UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

LUCRARE DE DISERTAȚIE

Rețele neuronale recurente in detectarea intenției si a entităților dintr-o conversație

Autor:

Ionuț N. Ciocoiu

Coordonator științific:

Prof. Dr. Liviu P. Dinu

IUNIE 2019

Cuprins

Listă de figuri **3**

1 Introducere **5**

1.0.1 Progres precedent 6

1.1 Nevoia de a comunica 7

1.1.1 Contribuția noastră 8

1.2 Rezumatul capitolelor 9

2 Dialog **11**

2.1 Conversația naturală 11

2.1.1 Acte de vorbire 12

2.2 Sistem de dialog 12

2.2.1 Componente 13

3 Sistem de dialog **15**

3.1 Înțelegerea limbajului natural 16

3.1.1 Abordări anterioare 17

3.1.2 Model propus 17

3.1.3 Mulțimea de antrenare 20

3.1.4 Rezultate 21
3.1.5 Analiza erorilor 21
3.2 Administrator de dialog 23
3.2.1 Abordări anterioare 24
3.2.2 Model propus 27

4 Noțiuni teoretice **31**
4.1 Rețele Neurale Recurente 31
4.1.1 Problema diminuării si a exagerării gradientului 33
4.1.2 Soluții pentru dispariția/exagerarea gradientului 35
4.1.3 Seq2Seq 35
4.1.4 GRU 36
4.1.5 LSTM 37

5 Expunerea tehnologiei **39**
5.1 Serviciul NLU 39
5.2 Tehnologii folosite 40
5.2.1 Python 40
5.2.2 Numpy 40
5.2.3 PyTorch 41

6 Concluzii **42**

Bibliografie **43**

Anexe 45

Listă de figuri

2.1 Sistem de Dialog Modular 13

3.1 Administrator de dialog 16

3.2 Seq2Seq-x 18

3.3 Bidirectional Encoder 19

3.4 Attention Decoder 20

3.5 Rezultat eronat 24

3.6 Rezultat corect 25

3.7 Administrator de dialog 26

4.1 Arhitectura RNN [1] 32

5.1 Arhitectura serviciului de NLU 39

Lista anexe

6.1 Ontologie pentru ATIS 1

6.2 Structura dialogului pentru ATIS 4

Capitolul 1

Introducere

Privind dincolo de utilitatea practică de a transmite informații, limbajul reprezintă principala noastră unealtă în evoluție și nu numai, el este cel care intensifică interacțiunea și menține legăturile interumane. Sunt rare momentele în care ne gândim la această capacitate de a comunica, totuși o folosim la fel de des și ne este la fel de indispensabilă precum o funcție vitală.

Înțelesul este conceptul ce are ca principală sursă limbajul, această relație scoate în evidență contrastul dintre complexitatea inferenței și simplitatea cuvintelor. Reprezentativ pentru această legătură este proverbul *"Vorba dulce mult aduce"* ce surprinde în esență trăsături definitorii precum puterea cuvântului, capacitatea afectivă și în principal caracterul contemplativ al limbajului. El în sine reprezintă o formă de artă, în detaliu paradoxală și recurentă, poate din cauza faptului ca ne reprezintă ajunge să se confunde cu noi.

În cartea sa Speech and Language Proccessing [2], profesorul Daniel Jurafsky observă faptul că există numeroase exemple în care omul încearcă să comunice cu creația sa. Acest comportament este surprins adesea și în literatură unde personificarea nu se oprește numai la natură. În lumina acestor observații se remarcă longevitatea și neoboseala dorinței creatoare, a cărei împlinire se concretizează în această ramură a lingvisticii matematice unde puterea de a înțelege limbajul este folosită la implementarea unor modele matematice capabile de o conversație.

1.0.1 Progres precedent

Mergând pe acest drum, în literatura de specialitate întâlnim două abordări care tratează construcția unui sistem de dialog.

Prima vorbește despre asistenți digitali care rezolvă diferite cerințe (taskbots). În general conversația nu este menită să dureze mult, iar rolul agentului este de a capta informații din conversația cu utilizatorul și de a-l ajuta să ducă la bun sfârșit o operație [3, 4].

