inversă* reprezintă chintesența antrenării rețelelor neuronale, fiind o abordare standard a rețelelor neuronale artificiale. Propagarea inversă se referă la procedeul de reglare fină a ponderilor rețelei pe baza ratei de eronare obținută în epoca anterioară. Mai concret, această metodă permite calcularea gradientului funcției de pierdere raportat la ponderile vehiculate în rețea.[11]



*Rețea neuronală cu propagare inversă a erorii*

Modul de funcționare al rețelei cu propagare inversă:

1. Datele de intrare ajung în rețea prin intermediul stratului de intrare.
2. Intrarea este modelată utilizând ponderi reale. Ponderile sunt adesea selectate random în primă instanță.
3. Calcularea valorii de ieșire pentru fiecare nod de la stratul de intrare, prin stratul “ascuns”, până la stratul de ieșire.
4. Calculul erorii datelor de ieșire.

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. Întoarcerea din stratul de ieșire în stratul “ascuns” cu scopul de a ajusta ponderile astfel încât eroarea să fie redusă.
2. Pentru a estima acuratețea valorii de ieșire , se determină funcția de cost C, fiind adesea *Eroarea Medie Pătratică* (MSE):

|  |  |  |
| --- | --- | --- |
|  |  |  |

Pentru a realiza actualizarea ponderilor de care am amintit mai sus, trebuie să calculăm *gradienții*, care reprezintă rata de variație la un anumit punct pentru fiecare pondere a fiecărui strat.

*„Gradientul este generalizarea derivatei la funcțiile multivariate. Acesta surprinde panta locală a funcției, permițându-ne să prezicem efectul efectuării unui pas mic de la un punct în orice direcție.”*[12]

Gradientul este calculat conform formulei ???:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Parcurgem fiecare pondere, de exemplu din stratul de ieșire, și scădem valoarea *ratei de învățare (LR)* multiplicată cu *costul* acelei ponderi, din valoarea inițială pe care o avea ponderea conform formulei ????.

|  |  |  |
| --- | --- | --- |
|  |  |  |

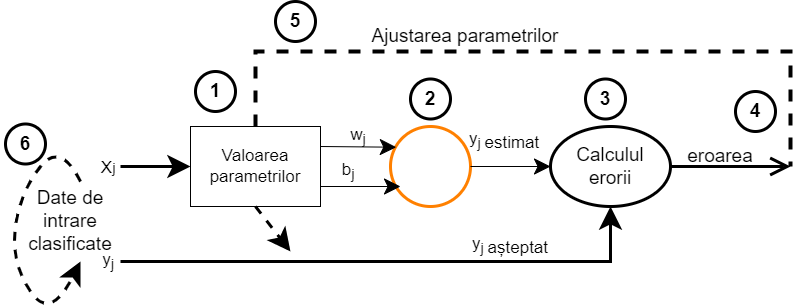
## Procesul de învățare

Procesul de învățare într-o rețea neuronală este strâns legat de modul în care învățăm în viața și activitățile obișnuite: efectuăm o acțiune și suntem fie recompensați, fie corectați de o persoană care ne supraveghează activitatea pentru a înțelege cum să devenim mai buni într-o anumită sarcină. În mod similar, rețelele neuronale au nevoie de un formator pentru a descrie ceea ce ar fi trebuit să fie produs ca răspuns la datele de intrare. Pe baza diferenței dintre valoarea reală și valoarea care a fost prezisă de rețea, se calculează o *valoare de eroare* care este trimisă înapoi prin sistem. Pentru fiecare strat al rețelei, valoarea de eroare este analizată și utilizată pentru a ajusta pragul și ponderile pentru următoarea intrare. În acest fel, eroarea continuă să devină mai mică la fiecare rulare, pe măsură ce rețeaua învață cum să analizeze valorile.

Procedeul poartă denumirea de propagare inversă și a fost descris în subcapitolul anterior.

Esențial în procesul de învățare al rețelei este *funcția de activare* asociată fiecărui neuron, acesta introduce *neliniaritate* în procedeele de modelare ale rețelei.

Astfel, procesul de învățare presupune parcurgerea următoarelor etape:



1. Începe cu valorile parametrilor rețelei (ponderile și bias-urile) alese *aleator*.
2. Preia un set de date de intrare în funcție de dimensiunea lotului care parcurg rețeaua în scopul obținerii *predicției*.
3. Compară predicția cu valoarea așteptată și pe baza acestei comparații calculează *eroarea*.
4. Efectuează propagarea inversă în scopul minimizării costului pentru optimizarea eodelului neuronal.
5. Utilizează informația propagată pentru *a ajusta parametrii rețelei neuronale* cu ajutorul *gradientului descendent* în scopul reducerii pierderii totale și a obținerii unui model mai bun.
6. Contiunarea iterării pașilor de mai sus până la obținerea unor predicții cu o acuratețe bună.

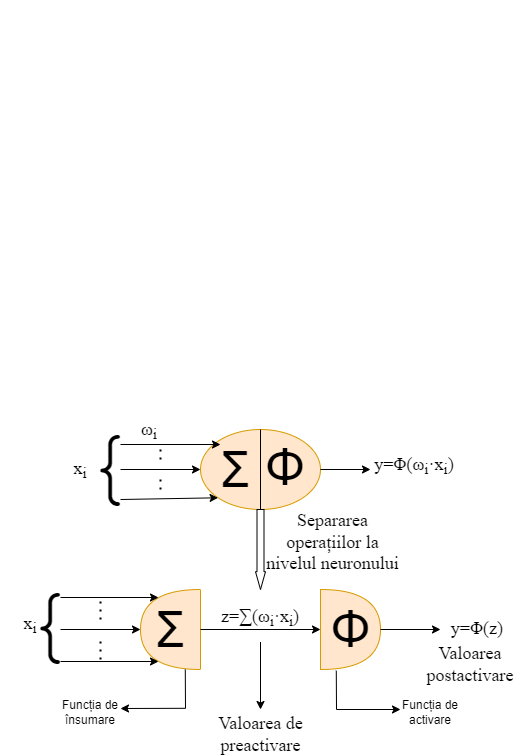
### Funcția de activare

Cum am explicat și în subcapitolele anterioare, neuronii primesc semnale de intrare de la stratul precedent al unei rețele neuronale. Suma ponderată a acestor semnale este introdusă în funcția de activare a neuronului, apoi valoarea de ieșire a funcției este transmisă către următorul strat al rețelei.

Alegerea funcției de activare este o parte critică a proiectării rețelelor neuronale.

De exemplu, în cazul în care variabila *target* (termen de identificarepentru variabila de ieșire în contextul DL) care trebuie să fie prezisă este reală, atunci se utilizează funcția de activare liniară, iar algoritmul rezultat este același cu cel al regresiei prin metoda celor mai mici pătrate. În cazul în care trebuie prezisă o probabilitate a unei clase binare, se utilizează funcția Sigmoid pentru activarea nodului de ieșire, astfel încât predicția  să indice probabilitatea că valoarea observată, y, a variabilei dependente este 1.[13]

La nivelul neuronului sunt efectuate două funcții computaționale: însumarea valorilor neuronilor anteriori și funcția de activare. În figura ??? este prezentată descompunerea calculelor neuronlui în valori separate.



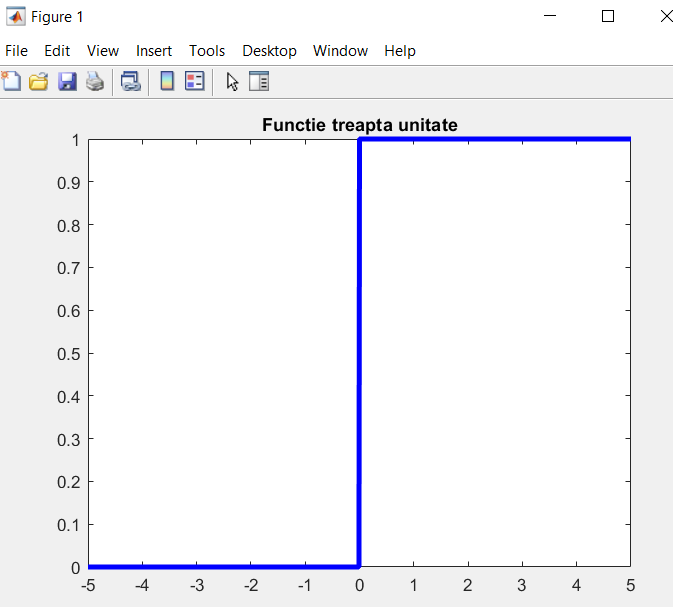
Separarea operatiilor realizate la nivelul neuronului

