UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA MATEMATICĂ ȘI INFORMATICĂ

LUCRARE DE LICENȚĂ

GENERAREA UNOR PIESE MUZICALE FOLOSIND

GENERATIVE ADVERSARIAL NETWORKS

COORDONATOR ȘTIINȚIFIC:

Prof. dr. DIOSAN LAURA

Absolvent: REBEGEA DRAGOȘ-ALEXANDRU

CLUJ-NAPOCA

2019

Cuprins

[1.Introducere 3](#_Toc11755249)

[2.Problema abordata 4](#_Toc11755250)

[3.Elemente teoretice 4](#_Toc11755251)

[3.1 Inteligenta artificiala 4](#_Toc11755252)

[3.2 Invatarea automata 5](#_Toc11755253)

[3.2.1 Tipurile de invatare automata 6](#_Toc11755254)

[Invatare supervizata 6](#_Toc11755255)

[Invatarea nesupervizata 7](#_Toc11755256)

[Invatarea prin intarire 8](#_Toc11755257)

[Retelele neuronale 9](#_Toc11755258)

[Modelul unui neuron 14](#_Toc11755259)

[Modelul deep learning 15](#_Toc11755260)

[Gradient Descent 15](#_Toc11755261)

[4.Studiu de caz 21](#_Toc11755262)

[4.1 Functionalitati 21](#_Toc11755263)

[4.2 Analiza si proiectare 21](#_Toc11755264)

[4.2.1 Diagrama cazurilor de utilizare 21](#_Toc11755265)

[4.2.2 Diagramele cazurilor de interacțiune 22](#_Toc11755266)

[4.3 Implementare 24](#_Toc11755267)

[Python 24](#_Toc11755268)

[4.4 Validare si testare 25](#_Toc11755269)

[4.5 Validare experimentala 25](#_Toc11755270)

[5.Bibliografie 28](#_Toc11755271)

# 1.Introducere

De la ascensiunea si popularitatea de care s-a bucurat domeniul învățării automate care a adus rezolvarea numeroase probleme din diferite domeniile: medicina (depistarea bolilor, anomaliilor umane), securitate cibernetica(depistarea virusilor si malware-urilor, atacurilor cibernetice), tehnologie ( masini autonome, roboti umanoizi), generarea de orice tip a devenit si el un subiect de interes in lumea informaticii. In aceasta conjunctura, pornind de la generare de text dupa anumiti autori, cu stiluri unice, si incercarea de a replica aceste stiluri creand totusi opera noi, care sunt doar inspirate de aceste personalitati si nu copiaza nici macar partial operele acestea, voi incerca sa creez o aplicatie care poate face acest lucru, doar ca scopul aceste lucrari nu este acela de a genera o opera literara, ci una muzicala.

## Boom-ul invătării automate

În 2006, Geoffrey Hinton et al. a publicat un document care arăta cum o rețea neuronală este capabilă să recunoască cifre scrise de mână cu o precizie uimitoare(> 98%). Ei au numit această tehnică "Deep Learning". Formarea unei rețele neuronale fost în general considerat imposibil la acel moment, și majoritatea cercetătorilor au abandonat ideea din anii '90. Această lucrare a reînviat interesul comunității științifice și au urmat multe lucrări noi ce demonstrat că Deep Learning nu a fost numai posibil, dar capabil să realizeze lucruri uimitoare, pe care nici o altă tehnică de invățare automată nici nu ar putea spera să le atingă. Acest entuziasm sa extins în curând în multe alte domenii ale învătării automate.

După doar 10 ani acest domeniu a cucerit industria: este acum in aproape toate produsele high-tech de astăzi, clasificatori web pentru rezultatele căutării, înbunătățirea recunoașterii vocale de pe orice dispozitiv, recomandări video, bate campionul mondial la jocul Go. Toate acestea sunt doar o mică parte din potențialul uriaș pe care continuă sa il ofere.

Majoritatea oamenilor cand aud „inteligentă artificială” își imaginează un robot: un majordom de încredere, un asistent de orice tip. Dar acest lucru nu este doar o fantezie futuristă, este deja aici. De fapt, a fost în jur de zeci de ani în unele aplicații specializate, cum ar fi recunoaștere optică a caracterelor .Totuși, prima aplicație din acest domeniu care a fost într-adevăr folosită de întreaga populație, îmbunătățind viața a sute de milioane de oameni, a fost creată tocmai în anii 1990: filtrul de spam-uri. Nu este chiar o aplicație de sine stătătoare, dar se califică din punct de vedere tehnic ca învățare automată (de fapt a învățat atât de bine încât rareori trebuie să semnalizați un e-mail ca spam). A fost urmată de sute de aplicații care sunt încorporate in produse și caracteristici pe care le utilizați în mod regulat, de la recomandări mai bune de filme, videoclipuri, muzica la căutarea vocală,asistenți vocali.

# 2.Problema abordata

## 3.Elemente teoretice

## 3.1 Inteligenta artificiala

Inteligența artificială sa născut în anii 1950, când o mână de pionieri din

domeniul informaticii au început să se întrebe dacă computerele ar putea fi făcute sa

"gândeasca" - o întrebare a cărei ramificații încă o explorăm astăzi. O definire concisă a domeniului ar fi după cum urmează: efortul de automatizare a sarcinilor intelectuale efectuate în mod normal de către oameni. Ca atare, AI este un domeniu general care cuprinde învățarea mașinilor și

învățare profundă, dar care include și mai multe abordări care nu implică nici una

învăţare.Primele programele de șah, de exemplu, au implicat doar regulile brute elaborate de

programatori, și nu s-au calificat drept mașini de învățare. Timp destul de lung, mulți

experții credeau că inteligența artificială la nivel uman ar putea fi atinsă programarea unui set suficient de mare de reguli explicite pentru manipulare cunoştinţe. Această abordare este cunoscută sub numele de IA simbolic, și a fost paradigma dominantă în IA din anii 1950 până la sfârșitul anilor 1980. Ea a atins popularitatea maximă în timpul boom-ul sistemelor experte din anii '80.

 Deși IA simbolic s-a dovedit a fi adecvata pentru a rezolva probleme bine definite, logice, cum ar fi

Jucul de șah, sa dovedit a fi dificil de găsit reguli explicite pentru rezolvarea unor probleme mult mai

complexe, problemelor mai neclare, cum ar fi clasificarea imaginilor, recunoașterea vorbirii și traducerea limbilor. O nouă abordare a apărut pentru a lua locul simbolic al Ia: invatarea automata.

Charles Babbage, inventatorul motorului analitic: primul calculator mecanic cu scop general cunoscut. Deși vizionar și cu mult înainte de timpul său, motorul analitic nu a fost conceput ca un calculator de uz general când a fost proiectat în Anii 1830 și 1840, deoarece conceptul de calcul general era încă inventat. A fost vorba doar de o modalitate de a utiliza operații mecanice pentru a automatiza anumite calcule din domeniul analizei matematice - de aici, denumirea de motor analitic. În 1843, Ada Lovelace, un prieten și colaborator al lui a adus o remarca cu privire la invenție: "Motorul analitic nu are pretenții indiferent de originea vreunui lucru. Poate face orice știm cum să să-i dăm ordin să facă ... Scopul lui este să ne ajute să punem la dispoziție cu ceea ce suntem deja familiarizat. "

  Această remarcă a fost ulterior citată de pionierul inteligentei artificiale,Alan Turing ca "obiecția doamnei Lovelace" în cartea sa de referință din 1950 “Computing Machinery and Intelligence,” in care a introdus testul Turing, precum și concepte-cheie care au dat o forma inteligenței artificiale. Turing a citat-o ​​pe Ada Lovelace în timp ce se gândea dacă computerele cu scop general ar putea să fie capabil de învățare și originalitate și a ajuns la concluzia că ar putea.

## 3.2 Invatarea automata

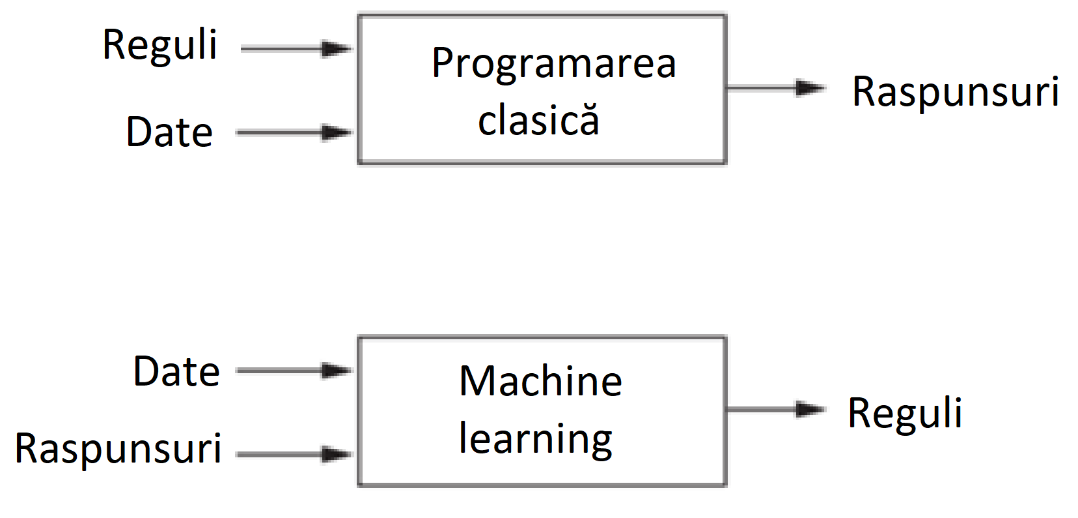
Invatarea automata rezultă din întrebarea: ar putea un computer să depășească "ceea ce noi știm să-l programam să efectueze " și să înveți singur cum să efectueze o sarcină specificată?

Ne-ar putea surprinde un computer? Mai degrabă decât programele care realizează prelucrarea de mana a datelor, ar putea un calculator să învețe automat aceste reguli prin analizarea datelor?