În cea de-a doua direcție este vorba despre agenți conversaționali (chatbots). Acest termen face referire la un agent orientat spre discuție la nivel general, servind mai degrabă unui scop subiectiv decât unuia obiectiv soluțional. În această situație dialogul poate dura mai mult, existând și aplicații practice ale acestuia cum ar fi cea de a testa anumite teorii psihologice [5]

Ghidat în mare parte de impactul asupra industriei am ales ca direcție de studiu sistemul de dialog orientat pe rezolvarea cerințelor (taskbots), unde accentul se pune în primul rând pe puterea de a simula determinismul cu care un agent uman ar răspunde. Din această cauză avem o suită de module: recunoaștere vocală (Automatic Speech Recognition - ASR sau Speech To Text - STT), înțelegerea limbajului natural (Natural Language Understanding - NLU), administrarea dialogului (Dialog Manager - DM), generarea de replici (Natural Language Generation) și într-un final conversia textului în voce (Text To Speech, TTS) toate acestea conectate într-un flux de informații ce are ca prim declanșator utilizatorul. La celălalt pol, în cercetare, întâlnim tot mai des abordări în care componente ([STT], NLU, DM, NLG, [TTS]) sunt văzute ca un singur model unde învățarea se realizează de la un capăt la altul (end-to-end) [6, 7, 8, 9], nemaifiind nevoie de antrenarea separată. Privind dintr-o perspectivă practică, aceste modele end-to-end duc lipsă de mulțimi de antrenare din cauza faptului că necesită o atenție mai sporită, iar în ceea ce privește antrenarea administratorului de dialog, avem nevoie de simularea unui mediu de utilizatori, cerință impusă de strategia de învățare prin recompensă (Reinforcement Learning). Totuși aceste abordări reușesc să surprindă în aplicațiile de tip conversație deschisă (chatbots) unde nu se pune accent mare pe contextul dialogului.

Revenind la sistemul de dialog orientat pe rezolvarea cerințelor, observăm că minimum viabil al acestei arhitecturi este dat de componenta care înțelege limbajul natural (NLU) și cea care ține contextul conversației (DM). Aceste două module sunt necesare și suficiente pentru a putem avea cel puțin în scris un dialog cu mașina.

Componenta de înțelegerea limbajului natural se ocupă cu detectarea intenției și a entităților aflate într-o replică venită din partea utilizatorului. De multe ori cele două chestiuni sunt văzute separat: detecția intenției ca o problemă de clasificare, respectiv recunoașterea constituenților semantici ca o problemă de etichetare a unei secvențe. Pentru detectarea intenției se folosesc abordări precum [10, 11, 12, 4], iar pentru recunoașterea entităților sunt folosite tehnici de etichetare ca [13, 14, 15, 16, 17, 4, 18].

Arhitectura de codificare-decodificare împreună cu rețelele neuronale recurente își aduc contribuția cu succes la un număr mare de probleme din procesarea limbajului natural. Rezultate promițătoare se remarcă în următoarele chestiuni: recunoașterea de entități [18], traducerea automată [**?**], recunoașterea vorbirii, generarea de limbaj, rezumarea de text. Acest tip de abordare o întâlnim tot mai des și în ipostaze hibride, unde se încearcă rezolvarea concomitentă a mai multor probleme. În esență se folosește capacitatea de a capta înțelesul unei secvențe de text cu ajutorul unui codificator iar apoi se folosesc diferite decodificatoare pentru a traduce înțelesul captat în etichete specifice.

1.1 Nevoia de a comunica

Privind comunicarea ca o nevoie de bază și limbajul ca un prim instrument al său, vedem de ce domenii științifice precum procesarea limbajului natural și lingvistica matematică reprezintă elemente cheie ale accesului nostru la informație, mai presus de atât curiozitatea stârnită aici ne poartă de la înțeles la autocunoaștere.

Stilul nostru de viață se schimbă odată cu tehnologia, acest dinamism aduce cu el noi straturi de informații ce dau naștere unor probleme de căutare, structurare și reprezentare a informației. Dat fiind rapiditatea cu care aceste inovații intră în contact cu oamenii, nevoia de a comunica cu un astfel de sistem se dorește a fi cât mai naturală.

Gradul de eficiență și acces la informație cerut de generațiile tinere este în continuă creștere, iar acest lucru sporește nevoia de a înțelege limbajul natural și de a menține unui dialog cât mai firesc între om și mașină.