Există cinci tipuri principale de funcții de activare:

* Funcția Prag;
* Funcția liniară sau funcția identitate (singura funcție de activare liniară);
* Funcția Sigmoid sau funcția de activare logistică;
* Funcția ReLU (“Rectifier Linear Unit”);
* Funcția Tanh (tangentă hiperbolică)

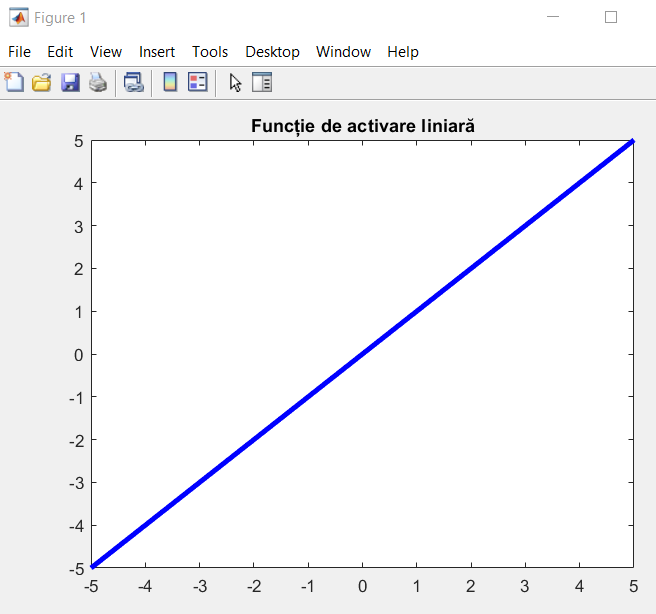
***Funcția prag***calculează valoarea de ieșire în funcție de condiția dacă valoarea de intrare se află peste sau sub un anumit prag*,* unde valoarea d eintrare este suma ponderată a valorilor de intrare din stratul precedent. Funcția prag mai poartă denumirea de *funcția treaptă unitate,* funcție adesea întâlnită în domeniul prelucrării semnalelor*.*

|  |  |  |
| --- | --- | --- |
|  |  |  |



***Funcția de activare liniară*** este practic funcția de identitate în care valoarea de intrare nu suferă nici o modificare.

|  |  |  |
| --- | --- | --- |
|  |  |  |

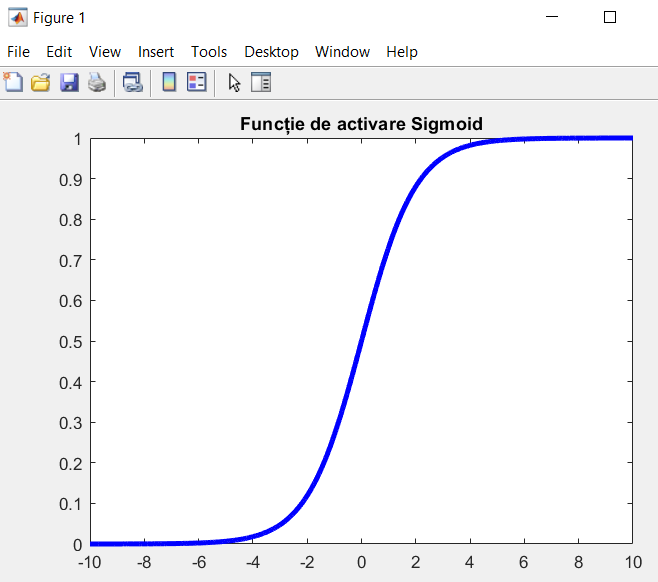


Derivata funcției de activare liniară este 1 în toate locurile.

***Funcția Sigmoid***  este bine cunoscută în zona data science datorită utilizării acesteia în regresia logistică, una dintre principalele tehnici ML utilizate pentru problemele de clasificare. Funcția sigmoid poate primi orice valoare de intrare, dar întotdeauna va emite o valoare între 0 și 1.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Un avantaj al funcției Sigmoid față de funcția prag dat de curba acestia, care permite calculul derivatei în orice punct de-a lungul curbei.

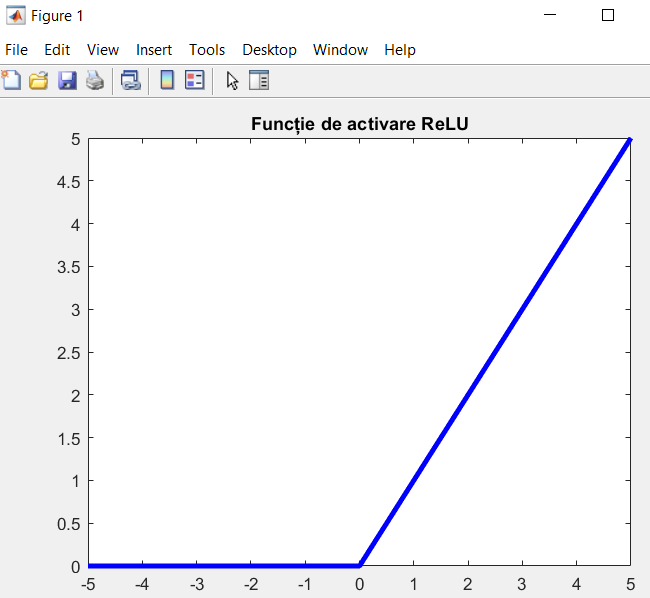


***Funcția ReLU*** nu prezintă aceeași proprietate de netezime precum funcția sigmoidă din ultima secțiune. Cu toate acestea,funcția de rectificare este încă foarte populară în domeniul DL.

Funcția de rectificare se definește după cum urmează:

* Dacă valoarea de intrare este mai mică decât 0, atunci funcția are la ieșire 0.
* În caz contrar, ieșirea este o relație liniară cu variabila de intrare de forma f(x)=x.

|  |  |  |
| --- | --- | --- |
|  |  |  |

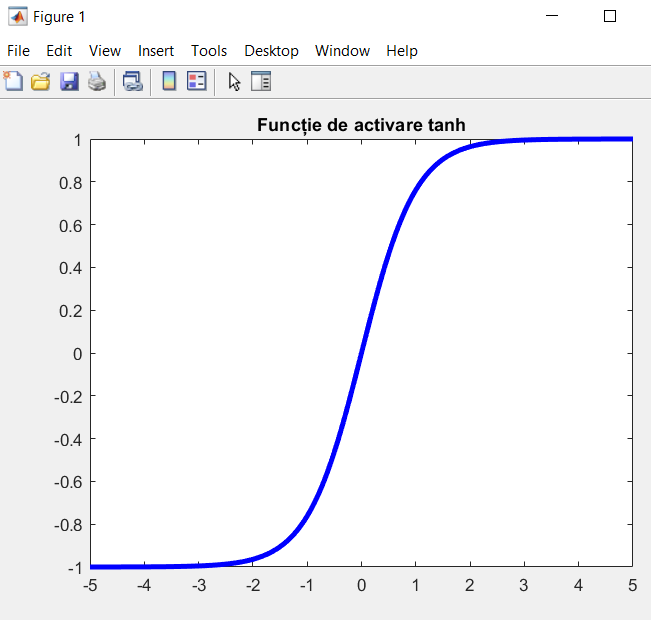


***Funcția de activare Tanh*** este de asemenea o funcție logistică, asemănătoare funcției Sigmoid, dar cu o rezoluție mai bună. Intervalul funției tahn este (-1,1), spre deosebire de funcția Sigmoid cu intervalul (0,1).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Avantajul utilizării funcției tangent hiperbolică constă în faptul că intrările negative vor fi mapate puternic negative, iar intrările de zero vor fi mapate aproape de zero în graficul tanh, facându-se diferența între acestea.

Funcția este diferențială și monotonă, pe când derivata acesteia nu este, drept dovadă este utilizată în rețelele de tip feedforward.

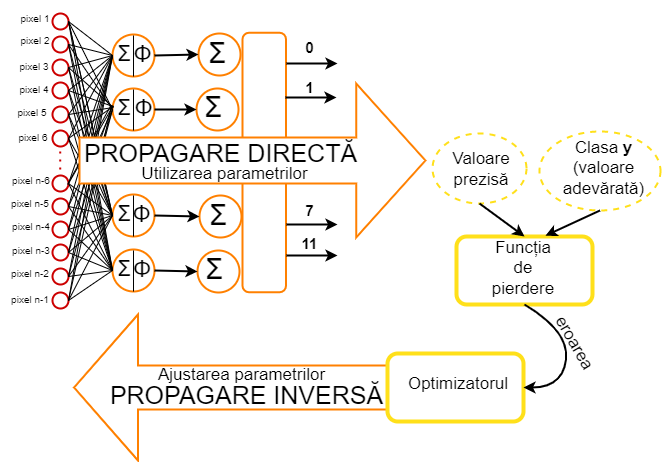


Funcția tanh este utilizată în principal pentru clasificarea intr-un sistem cu două clase.