  Această întrebare deschide ușa unei noi paradigme de programare. În programarea clasică, paradigma AI simbolică, oamenii introduc reguli (un program) și date care să fie prelucrate în conformitate cu aceste reguli, ca date de iesire rezultand raspunsurile (a se vedea figura de mai jos). Cu

Machine learning, oamenii introduc datele de intrare,precum și răspunsurile așteptate din date,

și reies regulile ca rezultat al rularii. Aceste reguli pot fi apoi aplicate la date noi pentru a produce răspunsuri originale.



Un sistem de tip de învățare a mașinilor este instruit mai degrabă decât programat explicit. Este prezentat cu multe exemple relevante pentru o sarcină, și găsește structură statistică în aceste exemple care permit în cele din urmă sistemului să vină cu reguli pentru automatizarea sarcinii. De exemplu, dacă doriți să automatizați sarcina de a eticheta fotografiile de vacanță, vă ar putea prezenta un sistem de învățare a mașinilor cu multe exemple de imagini deja etichetate de oameni, iar sistemul ar învăța reguli statistice pentru asocierea anumitor imagini la etichete specifice.

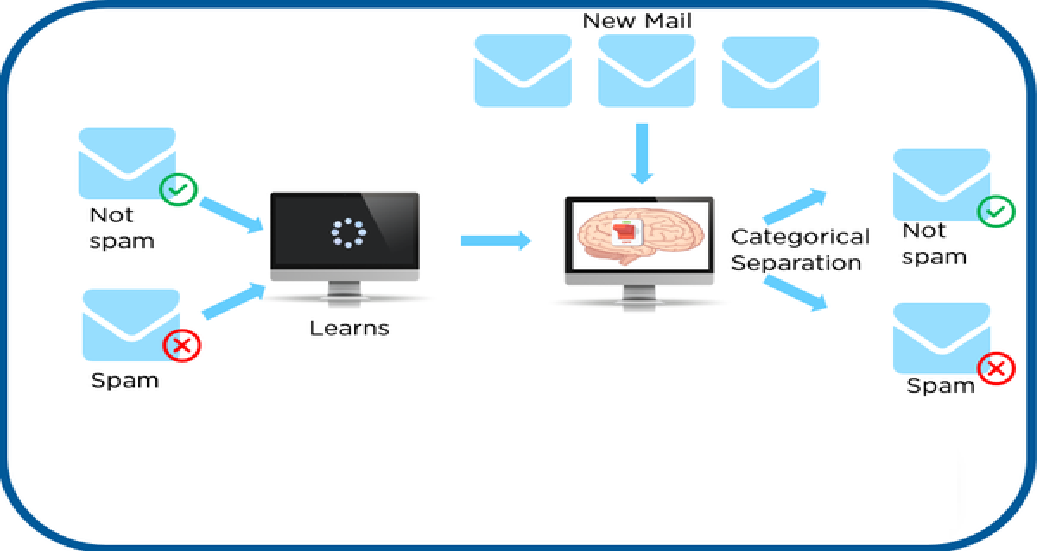
### 3.2.1 Tipurile de invatare automata

Acest domeniu poate fi impartit in 3 mari categorii:

* Invatare supervizata
* Invatare nesupervizata
* Invatare semi-supervizata
* Invatare prin intarire

### Invatare supervizata

În învățarea supervizata, un sistem AI este prezentat cu date care sunt etichetate, ceea ce înseamnă că fiecare data este etichetată cu eticheta corectă. Scopul este de a aproxima funcția de mapare atât de bine încât atunci când avem noi date de intrare (x), le putem prezice variabilele de ieșire (Y) pentru respectivele date.

*Exemplu învățare supervizată*

Așa cum am arătat în exemplul de mai sus, am luat câteva date și le-am marcat ca "Spam" sau "Nu este spam". Aceste date sunt utilizate de modelul de invatare supervizata, aceste date fiind folosite pentru formarea modelului. Odată ce este pregătit, putem testa modelul nostru prin testarea acestuia cu câteva e-mailuri de test și verificând că modelul este capabil să prezică rezultatul potrivit.

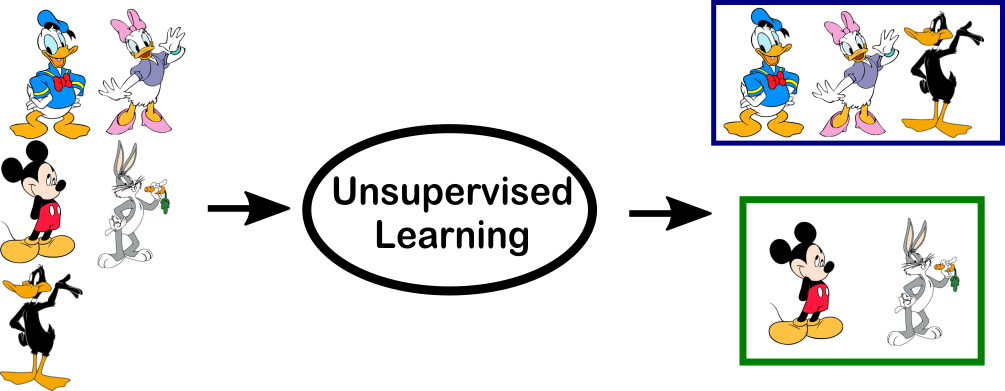
Se numește învățare supervizata deoarece procesul de învățare a algoritmului din setul de date de formare poate fi considerat ca un profesor care supraveghează procesul de învățare. Cunoaștem răspunsurile corecte, algoritmul face iterații cu privire la datele de antrenament și este corectat de către profesor. Învățarea se oprește atunci când algoritmul atinge un nivel acceptabil de performanță.

Problemele de învățare supervizata pot fi grupate în continuare în probleme de regresie și clasificare.

* Clasificare: O problemă de clasificare este atunci când variabila de ieșire este o categorie, cum ar fi "roșu" sau "albastru" sau "boală" și "fără boală".
* Regresie: O problemă de regresie este atunci când variabila de ieșire este o valoare reală, cum ar fi "dolari" sau "greutate". Unele tipuri de probleme comune, construite pe baza clasificării și a regresiei, includ recomandarea și predicția seriei de timp.

### Invatarea nesupervizata

În învățarea nesupervizata, un sistem AI este prezentat cu date neetichetate și neclasificate, iar algoritmii sistemului acționează asupra datelor fără o pregătire prealabilă. Ieșirea depinde de algoritmii codați. Impunerea unui sistem la învățarea nesupravegheată este o modalitate de a testa AI.

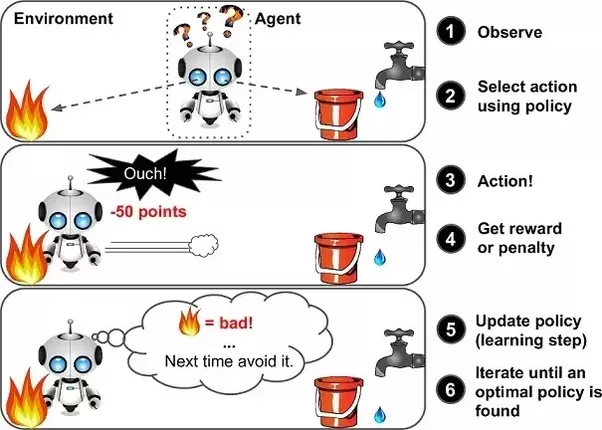
*Exemplu de învățare nesupervizata*

În exemplul de mai sus, am dat câteva caractere modelului nostru care sunt "Rata" și ”Nu este rata” . În datele noastre de instruire, nu furnizăm nicio etichetă datelor corespunzătoare. Modelul nesupravegheat este capabil să separe atât caracterele, analizând tipul de date și modelează structura sau distribuția subiacente în date, pentru a afla mai multe despre ele.

Tipuri de învățare nesupravegheată:

* Clustering: o problemă de grupare în care se doreste descoperirea unor grupări inerente din date, cum ar fi gruparea clienților prin comportamentul de cumpărare.
* Asociație: o problemă de învățare a regulilor de asociere în care se doriste să descoperirea unor reguli care descriu porțiuni mari de date, cum ar fi persoanele care cumpără X, de asemenea, tind să cumpere Y

### Invatarea prin intarire

**Un algoritm de învățare prin întărire, sau agent, învață interacționând cu mediul său. Agentul primește recompense prin efectuarea corectă și penalizările pentru efectuarea incorectă. Agentul învață fără intervenția unui om, maximizând recompensa și minimizând pedeapsa. Este un tip de programare dinamică care tratează algoritmi folosind un sistem de recompensă și pedeapsă.

*Exemplu de învățare prin intarir*

În exemplul de mai sus, putem vedea că agentului i se dau două opțiuni, adică o cale cu apă sau o cale cu foc. Un algoritm de întărire funcționează pe recompensarea unui sistem, adică dacă agentul utilizează traseul de incendiu, atunci recompensele sunt scăzute și agentul încearcă să afle că ar trebui să evite traseul de incendiu. Dacă ar fi ales calea de apă sau calea sigură, atunci s-ar fi adăugat câteva puncte în totalul de puncte, atunci agentul ar încerca să afle ce cale este sigură și ce cale nu este.

În principiu, se utilizează recompensele obținute, agentul își îmbunătățește cunoștințele de mediu pentru a selecta acțiunea următoare.

