UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IASI
**FACULTATEA DE INFORMATICA**

LUCRARE DE LICENTA

# CREAREA RETELELOR DE SORTARE FOLOSIND DEEPLEARNING

*DANILA MARIUS CRISTIAN*

**Data:** *Iulie, 2018*

**Coordonator**
***Lect. Dr. Frasinaru Cristian***

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IASI
**FACULTATEA DE INFORMATICA**

LUCRARE DE LICENTA

# CREAREA RETELELOR DE SORTARE FOLOSIND DEEPLEARNING

*DANILA MARIUS CRISTIAN*

**Data:** *Iulie, 2018*

**Coordonator**
***Lect. Dr. Frasinaru Cristian***

# Cuprins

# 1 Introducere

Scopul acestei lucrari este de a construi un algoritm de deeplearning care sa genereze compartorii dintr-o retea de marime $N$. $N$ reprezinta numarul de fire din retea. Desi exista abordari neimperative functionale, care genereaza rezultate intr-un timp multumitor, nu s-a mai incercat pana acum generarea de retele de sortare prin aceasta abordare. Rezultatul obtinut este un model antrenat care produce rezultate pentru retele de dimensiuni mici.

In continuare prezentam succint capitolele prezente in lucrarea de fata, cat si continutul fiecauri capitol:

1. **Problema**. In cadrul acestui capitol am descris problema abordata de prezenta lucrare fara a intra in detalii amanuntite despre fundamentele teoretice ce stau la baza acesteia. Am adaugat si un scurt istoric al abordarilor precedente relevante.

2. **Retele de sortare**. Acest capitol are ca scop prezentarea cititorului fundamentele teoretice esentiale pentru intelegerea corecta a domeniului problemei. Tot in cadrul acestui capitol am expus si descrierea amanuntita a unei solutii precedente a problemei printr-o abordare pe care se bazeaza prezenta lucrare.

3. **Retele Neuronale**. Acest capitol incepe cu o descriere foarte scurta a conceptului de *retea neuronala*, intrucat acest sub domeniu al informaticii este acoperit de programa de studii a facultatii, apoi in urmatoarele subcapitole se descrie conceptul de *Neural Turing Machine*. Acesta fiind arhitectura pe care se bazeaza nucleul proiectului. Descrierea acestui concept este una pe larg, intrucat este introdus relativ recent si nu este prezent in programul de studii de licenta.

4. **Descrierea solutiei**. In acest capitol este descrisa arhitectura intregului proiect, modul de constructie a modelului si arhitectura modelului. De asemenea sunt prezentate rezultatele obtinute (*retelele de sortare* generate), iar spre final sunt comentate bucatile relevante de cod din proiect.

5. **Tehnologii folosite**. Acest capitol are rolul de a expune motivatia, dar si partea din proiect unde este folosita o anumita tehnologie. In cazul tehnologiilor neprezente in programul de studii, se spun cateva cuvinte si despre istoricul lor si domeniul problemei lor.

6. **Concluzii**. In final, reiteram problema aborsata, rezultatele obtinute, dat se si prezinta directiile de viitor ale proiectului.

# 2 Contributii

In cadrul acestei lucrari mi-am propus sa dezvolt un model capabil sa genereze, pe baza unui numar $N$ de fire primit ca input, lista de comparatori necesari pentru o retea de sortare cu $N$ fire. In cadrul proiectului am dezvoltat si o interfata web ce permite vizualizarea cat si simularea unei sortari pe reteaua obtinuta de model. Cu aceasta ocazie scoatem in lumina si elementul de inovatie al acestei lucrari, si anume ca pana acum nu s-a mai incercat generarea de retele de sortare prin aceasta abordare. In spate, modelul foloseste o arhitectura de retele neuronale introdusa recent (in *20 Oct 2014*), optimizata pentru invatarea de algoritmi imperativi. Modelul obtinut produce rezultate pentru retele de dimensiuni mici, de 2, 3 sau 4 fire. Nu produce comparatori pentru retele care sa sorteze complet siruri de numere mai mari de 4 elemente, dar acest lucru lasa loc de extindere a proiectului pe viitor.

Motivatia din spatele alegerii proiectului de fata este data de viteza de generare ca timp a retelelor de sortare ce este semnificativ mai buna pe retele de dimeniuni mari fata celelalte abordari.

Proiectul inglobeaza 2 modele antrenate pe seturi de antrenament diferite, bazate pe rezultatele a 2 algoritmi imperativi. Acesti 2 algoritmi imperativi au la baza modul de generare a interschimbarilor din algoritmii de sortare *Bubble Sort* si *Insertion Sort*. Fiecare model incearca sa produca rezultatele similare cu unul dintre cei 2 algoritmi imperativi folositi. Cele doua modele sunt rulate de o a doua componenta a proiectului, *model manager*, intr-un *thread*[1] separat fiecare model. Scopul acestei componente este sa centralizeze rezultatele de la cele doua modele si sa comunice cu componenta de *backend* din spatele interfetei web. Intr-un final, ultima componenta a proiectului este cea de interfata web care afiseaza retelele produse de cele doua modele.

# 3 Problema

Problema pe care o atinge prezenta lucrare este cea a generarii retelelor de sortare. O retea de sortare reprezinta un model abstract din informatica,

[1]S-a recurs la aceasta abordare din motive de performanta

compus din fire si comparatori. Scopul comparatorilor este de a interschimba 2 numere de pe firele pe care sunt plasati asa in cat la finalul parcurgerii seriei de comparatori, outputul reteli se va afla in ordine sortata. Desi sunt modele simple, retelele de sortare au aplicatii numeroase in calculul paralel si sunt folosite la sortarea seturilor de numere pe placile grafice.

In aceasta lucrare am incercat sa abordam problema intr-un mod nou, si anume prin antrenarea unui model bazat pe o arhitectura specializata in invatarea de algoritmi imperativi care sa produca comparatorii dintr-o retea de sortare de dimensiune $n$.

# 4 Retele de sortare

In cadrul acestui capitol prezentam acestui capitol prezentam fundamentele teoretice necesare pentru intelegerea conceptului de *retea de sortare* cat si aplicatiile acestora in practica

## 4.1 Prezentare

O retea de sortare cu $n$ inputuri este o secventa fixa de comparatii si interschimbari (comparatori) ce sorteaza toate inputurile de marime $n$. Deoarece aceeasi secventa de comparatori sorteaza orice secventa de marime $n$, reprezinta un algoritm de sortare independent de date. Asta inseamna ca secventa de comparatii ce se executa nu depinde de inputul primit. Datorita structurii pe care o are o retea de sortare, sunt de preferat in implementarile paralele ale algoritmilor de sortare, precum cele de pe placile grafice.

Motivati de aceste aplicatii in practica, retelele de sortare sunt un subiect important de cercetare inca din 1950. Un interes deosebit il au retelele de sortare optimale, care folosesc numarul minim posibil de comparatori. Generarea retelelor de sortare optimale reprezinta o problema grea de optimizare, investigata pentru prima data de O'Connor si Nelson pentri $4 \leq n \leq 8$. Retelele gasite de cei 2 aveau numarul minim de comparatori pentru $4, 5, 6$ si 8 inputuri, dara veau nevoie de 2 comparatori in plus pentru 7 inputuri. Acest rezultata a fost imbunatatit in 1968 de Batcher care a gasit retele minimale pentru $n \leq 8$.