### Antrenare

Antrenarea unei rețele neuronale presupune procesul ajustare si de învățare a valorilor parametrilor: ponderilor (ωij) și a bias-urilor (bj). Antrenarea poate fi văzută ca un proces iterativ de “înaintare și întoarcere” de-a lungul straturilor rețelei neuronale, determinat de cele două etape ale antrenării: propagarea directă prin rețea și propagarea inversă.

Vizual, putem sumariza cele spuse mai sus prin următoarea diagramă:



### Funcția de pierdere

În primul rând vom clarifica diferența între funcția de pierdere și funcția de cost, care sunt utilizate în același context, deși sunt diferite. ***Funcția de pierdere***  este asociată cu fiecare eșantion din datasetul de antrenare, pe când ***funcția de cost***  reprezintă valoarea medie a pierderii tuturor eșantioanelor. În domeniul DL, în mod uzual ne dorim mai degrabă optimizarea funcției de cost decât a funcției de pierdere.

Funcțiile de pierdere se împart în 3 categorii în funcție scopul și arhitectura rețelei neuronale:

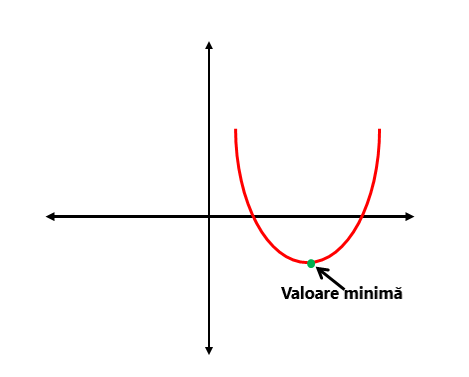
* pentru regresie
* pentru clasificare
* pentru clustering

***Funcții de cost utilizate în cazul regresiei:***

1. **Eroarea pătratică**

Se mai numește si pierderea L2 .Potrivit formulei, observăm că funcția MSE este o ecuație pătratică care are un singur minim global și niciun minim local. Acest aspect poate fi privit ca un avantaj matematic, fiind cea mai utilizată funcție de pierdere în rândul DL.

|  |  |  |
| --- | --- | --- |
|  |  |  |



Funcția de cost corespunzătoare este media acestor pierderi pentru toate eșantioanele și anume *Eroarea Medie Pătratică* (**MSE**).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Dacă eroarea este mare, atunci diferența între valoarea prezisă și cea reală este mai mare, iar în urma ridicării la pătrat aceasta diferență va fi mai proeminentă. Deci MSE este mai puțin robust la prezența valorilor aberante (valoare total diferită de restul datelor din dataset) și este indicat să nu se utilizeze în cazul în care sunt multe astfel de valori prezente în dataset.

1. **Eroarea absolută**

Se mai numește și pierderea L1. Spre deosebire de MSE, se ia în calcul valoarea absolută a diferenței și nu pătratul acesteia. În acest caz calculul gradienților implică tehnici de programare mai complexe. Această funcție este utilizată în cazul unor dataset-uri cu multe valori aberante.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Funcția de cost corespunzătoare este media pierderilor absolute pentru toate eșantioanele de antrenare și anume *Eroarea Medie Absolută* (**MAE**).

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. **Pierderea Huber**

Funcțiile de pierdere L1 și L2 sunt eficiente, dar au anumite limitări:

* Funcția L1 este mai robustă la valori aberante decât L2, adică diferența estee mai mare, iar L1 este mult mai stabilă decât L2.
* Funcția L2 este mult mai stabilă decât L1 atunci când diferența între valoarea prezisă și cea reală este mai mică.

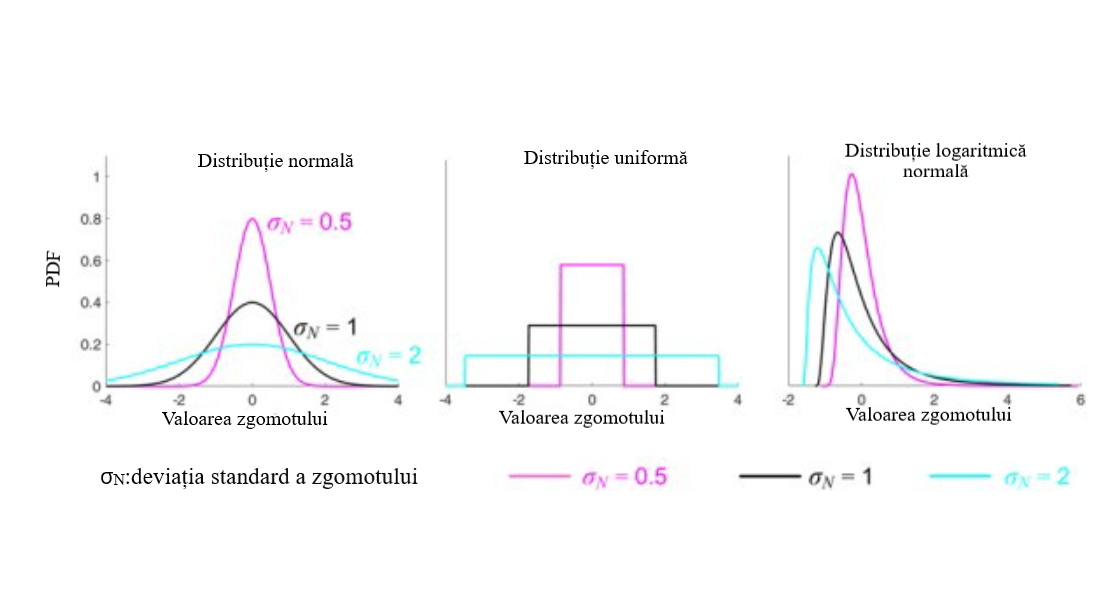
*Funcția de pierdere Huber* preia avantajele celor două funcții menționate anterior, mai exact acesta devine pătratică pentru erori mici și liniară pentru valori mai mari aler erorilor.

Funcția Huber este caracterizată de parametrul .

|  |  |  |
| --- | --- | --- |
|  |  |  |

***Funcții de cost utilizate în cazul clasificării:***

În cazul clasificării încercăm să prezicem variabile categorice cu ajutorul teoriei probabilităților. Astfel, vom trata probabilitățile prezise ca și eșantioane provenind de la o anumită funcție densitate de probabilitate (PDF) și probabilitățile reale provenind de la o altă funcție densitate de probabilitate. Obiectivul principal al rețelei este ca aceste funcții de densitate să coincidă.



Funcțiile de pierdere din cadrul clasificării au la bază *entropia*, care semnifică incertitudinea unui eveniment.

Pentru o variabilă oarecare X, având distribuția de probabilitate p(X), entropia este definită astfel:

|  |  |  |
| --- | --- | --- |
|  |  |  |

* 1. **Funcția de pierdere Binary Cross-Entropy**

Dacă probabilitatea de a fi în clasa x este **P**, atunci probabilitatea de a fi în clasa y este **(1-P)**. Atunci entropia pentru valoarea reală a lui Y (care poate fi între 0 și 1) și valoarea prezisă este:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Această funcție este numită și pierderea Log și se poate utiliza funcția sigmoid pentru a calcula P, unde Z reprezintă parametrii de intrare ai modelului:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Funcția de cost corespunzătoare este definită:

|  |  |  |
| --- | --- | --- |
|  |  |  |

* 1. **Funcția de pierdere Categorical Cross-Entropy**

În cazul acesta vom avea variabile catergoriale sub forma unor clase multiple. Calculul entropiei va rămâne același, dar vectorul Y va fi reprezentat printr-un vector codificat.

|  |  |  |
| --- | --- | --- |
|  |  |  |

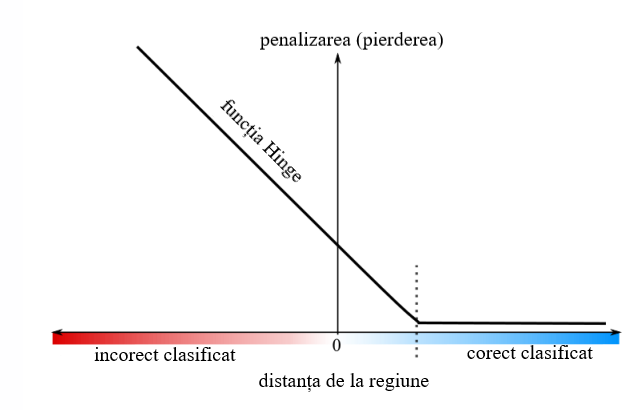
Yij va fi vectorul codificat și Pij va fi probabilitatea prezisă de a fi în clasa j când al i-lea eșantion al datei de intrare (Xi) va fi furnizat.