Actualele modele de a structura informația se dovedesc a fi instrumente bune la stocarea și procesarea ei, pe când reprezentarea și inferența sunt două concepte aduse în prim plan de inteligența artificială care cere noi moduri de a rezona cu informațiile stocate.

Lucrarea își propune cercetarea unui model îndeajuns de puternic încât să înțeleagă limbajul natural exprimat într-un anumit context (general sau specific unui domeniu) și punerea în aplicare a unui administrator de dialog, suficient de complex pentru a ține contextul unor sarcini bine definite.

Înțelegerea limbajului natural este un concept care poate îngloba o mulțime de probleme precum analiza de sentiment, clasificarea de text, rezumarea și multe alte chestiuni legate de semantică. În literatura de specialitate a sistemelor de dialog, NLU ține locul componentei ce este responsabilă cu detectarea intenției vorbitorului și recunoașterea entităților, aceste grupuri de cuvinte care au o însemnătate în cadrul unui domeniu. De exemplu: ”aș dori să îmi resetez parola, numele meu de utilizator este cioionut”, avem ca intenție resetarea parolei, iar ca entitate numele de utilizator.

În administratorul de dialog, aceste intenții sunt văzute ca reprezentări ale obiectivelor utilizatorului. Odată ce intenția este stabilită, pentru a duce la bun sfârșit un obiectiv s-ar putea să mai avem nevoie de anumite informații așa cum în exemplu anterior am avut nevoie de numele de utilizator, lucru ce face ca entitățile să fie văzute drept argumente ale unui obiectiv.

1.1.1 Contribuția noastră

Luând contact tot mai des cu mediul de cercetare, încerc să observ modul în care această comunitate reușește să aducă contribuții în societate. Un lucru care m-a făcut să prețuiesc fiecare efort, este acela că în general o descoperire se bazează pe cercetări anterioare și ca orice gând exprimat în scopul de a descoperi, poate fi valoros și dus mai departe. Dorința de a crea cu scopul de a face viața oamenilor mai ușoară este

imboldul intrinsec ce ghidează acțiunile mele. Așadar evaluându-mi cunoștințele am decis sa îmi aduc contribuția într-un domeniu atât de important.

Ideea de a cerceta un model pentru înțelegerea limbajului, vine în egală măsură din curiozitate și din conștientizarea nevoii. Abordarea propusă conține două module, unul pentru NLU și altul pentru DM (dialog manager). Decizia care a stat la baza acestei arhitecturi este legată în primul rând de rigoarea impusă de un administrator de dialog separat în favoarea unei soluții end-to-end ce impune mulțimi de antrenare mult mai rafinate și simularea unui mediu de utilizatori. S-a urmărit și posibilitatea de a pune în producție un astfel de sistem, dar și ușurința de a crea noi agenți doar adăugând date specifice domeniului.

Pentru a înțelege limbajul natural s-a folosit o arhitectură de tip encode-decoder, unde codificarea este realizată de o rețea neuronală recurentă, iar decodificarea se realizează în două faze diferite pentru fiecare problemă. Detectarea intenției se realizează cu ajutorul unui decodificator bazat pe o rețea de tip feed forward, care primește ca intrare secvența codificată și scoate la ieșire o distribuție de probabilitate peste intențiile cunoscute. A doua fază de decodificare este realizată de o a doua rețea recurentă care la fiecare pas prezice eticheta corespunzătoare unei entități. În ajutorul acestei arhitecturi s-au experimentat mai multe modele de atenție așa cum sunt descrise și în [19].

Un plus adus în puterea de generalizare îl reprezintă însumarea celor două stări ascunse (înainte și înapoi) din codificator. Spre deosebire de [4] unde se folosește o concatenare a celor două. O altă îmbunătățire a performanței se datorează ideii de a nu mai propaga eroarea din decodificatorul de intenții în codificator. Făcând astfel ca predicțiile de intenții să se realizeze pe baza reprezentărilor din codificator învățate din entități. Acest lucru simplifică procesul de învățare și ajută ambele decodificatoare să obțină rezultate mai bune.