### Retelele neuronale

Aparitia rețelelor neuronale artificiale, denumite în mod obișnuit "rețele neuronale" a fost motivată chiar de la începuturile sale prin recunoașterea faptului că creierul uman calculeaza într-un mod complet diferit de cel al computerului digital convențional. Creierul este un calculator extrem de complex, neliniar și paralel (sistem de procesare a informațiilor). Aceasta are capacitatea de a-și organiza constituenții structurali, cunoscuți sub numele de neuroni, astfel încât să efecteze anumite calcule (de exemplu, recunoașterea sabloanelor). Cum face un creier uman acest lucru? La naștere, un creier are o mare structură și capacitatea de a-și construi propriile reguli prin ceea ce face, de obicei ne referim la acest lucru prin "experiență". Într-adevăr, experiența este construită în timp, cu cea mai dramatică dezvoltare (adică, hard-cablare) a creierului uman care are loc în timpul primelor două ani de la naștere; dar dezvoltarea continuă mult dincolo de această etapă. Un neuron "în curs de dezvoltare" este sinonim cu un creier plastic: plasticitatea permite dezvoltarea sistemului nervos pentru a se adapta mediului înconjurător. Asa cum plasticitatea pare a fi esențială pentru funcționarea neuronilor ca unități de procesare a informațiilor în creierul uman, la fel si in cazul rețelelor neuronale formate din neuroni artificiali. În cele mai generală forma, o rețea neuronală este o mașină care este proiectată să reproduca modelul prin care creierul îndeplinește o anumită sarcină sau o funcție de interes; rețeaua este de obicei implementat prin utilizarea componentelor electronice sau simulat în software pe un calculator digital.

Pentru a realiza performanțe bune, rețelele neuronale folosesc o interconectare masivă a punând celulele denumite "neuroni" sau "unități de procesare". Prin urmare, unei rețele privită ca o mașină adaptivă neuronale, ii putem oferi urmatoarea definiție:

O rețea neurală este un procesor distribuit masiv paralel, alcătuit din unitati simple de procesare, care au o tendință naturală de a stoca cunoștințele experiențiale și de a le face utilizabile. Seamănă cu un creier uman în două privințe:

1. Cunoașterea este dobândită de rețea din mediul său printr-un proces de învățare.

2. Conexiunile interneuronale, cunoscute sub denumirea de sinaptice, sunt utilizate pentru a stoca cunostiintele dobandite.

Procedura folosită pentru realizarea procesului de învățare se numește algoritm de învățare, scopul acestuia este de a modifica valorile sinaptice ale rețelei pentru a modela si a ajunge design-ul dorit. Modificarea valorilor sinaptice oferă metoda tradițională pentru proiectarea rețelelor neuronale. O astfel de abordare este cea mai apropiată de teoria filtrului adaptiv liniar.Cu toate acestea, este posibil pentru o rețea neuronala si modifica propriei topologii, care este motivat de faptul că neuronii în creierul uman pot muri și că noile conexiuni sinaptice pot crește. Rețelele neuronale sunt, de asemenea, menționate în literatură ca neurocomputere , procesoare parale distribuite, etc.

#### Beneficiile rețelelor neuronale

Este evident că o rețea neuronală își obține puterea de calcul prin, pe de o parte, structura distribuită masiv paralelă și, pe de alta parte, capacitatea sa de a învăța și, prin urmare, de a generaliza. Generalizarea se referă la rețeaua neuronală care produce rezultate rezonabile pentru intrările care nu au fost întâlnite în timpul instruirii (învățării). Aceste două capacități de procesare a informației fac posibil ca rețelele neuronale să rezolve probleme complexe (la scară largă) care sunt în prezent imposibil de rezolvat. În practică, totuși, rețelele neuronale nu pot oferi soluția prin lucrul individual. Mai degrabă, ele trebuie integrate într-o abordare inginerică sistemică consistentă. În mod specific, o problemă complexă de interes este descompusă într-un număr de sarcini relativ simple, iar rețelelor neuronale li se atribuie un subset de sarcini care se potrivesc cu capacitățile lor inerente. Este important să recunoaștem,cu toate acestea, avem o cale lungă de parcurs (înainte de a putea construi o arhitectură a calculatorului care să imite un creier uman).

Utilizarea rețelelor neuronale oferă următoarele proprietăți și capabilități utile:

1. Nonlinearitate. Un neuron artificial poate fi liniar sau neliniar. O rețea neurală, formată dintr-o interconectare a neuronilor neliniari, este ea însăși neliniară. În plus, aceasta este de o natură specială, în sensul că este distribuită în întreaga rețea. Nonlinearitatea este o proprietate extrem de importantă, în special dacă mecanismul fizic care stă la baza generării semnalului de intrare (de exemplu, semnalul de vorbire) este inerent neliniar.
2. Maparea intrărilor-ieșirilor. O paradigmă populară a învățării supervizata implică modificarea valorilor sinaptice ale unei rețele neuronale prin aplicarea unui set de eșantioane de instruire etichetate sau exemple de sarcini. Fiecare exemplu constă dintr-un semnal unic de intrare și un răspuns dorit corespunzător. Rețeaua este prezentată cu un exemplu extras din întâmplare din set și valorile sinaptice (parametrii liberi) ai rețelei sunt modificați pentru a minimiza diferența dintre răspunsul dorit și răspunsul real al rețelei produse de semnalul de intrare în conformitate cu un criteriu statistic adecvat. Formarea rețelei este repetată pentru numeroase exemple din set, până când rețeaua ajunge la starea de echilibru în care nu există alte modificări semnificative ale valorilor sinaptice. Exemplele de instruire aplicate anterior pot fi reaplicate în cursul sesiunii de formare, dar într-o ordine diferită. Astfel, rețeaua învață din exemple construind o maparedintre intrari si ieșiri pentru problema la îndemână. O astfel de abordare aduce în minte studiul deductiilor statistice nonparametrice; termenul "nonparametric" este folosit aici pentru a indica faptul că nu se fac presupuneri anterioare privind un model statistic pentru datele de intrare. De exemplu,daca luam o sarcină de clasificare a modelelor, în cazul în care cerința este de a atribui un semnal de intrare reprezentând un obiect sau eveniment fizic la una dintre mai multe categorii (clase) pre-specificate. Într-o abordare nonparametrică la această problemă, cerința este de a "estima" limitele arbitrare de decizie în spațiul semnalului de intrare pentru sarcina de clasificare a modelului utilizând un set de exemple și fără a invoca un model de distribuție probabilistă. Un punct de vedere similar este implicit în paradigma învățării supervizate, ceea ce sugerează o analogie apropiată între maparea intrare-ieșire efectuată de o rețea neuronală și deductiile statistice nonparametrice.
3. Adaptivitate. Rețelele neuronale au o capacitate încorporată de a adapta valorile lor sinaptice la schimbările din mediul înconjurător. În special, o rețea neurală instruită să opereze într-un mediu specific poate fi ușor recalificată pentru a face față schimbărilor minore în condițiile de funcționare a mediului. Mai mult, atunci când funcționează într-un mediu non-staționar (adică unul în care statisticile se schimba cu timpul), o rețea neuronală poate fi proiectată pentru a schimba valorile sale sinaptice în timp real. Arhitectura naturală a unei rețele neuronale pentru clasificarea modelelor, procesarea semnalelor și aplicațiile de control, împreună cu capacitatea adaptivă a rețelei, îl fac un instrument util în clasificarea adaptabilă a modelelor, procesarea adaptivă a semnalelor și controlul adaptiv. Ca regulă generală, se poate spune că cu cât mai adaptabil facem un sistem, asigurându-ne tot timpul stabilitatea sistemului, cu atât performanța sa va fi mai robustă atunci când sistemul este obligat să funcționeze într-un mediu nestationar. Trebuie subliniat totuși că adaptivitatea nu duce întotdeauna la robustețe;aceasta poate face exact opusul. De exemplu, un sistem adaptiv cu constante de timp scurte se poate schimba rapid și, prin urmare, tinde să răspundă la perturbații false, provocând o degradare drastică a performanțelor sistemului. Pentru a realiza beneficiile integrale ale adaptivității, constantele de timp principale ale sistemului ar trebui să fie suficient de lungi pentru ca sistemul să ignore perturbațiile falsificate și totuși suficient de scurt pentru a răspunde schimbărilor semnificative din mediul înconjurător; problema descrisă aici este menționată ca dilema stabilitatii-plasticitatii (Grossberg, 1988b).

4. Raspunderea evidentiala

În contextul clasificării modelului, o rețea neurală poate fi proiectată pentru a furniza informații, nu numai despre ce șablon particular va fi selectat, ci și despre încrederea avută în decizia luată. Aceste informații din urmă pot fi folosite pentru a respinge șabloanele ambigue, în cazul în care apar, și evident pentru a îmbunătăți performanța rețelei.

5. Informații contextuale

Cunoașterea este reprezentată de structura și gradul de activare al unei rețele neuronale. Fiecare neuron din rețea poate fi afectat de activitatea globală a tuturor celorlalți neuroni din rețea. În consecință, o rețea neuronală va trata în mod natural informațiile textuale.

6. Toleranța la defecțiuni

O rețea neuronală, în formă hardware, are potențialul de a fi moștenit toleranța la erori sau capacitatea de a computaționa în mod robust, în sensul în care performanța sa scade în condiții de funcționare nefavorabile. De exemplu, dacă un neuron sau legăturile sale sunt deteriorate, reapelarea unui șablon stocat va avea calitatea afectată. Cu toate acestea, datorită distribuției informației stocate în rețea, daunele trebuie să fie majore înainte ca răspunsul general al rețelei să fie degradat serios. În principiu, o rețea neuronală prezintă o degradare treptată a performanței, mai degrabă decât un eșec brusc și catastrofal. Există anumite dovezi empirice referitoare la computaționarea robustă, însă de obicei este necontrolată. Pentru a ne asigura că rețeaua neuronală tolerează erori, ar fi necesar să se ia măsuri corective în proiectarea algoritmilor utilizați pentru instruirea rețelei. (Kerlirzinși Vallet, 1993).

7. Implementabilitatea VLSI

Natura paralelă a unei rețele neuronale o face potențial mai rapidă pentru a computaționa anumite sarcini. Aceeași caracteristică face o rețea neuronală foarte potrivită pentru implementarea utilizând tehnologii (VLSI) integrate pe scară o largă. O calitate benefică a VLSI este că oferă o modalitate de a capta comportamente cu adevărat complexe într-un mod ierarhic (Mead, 1989).