* 1. **Funcția de pierdere Hinge**

Este utilizată doar în cazul algoritmilor **Suport Vector Machine (SVM),** având clase -1 și 1. SVM este un algoritm de ML utilizat în mod special pentru clasificare binară, utilizând *regiuni de decizie* pentru a separa două clase.

Funcția Hinge penalizează predicțiile eronate precum și care modelnu este de încredere.

|  |  |  |
| --- | --- | --- |
|  |  |  |



Funcția Hinge nu este o funcție diferențiabilă, dar are o natură convexă, care o ajută în găsirea pierderii optime.

### Calculul gradienților și optimizatorul

1. ***Calculul gradienți***

Pe baza noțiunilor prezentate în subcapitolul de Definiții, vom abstractiza fiecare strat al rețelei conform următoarelor relații:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

,unde L reprezintă indicele ultimului strat al rețelei

Pentru a calcula relația dintre componentele rețelei și funcția de cost trebuie să utilizăm derivatele parțiale. Astfel, înțelegând necesitatea acestora, vom putea minimza funcția de cost prin modificarea ponderilor corespunzătoare și a bias-urilor, mai exact vom determina un raport între ponderi și funcția de cost, iar .elementele cu cel mai mare raport vor avea impactul cel mai proeminent asupra funcției de cost.

Aplicând derivata parțială asupra relațiilor ??? vom obține relația ??:

|  |  |  |
| --- | --- | --- |
|  |  |  |

În scopul minimizării funcșiei de cost, este necesară propagarea inversă în rețea și actualizarea ponderilor și a bias-urilor. În principiu acest proces presupune calculul *gradientului* pe baza a trei relații: una pentru ponderi, un pentru bias și una pentru funcția de activare:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Fiecare dintre relațiile de mai sus (corespunzătoare stratului de ieșire al rețelei) calculează cum anumite ponderi afectează funcția de cost, pe care ne dormim să o *optimizăm.* Fiecare derviată a ponderilor și a bias-urilor este salvată într-un *vector gradient* ∇, a cărui dimensiune este egală cu numărul ponderilor și al bias-urilor:

|  |  |  |
| --- | --- | --- |
|  |  |  |

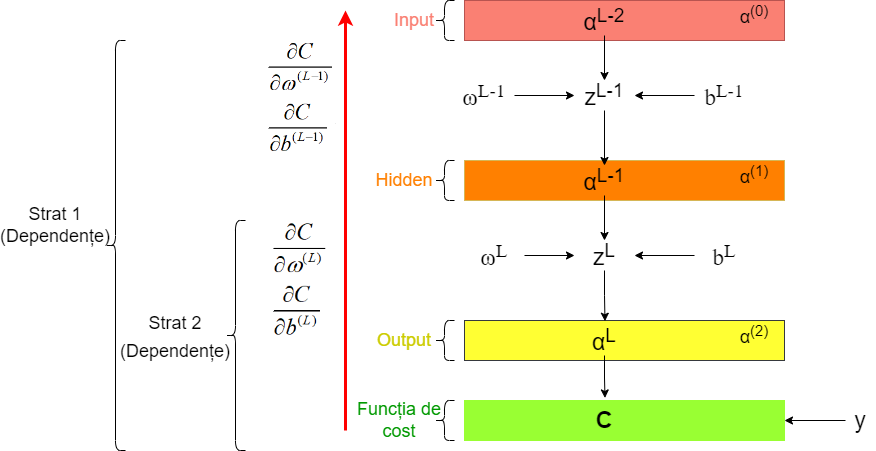
Unde n reprezină numărul de ponderi/ bias-uri și ∇ este vectorul gradient.Gradientul ∇C(ω1,b1…,ωn,bn) ne furnizează direcția în care funcția de cost C are *cea mai rapidă scădere*, demonstrând și necesitatea semnului minus înaintea gradientului

Gradientul este calculat în funcție de un *mini-batch* (va fi dicutat ulterior, adesea este de 16 sau 32 în funcție de puterea computațională) de date. Pentru fiecare mini-batch este calculată media ponderilor și a bias-urilor. Ulterior, acestă medie devine ieșirea gradientului care creează în direcția medie optimă peste dimensiunea mini-batch-ului. Astfel, trebuie actualizate toate ponderile și bias-urile dupa fiecare parcurgere a mini-batch-ului, acestea având o anumită valoare pentru fiecare *strat l,* conform formulelor ???:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Unde η reprezintă rata de învățare.

Ecuațiile ??? vizează stratul de ieșire al rețelei, dar dacă luăm în considerare și restul straturilor, sunt înlănțuite mai multe derivate parțiale pentru a determina și ponderile specifice primului strat și pentru a exemplifica acest aspect vom folosi un graf de dependeță:

**

*Dependențe pentru calcularea gradienților pentru o rețea cu 3 straturi. Stratul 1 se bazează pe stratul 2, deoarece reutilizează calculele gradienților din stratul 2*

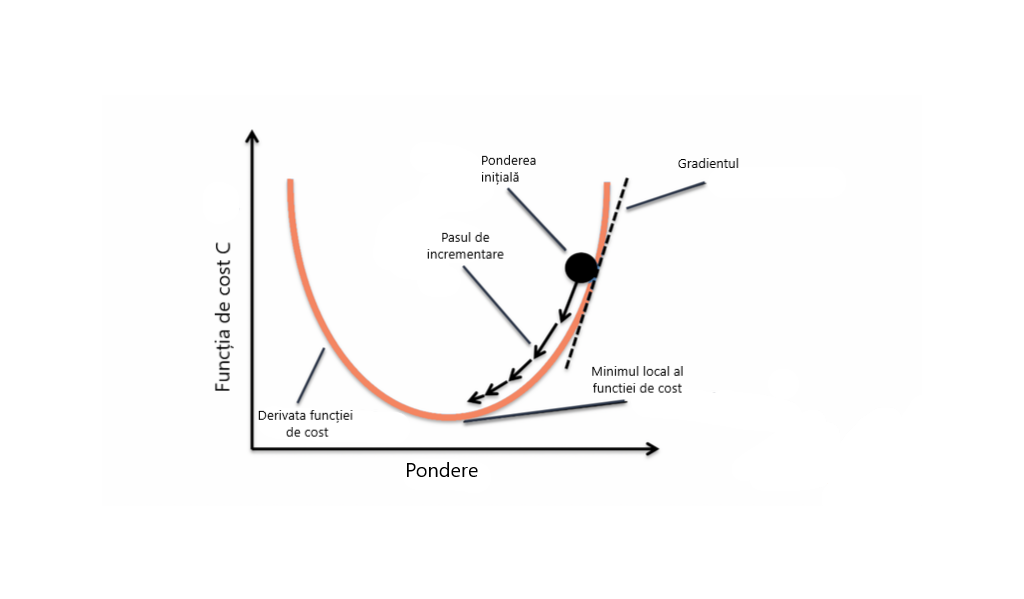
Conform figurii ???, actualizarea ponderilor și a bias-urilor din stratul 2 (sau L) depinde numai de funția de cost și ponderile și bias-urile de la stratul respectiv. Similar, pentru actualizarea stratului 1 (sau L-1), sunt dependente de calculele de la strayul 2 și ponderile și bias-urile de la stratul1. Dacă ar fi o rețea cu mai multe straturi, ar fi tot mai multe dependențe între starturi și din acest motiv tehnica poartă denumirea de propagare inversă.