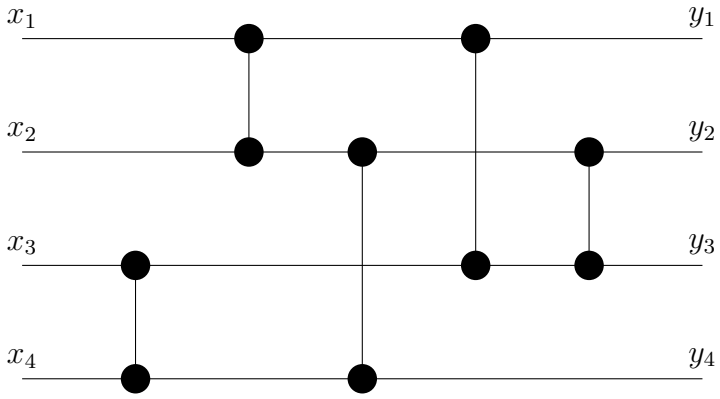



Fig. 1: O retea de sortare cu 4 inputuri. Valorile de input $x_1, , x_2, , x_3, , x_4$ din partea stanga a liniilor orizontale trec printr-o secventa de operatii de comparatii si interschimbari, reprezentate prin linii verticale ce conecteaza perechi de linii orizontale. Fiecare comparator de acest fel isi sorteaza cele doua valori de input, rezultand intr-un final in liniile orizontale ce contin valorile sortate de output $y_1 \leq y_2 \leq y_3 \leq y_4$, Aceasta este o retea minimala din punct de vedere a numarului de comparatori folositi.

| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | 0 | 1 | 3 | 5 | 9 | 12 | 16 | 19 | 25 | 29 | 35 | 39 | 45 | 51 | 56 | 60 |

Fig. 2: Numarul minim de comparatori cunoscut pentru retele de sortare de dimensiune $n \leq 16$. Aceste retele au fost studiate intens, dar s-a dovedit ca doar rezultatele pentru $n \leq 8$ sunt minimale.

Pana in 2013, se cunosteau retele de sortare minimale doar pentru inputuri $n \leq 8$ (vezi Tabelul 1). In 2013, Valsalam a introdus un articol in care prezinta modalitati de generare a retelelor optimale prin algoritmi evolutivi si reuseste sa obtina noi rezultate pentru $n$ de marime 17 , 18, 19, 20, , 21 si 22. In continuare, gasirea de retele optimale pentru inputuri $n > 8$ ramane o problema deschisa. Un caz particualr de interes este reprezentat de retelele descoperite de Green.

## 4.2 Principiul 0-1

### 4.2.1 Enunt

*Principiul 0-1* este o modalitate de verificare a corectitudinii unei retele de sortare. Conform acestuia, o retea de sortare care sorteaza corect orice de secventa de 0 si 1, va sorta corect orice secventa de numere.

### 4.2.2 Demonstratie

**Lemma:** Presupunem $f$ o functie monotona, crescatoare. Atunci, daca o retea mapeaza $x_1$, $x_2$,...,$x_n$ la $y_1$, $y_2$,...$y_n$, va mapa $f(x_1)$, $f(x_2)$,..., $f(x_n)$ la $f(y_1)$, $f(y_2)$,...,$f(y_n)$.

**Demonstratie:** Prin inductie la numarul de comparatori din retea folosind:

$$f(min(a,b)) = min(f(a), f(b))$$
$$f(max(a,b)) = max(f(a), f(b))$$

Sa presupunem ca o *retea* nu este retea de sortare. Atunci, va mapa valorile $x_1$, $x_2$,...,$x_n$ arbitrare, la $y_1$, $y_2$,..., $y_n$. Unde $y_i > y_{i+1}$ pentru $1 \leq i < n$. Presupunem $f(x) = 1$ daca si numai daca $x \geq y_i$, 0 altfel. Reteaua va mapa $f(x_1)$, $f(x_2)$,..., $f(x_n)$ la $f(y_1)$, ..., $f(y_i) = 1$, $f(y_{i+1}) = 0$, ..., $f(y_n)$. Astfel, reteaua nu va sorta toate inputurile de tipul $0 - 1$.

## 4.3 Generarea retelelor de sortare folosind algoritmi evolutivi

In acest subcapitol descriem succint modalitatea de generare a retelelor de sortare optimale folosind algorimti evolutivi. Motivul pentru care am ales sa aducem in lumina si aceasta abordare este ca, proflema fiind una dificila de rezolvat prin metode imperative, s-a incercat abordarea acesteia prin metode evolutive, iar rezultatele sunt surprinzatoare.

In principal, vom descrie metoda introdusa de *Valsalam* si *Miikkulainen*, (SENSO[15]).

### 4.3.1 Reprezentarea functiilor booleane

Folosind principiul *0-1* descris mai sus, putem exprima inputul unei retele de sortare ca variabile booleane, iar outputul retelei ca functii de aceste variabile. Astfel, problema gasirii comparatorilor pentru o retea de sortare se reduce la a numara firele ce au ca input valoarea 1 si a seta outputul firelor de output *de mai jos* la valoarea 1, iar outputul firelor ramase la 0. Deoarece aceste functii sunt implementate de comparatorii din retea, problema gasirii unei retele de sortare se poate reduce la problema gasirii unei secvente de comparatori care genereaza functiile de output ale retelei.

### 4.3.2 Simetrii in retele de sortare

O simetrie intr-o retea de sortare este o operatie pe setul de functii de output al retelei care nu afecteaza outputul retelei. Valsalam defineste formal operatie de simetrie $\sigma_i$ pentru $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$ care opereaza pe setul de functii de output din retea prin interschimbarea functiei $f_i$ si dualei $f_{n+1-i}$, interschimband conjunctia si disjuctia dintre ele. Compozitia simetriilor este tot o simetrie, fiind inchisa la compozitie.

### 4.3.3 Minimizarea numarului de comparatori necesari

Minimizarea numarului de comparatori presupune gasirea comparatorilor ce pot fi impartiti pe mai multe fire.