8. Uniformitatea analizei și designului

În principiu, rețelele neuronale se bucură de tehnologii universale, precum procesoarele de informații. Spunem acest lucru în sensul că se utilizează aceeași notație în toate domeniile care implică aplicarea rețelelor neuronale. Această caracteristică se manifestă în diverse feluri:

• Neuronii, într-o formă sau alta, reprezintă un ingredient comun tuturor rețelelor neuronale.

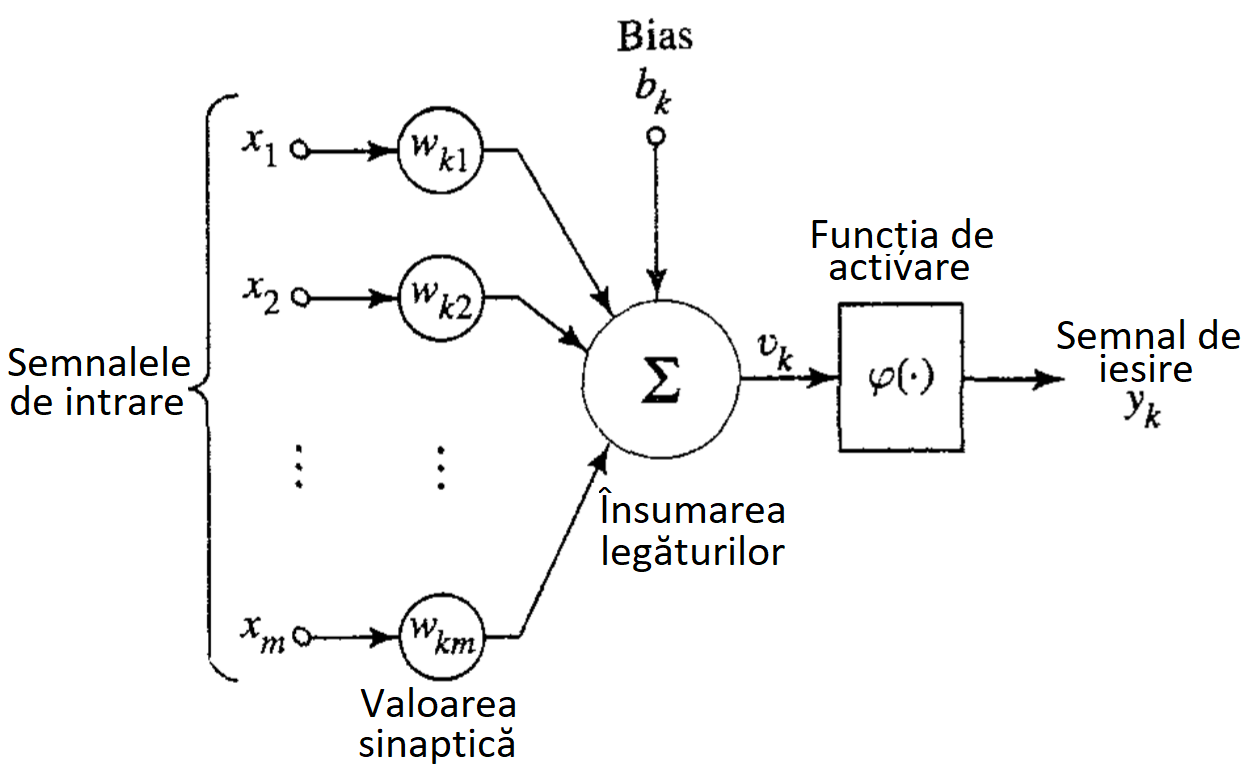
• Fiind atât de comuni, este posibilă partajarea teoriilor și algoritmilor de învățare în diferite domenii ale rețelelor neuronale.

• Rețelele modulare pot fi construite printr-o integrare fără întreruperi a modulelor.

9. Analogie neurobiologică

Creierul uman stă la baza designului unei rețele neuronale. Acest lucru dovedește faptul că prelucrarea paralelă tolerantă la erori este nu numai fizic posibil, dar și rapidă și puternică. Neurobiologii utilizează, într-o manieră artificială, rețele neuronale ca un instrument de cercetare pentru interpretarea fenomenelor neurobiologice. Pe de altă parte, inginerii folosesc neurobiologia pentrua găsi idei noi de rezolvare a unor probleme mai complexe decât cele bazate pe designul convențional, tehnic.

### Modelul unui neuron



*Modelul nonlinear al unui neuron*

Un neuron este o unitate de procesare a informațiilor care este fundamentală pentru funcționarea unei rețele neuronale. Diagrama de mai sus prezintă modelul unui neuron, care formează baza pentru proiectarea rețelelor neuronale (artificiale). Aici identificăm trei elemente de bază ale modelului neuronal:

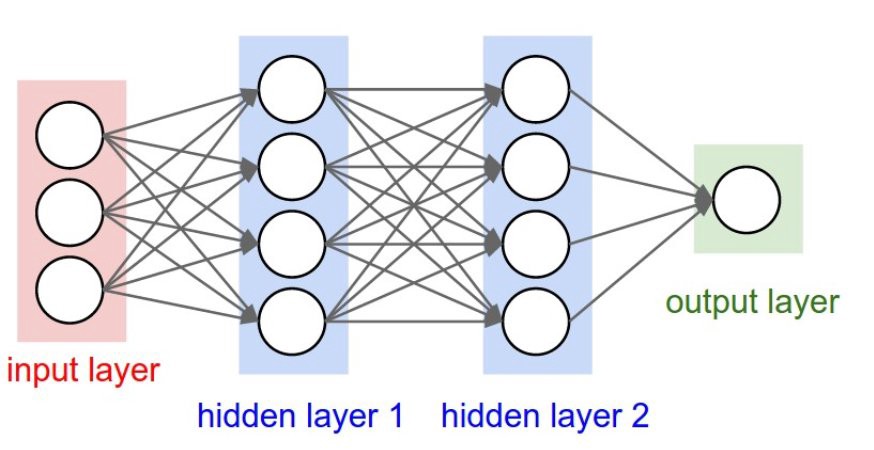
1. Un set de sinapse, fiecare caracterizată printr-o valoare sau putere proprie. În mod specific, un semnal xj la intrarea sinapsei j conectat la neuron k este înmulțit cu greutatea sinaptică wkj. Este important să notăm modul în care sunt scrise indiciile greutății sinaptice wkj. Prima coloană se referă la neuronul în cauză și a doua coloană indică capătul de intrare al sinapselor la care se referă valoarea. Spre deosebire de o sinapsă a creierului uman, valoarea sinaptică a unui neuron artificial poate fi într-un interval care include valori negative, precum și valori pozitive.

2. Un nucleu pentru însumarea semnalelor de intrare, ponderate de sinapsele respective ale neuronului; operațiile descrise aici constituie un combinator liniar.

3. Funcție de activare pentru limitarea amplitudinii ieșirii unui neuron. Funcția de activare este de asemenea menționată ca o funcție de strivire prin faptul că scade (limitează) domeniul admis de amplitudine al semnalului de ieșire la o anumită valoare finită.

### Modelul deep learning

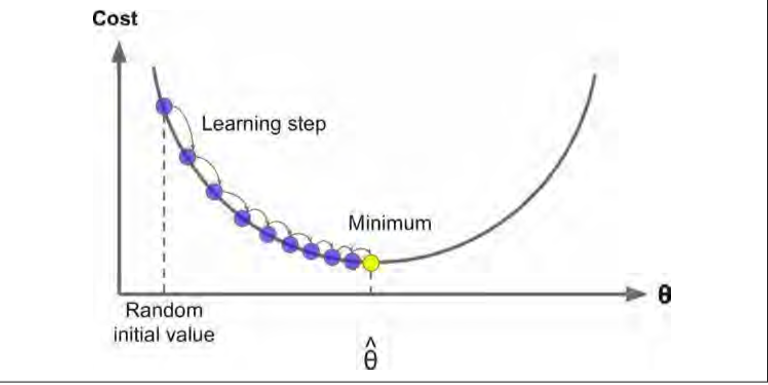
Modelele deep learning sunt construite folosind rețele neuronale. O rețea neurală ia intrările, care sunt apoi procesate în numeroare straturi numite „ascunse„ folosind valori sau greutăți care sunt ajustate în timpul antrenamentului. Apoi, modelul face o predicție. Valorile sunt ajustate pentru a găsi modele ce pot face predicții mai bune. Utilizatorul nu trebuie să specifice tiparele de cautare - rețeaua neurală învață singură.



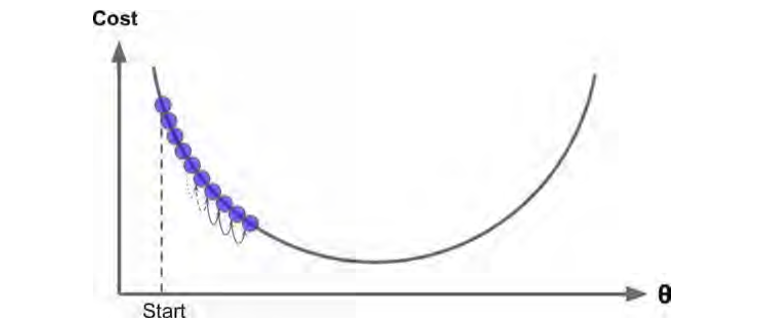
### Gradient Descent

Gradient Descent este un algoritm generic, capabil să găsească soluțiile optimale pentru o gamă largă de probleme. Ideea generală a algoritmului Gradient Descent este de a optimiza iterativ parametrii pentru a minimiza o funcție de cost. Să presupunem că v-ați pierdut în munți și este o ceață densă. O strategie bună de a ajunge cât mai repede în partea de jos a văii este să mergeți în direcția celei mai abrupte pârtii. Același lucru îl presupune și algoritmul Gradient Descent: măsoară gradientul local al funcției de eroare în ceea ce privește vectorul parametru θ, mergând în direcția unui gradient descendent. Odată ce gradientul este zero, s-a atins un minimum.