Generalizând, pentru o rețea cu *n straturi*, derivata funcției de cost raportată la valoarea ponderii din stratul 1 va fi dată de formula ???:

|  |  |  |
| --- | --- | --- |
|  |  |  |

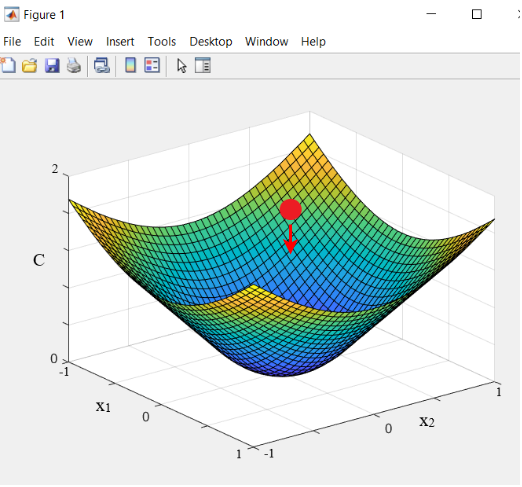
Așadar, algoritmul de calcul al gradientului care reprezintă esența procesului de învățare a rețelei neuronale poartă denumirea de *propagare inversă* (explicată în subcapitolul anterior)*.*

Este vital ca valorile de ieșire ale funcției de cost să fie *uniforme* pentru a putea determina *minimul local* al funcției prin pași mici descrescători. Acest proces repetat de actualizare a unei valori de intrare bazate pe multiple valori negative ale gradientului poartă denumirea de ***Gradient Descent (***“descreșterea gradientului”***)***. [14]



*Exemplificare a modului de funcționare a gradientului în scopul minimizării funcției de cost*

Rezumând, modul în care funcționează algoritmul Gradient descent este de a calcula în mod repetat gradientul ∇C și apoi de a se deplasa în direcția opusă, „căzând” pe panta văii. O putem vizualiza astfel, în cazul unui sistem bidimensional, adica C reprezintă o funcție cu doar două variabile (de cele mai multe ori rețelele neuronale sunt mult mai profunde de atât și nu pot fi vizualizate grafic):



Există trei variante de ***Gradient Descent***, care diferă prin cantitatea de date pe care o folosim pentru a calcula gradientul funcției cost. În funcție de cantitatea de date, se face un compromis între acuratețea ajustării parametrilor și timpul necesar pentru a efectua o ajustare:

* **Batch Gradient Descent**
* **Stochastic Gradient Descent (SGD)**
* **Mini-batch Gradient Descent**

În conformitate cu literatura de specialitate din domeniul DL, notațiile uzuale întâlnite sunt:

* J(θ)-funcția cost(funcție reală multidimensională), iar J provine de la matricea Jacobiană care ne furnizează derivatele parțiale din toate direcțiile;
* θ- ponderile determinate de conexiunile rețelei neuronale
* η-rata de învățare

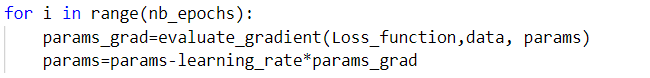
1. **Batch Gradient Descent:**

Vanilla Gradient Descent sau *gradient descendent în lot* (BGD), calculează gradientul funcției de cost în raport cu ponderile θ pentru *întregul set de date* de antrenare:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Deoarece trebuie să calculăm gradienții pentru întregul set de date pentru a efectua doar o singură ajustare a ponderilor, descreșterea gradientului pe loturi poate fi foarte lentă și este dificil de realizat pentru seturile de date care nu încap în memorie. Astfel, pentru procesul de antrenare pe un dataset mare este nevoie de o putere de calcul considerabilă, *resurse GPU mai mari*.

La nivel de cod, BGD este implementat în felul următor:



Pentru un număr prestabilit de epoci, mai întâi calculăm vectorul gradient *params\_grad* al funcției de pierdere pentru întregul set de date în raport cu vectorul nostru de parametri *params*. Apoi actualizăm parametrii în *direcția opusă gradienților*(-*∇*), rata de învățare determinând cât de mare este modificarea ponderilor. *Descreșterea gradientului* pe loturi converge la un *minim global* pentru suprafețele de eroare *convexe* și la un *minim local* pentru suprafețele *neconvexe*.

În cazul acestei metode de optimizare, dimensiunea loturilor prelucrate în pașii de antrenare (batch size) va fi egală cu dimensiunea datasetului de antrenare.

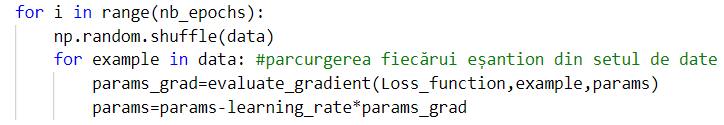
1. **Stochastic Gradient Descent (SGD)**[15]**:**

Spre deosebire de BGD, Stochastic Gradient Descent presupune actualizarea parametrilor pentru fiecare eșantion din setul de date, x(i), și pentru fiecare nume de clasă, y(i), de antrenament. Acest aspect înseamnă un batch size egal cu 1.

|  |  |  |
| --- | --- | --- |
|  |  |  |

BGD realizează calcule redundante pentru seturi mari de date, deoarece recalculează gradienții pentru datele asemănătoare înainte fiecărei actualizări a ponderilor. În schimb, SGD elimină această redundanță prin efectuarea unei actualizări la un moment dat. Deoerece SGD realizează des actualizările parametrilor cu o varianță mare, graficul funcției de cost va avea o fluctuație mare.

În timp ce BGD converge către minimul zonei convexe în care se află, SGD, datorită fluctuațiilor multe, permite saltul către minime locale noi și mai bune. La nivelul codului, poate fi implementat astfel:



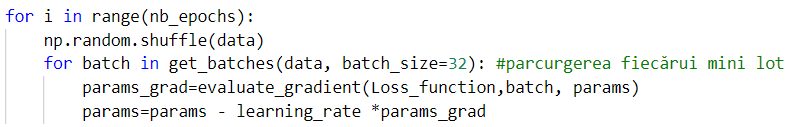
**Mini-batch Gradient Descent:**

*Metoda gradientului descendent pe mini loturi* reunește avantajele celor două metode menționate mai sus și realizează actualizarea parametrilor pentru fiecare mini lot a câte n elemente (confrom literaturii de specialitate este utilizat pentru n=32):

|  |  |  |
| --- | --- | --- |
|  |  |  |

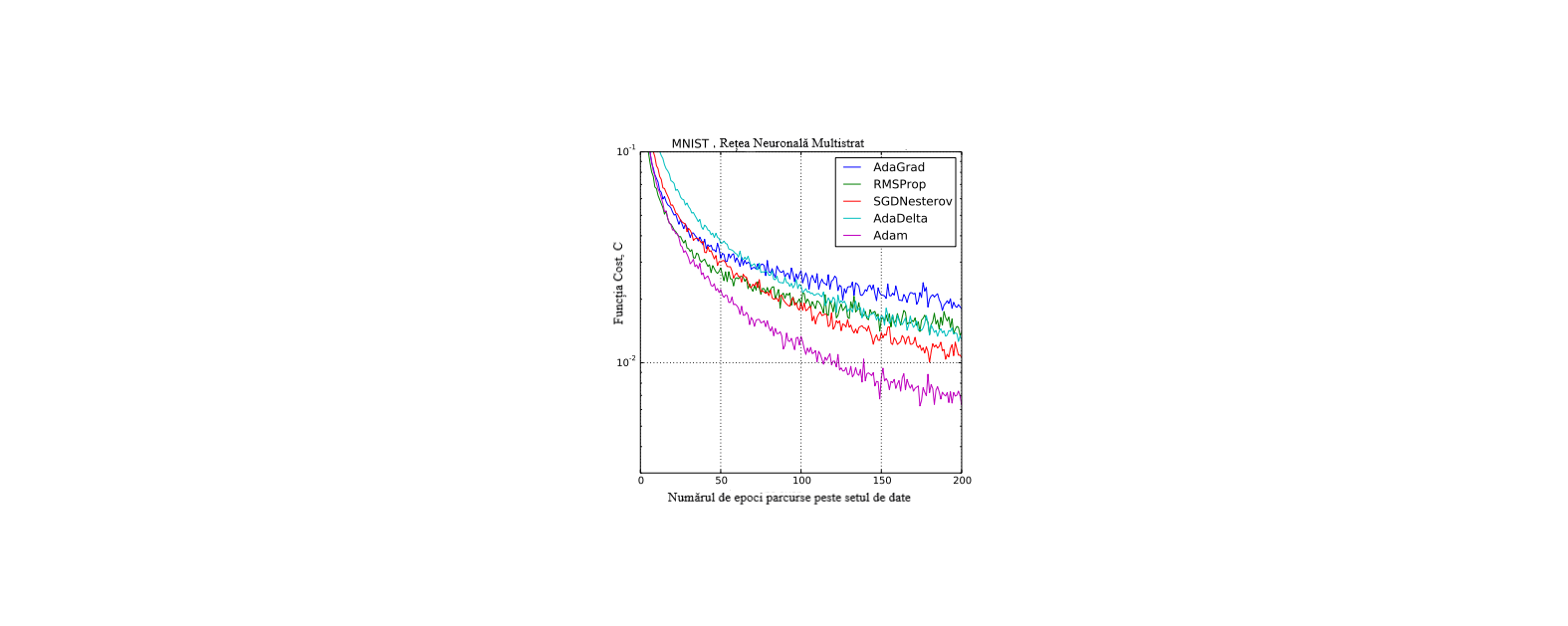
Avantajele gradientului descendent pe mini loturi sunt: reducerea varianței determinată de modificările dese ale parametrilor, determinând astfel o mai bună convergență și compatibilitatea cu bibliotecile de DL care pot permite calculul eficient al gradientului.