1.2 Rezumatul capitolelor

Introducere - Capitolul întâi vorbește in principal despre modul în care această lucrare își propune să rezolve nevoia de interacțiune cu tehnologia. Este prezentată

problematica, motivația, abordările anterioare și contribuția adusă domeniului.

Dialog - În această parte este vorba despre încercarea noastră de a structura limbajul și de a-l pune într-un cadru științific, ușor de modelat matematic, dar fără a pierde din vedere puterea cuvintelor.

Sistem de dialog - Capitolul al treilea vorbește despre experimentele noastre și descrie abordările cu cele mai bune rezultate. Aici este prezentată mulțimea de antrenare împreună cu performanța obținută, dar și o analiză privind erorile modelului.

Noțiuni teoretice - In al patrulea capitol vor fi detaliate modelele matematice care stau în spatele modelelor decizionale.

Expunera tehnologiei - Atunci când se rostește "progres" am în minte o spirală a cunoștințelor care se bazează unele pe altele. Precum această imagine implementarea unui sistem de dialog impune un anumit progres precedent, așa că aici se vor prezenta instrumentele care fac posibilă aceasta tehnologie.

Concluzii - În finalul tezei se găsesc concluziile referitoare la studiul elaborat, împreună cu observațiile privind puterea de scalabilitate a unei astfel de abordări și înțelegerea diferenței dintre abordările din industrie și cele din spațiul academic.

Capitolul 2

Dialog

Conform definiției din limba română, dialogul este modul de expunere care prezintă succesiunea replicilor dintr-o conversație care are loc între două sau mai multe persoane.

Această lucrare își propune să dea formă înțelegerii limbajului natural dintr-o perspectivă matematică, prezentând sub forma unei soluții programabile un întreg sistem de micro servicii toate funcționând sub umbrela aceluiași scop, comunicarea.

Pe parcursul lucrării se va face referire la dialog ca o secvență de replici între un om și un calculator pentru a transmite informații. Referitor la componentele unui dialog, se va prezenta doar o abordare bazată pe componenta *verbală*, celelalte componente - *nonverbală* (gesturi, mimică, poziția corpului) și *paraverbală* (accentul, ritmul și intensitatea vorbirii) - făcând obiectul altor lucrări viitoare.

2.1 Conversația naturală

Un factor cheie într-o conversație este acela că fiecare replică dintr-un dialog este o formă de **acțiune** venită din partea vorbitorului [20]. În literatura de specialitate **actele de vorbire** sunt cele ce dau tipul acestor acțiuni.

De-a lungul timpului s-au propus numeroase moduri de a grupa acte de vorbire, iar modul de clasificare potrivit abordării de față conține doar 4 categorii și se concentrează

pe intenția comunicată, conținutul propozițional și contextul de producere [21].

2.1.1 Acte de vorbire

1. **Reprezentative (asertive)** - sunt acele acte care definesc o constatare. Cu ajutorul lor se comunică informații se impun constrângeri, iar la nivel de conținut putem spune că descriu realitatea. Se descriu anumite evenimente (”Coletul a fost livrat”), sau se impun anumite constrângeri (”Voi pleca maine”). Mărcile acestui act de vorbire sunt date de verbe precum: *a afirma, a admite, a anunța, a avertiza, a declara, a insista, a înștiința, a zice, etc.*

2. **Directive** - sunt actele ce urmăresc impunerea unei acțiuni. Scopul lor este de a determina participanții la dialog să execute o sarcină. Verbele marcante sunt *a întreba, a interzice, a ordona, a cere*

3. **Expresive** - reprezintă o manifestare în plan verbal a sentimentelor, emoțiilor și a atitudinilor vorbitorului cu referire la acțunile întreprinse sau nu de către interlocutor. Verbele reprezentative ale acestui act sunt: *a mulțumi, a lăuda, a felicita, a regreta, a reproșa, a critica, a acuza, etc*

4. **Comisive (promisive)** - vizează angajamentul locutorului de a da curs unor acțiuni/convingeri *a promite, a plănui, a jura, a paria, a se opune*

Pentru stabilirea intenției, actul de vorbire joacă un rol foarte important, iar sistemele de dialog extind aceste clasificări în intenții specifice domeniului cu scopul de a caracteriza cât mai bine conversația.