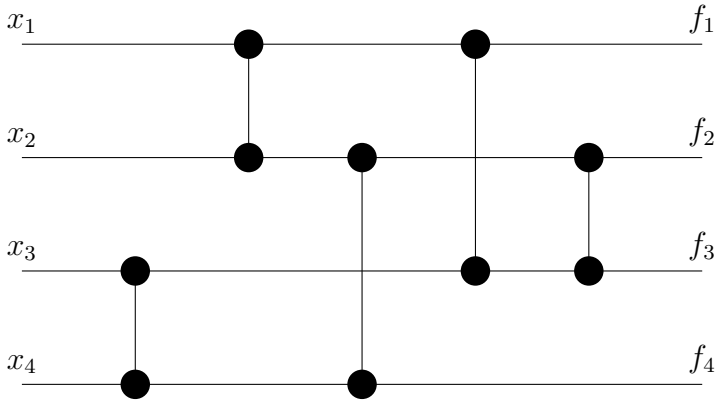



Fig. 3: Potrivit documentului introdus de Valsalam, outputurile retelei pot fi exprimate astfel:
$f_1 = x_1 \wedge x_2 \wedge x_3 \wedge x_4$,
$f_2 = (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_4) \vee \cdots$
$f_3 = (x_1 \wedge x_3) \vee (x_1 \wedge x_3) \vee \cdots$
$f_4 = x_1 \vee x_2 \vee x_3 \vee x_4$

Fiecare comparator produce o conjuctie a inputurilor sale pe firul superios si o disjunctie a inputurilor sale pe firul inferior. In consecinta, functiile de output ale retelei sunt combinatii de disjunctii si conjunctii ale inputurilor sale. Prin urmare, o retea de sortare poate fi privita ca o secventa de comparatori care genereaza functiile de output pe baza inputurilor sale. Valsalam a folosit aceasta observatie in proiectarea algoritmului evolutiv descris mai jos.

### 4.3.4 Evoluarea retelelor minimale

Valsalam initializeaza evolutia cu o populatie de solutii produse de algoritmul greedy. Valoarea de *fitness* a fiecarei solutii este $-1 \cdot$ `numarul de comparatori`, pentru ca imbunatatirea *fitness*-ului sa rezulte intr-un numar mai mic de comparatori generati. La fiecare noua generatie, selectia se face prin metoda *turneu* pe baza valorii de *fitness* pentru a selecta indivizii din populatie pentru reproducere. La reproducere se muteaza reteaua parinte si se creeaza o retea de *offspring* in doi pasi:

1. Un comparator este ales in mod aleator din retea si apoi se trunchiaza reteaua dupa el, eliminand comparatorii ce urmeaza dupa el.

2. Algoritmul greedy este folosit pentru a adauga noi comparatori construind o noua retea. Pentru ca algoritmul greedy alege un comparator cu cea mai mare utilitate in mod aleator, aceasta mutatie exploreaza o noua combinatie de comparatori ce ar putea fi mult mai folositori decat comparatorul parinte.

Aceasta abordare restrange spatiul de cautare la comparatorii generati de algorimtul greedy. SENSO[15] a fost rulat pe o populatie de marime 200 pentru 500 de generatii ca sa evolueze de retele de sortare optimale. La fiecare generatie, au selectate cele 100 de retele cu numarul minimal de comparatori pentru a estima modelul. Acelasi set de retele a fost copiat la urmatoarea generatie, fara modificari, ....

Experimentul de mai sus a fost repetat de 20 de ori pentru fiecare varianta a algoritmului greedy si pentru fiecare input $n \leq 23$

Numarul de comparatori gasit de SENSO in corespondenta cu marimea reteli este reprezentat in figura de mai jos.

## 4.4 Retelele optimale

### 4.4.1 Prezentare

In aceasta sectiune a lucrarii dorim sa amintim despre *retelele lui Green*, intrucat folosind metoda lui *Green* de construire a retelelor obtinem numarul minimal de comparatori.

Mai jos afisam rezultatul metodei lui *Green* pentru o retea de marime 16:

Acesta retea are 60 de comparatori, care reprezinta cea mai mica valoare cunoscuta pentru o retea cu 16 inputuri[16][17]. Comparatorii dintr-o astfel

| n | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $C$ | 39 | 45 | 51 | 56 | 60 | 73 | 79 | 88 | 93 | 103 | 110 | 118 |
| $C_2$ | 39 | 45 | 51 | 56 | 60 | **71** | **78** | **86** | **92** | **102** | **108** | 118 |

Fig. 4: Rezultate obtinute de Valsalam. Rezultatele sunt extrase din [15]. $C$ reprezinta numarul minim de comparatori cunoscut pana atunci pentru o retea cu $n$ inputuri, iar $C_2$ reprezinta numarul de comparatori obtinuti prin abordarea evolutiva. Au fost marcate in bold, retelele pentru care au fost scosi un numar de comparatori mai bun.

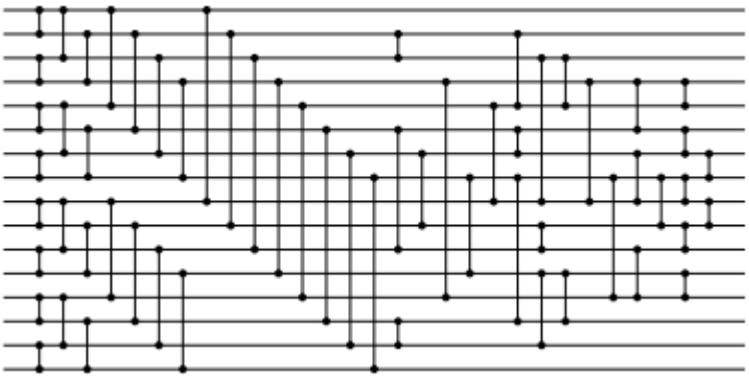


Fig. 5: Reteaua de comparatori a lui green pentru un input $n = 16$

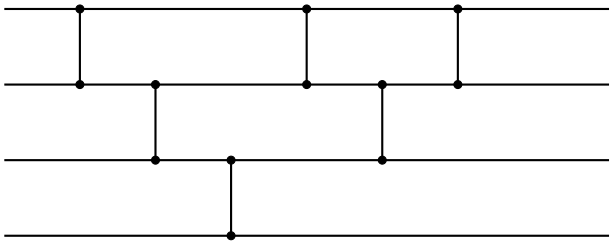


Fig. 6: Retea de sortata generata pe baza algoritmului *Bubble Sort* cu $n = 4$ inputuri.

de retea sunt simetric aranjati de la axa orizontala pana in mijlocul retelei. Acest tip de retele au reprezentat pentru unii cercetatorii obiectiv de studiu pentru dezvoltarea algoritmilor evolutivi care genereaza retele de sortare.

## 4.5 Metode imperative de generare a retelelor de sortare

Exista mai multe metode imperative pentru a genera o retea de sortare. Cele mai usoare modalitati sunt de a generea o retea bazata pe algoritmii de sortare bubble sort si insertion sort. In cazul primei optiuni se adauga un comparator intre firele 0 si 1, 1 si 2, ..., $n-1$ si $n$. Se reaia procesul de creere a comparatorilor inserand un comparator intre 0 si 1, ..., $n-2$ si $n-1$. Se repeta aceasta procedura de creere a comparatorilor pana la pasul in care inseram un singur comparator intre firele 0 si 1.

In cazul unei insertion sorting network, o retea bazata pe insertion sort procesul este similar, singura diferenta este ca buclele de creare a comparatorilor sunt in ordine inversa fata de metoda precedenta. Astfel, prima data se creeaza un comparator intre firul 0 si 1. Se creeaza un comparator intre 0 si 1, 2. Pana cand se ajunge la pasul in care se generaza un comparator intre 0 si 1, ..., $n-1$ si $n$.