La nivelul codului, această metodă poate fi implementată astfel:



Algoritmi de optimizare cunoscuți: *Momentum, Gradientul Accelerat Nesterov, Adagrad, Adadelta, RMSprop, Adam, AdaMax, Nadam, AMSGrad, QHAdam, AggMo* și *AdamW.*

O comparație între rezultatele generate de fiecare algoritm se poate observa în figura de mai jos:



*Antrenarea rețelelor neuronale multistrat pe imagini MNIST cu optimizatori diferiți* [16]

În cadrul acestui subcapitol voi aborda doar modul de funcționare ale algoritmilor de optimizare utilizați în partea practică a proiectului și anume ***SGD cu moment și ADAM(Adaptive Moment Estimation).***

**SGD cu moment sau Momentum**[17] este o metodă care *accelerează* SGD într-o direcție dorită și atenuează oscilațiile. Realizează asta prin adăugarea unei fracțiuni γ din vectorul de la pasul anterior vectorului curent pentru a accelera pasul de determinare a minimului global:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

*Valoarea fracțiunii γ este adesea setată la 0.9.*

Ajustarea parametrilor se face prin creșterea impulsului în cazul gradienților care îsi păstrează direcția sau reducerea acestuia dacă gradienții își schimbă direcția. Ca rezultat, este obținută o convergență mai rapidă cu oscilații reduse.

**ADAM** [16] este *o metoda adaptivă de adjustare a ratei de învățare* pentru fiecare parametru. Pentru a ține evidența unei medii exponențiale descrescătoare a gradienților anteriori, ADAM menține o descreștere a gradienților trecuți *mt.*Pentru o înțelegere mai bună, diferența între Momentum și ADAM poate fi privită conform analogiei următoare: momentul poate fi privit ca o bilă ce se rostogolește de-a lungul unei pante, pe când în cazul ADAM bila este grea și alunecă cu frecare, găsind astfel minime plate în zona de eroare.[18]

Media de scădere a gradienților trecuți poate fi calculată astfel:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Unde mt și vt sunt aproximările pentru momentul inițial (principal) și momentul secundar, de unde și denumirea acestei metode (ADAM= estimarea adaptivă a momentelor). Mai exact, “netezește” exponențial gradientul de ordinul întâi pentru a integra momentul (impulsul) în ajustare. De asemenea, integrează în mod direct și bias-ul în medierea exponențială, în cazul în care estimarea curentă a unei valori “netezite” este inițializată în mod nerealist cu 0.[13]

Astfel, calculul estimărilor primului și celui de-al doilea moment în funcție de bias devin:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Ecuația de ajustare a ponderilor în cazul aplicării algoritmului ADAM devine:

|  |  |  |
| --- | --- | --- |
|  |  |  |

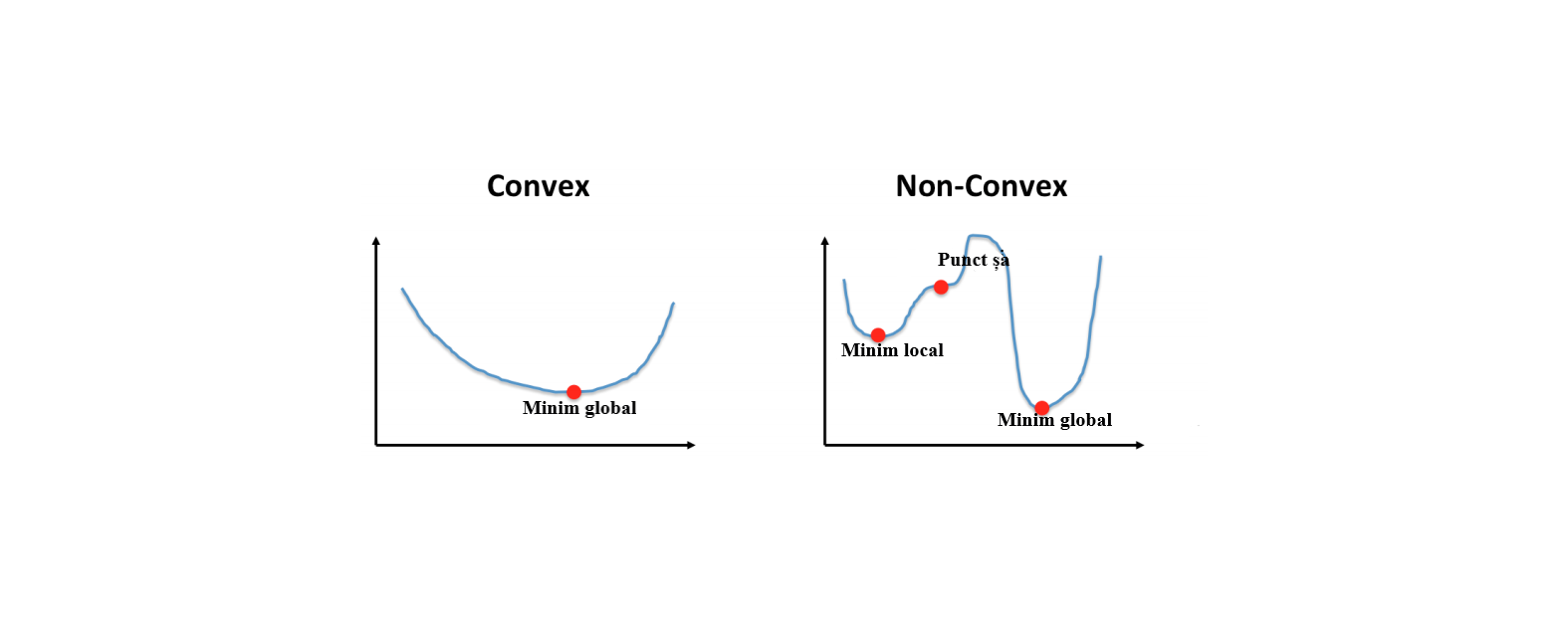
Unde ε are valoarea 10-8 propusă de autorii metodei.

### Parametrii și hiperparametrii rețelelor neuronale convoluționale. Probleme impuse în urma modificării acestora

Antrenarea unui model de rețea neuronală de învățare profundă cu SGD și propagare inversă implică configurarea unor parametri și hiperparametri pentru a putea combate problemele esențiale impuse de o rețea neuronală multistrat și pentru obținerea unui model optim.