2.2 Sistem de dialog

Conversația cu un robot presupune o înlănțuire de servicii văzută ca un ciclu de informații ce se mulează pe intențiile vorbitorului. În continuare sunt prezentate componente care fac posibilă această conexiune.

2.2.1 Componente

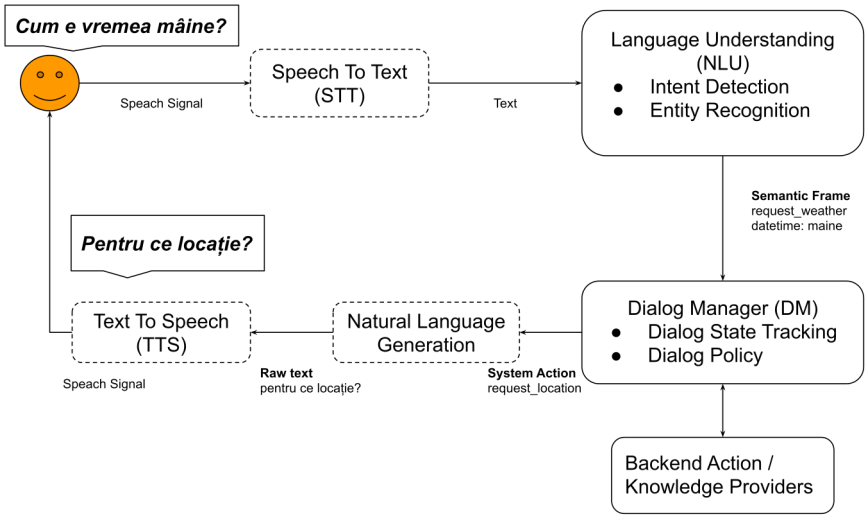

Figura 2.1: Sistem de Dialog Modular

Din voce în text - Speech To Text (STT) - Este serviciul care se ocupă cu recunoașterea și conversia în text a semnalelor vocale

Înțelegerea limbajului natural - Natural Language Understanding (NLU) - Componenta responsabilă de cadrul semantic ce descrie intenția și entitățile menționate de utilizator

Administrator de dialog - Dialog Manager (DM) - Este modulul ce gestionează dialogul, având ca funcții principale reprezentarea contextului și inferența pe baza acestuia

Generarea de limbaj natural - Natural Language Generation (NLG) - Este modulul care se ocupă cu transformarea stării interne într-o replică de dialog

Din text în voce - Text To Speech - Ultima procesare este produsă de componenta care generează semnale vocale pe baza textului primit de la NLG

Întregul proces este descris de diagrama din figura 2.1. Totul începe cu utilizatorul care formulează o cerere către sistem, vocea acestuia este procesată de către modulul de recunoaștere vocală care întoarce textul rostit de către vorbitor. Cererea în format text este apoi trimisă la modulul de înțelegere a limbajului care detectează la rându-i,

intenția utilizatorului dar și entitățile menționate. Acest cadru semantic extras este apoi trimis modulului care ține firul dialogului și în funcție de definiția sarcinii pe care utilizatorul o dorește a fi îndeplinită se iau acțiunile în consecință și se trimite răspunsul către utilizator.

În tabelul 2.1 avem un exemplu de dialog între om și robot. Aici este prezentat rezultatul componentei de NLU care ne indică diferite stagii ale dialogului.

	Replică	**Cadru semantic (acte de vorbire + entități/sloturi)**
Sistem:	Hello! I am your flights assistant. How can I help you?	`welcome()`
Utilizator:	I want to go from Boston to Atlanta on monday	`inform(fromloc.city_name=Boston, toloc.city_name=Atlanta, depart_date.day_name=monday)`
Sistem:	Please let me know the arrival date	`request(arrival_date)`
Utilizator:	arrive on june 5 at 5 am	`inform(arrive_date.month_name=june, arrive_date.day_number=5, arrive_time.time=5 am)`
Sistem:	Done! That's your flights for monday flying from boston to atlanta <system_call>	`inform(fromloc.city_name=Boston, toloc.city_name=Atlanta, depart_date.day_name=monday)`
Utilizator:	Great, thank you	`acknowledgement()`
Sistem:	Can I help you with other things?	`request(intent)`
Utilizator:	No, goodbye	`bye()`

Tabela 2.1: Conversație între un sistem de dialog și un utilizator

Capitolul 3

Sistem de dialog

Construcția unui agent virtual menit să întrețină cursul unei conversații a fost întotdeauna un etalon al performanței de cercetare, de aceea testul ce poartă numele cercetătorului A. M. Turing [22] a fost până de curând un criteriu în această direcție.