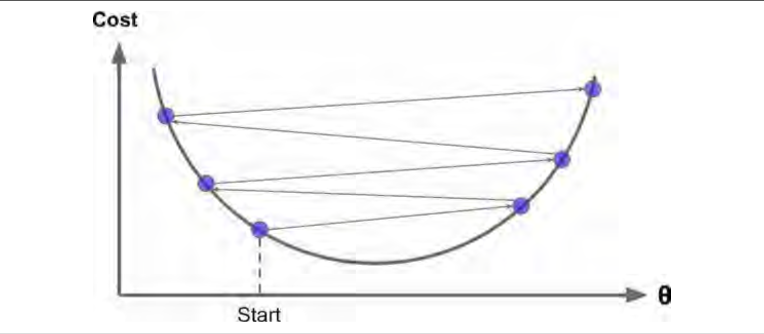
Concret, se începe prin a da valori aleatorii lui θ (această acțiune se numește inițiere aleatorie). Apoi se îmbunătățește treptat, fiecare pas încercând să scadă funcția de cost (de exemplu MSE), până când algoritmul converge la un nivel minim.



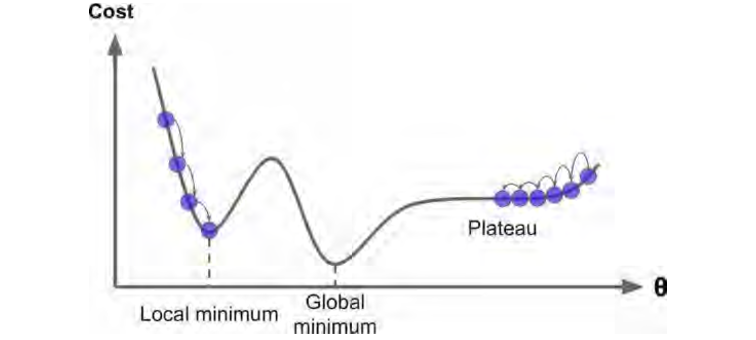
Un parametru important al algorimului Gradient Descent este dimensiunea pașilor determinată de rata de învățare. Dacă rata de învățare este prea scăzută, atunci algoritmul va avea de executat multe iterații pentru a converge, iar acest lucru va dura un timp îndelugat.



Pe de altă parte, dacă rata de învățare este prea ridicată, s-ar putea ca algoritmul să diveargă cu valori din ce în ce mai mari, eșuând să găsească o soluție bună.

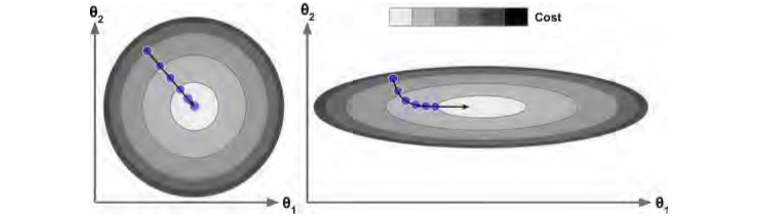


Însă, nu toate funcțiile de cost arată precum niște boluri normale. Ar putea apărea găuri și tot felul de suprafețe iregulate, făcând convergența către minimum foarte dificilă. Următoarea figură ne arată două dintre principalele provocări ale algoritmului Gradient Descent: dacă inițializarea aleatoare pornește algoritmul din stânga, atunci va converge la un minim local, care nu va fi la fel de bun ca minimul global. Dacă va începe din dreapta, atunci va dura un timp foarte lung pentru a traversa platoul, iar dacă se va opri prea devreme, nu va ajunge niciodată la minimul global.



Din fericire, funcția costului MSE pentru un model de regresie liniară este o funcție convexă, ceea ce înseamnă că dacă alegeți două puncte pe curbă, segmentul de linie care le unește nu va tăia niciodată curba. Aceasta implică faptul că nu există minime locale, doar un minim global. De asemenea, este o funcție continuă cu o pantă care nu se schimbă niciodată brusc. Aceste lucruri au o mare consecință: algoritmul Gradient Descent se va apropia garantat arbitrar de minimul global (dacă se va aștepta destul de mult și dacă rata de învățare nu este prea mare).

De fapt, funcția de cost are forma unui bol, dar poate fi și un castron alungit dacă caracteristicile sunt scalate foarte diferit. Figura de mai jos, prezintă algoritmul Gradient Descent pe un set de antrenament în care caracteristicile 1 și 2 au aceeași scală (în stânga) și un set de antrenament în care caracteristica 1 are valori mult mai mici decât caracteristica 2 (în partea dreaptă).



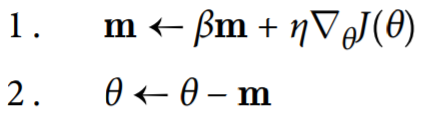
*Problema bolului alungit*

Precum se poate observa, algoritmul Gradient Descent, în cazul din stânga, ajunge rapid până la minimum, spre deosebire de cazul din dreapta, în care prima dată merge într-o direcție aproape ortogonală cu direcția minimului global, ajungând aproape pe o zonă netedă. Va ajunge ulterior la un minim, însă după o perioadă îndelungată de timp.

Diagrama ilustrează, de asemenea, faptul că învățarea unui model înseamnă găsirea unei combinații de parametri, care vor minimiza funcția de cost. Este o căutare în spațiul parametrilor modelului: cu cât sunt mai mulți parametri pe care un model îi are, cu atât mai multe dimensiuni are acest spațiu și cu atât mai mare este căutarea.

### Momentum

Imaginați-vă o minge de bowling care se rostogolește pe o pantă blândă pe o suprafață netedă: se va porni încet, dar viteza va crește rapid până când va ajunge în cele din urmă la viteza terminală (dacă există o anumită fricțiune sau rezistență la aer). Aceasta este ideea foarte simplă din spatele optimizării Momentum, propusă de Boris Polyak în 1964. În schimb, Gradient Descent va lua pur și simplu mici pași pe pantă, așa că va dura mai mult timp pentru a ajunge la optim. Reamintim că Gradient Descent nu ține cont de de gradientele anterioare. Dacă gradientul local este mic, merge foarte lent. Strategia optimizării Momentum ține foarte mult cont de gradientele anterioare: la fiecare iterație, se adaugă gradientul local vectorului impuls m (înmulțit cu rata de învățare η) și actualizează greutățile prin simpla scădere a vectorului de impuls. Cu alte cuvinte, gradientul este folosit ca accelerare, nu ca viteză. Pentru a simula un fel de mecanism de frecare și pentru a împiedica creșterea dinamică a impulsului, algoritmul introduce un nou hiperparametru β, pur și simplu numit impuls, care trebuie să fie setat între 0 (fricțiune mare) și 1 (fără frecare). Valoarea momentului tipic este de 0,9.

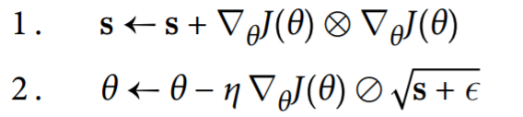
Ecuația algoritmului Momentum:

Gradient Descent coboară pe panta abruptă destul de repede, dar apoi durează foarte mult timp pentru a merge în jos în vale. În contrast, optimizatorul Momentum va derula în josul văii din ce in ce mai rapid până când ajunge la optim.

Singurul dezavantaj al optimizării Momentum este că adaugă încă un alt hiperparametru. Cu toate acestea, valoarea momentului de 0.9, de obicei, funcționează bine în practică și aproape întotdeauna merge mai repede decât Gradient Descent.

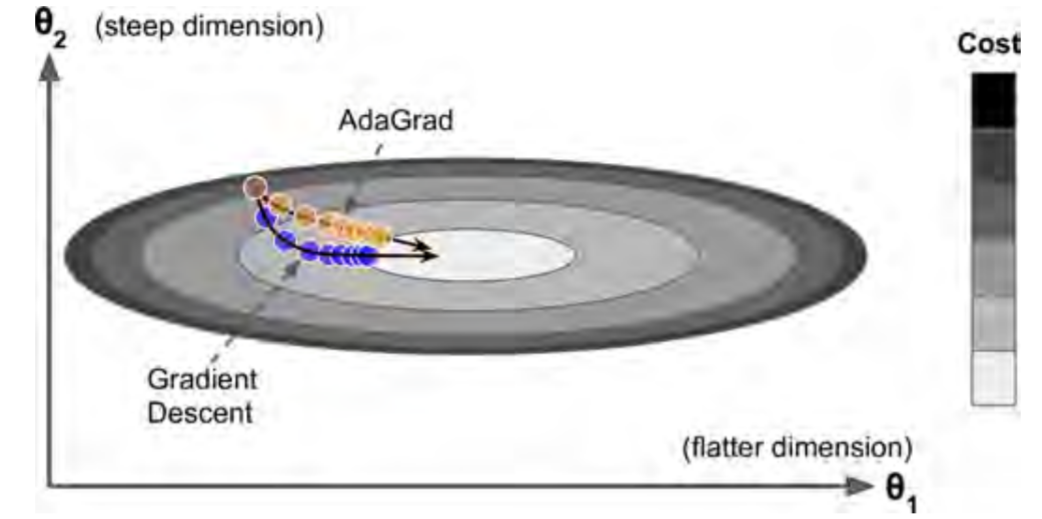
### AdaGrad

Luați în considerare problema bolului alungit: Gradient Descent începe prin a merge rapid pe cea mai abruptă pantă, apoi se duce încet pe fundul văii. Ar fi drăguț dacă algoritmul ar putea detecta acest lucru mai devreme și corecta direcția sa de a puncta un pic mai mult spre optimul global. Algoritmul AdaGrad realizează acest lucru prin scalarea în jos a vectorului de gradient de-a lungul cele mai abrupte dimensiuni.

Ecuatia algoritmului AdaGrad:

Primul pas acumulează pătratul gradientilor în vectorul s (simbolul ⊗ reprezintă multiplicarea elementului). Această formă vectorizată este echivalentă cu a calcula fiecare element si ← si + (∂/∂ θiJ(θ))2  a vectorului s; cu alte cuvinte,fiecare si acumulează pătratele derivatelor parțiale a funcției de cost cu privire la parametrul θi. Dacă funcția de cost este abruptă de-a lungul dimensiunii i, atunci si va deveni mai mare și mai mare la fiecare iterație.