Un exemplu de bubble sorting network:

Un exemplu de insertion sorting network:

Algoritmul imperativ pe care l-am folosit pentru a genera un bubble sorting network este:

```
def bubble_sorter(n):
    if n <= 1:
        return []
```

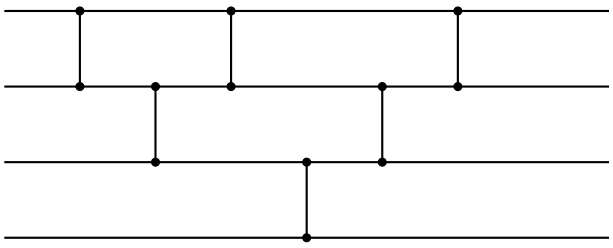


Fig. 7: Retea de sortare generata pe baza algoritmului *Insertion Sort* pentru $n = 4$ inputuri.

```
ret_val = []
for i in range(0, n - 1):
    ret_val.append((i, i + 1))
ret_val += self.dummy_sorter(n - 1)
return ret_val
```

Pentru un numar de fire dat, $n$, generam lista de comparatori, adica conexiunile dintre firele din retea.

Aceste arhitecturi de retele de sortare nu sunt optimale si vor sorta un sir de numere ca input in timp $O(n^2)$. Retelele optimale si cele mai des folosite in practica sunt bazate pe algoritmul bitonic sort, acestea ating complexitate de $O(nlog^2(n))$. Insa motivul pentru care am prezentat bubble sorting network si insertion sorting network este ca bazandu-ne pe acesti algoritmi am generat datele de antrenament pentru modelul dezvoltat. Motivatia ar fi ca, cel putin din experimentele realizate in cadrul acestui proiect, este mai usor de antrenat un model care sa produca date relevante atunci cand primeste date de antrenament generate pe baza acestor algoritmi.

## 4.6 Aplicatii

Cea mai la indemana aplicatie in practica a retelelor de sortare o reprezinta sortarea pe placile grafice. Placile grafice fiin ddispozitive orientate puternic spre paralelizarea instructiunilor, pe arhitectura *single-instruction, multiple data (SIMD*. Si avand in vedere si ca placile grafice depasesc in performanta CPU-ul cand vine vorba de algoritmi cu limita de memorie si timp de calcul, retelele de sortare devin o optiune eficienta.

# 5 Retele neuronale

Retelele neuronale reprezinta o ramura a inteligentei artificiale. Sunt modele abstracte ce incearca sa simuleze creierul uman in modul de intercationare cu natura. O retea neuronala este compusa din neuroni artificiali. Neuronii artificiali din retea sunt dispusi pe mai multe straturi. Din exterior le putem vedea ca pe un black box conectat la un strat de neuroni de input, responabil de preluarea inputlui si la un strat de neuroni de output, responsabil pentru a genera outputul. In functie de neuronii din stratul de output care se vor activa se decodifica rezultatul produs de retea.

## 5.1 Arhitecturi de retele neuronale

Retelele neuronale convolutionale sunt modele simple, proiectate sa primeasca un set de input fix si sa produca un set output output de marime fixa. In practica, acest tip de retele sunt ideale pentru computer vision. Sunt mult mai eficiente din punct de vedere al resurselor consumate la sarcini precum recunoasterea de obiecte in imagini.

Retelele neuronale recurente se deosebesc de cele convolutionale prin posibilitatea de a primi un set de input variabil si a produce un set de output variabil. In practica se folosesc la procesarea limbajului natural. Acest tip de retele neuronale sunt ideale si pentru rezolvarea problemei de generare a retelelor de sortare, intrucat putem codifica numarul de fire din retea ca o secventa de lungime variabila, iar rezultatul, respectiv setul de comparatori pentru numarul de fire dat poate varia. Pe langa aceste considerente, acest tip de retele contin si o memorie atasata.

Un caz particular de retele neuronale recurente sunt *Neural Turing Machines*. Acestea fiind optimizate in a invata algoritmi imperativi simpli. Un exemplu ar fi problema sortarii, rezolvata in mod traditional cu un algoritm imperativ poate fi rezolvata folosind acest tip de retele. Pentru aceasta, reteaua va primi ca input tuple de forma $(X, Y)$. Unde $X$ reprezinta un sir de numere aflate in ordine aleatoare, iar $Y$ reprezinta varianta sortata a sirului $X$ cu un algoritm imperativ de sortare. Odata antrenata pe un set suficient de mare de date de acest fel, modelul va invata practic sa aplice algoritmul de sortare folosit pe componenta $Y$.

Motivul prezentarii scenariului de mai sus este ca hiperparametrii si structura straturilor de neuroni dintr-o retea neuronala folosita la sortarea de numere sunt similari in proportie foarte mare cu cei din reteaua proiectata sa

genereze retele de sortare.

## 5.2 Neural Turing Machine (NTM)

Retelele NTM fac partea din clasa retelelor neuronale recurente si sunt introduse relativ recent. Au fost introduse intr-un articol publicat de Alex Graves in 2014. Acest model de retea neuronala combinata capacitatea de pattern matching a retelelor neuronale cu capacitatea algoritmica a calculatoarelor programabile. Mai exact, putem folosi acest model de retea pentru a invata algoritmti simpli. In paperul introdus de Graves sunt aduse rezultate pentru algoritmi de copiere a unui sir sau sortare. Aceste aspecte ne inspira o probabilitate mare ca o retea bazata pe acest model poate produce rezultate multumitoare pentro problema abordata.

### 5.2.1 Descriere

Neural Turing Machines sunt inspirate din doua lumi: Machine Learning si Teoria Computabilitatii. Prima le permite calculatoarelor sa realizeze taskuri care, desi pentru oameni sunt usoare, se credeau grele pentru calculatoare, precum computer vision. A doua defineste formal lucrurile de care este capabil un calculator.

Legatura dintre inteligenta artificiala si teoria computatiei in 1940. Modelul introdus de el, Masina Turing, e un model computational clasic care opereaza pe o banda de memorie infinita si un cap care poate sa scrie sau sa citeasca. Pe baza acestor aceste rezultate se inspira si Neural Turing Machines.

Un exemplu de NTM desfasurat in timp:

Componenta de controller din cadrul unui Neural Turing Machine este o retea neuronala care pune la dispozitie reprezentarea interna a inputului care este folosit de capetele de scriere si citire ca sa poate interactiona cu memoria. Reprezentarea aceasta nu este identica cu cea stocata in memorie. Tipul de controler pe care il alegem reprezinta cea mai semnificativa alegere arhitecturala. Controllerul poate fi fie o retea feed-forward, fie o retea recurenta.