Au fost propuse diferite arhitecturi și moduri de a schița matematic o conversație, printre ele și ELIZA [5] compus dintr-un set de reguli elaborate pe baza mai multor studii având la bază logica. Bineînțeles industria cere și abordări mai modulare, mai robuste dar care să nu iasă din tipare, în această direcție făcându-și apariția prima arhitectură bazată pe umplerea de sloturi (GUS [23]), adică pentru fiecare replică din dialog se extrag constituenți semantici specifici domeniului care mai apoi sunt completați într-o structură tabelară (frame) urmând să servească drept parametrii în interogările cu sistemul.

Dacă privim la scopul final al unei conversații distingem două clase și anume: agenți orientați pe rezolvarea de cerințe și agenți orientați pe discuție la nivel general.

Cum majoritatea asistenților virtuali sunt dezvoltați în scopuri comerciale, rezolvarea cerințelor primează în funcționalitatea unei astfel de aplicații, așadar industria se concentrează pe arhitecturi modulare care să necesite cât mai puține date de antrenare. Servicii precum RASA, SNIPS, WATSON, DialogFlow, Wit.ai pun la dispoziție instrumente pentru a crea acest gen de agenți.

Prin natura sa lumea academică explorează noi metode de a gestiona un dialog,

iar acum cele mai populare abordări se concentrează pe dezvoltarea unui singur model care să descopere strategii de a acționa în diferite situații, mai exact se folosesc rețele neurale recurente pentru a capta contextul unei replici și pentru a genera răspunsuri, iar pentru a reprezenta contextul conversației și a genera politici de decizie se folosește învățarea prin recompensă. [6, 7, 8, 9]

În cadrul studiului curent accentul este pus pe structura ce îmbină mai multe modele, drept pentru care vor fi detaliate experimentele ce au condus la alegerea unui model care să înțeleagă textul (NLU).

3.1 Înțelegerea limbajului natural

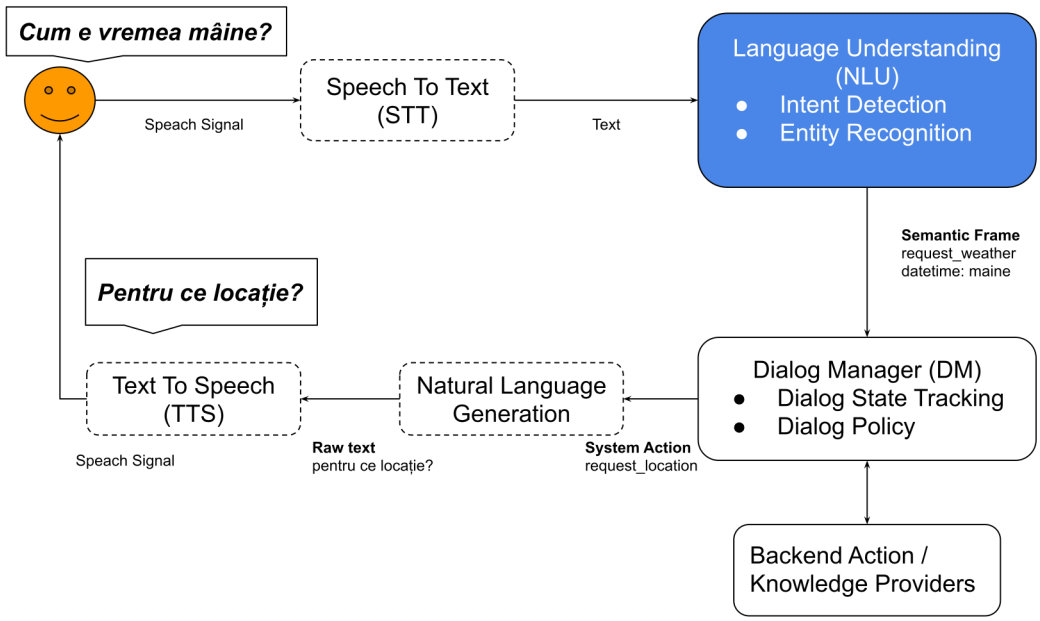

Figura 3.1: Administrator de dialog

Înțelegerea limbajului natural (NLU) este un concept destul de vast. În terminologia sistemelor de dialog acesta joacă rolul componentei care detectează intenția

vorbitorului si extrage constituenți semantici din limbajul natural. Detecția intenției poate fi tratata ca o problema de clasificare a unei replici din punct de vedere semantic, iar recunoașterea entităților poate fi văzută ca o etichetare de secvențe.