Al doilea pas este aproape identic cu Gradient Descent, dar cu o mare diferență:vectorul de gradient este micșorat cu un factor de (simbolul ⊘ reprezintă împărțirea, și ε este un termen de netezire pentru a evita împărțirea la zero, de obicei setat la 10-10). Această formă vectorizată este echivalentă cu cea de calculare a lui θi ←θi − η ∂/∂θi J(θ) / pentru toti parametrii θi.

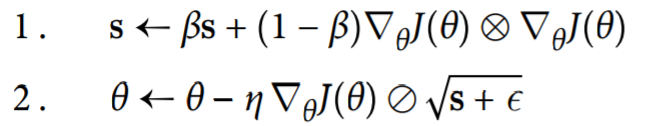
Pe scurt, acest algoritm descompune rata de învățare, dar se face mai rapid pentru dimensiuni abrupte decât pentru dimensiuni cu pante mai blânde. Aceasta se numește o rată de învățare adaptivă. Ajută la direcționarea actualizărilor rezultate mai direct spre optimul global. Un avantaj suplimentar constă în faptul că necesită mult mai puțin reglarea hiperparametrului pentru rata de învățare η.

*Diferența dintre algoritmii AdaGrad si Gradient Descent*

*AdaGrad funcționează adesea bine pentru probleme simple, dar, din păcate, se oprește adesea prea devreme în procesul de învățare a rețelelor neuronale. Rata de învățare se reduce atât de mult încât algoritmul se oprește în întregime înainte de a ajunge la optimul global (poate fi eficient pentru sarcini mai simple, cum ar fi Regresia Lineară, totuși).*

### RMSProp

#### Deși AdaGrad încetinește puțin prea repede și nu converge niciodată la optimul global, algoritmul RMSProp fixează acest lucru prin acumularea doar a gradientilor din cele mai recente iterații (spre deosebire de toate gradientele de la începutul antrenamentului). Ea face acest lucru prin folosirea decăderii exponențiale în primul pas.

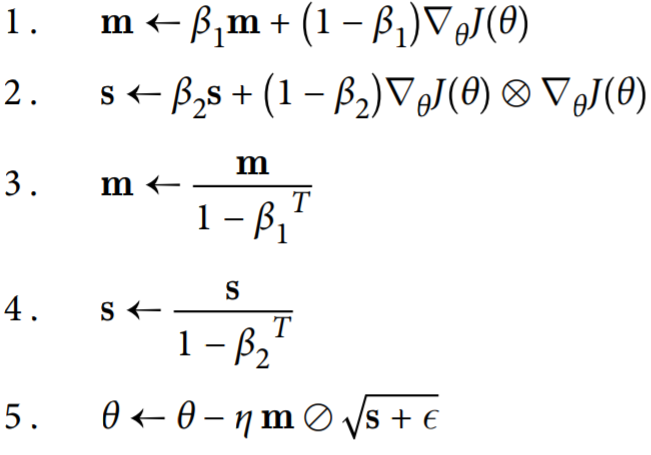
Ecuatia algoritmuluiRMSProp**:**

Rata de decădere β este de obicei stabilită la 0,9. Da, este încă o dată un nou hiperparametru, dar această valoare implicită funcționează adesea bine, deci este posibil să nu fi nevoie deloc de ajustarea acestuia.

Cu excepția problemelor foarte simple, acest optimizator aproape întotdeauna se comportă mult mai bine decât AdaGrad. În general, acesta performanță este mai bună și decât a optimizării Momentum.De fapt, a fost algoritmul de optimizare preferat al multor cercetători până la descoperirea optimizatorului Adam.

# Adam Optimization

Numele de Adam vine de la estimarea momentului adaptiv ( eng: adaptive moment estimation), combină ideile optimizării Momentum și RMSProp: la fel ca optimizarea Momentum, ține evidența unei medii exponențial decalate a gradienților anteriori și, la fel ca RMSProp, ține cont de media diferențelor exponențiale ale pătratelor gradienților anteriori.

Ecuatia algoritmului Adam:

Dacă ne uităm doar la pașii 1, 2 și 5, putem observa asemănarea apropiată a lui Adam atât cu optimizarea Momentum, cât și cu RMSProp. Singura diferență constă în faptul că pasul 1 calculează mai degrabă o medie exponențială decât o sumă exponențială, dar acestea sunt de fapt echivalente, cu excepția unui factor constant (media decăderii este de 1 - β1 ori mai mare decât suma decăderii). Etapele 3 și 4 sunt oarecum de detaliu tehnic: deoarece m și s sunt inițializate la 0, ele vor avea o valoare foarte apropiata de 0 la începutul antrenamentului, astfel încât acești doi pași vor ajuta la mărirea parametrilor m și s la începutul antrenamentului.

Hiperparametrul β1 de degradare a impulsului este de obicei inițializat la 0.9, în timp ce hiperparametrul β2 de descompunere a scalării este adesea inițializat la 0.999. Ca și mai devreme, termenul de netezire ε este de obicei inițializat la un număr mic, cum ar fi 10-8. De fapt, din moment ce Adam este un algoritm de adaptare a ritmului de învățare (cum ar fi AdaGrad și RMSProp), acesta necesită o reglare mai mică a hiperparametrului rata de învățare η. Se poate folosi de multe ori valoarea implicită η = 0.001, făcând Adam mai ușor de utilizat decât Gradient Descent.

#### 

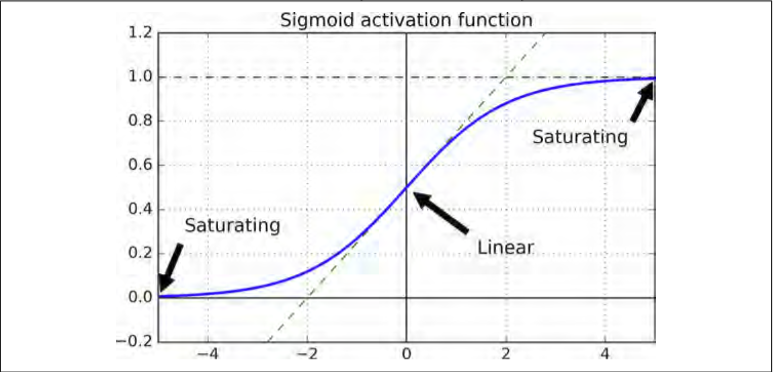
#### Problemele de eliminare/explodare a gradienților

Algoritmul de backpropagation funcționează trecând de la stratul de ieșire la stratul de intrare, propagând gradientul de eroare odată cu trecerea sa. Odată ce algoritmul a calculat gradientul funcției de cost cu privire la fiecare parametru din rețea, utilizează acești gradienți pentru a actualiza fiecare parametru cu pas Gradient Descent.

Din păcate, gradienții de obicei devin mai mici pe măsură ce algoritmul avansează către straturile inferioare. Ca o consecință, actualizarea Gradient Descent păstrează greutățile de legătură ale stratului inferior practic neschimbate, iar învățarea nu va converge niciodată la o soluție bună. Aceasta se numește problema de decolorare. În unele cazuri, se poate întâmpla contrariul: gradiențele pot să crească și mai mari, astfel încât multe straturi să primească actualizări cu greutate mare și algoritmul să se diferențieze. Aceasta este problema de eliminare a gradienților, care se întâlnește în majoritatea rețelelor neuronale recurente. În unele cazuri, se poate întâmpla contrariul: gradienții pot crește foarte mult, așa încât straturile vor primi actualizări imense referitoare la greutate și algoritmul va diverge. Aceasta se numește problema de explodare a gradienților, care poate fi întâlnită în majoritatea rețelelor neuronale recurente. Generalizând, rețelele neuronale profunde pot suferi de gradienți instabili, deci straturile diferite învăță la viteze foarte diferite.

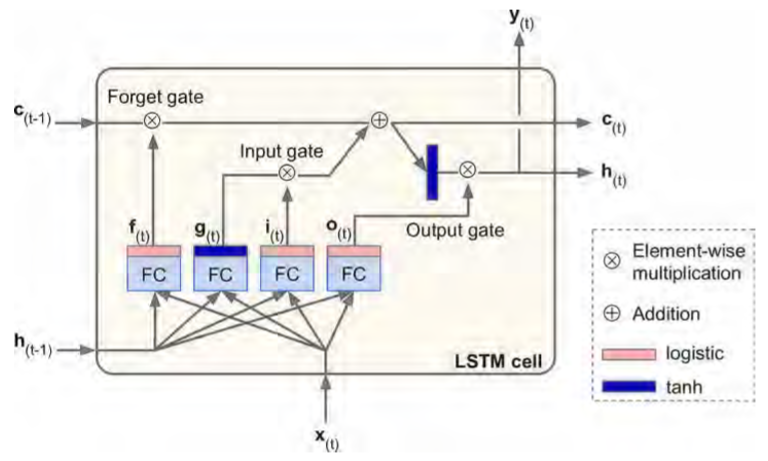
Deși acest comportament nefericit a fost empiric observat pentru o perioadă îndelungată de timp, fiind unul dintre motivele pentru care rețelele neuronale profunde nu au mai fost studiate o perioadă, abia în jurul anului 2010 s-au înregistrat progrese semnificative în înțelegerea acestuia. O lucrare intitulată “Understanding the Difficulty of Training Deep Feedforward Neural Networks”, realizată de Xavier Glorot și Yoshua Bengio a găsit câteva cauze precum: combinația dintre funcția logică populară de activare a sigmoidului și tehnica de inițiere a greutății (care a fost cea mai populară la acea dată), inițializare folosind o distribuție normală cu o medie de 0 și o abatere standard de 1. Pe scurt, au arătat că, prin această funcție de activare și această schemă de inițializare, variația ieșirilor fiecărui strat este mult mai mare decât variația intrărilor sale. Mergând înainte în rețea, varianța continuă să crească după fiecare strat, până când funcția de activare saturează în straturile superioare. Acest fapt este înrăutățit de funcția logistică, deoarece aceasta are o medie de 0.5, nu 0 (funcția hiperbolică tangentă are o medie de 0 și se comportă puțin mai bine decât funcția logistică în rețele profunde).