În primă instanță, problemele lansate de de antrenarea unei rețele neuronale au la bază găsirea *minimului global în suprafața de eroare: “Optimizarea în general este o sarcină extrem de dificilă. […] Când antrenăm rețele neuronale, trebuie să ne confruntăm cu cazul general neconvex.”*[19]

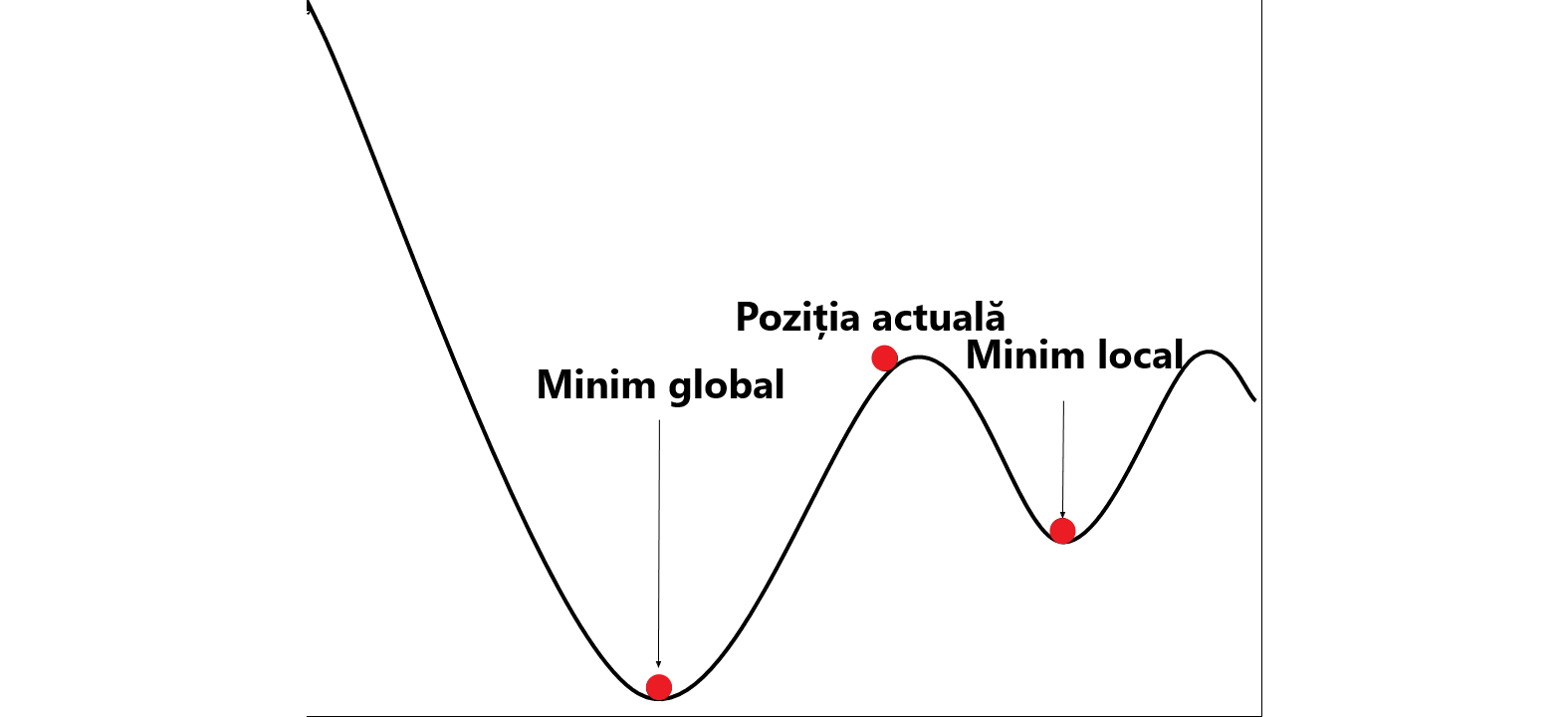
Majoritatea problemelor de optimizare ale modelului derivă din forma neconvexă a suprafeței de eroare, pe când găsirea minimului global în cazul unei suprafețe convexe este o sarcină relativ ușoară:



Rezumând astfel, sunt trei probleme principale determinate caracterul neconvex al suprafețelor de eroare:

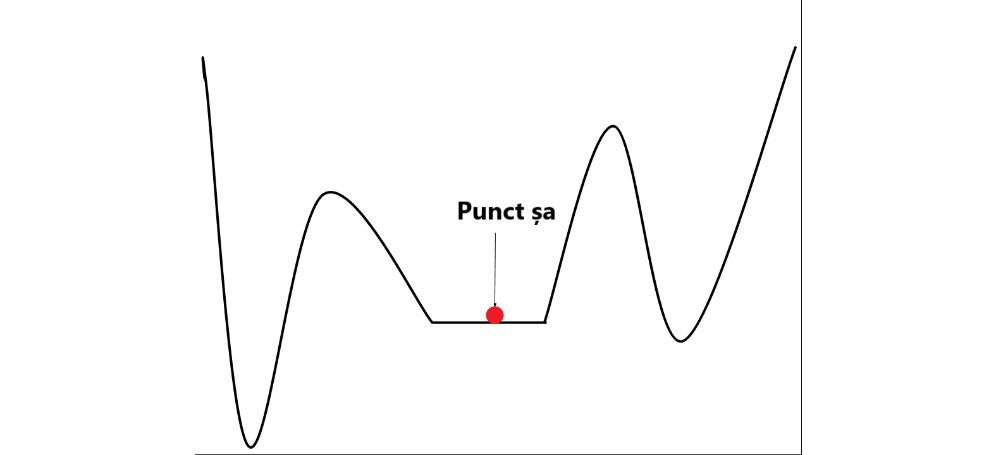
* **Minimul local**

Rețelele neuronale pot avea unul sau mai multe minime globale, iar diferența dintre minimele locale și cele globale poate să nu fie sesizată adesea.



* **Punctele șa**

O zonă plată sau un punct șa este punctul din grafic în care gradientul este 0 (funcția de cost este constantă). Problema determinată de valoarea nulă a gradientului este că algoritmul de optimizare nu știe în ce direcție să se deplaseze pentru a îmbunătăți modelul.

****

* **Multi-dimensionalitatea**

Fiecare pondere a rețelei reprezintă o dimensiune a suprafeței de eroare. DNN au milioane de astfel de parametri, ceea ce determină o dimensionalitate foarte mare a funcției de cost.

Problema determinată de adăugarea unei noi dimensiuni constă în creșterea distanței dinstre punctele din spațiu.

Subliniind o parte din problemele de optimizare din cadrul DNN, parametrii și hiperparametrii, care prin ajustarea optimă a acestora pot conduce la rezultate bune, sunt:

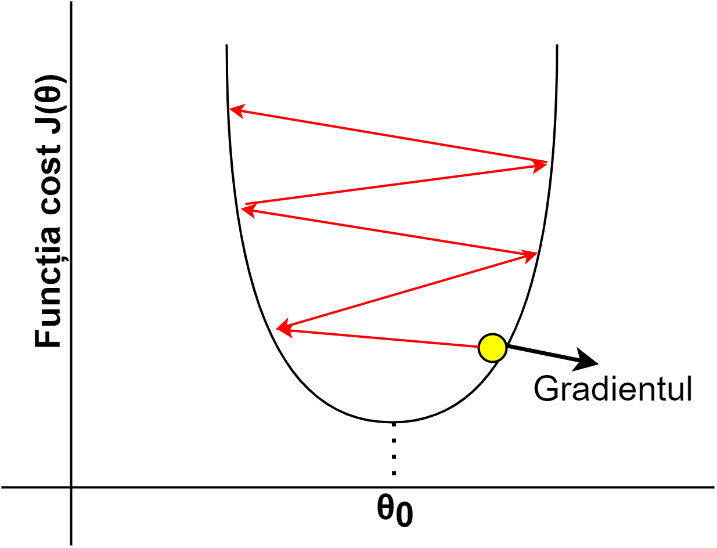
1. **Rata de învățare (learning rate η)**

*Rata de învățare* este un hiperparametru care controlează cât de mult trebuie ajustate ponderile rețelei în funcție de gradientul de pierdere.

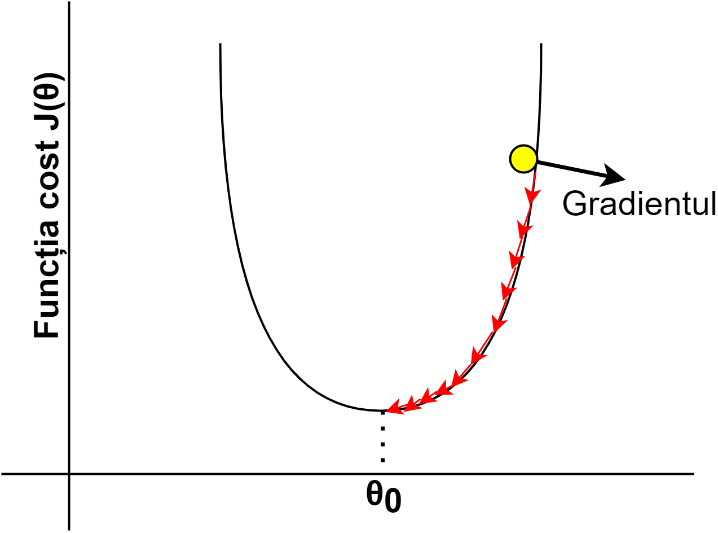
Astfel, cu cât eroarea este mai mică, cu atât mai încet se deplasează gradientul de-a lungul pantei descendente, sugerând utilizarea unei rate de învățare mici pentru a nu evita niciun minim local, dar dacă este setată prea mică poate determina un timp îndelungat al rețelei ca să conveargă, mai ales dacă rămâne blocată într-un punct de șa și multe ajustări ale parametrilor, necesitând o putere mare de calcul.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Unde η este rata de învățare, ∇θJ(θ) gradientul funcției de cost și θ1 parametrul corespunzător ponderii ω1 și bias-ului b1.