3.1.1 Abordări anterioare

Pentru detectarea intenției se pot folosi tehnici standard de clasificare a unui text precum [10, 11, 12, 4], chiar si rețele neurale convoluționale (CNNs) [13] întâlnite adesea în procesarea imaginilor.

În recunoașterea entităților se folosesc abordări precum (MEMMs) (McCallum et al., 2000), conditional random fields (CRFs) (Raymond and Riccardi, 2007), și rețele neurale recurente (RNNs) [18, 16, 15].

Având la baza înțelegerea limbajului, majoritatea lucrărilor actuale se concentrează pe rezolvarea concomitenta a acestor doua probleme, întrucât combinarea celor doua modele ajuta la învățarea unei reprezentări cat mai precise a textului. [10, 4, 3, 24]

3.1.2 Model propus

Rețelele neuronale recurente (RNN) sunt unele dintre cele mai populare modele matematice folosite în procesarea limbajului natural. Avantajul pe care acestea îl oferă se datorează într-o mare măsură formelor sale LSTM, GRU, dar și naturii lor de a învăța din date cu caracter secvențial. Aceste tipuri de rețele sunt folosite adesea ca structuri de bază pentru a crea alte modele de calcul, un exemplu este rețeaua de codificare-decodificare (seq2seq), formată din două rețele recurente, una pentru captarea (codificarea) înțelesului unei secvențe iar cealaltă pentru operația de decodificare într-o secvență de ieșire.

Modelul propus pentru înțelegere a limbajului rezolvă ambele probleme simultan folosind un codificator pentru reprezentarea înțelesului unei propoziții și două decodificatoare câte unul pentru fiecare chestiune. Figura 3.2 sintetizează procesele implicate în acest model.

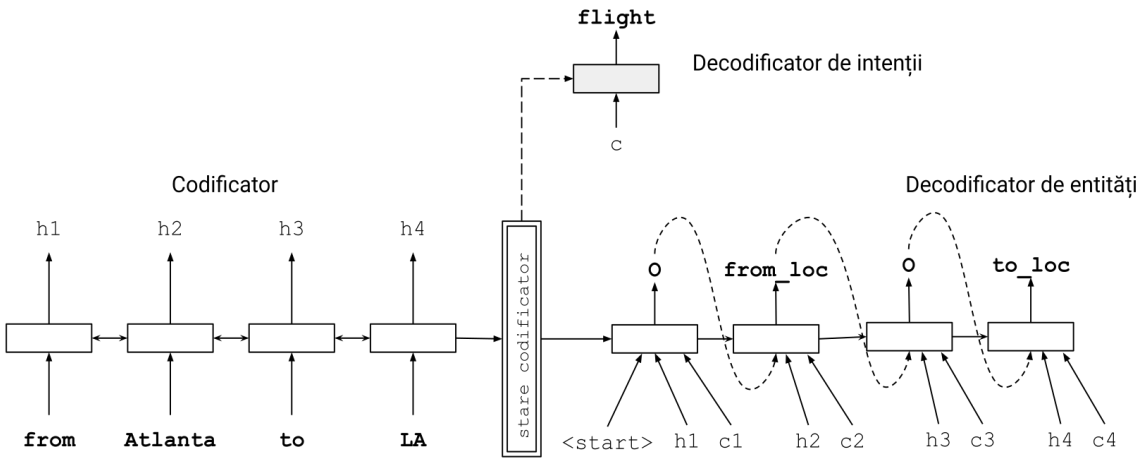

Figura 3.2: Seq2Seq-x

Pentru recunoașterea entităților se dorește etichetarea unei secvențe de cuvinte $x = (x_1, ..., x_T)$ într-o altă secvență de nume de entități corespunzătoare $y = (y_1, ..., y_T)$.