Privind la funcția de activare logistică, din figura de mai jos, se poate observa că atunci când intrările devin mari (negative sau pozitive), funcția saturază la 0 sau 1, cu un derivat extrem de apropiat de 0. Astfel, atunci când backpropagation funcționează, nu are practic niciun gradient de propagat înapoi prin rețea, și numărul mic de gradienți care există continuă să se dilueze, deoarece backpropagation progresează în jos prin straturile de superioare, astfel încât nu va mai exista nimic pentru straturile inferioare.



Celula LSTM

Celula de memorie pe termen scurt (eng: The Long Short-Term Memory) (LSTM) a fost propusă în 1997 de Sepp Hochreiter și Jürgen Schmidhuber și a fost îmbunătățită treptat de-a lungul anilor de câțiva cercetători, cum ar fi Alex Graves, Hașim Sak, Wojciech Zaremba și multe altele. Dacă luați în considerare celula LSTM ca o cutie neagră, aceasta poate fi folosită foarte mult ca o celulă de bază, cu excepția faptului că va funcționa mult mai bine; formarea va converge mai rapid și va detecta dependențele pe termen lung în date.

Deci, cum funcționează o celulă LSTM? Mai jos este prezentată arhitectura unei celule LSTM de bază.

*Arhitectura celului LSTM*

Ideea de bază este că rețeaua poate învăța ce trebuie să stocheze pe termen lung, ce ar trebui să arunce și ce să citească de la ea. Pe măsură ce starea c(t-1) traversează rețeaua de la stânga la dreapta, puteți observa că mai întâi trece printr-o așa-numită poartă de uitare, renunțând la unele amintiri și apoi adaugă câteva amintiri noi prin operația de adăugare (amintirile selectate de o poartă de intrare). Rezultatul c(t) este trimis direct, fără nicio altă transformare ulterioară. Deci, la fiecare pas, unele amintiri sunt abandonate si cateva amintiri sunt adaugate. Mai mult, după operația de adăugare, starea este copiată și trecută prin funcția tanh, iar rezultatul este filtrat de poarta de ieșire. Aceasta produce starea h(t) (care este egală cu rezultatul celulei pentru acest moment de timp y(t)). Mai departe, voi prezenta de unde provin amintirile noi și cum funcționează porțile.

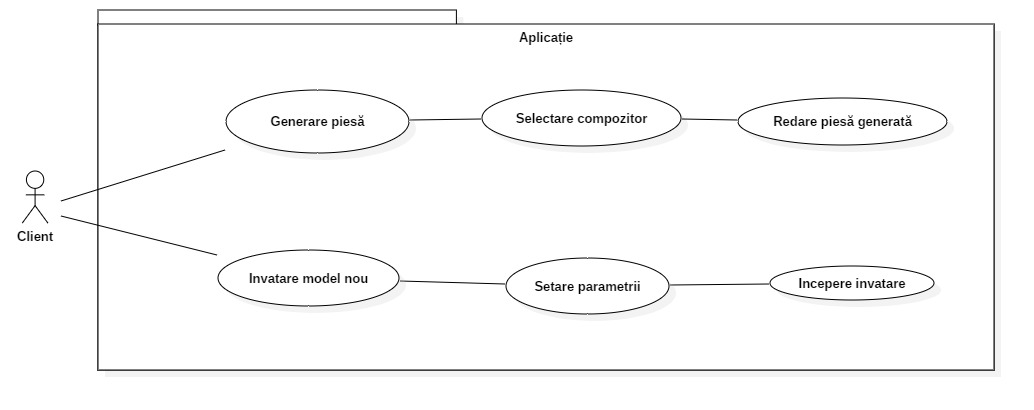
# 4.Studiu de caz

## 4.1 Functionalitati

## 4.2 Analiza si proiectare

### 4.2.1 Diagrama cazurilor de utilizare

În continuare voi prezenta diagrama cazurilor de utilizare care conține funcționalitățile aplicației, menționate mai sus.



*Diagrama cazurilor de utilizare*

Aplicația va fi folosită de către un singur tip de utilizatori, respectiv Clientul-ul. Aceasta permite utilizatorului să aleaga daca dorește să folosească un model deja invățat pentru a genera o piesa sau dacă dorește sa seteze parametrii pentru invățarea unui nou model.

### 4.2.2 Diagramele cazurilor de interacțiune

Următorul pas în realizarea aplicației este crearea **diagramelor de interacțiune** pentru principalele cazuri de utilizare.

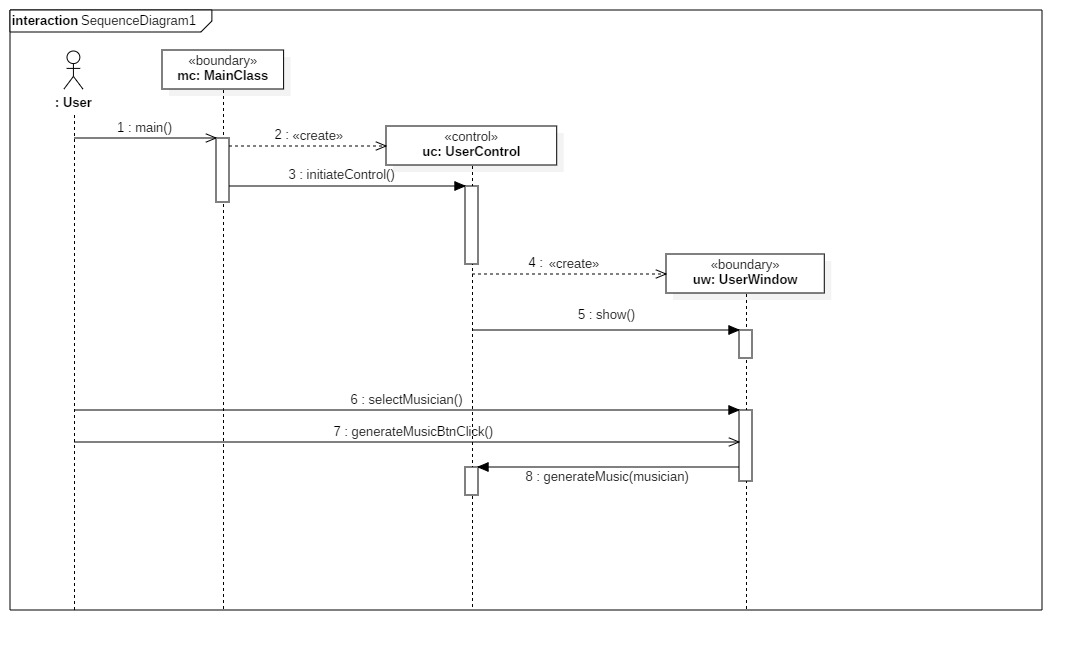
### G**enerarea unei piese**

#### 

*Diagrama de interactiune- Generarea unei piese*

La pornirea aplicației de către utilizator, se apelează metoda principală care creează un obiect de tip control – UserControl și îl inițializează. Acesta creează la rândul lui obiectul de tip GUI – UserWindow si il afișează.După apăsarea butonului „Generate Music”, utilizatorului îi va apărea pe ecran o lista de interpreți din care poate ale. După selectarea interpretului de către utilizator si acționarea butonului „Generate” are loc creearea piesei care ii va apărea pe ecran, cu posibilitatea de a o reda. În funcție de muzicianul ales, piesa generata va avea anumite influențe de la lucrările acelui compozitor.

### Învățarea unui nou model



*Diagrama de interacțiune - Învățarea unui nou model*

La fel ca la generarea unei noi piese la pornirea aplicației de către utilizator, se apelează metoda principală care creează un obiect de tip control – UserControl și îl inițializează. Acesta creează la rândul lui obiectul de tip GUI – UserWindow si il afișează. După apăsarea butonului „Create new model”, utilizatorului îi va apărea anumite câmpuri legate de parametrii modelului. După setarea fiecărui parametru si acționarea butonului „Start” se va incepe invățarea unui nou model. La sfârșitul procesului, utilizatorul poate genera o piesă folosind model creat.

## 4.3 Implementare

Pentru implementarea rezolvarii problemei am ales sa folosesc Python in combinatie cu libraria Tensorflow.

### Python

### 

Python este un limbaj de programare interpretat, de nivel înalt, cu scop general. Creat de Guido van Rossum și lansat pentru prima dată în 1991, filosofia designului Python accentuează lizibilitatea codului cu utilizarea notabilă a spațiului alb. Construcțiile lingvistice și abordarea orientată pe obiecte vizează să ajute programatorii să scrie cod clar, logic pentru proiectele mici și la scară largă. Acesta este tipizat dinamic și colectat de gunoi. Acesta susține mai multe paradigme de programare, inclusiv programare orientată pe obiecte și funcțională.

Acesta era descris de Guido van Rossum in cartea sa “python reference manual” in felul urmator: “Acesta este un limbaj de programare simplu, dar puternic, interpretat, care umple golul dintre limbajul C si programarea shell. Este ideal pentru prototiparea rapida si ‘’throw-away programming’’. Sintaxa lui este realizata prin punerea impreuna a constructiei unei varietati de alte limbaje, cele mai proeminente fiind ABC, C, Modula-3, Icon. Interpretorul din Python este usor extins cu noi functionalitati si tipuri de date implementate in C. Acesta este de asemenea potrivit ca limbaj de extensie pentru aplicatii C foarte personalizabile”.