Capetele de scriere si citire sunt singurele componente din cadrul unui Neural Turing Machine care interactioneaza in mod direct cu memoria. Intern, comportamentul fiecarui dintre capete este controlat de vectorul de

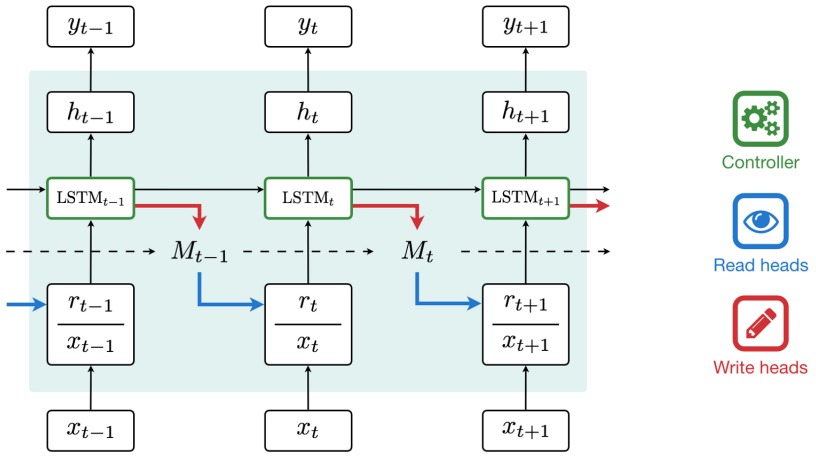



Fig. 8: Arhitectura unui Neural Turing Machine. Sursa: `https://medium.com/snips-ai/ntm-lasagne-a-library-for-neural-turing-machines-in-lasagne-2cdce6837315`

weighturi care este actualizat la fiecare pas in timp. Fiecare weight din vector corespunde cu gradul de interactiune cu fiecare celula din memorie. Un weight de 1 focuseaza atentia NTM-ului spre celula respectiva.

Reprezentare vizuala a notiunilor descrise mai sus:

In continuare prezentam succint partile critice din aceasta arhitectura, asa cum au fost descrise de *Graves* in [21].

### 5.2.2 Operatia de citire

Fie $M_t$ continutul matricii $N \times M$ de memorie la timpul $t$. $N$ reprezentand numarul de celule de meorie, iar $M$ este marimea vectorului de la fiecare locatie. Fie $w_t$ un vector de *weighturi* peste $N$ locatii generat de capul de citire. Pentru ca toate *weight*-urile sunt normalizate, ce $N$ elemente ale $w_t(i)$ suporta urmatoarea constrangere:

$$\sum_t w_t(i) = 1, 0 \leq w_t(i) \leq 1, \quad \forall i$$

Vectorul $r_t$ de lungime $M$ returnat de capul de citire este definit:

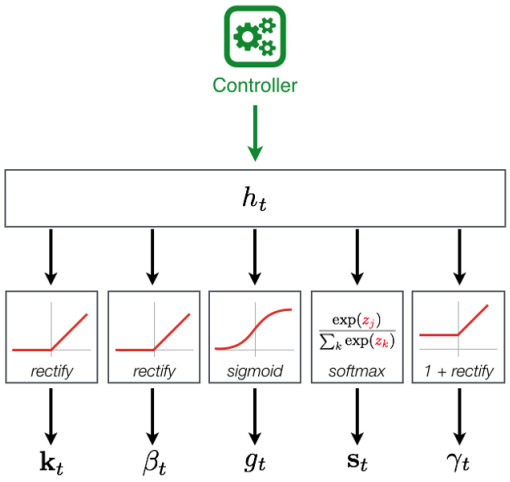



Fig. 9: Arhitectura capului de scriere/citire a unui *Neural Turing Machine*. Sursa: `https://medium.com/snips-ai/ntm-lasagne-a-library-for-neural-turing-machines-in-lasagne-2cdce6837315`

$$r_t \leftarrow \sum_t w_t(i) M_t(i)$$

### 5.2.3 Operatia de scriere

Fiecare operatie de scriere din aceasta arhitecura este impartita in doua parti:

1. o operatie de stergere (*erase*)
2. o operatie de adaugare (*add*)

Fiind dat un vector de weighturi $w_t$ generat de capul de scriere la timpul $t$, si un vector de stergere (*erase vector*) $e_t$ cu elemente in $(0, 1)$, vectorii de memorie $M_{t-1}(i)$ sunt modificati astfel:

$$\tilde{M}_t(i) \leftarrow M_{t-1}(i)[1 - w_t(i)e_t]$$

unde 1 reprezinta un vector linie cu elemente 1. Elementele de la o locatie de memorie sunt setate la 0 daca atat *weight*-urile de la acea locatie, cat si

elementul de trebuie sters sunt 1. Altfel, daca oricare dintre *weight*-uri sau elementul ce trebuie sters sunt 0, meoria este lasata neschimbata.

Capul de citire si un vector $a_t$ de lungime $M$ ce este adaugat in memorie dupa pasul de stergere:

$$M_t(i) \leftarrow \tilde{M}_t(i) + w_t(i)a_t$$

### 5.2.4 Mecanismul de adresare

In articolul introdus de *Graves*, adresarea se realizeaza prin doua modalitati:

1. **In functie de continut**.
2. **In functie de locatie**.

Vom descrie doar prima metoda

In *adresarea pe baza de continut* fiecare cap de citire sau scriere va produce un vector $k_t$ de lungime $M$ care este comparat cu fiecare vector $M_t(i)$ printr-o masura de similaritate $K[\cdot, \cdot]$. Sistemul de adresare pe baza continutului produce un vector normalizat $w_t^e$:

$$w_t^e = \frac{\exp\left(\beta_t K[k_t, M_t(i)]\right)}{\sum_j \exp\left(\beta_t K[k_t, M_t(j)]\right)}$$

In articolul lui *Graves* $K$ este definit:

$$K[u, v] = \frac{u \cdot v}{||u|| \cdot ||v||}$$

### 5.2.5 Aplicatii

*Neural Turing Machines* inca se afla sub cercetare activa. Utilitatea acestei arhitecturi de retele neuronale este data posibilitatea unui model sa invete un *program*.

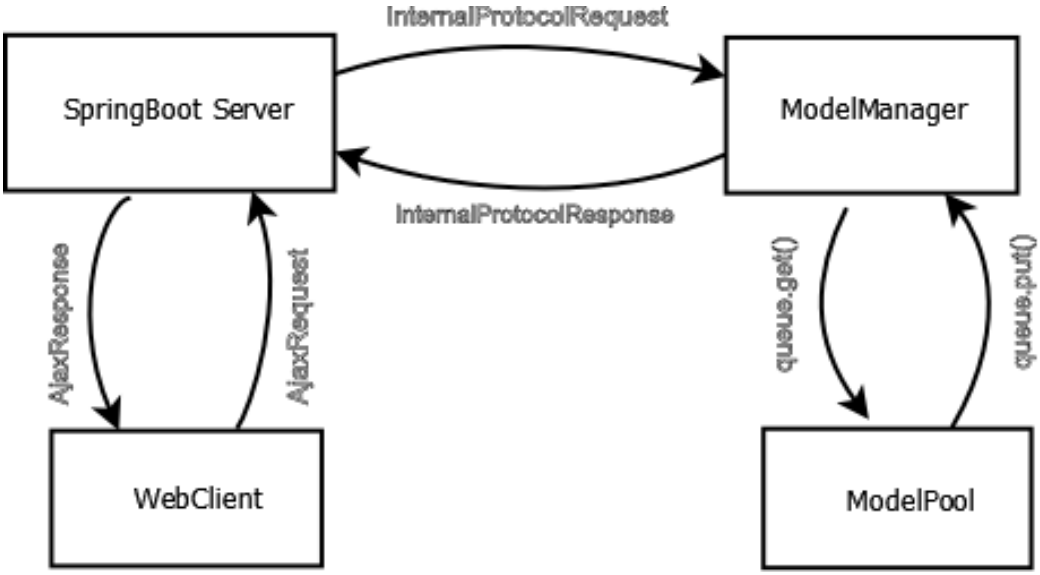
# 6 Descrierea solutiei

Solutia problemei consta intr-un model antrenat pe un set de date produs de un algoritm imperativ de generare a retelelor de tip bubble sorting network si insertion sorting network. Peste modelele generate avem o interfata web ce permite vizualizarea in browser a retelei generate.