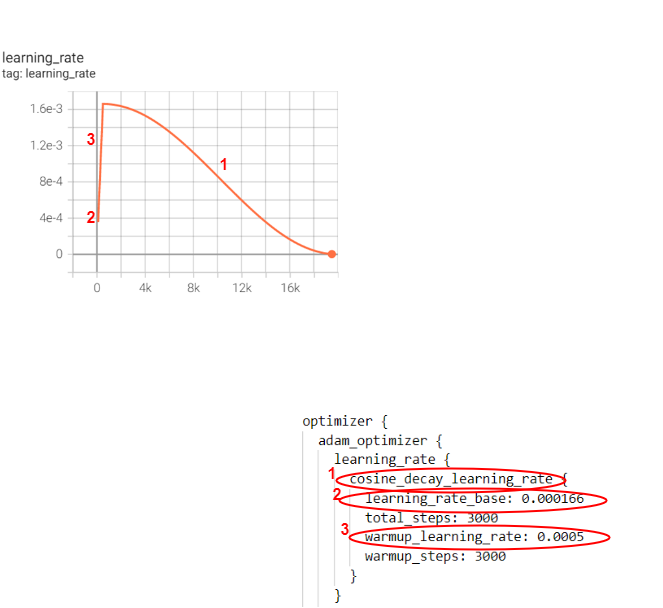
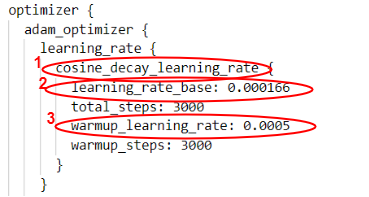


Totodată, dacă eroarea este mare, este sugerată o rată mai mare de învățare, dar dacă *rata de învățare este prea mare* va determina comportamentul divergent al gradientului și “saltul” acestuia peste minimele locale.



În plus, rata de învățare afectează și timpul în care modelul poate converge către un minim local (adică ajunge la cea mai bună precizie). Prin urmare, dacă o rata este aleasă bine de la început, va fi nevoie de mai puțin timp pentru a antrena modelul.

În cadrul proiectului rezultatele optime au fost obținute pentru o ajustare adaptivă a ratei de învățare pe tot parcursul antrenării după cum se observă în figura ???:

*Metodele adaptive cu pas descrescător* al ratei de învățare de-a lungul antrenării rezolvă problemele genereate de alergerea unei rate prea mari sau prea mici, ajustând rata de învățare în funcție de evoluția antrenării

*Utilizarea cosinusului descrescător* cu un optimizator mai avansat, cum ar fi Adam, îmbunătățește semnificativ procesul de învățare și ajută la evitarea minimului local. Dar metoda utilizată în proiect are propriuldezavantaj: în primul rând, nu știm care este rata inițială optimă de învățare și, în unele cazuri, scăderea ratei de învățare poate duce la "blocarea" rețelei în zone plate din suprafața de eroare.

După cum se observă în figura, am utilizat *tehnica de warm-up* (“încalzire”) ce presupune creșterea ratei de învățare de la 0 (sau de la o valoare foarte mică, precum cea de 0.0005) la o valoare stipulată pe un anumit număr de pași pentru a nu expune prematur rețeaua la o rată mare de învățare (în primele epoci rețeaua are cea mai mare rată de eroare), determinând comportamentul divergent al gradientului.

1. **Dimensiunea loturilor (batch size bs)**
2. **Numărul epocilor/ pașilor de antrenare (step size)**
3. **Inițializarea ponderilor**
4. **Regularizarea**

## Rețelele neuronale convoluționale (CNN)

### Arhitectura

### Convoluția

### Pooling

### Fully-Connected

## Modalități de antrenare

### Fine Tuning

### Transfer Learning

### Learning from scratch

## Modelul neuronal utilizat

## Procesarea imaginilor

### Adnotarea

### Augumentarea

# METODE DE IMPLEMENTAREA ȘI VERIFICARE

## Implementarea software

### Mediul de dezvoltare, limbajul de programare și librării utilizate

### Construirea setului de date

### Implementarea CNN pe baza principiului de Transfer Learning

### Modelul SSD MobileNet v2 320x320

### Procesorul grafic Nvidia

### CPU și GPU. Limitări impuse de placa grafică.

### Coral USB Accelerator (TPU Edge)

### Tensorflow. Framework-ul TFLite

### Tensorboard

### OpenCV

### Probleme de implementare și soluții

## Implementare hardware

### Platforma computațională Raspberry Pi 4B

### Driver-ul L298N Dual Motor

### Modulul cameră v2 Raspberry Pi

### Probleme hardware de implementare și soluții

# REZULTATE EXPERIMENTALE

## Prezentarea evoluției sistemului autonom analizat

## Rezultate obținute

## Analiza performanțelor sistemului obținut

# CONCLUZII ȘI PERSPECTIVE DE VIITOR

# Bibliografie

[1] The MathWorks Inc., “What Is Deep Learning?,” 2021. https://www.mathworks.com/discovery/deep-learning.html

[2] Magnimind Academy, “Deep Learning and Its 5 Advantages,” Jan. 28, 2020. https://becominghuman.ai/deep-learning-and-its-5-advantages-eaeee1f31c86 (accessed Feb. 16, 2022).

[3] I. C. E. IBM Cloud Education, “AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference?,” May 27, 2020.

[4] Michael Middleton, “Deep Learning vs. Machine Learning — What’s the Difference?,” Feb. 08, 2021. https://flatironschool.com/blog/deep-learning-vs-machine-learning/ (accessed Feb. 17, 2022).

[5] Alyse Falk, “Deep Learning vs Machine Learning: What’s the Difference,” Jun. 15, 2021. https://www.computer.org/publications/tech-news/trends/deep-learning-vs-machine-learning-whats-the-difference (accessed Feb. 16, 2022).

[6] Sorin Grigorescu, Tiberiu Cocias, Bogdan Trasnea, and Gigel Macesanu, “A Survey of Deep Learning Techniques for Autonomous Driving,” Brasov, 2020.

[7] Michael Barnard, “Tesla & Google Disagree About LIDAR — Which Is Right?,” Jul. 29, 2016. https://cleantechnica.com/2016/07/29/tesla-google-disagree-lidar-right/ (accessed Feb. 18, 2022).

[8] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller, “Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles,” *IEEE Intelligent Vehicles Symposium (IV)*, pp. 1671–1678, 2017.

[9] Nick McCullum, *Pragmatic Machine Learning*.

[10] Rinu Gour, “Artificial Neural Network for Machine Learning — Structure & Layers,” Feb. 15, 2019. https://medium.com/@rinu.gour123/artificial-neural-network-for-machine-learning-structure-layers-2a275f73f473 (accessed May 01, 2022).

[11] CASPER HANSEN, “Neural Networks: Feedforward and Backpropagation Explained & Optimization,” Aug. 05, 2019. https://mlfromscratch.com/neural-networks-explained/#/ (accessed May 02, 2022).

[12] Mykel J. Kochenderfer and Tim A. Wheeler, *Algorithms for Optimization (The MIT Press)*, Illustrated Edition. 2019.

[13] Charu C. Aggarwal A, *Neural Networks and Deep Learning:A Textbook*. Yorktown Heights, NY, USA: Springer.

[14] Michael Nielsen, *Neural Networks and Deep Learning*.

[15] H. Robinds and S. Monro, *“A stochastic approximation method,” Annals of Mathematical Statistics*, vol. 22. 1951.

[16] Diederik P. Kingma and Jimmy Lei Ba, “Adam: a Method for Stochastic Optimization.,” International Conference on Learning Representations, 2015.

[17] Ning Qian, *On the momentum term in gradient descent learning algorithms*, vol. 12. Neural Networks : The Official Journal of the International Neural Network Society, 1999. doi: 10.1016/s0893-6080(98)00116-6.

[18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter, “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium,” *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, doi: https://doi.org/10.48550/arXiv.1706.08500.

[19] Aaron Courville, Ian Goodfellow, and Yoshua Bengio, *Deep Learning (Adaptive Computation and Machine Learning series)*. London, Cambridge, Massachusetts: The MIT Press, Year: 2016, 2016.

# ANEXE