Tensorflow

TensorFlow este un sistem de machine learning care funcționează la scară largă și în medii eterogene. Tensor-Flow folosește grafice de fluxuri de date pentru a reprezenta calculele, starea comună și operațiile care mută acea stare. Maparea nodurilor unui grafic de flux pe multe mașini dintr-un grup și într-o mașină pe mai multe dispozitive computerizate, inclusiv procesoare multicore, GPU-uri de uz general și ASIC-uri proiectate la comandă, cunoscute sub denumirea de Unități de procesare a tensorului (TPU). Această arhitectură oferă flexibilitate dezvoltatorului de aplicații: în timp ce în modelele anterioare de "server de parametri" gestionarea stării partajate este integrată în sistem, TensorFlow permite dezvoltatorilor să experimenteze cu noi optimizări și algoritmi de antrenament. TensorFlow susține o varietate de aplicații, cu accent pe instruire și inferență asupra rețelelor neuronale profunde. Mai multe servicii Google folosesc TensorFlow în producție, fiind lansat ca un proiect open-source și a devenit utilizat pe scară largă pentru cercetarea în domeniul machine learning.

Music21

Music21 este un set de instrumente pentru analizarea, căutarea și transformarea muzicii în forme simbolice .Abordarea modulară a proiectului permite muzicienilor și cercetătorilor să scrie rapid scenarii simple și să le reutilizeze în alte proiecte. Setul de instrumente își propune să ofere instrumente puternice software integrate cu cunoștințe muzicale sofisticate atât muzicienilor cu experiență mică de programare (în special muziciologi), cât și programatorilor cu abilități modeste de teorie a muzicii.

NumPy

NumPy scurt pentru Numerical Python, este pachetul de bază pentru calculul științific în Python. Acesta oferă, printre altele: Un rapid și eficient Multidimensional array object ndarray. Funcții pentru efectuarea calculelor elenlent-wise cu matrice sau matematice operațiuni între matrice. Instrumente pentru citirea și scrierea seturilor de date bazate pe array pe disc. Operații de algebră liniară, transformari Fourier și generare de numere aleatorii. Instrumente pentru integrarea codului de conectare C, C ++ și Fortran în Python. Dincolo de capacitățile de procesare rapidă a matricelor pe care NumPy le adaugă Python-ului scopurile primare în ceea ce privește analiza datelor este ca si container primar pentru stoarea eficienta și manipularea, mult mai efficient in comparative cu celelalte tipuri de date din Python. De asemenea, bibliotecile scrise într-un limbaj inferior, cum ar fi C sau Fortran, pot opera pe datele stocate într-o matrice NumPy fără a copia orice date.

## 4.4 Validare si testare

## 4.5 Validare experimentala

Ca si set de date am folosit o colectie de piese de muzica clasica pentru pian in format midi ce contine diferiti compozitori: Isaac Albéniz, Johann Sebastian Bach, Ludwig van Beethoven, Frédéric Chopin, Wolfgang Amadeus Mozart, etc.

Formatul MIDI

MIDI nu este un dispozitiv specific, ci un acord internațional standard pentru conectarea a două sau mai multe electronice instrumente. Linkul MIDI conține informații despre orele de expediere începutul și terminarea unei note, a graficului și a altor date astfel încât, pentru de exemplu, un organ electronic sau un PC obișnuit cu drept software-ul poate "declanșa" alte echipamente. Un instrument tipic ca un sintetizator ar putea fi capabil nu numai de a genera date MIDI, dar de asemenea, să primească MIDI dintr-o altă sursă și apoi să o transmită unui a instrument suplimentar.

Modelarea datelor

Pentru reprezentarea datelor in memorie am folosit music21, ceea ce mi-a permis sa iterezez notele fiecarei piese. Astfel, dintr-un fisier de tip midi rezulta o lista de note sau acorduri.

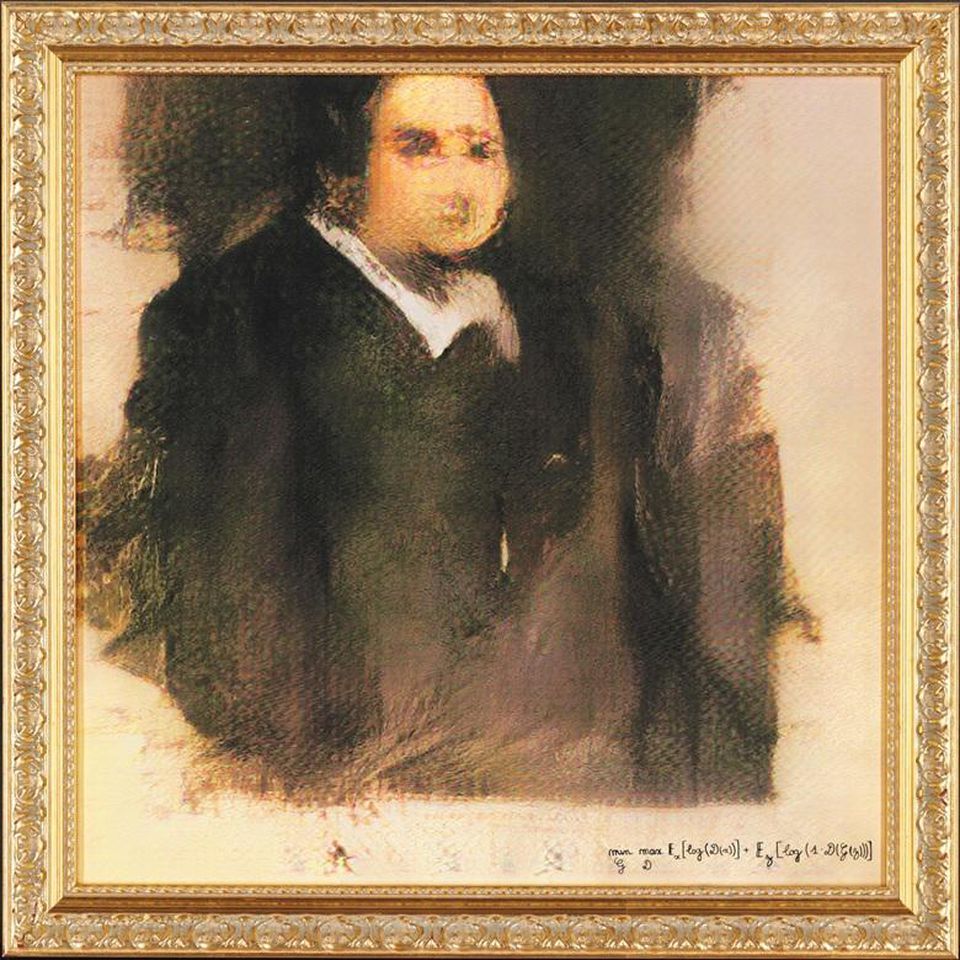
Codificarea unică

Codificarea unică este cea mai obișnuită și cea mai elementară metodă de a transforma un token într-un vector. Se compune din asocierea unui indice întreg unic cu fiecare cuvânt și transformând apoi acest index întreg într-un vector binar de mărimea N (dimensiunea vocabularului); vectorul este tot cu zero, cu excepția celei de-al i-lea element, care este 1.

In cazul nostru notele muzicale sunt dispuse pe 8 octave, fiecare octava avand 12 note: C,C#,D,E-,E,F,F#,G,G#,A,B-,B, plus o nota C adionala, rezultand un total de 97 de note. Prin urmare, fiecare nota va fi reprezentata printr-un vector de lungime 97 in care doar indicele notei va avea valoarea 1, restul valorilor fiind setate la 0. Deoarece, pot exista si note si acorduri, pentru fiecare nota sau acord voi avea un numar fix, mai exact 5, ce reprezinta numarul notelor cantate simultan. In acest mod, un acord va putea avea maxim 5 note reprezentate prin 5 codificari binare, iar in cazul in care este o singura nota sau mai putin de 5, vectorii nefolositi vor avea in intregime valoarea 0. In concluzie, fiecarea nota sau acord este reprezentata printr-o matrice de dimensiune 5x97.

Arhitectura dezvoltata

Ca si arhitectura folosita, inspirat de povestea portretului lui Edmond Belamy, creat de un GAN (Generative Adversarial Network) si vandut pentru $432,500, m-am decis sa il folosesc si pentru problema mea.



Principalul avantaj al utilizării rețelelor adversare generatoare (GAN) este că, atunci când funcționează, funcționează foarte bine, după cum au arătat documentele recente care produc imagini foarte realiste ale fețelor, scaunelor și animalelor. Motivul pentru aceasta este că obiectivul optimizat de GAN-uri - de a genera date artificiale care nu pot fi diferențiate de datele reale de către o altă rețea neuronală - este foarte aliniată cu scopul de a produce date realiste. Pe lângă faptul că au un obiectiv mai bun, GAN-urile nu necesită multe calcule de probabilitate anterioare și posterioare, adesea necesare pentru o altă abordare concurențială, probabilitate maximă.

Cu toate acestea, marele dezavantaj este că aceste rețele sunt foarte greu de antrenat. Funcția pe care aceste rețele încearcă să o optimizeze este o funcție de pierdere care, în esență, nu are nicio formă închisă (spre deosebire de funcțiile de pierdere standard, cum ar fi pierderea logică sau eroarea pătrată). Astfel, optimizarea acestei funcții de pierdere este foarte dificilă și necesită multă încercare și eroare în ceea ce privește structura rețelei și protocolul de instruire. Deoarece RNN-urile sunt în general mai stricte decât CNN-urile, foarte puțini au putut să aplice GAN-uri la ceva mai complex decât imagini, cum ar fi textul sau vorbirea.

# 5.Concluzii

# 6.Bibliografie

Rossum, Guido. "Python reference manual." (1995).

Abadi, Martín, et al. "Tensorflow: A system for large-scale machine learning." *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016.

Cuthbert, Michael Scott, and Christopher Ariza. "music21: A toolkit for computer-aided musicology and symbolic music data." (2010).

McKinney, Wes. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.", 2012.

Talbot-Smith, Michael. *Sound engineering explained*. Routledge, 2012.

Chollet, Francois. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. "Overview of supervised learning." *The elements of statistical learning*. Springer, New York, NY, 2009. 9-41.

<https://en.wikipedia.org/wiki/Python_(programming_language)>

<https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08>