## 6.1 Arhitectura proiectului

Proiectul complet este compus din 4 componente:

- Un server web, bazat pe SpringBoot. Rolul serverului web este sa-i livreze utilizatorului codul interfetei web a aplicatiei. De asemenea mediaza requesturile dintre requesturile unui utilizator de a primi comparatorii pentru un anumit numar de fire si *model manager*.
- O interfata in browser. Rolul interfetei in browser este acela de a se putea vizualiza efectiv reteaua generta de un modelul selectat pe un anumit numar de fire. Utilizatorul alege nuamrul de fire si modelul pe baza caruia se genereaza outputul.
- O componenta numita model manager. Managerul este compus dintr-un server TCP/IP, prin care care primeste requesturi de la serverul web. O clasa care proceseaza requestul primit de la serverul web si in functie de parametrii din request cere output de la thread pool-ul care ruleaza modelele.
- Un thread pool, fiecare model antrenat ruleaza intr-un thread separat din acest pool. Aceasta componenta incarca modelul antrenat de pe disc si ruleaza intr-un thread separat functia de predictie. Motivatia acestei abordari este legata de directiile de extindere pe viitor a proiectului.




Arhitectura generara a proiectului

In diagrama de mai sus observam schematic, abstractizat, principalele componenente ale proiectului si relatiile dintre ele. In continuare descriem in detaliu aceste relatii:

- **SpringBoot Server - WebUI**, etapa principala din ciclul de viata al proiectului este livrarea codului interfetei clientului, apoi, de interes este interactiunea dintre *WebUI* si server. De obicei fluxul normal este:

    1. Utilizatorul selecteaza numarul de fire, modelul antrenat folosit si asteapta producerea rezultatului. In spate, *WebUI* impacheteaza optiunile utilizatorului intr-un pachet *JSON* si trimite prin *AJAX* un request catre server.
    2. *Serverul* primeste requestul de la *WebUI* printr-un endpoint. In functie de optiunile alese, se creeaza un apel catre clasa *ManagerClient* care are rolul de a fi un adaptor intre server si *ModelManager*.
    3. *Server*-ul impacheteaza intr-un *JSON* lista de comparatori returnata de *ManagerClient* si il trimite *WebUI*-ului.
    4. *WebUI*-ul primeste *response*-ul *AJAX* cu lista de comparatori si initiaza un apel spre modulul de desenare, care creeaza reprezentarea grafica a retelei produse.

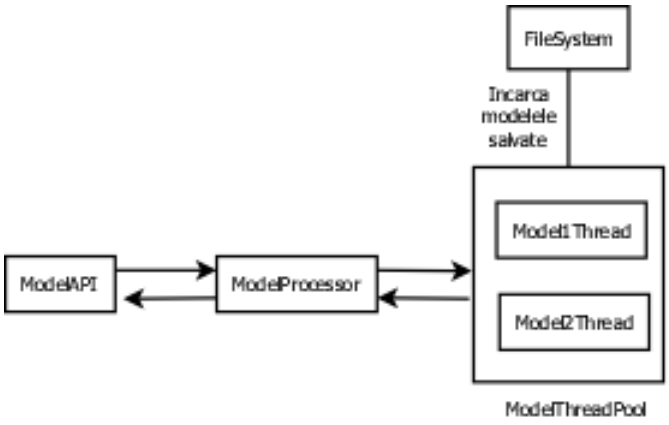
- **SpringBoot Server - ModelManager**, comunicarea dintre aceste componente este, de regula, initializata de *ManagerClient* si cronologic este structurata astfel:

    1. *ManagerClient*-ul initiaza o conexiune TCP/IP cu *ModelManager*-ul. Ii trimite acestuia un pachet codificat intr-un protocol propriu ce contine numarul de fire si modelul pe baza caruia sa se genereze comparatorii.
    2. *ModelManagerul*-ul accepta conexiunea de la *ManagerClient*, primeste pachetul, il decodifica si realizeaza un apel catre *ModelThreadPool*, apoi impacheteaza lista de comparatori generata si o trimite inapoi la *ManagerClient*.

- **ModelManager - ModelPool**. De la initializare *ModelPool*-ul incarca de pe disc fiecare dintre modele si porneste un thread separat ce le ruleaza.

    1. In functie de modelul ales *ModelManager*-ul realizeaza un apel la metoda respectiva din *ModelPool*.
    2. Prin intermediul unei cozi *thread-safe*, *ModelPool*, semnalizeaza threadul cu modelul respectiv ca trebuie evaluat un nou rezultat. Rezultatul modelului este transmis inapoi prin acelasi mecanism.

In continuare dorim sa analizam mai atent structura si modul de functionare al *ModelManager*-ului.




Arhitectura interna a model managerului

In diagrama de mai sus sunt expuse principalele submodule ale managerului. Expunem succint scopul fiecarui submodul:

- *ModelApi*, reprezinta un server TCP/IP ce ruleaza pe portul *2018*. Comunicarea cu componenta de *ModelManager* se face prin trimiterea de TCP/IP pe acest port.

- *ModelProcessor*, decodifica pachetele primite de eventualii utilizatori apeleaza metoda potrivita din urmaorul submodul. *ModelProcessor*-ul are si rolul de a aproxima probabilitatile din lista de comparatori generata de model.

- *ModelThreadPool*, incarca de pe disc fiecare dintre modelele salvate si apoi creeaza un thread in care ruleaza fiecare model. Comunicarea dintre threadul principal se face prin intermediul a doua cozi *thread-safe*. Threadul principal pune un coada numarul de fire dorit, iar threadul ce ruleaza modelul respectiv pune in a doua coada lista de comparatori generarata.

## 6.2 Arhitectura modelului

Modelul este un "Neural Turing Machine". Pentru constructia modelului am folosit in spate un framework numit ntm-lasage, optimizat pentru crearea de modele de acest fel. Se abstractizeaza in acest fel detalii de implementare.

Modelul contine o memorie cu un layout de 1024x160 celule. Ca functie de loss am folosit crossentropia binara, pentru ca intern numerele ce vin ca input in model sunt reprezentate ca siruri binare.

Procesul de backpropagation este abstractizat de model. Actualizarea weighturilor din retea este optimizata cu rmsprop. Datele de antrenament sunt procesate de retea in batchuri de 1.

Structura inputului retelei este de un vector de 9 elemente ce reprezinta codificarea in binar a numarului de fire din retea. Outputul retelei este un vector tridimensional cu $2 * numar\_comparatori$ elemente. Elementele din vector sunt semantic grupate 2 cate 2, codificate ca un sir de biti, si reprezinta pozitiile comparatorilor din retea.

## 6.3 Sursa modelului

Mai jos introducem componenta principala din cadrul proiectului:

```
def model(input_var, batch_size=1, size=8, num_units=100, memory_shape=(1024, 160)):

    l_input = InputLayer((batch_size, None, size + 1), input_var=input_var)
    _, seqlen, _ = l_input.input_var.shape

    memory = Memory(memory_shape, name='memory',
        memory_init=lasagne.init.Constant(1e-6), learn_init=False)
    controller = GRUController(l_input, memory_shape=memory_shape,
        num_units=num_units, num_reads=1,
        nonlinearity=lasagne.nonlinearities.rectify,
        name='controller')
    heads = [
        WriteHead(controller, num_shifts=3, memory_shape=memory_shape,
                name='write', learn_init=False,
            nonlinearity_key=lasagne.nonlinearities.rectify,
            nonlinearity_add=lasagne.nonlinearities.rectify),
        ReadHead(controller, num_shifts=3, memory_shape=memory_shape,
                name='read', learn_init=False,
            nonlinearity_key=lasagne.nonlinearities.rectify)
    ]
    l_ntm = NTMLayer(l_input, memory=memory, controller=controller, heads=heads)

    l_output_reshape = ReshapeLayer(l_ntm, (-1, num_units))
    l_output_dense = DenseLayer(l_output_reshape, num_units=size + 1,
        nonlinearity=lasagne.nonlinearities.sigmoid, \
        name='dense')
    l_output = ReshapeLayer(l_output_dense, (batch_size, seqlen, size + 1))

    return l_output, l_ntm
```

Modelul este construit in intregime avand ca baza frameworkul *Lasagne*.

## 6.4 Antrenarea modelului

Modelul ce genereaza retele bubble sorter a fost antrenat pe un set de input 1000 de elemente, pe numere mai mari de atat timpul necesar finalizarii procesului de antrenament devine foarte mare. Datele din setul de antrenament au fost generate conform algoritmului:

```
def bubble_sorter(n):
    if n <= 1:
        return []

    ret_val = []
    for i in range(0, n - 1):
        ret_val.append((i, i + 1))
    ret_val += self.dummy_sorter(n - 1)
    return ret_val
```

si arata astfel pentru un set de antrenament de 5 elemente:

```
[2, [(0, 1)]]
[3, [(0, 1), (1, 2), (0, 1)]]
[4, [(0, 1), (1, 2), (2, 3), (0, 1),
        (1, 2), (0, 1)]]
[5, [(0, 1), (1, 2), (2, 3), (3, 4),
        (0, 1), (1, 2), (2, 3), (0, 1), (1, 2), (0, 1)]]
[6, [(0, 1), (1, 2), (2, 3), (3, 4),
        (4, 5), (0, 1), (1, 2), (2, 3),
        (3, 4), (0, 1), (1, 2), (2, 3), (0, 1), (1, 2), (0, 1)]]
```

Insa date de mai sus reprezinta, la nivel abstract, inputul si outputul dorit pentru retea. In proiect datele sunt reprezentate in format binar ca un sir de 9 biti, iar interschimbarile nu sunt reprezentate explicit ca tuple. Reteaua va produce, practic, o lista de siruri biti ce vor fi grupate, in ordine, 2 cate 2.

Mai precis:

1. **Setul de antrenament** este de forma $(x, Y)$ unde $x$ este un numar intreg, iar $Y = ((a_1, b_1), (a_2, b_2), ..., (a_n, bn))$, unde $a_i, b_i$ sunt numere intregi.

2. **Outputul retelei** este de forma:

$$Out = (\underbrace{a_1, b_1}_{\text{C1}}, \underbrace{a_2, b_2}_{\text{C2}}, \quad ..., \quad c\underbrace{a_n, b_n}_{\text{Cn}})$$

   . Unde $a_i$ reprezinta firul superior pe care va fi amplasat comparatorul $C_i$, iar $b_i$ reprezinta firul inferior.

Spre exemplu, pentru urmatorul input:

```
numpy.array([[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0]]])
```

reprezinta dimensiunea 3, adica reteaua va trebui sa genereze setul de comparatori pentru o retea de 3 elemente. Inputul de mai sus a fost testat pe o retea antrenata pe un set de date 100 de elemente, modelul este *Bubble Sorter*. Outputul produs este urmatorul:

```
[[[0.000751546365971,                0.370771723772,
0.000769892576671,                   0.465849417004],
0.000637261131599,
0.000600783421602,                   [9.00920625725e-05,
0.12773718494,                       7.82020981282e-05,
0.239040904467,                      6.5095684253e-05,
0.214272843324,                      6.73165195701e-05,
```

```
0.180481531645,
0.244833481652,
0.273710496834,
0.454686659751,
0.496889833317],

[4.46837405637e-05,
3.44817119736e-05,
2.6885035585e-05,
3.21667374046e-05,
0.262205621903,
0.240104608732,
0.341346314965,
0.501759117299,
0.507289515972],

[3.48919682303e-05,
2.59198402876e-05,
1.83589819237e-05,
2.46684454266e-05,
0.330040528556,
0.237285575094,
0.394284580217,
0.525120693702,
0.510196603956],

[3.10455175941e-05,
2.33229850784e-05,
1.51453999616e-05,
2.18735040572e-05,
0.376419535208,
0.237146807326,
0.429571992622,
0.537611938861,
0.511490463268],

[2.89867797738e-05,
2.23753419482e-05,
1.35824510377e-05,
2.04069513415e-05,
0.406359755172,
0.238643669264,
0.452255805808,
0.54490706061,
0.513243797725]]]
```

Semnificatia fiecarui double din lista este probabilitatea ca bitul respectiv sa fie 1. Acuratetea probabilitatilor se sporeste cu antrenarea sporita a modelului. Intr-un final, se grupeaza 2 cate 2 elementele din lista si se decodifica din binar in decimal.

# 7 Tehnologii folosite

In cadrul proiectului am facut uz de o serie de frameworkuri si biblioteci pentru a fi scutiti de unele detalii de implementare. In aceasta sectiune vom prezenta pe scurt tehnologiile principale folosite, motivatia alegerii unei tehnologii, dar si unde a fost utilizata.

## 7.1 NTMLasagne

NTMLasagne este frameworkul pe care l-am folosit in proiect ca sa creem modelul. Pe scurt, este un wrapper peste Lasagne ce ne permite sa creeam "Neural Turing Machines" intr-un mod simplificat, abstractizand detalii de implementare. Ne sunt puse module pentru stratul NTM (en. NTMLayer) unde toate componentele precum capetele de scriere, controllerul si memoria pot fi customizate. In cadrul acestui proiect am ales sa nu customizam

niciuna dintre componente. In schimb, le-am folosit pe cele default oferite de framework.

Motivatia din spatele alegerii acestui framework este, chiar daca documentatia este relativ putina, comunitatea puternica din spatele proiectului si faptul ca procesul de dezvoltare este activ.

## 7.2 Lasagne

Lasagne este o biblioteca minimalista folosita pentru construirea de retele neuronale. Permite construire de retele feed-forward, cat si de retele convolutionale, ofera implementari pentru metode de optimizare folosite des (i.e. ADAM si RMSProp). De asemenea, modele construite in Lasagne pot fi antrenate fie pe CPU sau GPU pentru ca biblioteca foloseste Theano in spate.

## 7.3 Theano

Theano este o biblioteca de python ce permite definirea si optimizarea de expresii matematice ce contin array-uri multidimensionale intr-un mod eficient. Chiar daca in momentul de fata Tensorflow este alegerea facuta de majoritatea proiectelor, theano este folosit de cele 2 biblioteci specificate anterior si este in general mai flexibil.

## 7.4 Python

*Python* este un limbaj de scripting, multi paradigma. Motivul pentru care am ales sa implementam modelul in python este sintaxa usoara, suportul mare pentru biblioteci si frameworkuri de *deeplearning*, atat ca documentatie, cat si ca implementari efective.

De asemenea componenta de *model manager* care ruleaza intr-un thread fiecare model este scrisa in tot *python*.

In cadrul proiectului am folosit mai multe module pe care le mentionam succint in aceasta sectiune:

1. **numpy** pentru reprezentarea inputului unui model, cat si pentru dependintele de mai sus (*Lasagne*, *NTM-Lasagne*)
2. **cPickle** pentru a serializa un model antrenat.
3. **json** pentru a impacheta mesajele trimise la serverul Java.

## 7.5 Java

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de ctre James Gosling la Sun Microsystems (acum filial Oracle) la nceputul anilor 90, fiind lansat n 1995. Cele mai multe aplicaii distribuite sunt scrise n Java, iar noile evoluii tehnologice permit utilizarea sa i pe dispozitive mobile gen telefon, agenda electronic, palmtop etc. n felul acesta se creeaz o platform unic, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat n prezent cu succes i pentru programarea aplicaiilor destinate intranet-urilor

In proiect, Java a fost folosit la crearea *back-endului* pentru interfata web. Motivatia alegerii este experienta puternica anterioara a autorului cu acest limba de programare.

## 7.6 Spring Boot

In cadrul proiectului am pus la dispozitie si o interfata web in care se afiseaza reprezentarea vizuala a retelei generate de modelul ales de utilizator. Back-endul acestei interfete este scris in Java si are in spate Spring Boot.

Pe scurt Spring Boot este frameworkul web pe care l-am folosit pentru a scrie serverul proiectului[2]. Ni se pune la dispozitie posibilitatea de a scrie relativ usor o aplicatie web pe arhitectura *MVC* [3].

Motivul pentru care s-a ales Spring Boot in pofida unui alt framework web de Java este documentatia vasta si experienta anterioara a autorului cu aceasta tehnologie.

## 7.7 RaphaelJS

RaphaelJS reprezinta biblioteca de JavaScript folosita pentru desenarea in browser a retelei generate de modelul ales de utilizator. RaphaelJS este un wrapper peste WebGL cu un API relativ usor de folosit.

## 7.8 Twitter Bootstrap

Twitter Bootstrap este un framework web pentru componenta de frontend. Se pun la dispozitie mai multe clase CSS pentru crearea de componente

[2]Serverul ruleaza pe *tomcat 8*

[3]Designt Patternul Model-View-Controller

vizuale precum butoane, bare de meniu, cat si un mecanism specific de de pozitionare a continutului.

Motivul pentru care am introdus si aceasta dependinta in stiva de tehnologii folosita in proiect si nu am creat toata interfata in RaphaelJS este performanta. Desenarea intregului design si implementarea si mecanicilor din spate (precum gestionarea apasarilor de buton) ar fi fost remarcabil mai lenta. Ar fi fost o scadere redundanta in performanta.

# 8 Concluzii

In momentul de fata, cele doua modele antrenate, nu produc rezultate multumitoare, nu sunt capabile sa genereze comparatori pentru retele de dimensiuni mari. Optimizarea acestor modele pentru inputuri $N > 4$ poate reprezenta o directie de viitor pentru proiect. Acest lucru ar putea fi realizat printr-o metoda introdusa relativ recent (*Mai 2018*)[18].

Readucem in atentie ca aceasta lucrare contine un element de noutate, intrucat aceasta abordarea de a genera retele de sortare folosind *deeplearning* nu a mai fost atinsa pana in acest moment.

# Bibliografie


[1] https://github.com/primaryobjects/nnsorting

[2] *https* : *//github.com/drforester/Sequence_to_Sequence_Sorting*

[3] https://github.com/aditya-prasad/dnnet

[4] https://adventuresinevolutionblog.wordpress.com/2016/09/17/minimal-sorting-networks/

[5] http://aclweb.org/anthology/I17-3017

[6] http://psycnet.apa.org/record/1999-02657-007 – Seven times seven is about fifty

[7] https://www.sciencedirect.com/science/article/pii/S0020025512007670

[8] https://github.com/apache/incubator-mxnet/tree/master/example/bi-lstm-sort – sort numbers using lstm architecture

[9] *https* : *//rylanschaeffer.github.io/content/research/neural_turing_machine/main.html*

[10] https://github.com/carpedm20/NTM-tensorflow

[11] http://www.robots.ox.ac.uk/ tvg/publications/talks/NeuralTuringMachines.pdf

[12] https://medium.com/snips-ai/ntm-lasagne-a-library-for-neural-turing-machines-in-lasagne-2cdce6837315

[13] http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/nulleinsen.htm

[14] http://www.cs.tau.ac.il/ zwick/Adv-Alg-2015/Sorting-Networks.pdf

[15] Valsalam, Vinod K. and Miikkulainen, Risto Using Symmetry and Evolutionary Search to Minimize Sorting Networks *J. Mach. Learn. Res.*, 14(1): 303–331, feb 2013

[16] D. E. Knuth. Art of Computer Programming: Sorting and Searching, volumul 3, capitolul 5, paginile 219229. Addison-Wesley Professional, 2 edition, April 1998

[17] M. W. Green. Some improvements in non-adaptive sorting algorithms. In Proceedings of the Sixth Annual Princeton Conference on Information Sciences and Systems, paginile 387391, 1972.

[18] Stephen McAleer, Forest Agostinelli, Alexander Shmakov, Pierre Baldi Solving the Rubik's Cube Without Human Knowledge, 18 Mai 2018

[19] Alex Graves, Greg Wayne, Ivo Danihelka Neural Turing Machines, 20 Oct 2014

[20] Peter Kipfer, Rdiger Westermann Improved GPU Sorting, *https* : *//developer.nvidia.com/gpugems/GPUGems*2/*gpugems*2_*chapter*46.*html*, Aprilie 2005

[21] Alex Graves, Greg Wayne, Ivo Danihelka Can neural nets learn a program? http://www.robots.ox.ac.uk/ tvg/publications/talks/Neural-TuringMachines.pdf