Operația de codificare se realizează prin citirea întregii secvențe de către o rețea recurentă bidirecțională. Înțelesul secvenței de intrare este reprezentat de suma celor două stări ascunse finale, adică pentru citirea de la stânga la dreapta obținem stările ascunse $(fh_1, ..., fh_T)$ și invers $(bh_T, ..., bh_1)$, apoi suma celor două $h_i = [fh_i + bh_i]$ este folosită drept codificare a înțelesului secvenței de intrare. Alte lucrări [3] folosesc ca stare ascunsă finală h_i o concatenare a celor două stări. Alegerea de a aduna cele două reprezentări în schimbul concatenării, ține de faptul că dimensiunea reprezentării finale crește în cazul concatenării, fiind mai dificil de antrenat, dublând în acest caz și jumătate din parametrii rețelei. Figura 3.3 descrie procesul de codificare.

Vom folosi reprezentarea captată de codificator pentru a inițializa operația de decodificare, aceasta este produsă de către o rețea recurentă unidirecțională, văzută ca o funcție $s_i = f(s_{i-1}, y_{i-1}, h_i, c_i)$ care la fiecare pas i, calculează o nouă stare s_i, bazată pe: starea anterioară a decodificatorului s_{i-1}, eticheta anterior prezisă, y_{i-1}, starea ascunsă din codificator corespunzătoare pasului curent, h_i și un vector de context c_i, calculat ca o sumă ponderată peste stările ascunse din codificator.

Inspirat din [19] am experimentat mai multe modele de atenție.

$$c_i = \sum_{j=1}^{T} \alpha_{i,j} h_j$$

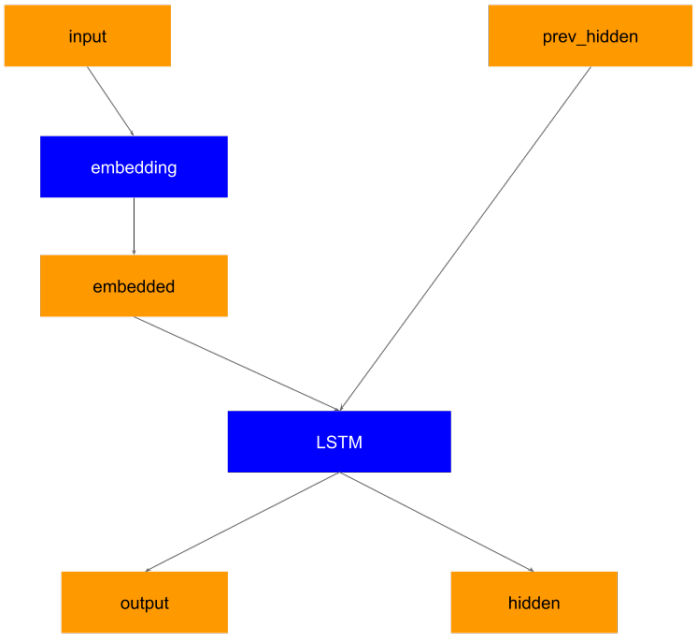

Figura 3.3: Bidirectional Encoder

$$\alpha_{i,j} = \frac{\exp(score_{i,j})}{\sum_{k=1}^{T} \exp(score_{i,k})}$$

$$score(i,k) = g(s_{i-1}, h_k) = \begin{cases} s_{i-1}^{\top} h_k & \text{dot} \\ s_{i-1}^{\top} W_a h_k & \text{general} \\ v_a^{\top} \tanh(W_a[s_{i-1}; h_k]) & \text{concat} \end{cases}$$

Modelul de atenție poate fi văzut ca o rețea de tip feed-forward $g(s_{i-1}, h_k)$, care la fiecare pas din operația de decodare calculează ponderile pentru stările ascunse ale codificatorului, folosindu-se de ultima stare ascunsă din decodificator, s_{i-1} și stările ascunse $(h_1, ..., h_T)$.

Decodificatorul de intenții este o rețea de tip feed-forward ce primește la intrare ultima stare ascunsă din codificator, h_T și stările ascunse $(h_1, ..., h_T)$. Cu ajutorul acestora calculează o atenție de tip *concat*, apoi folosind h_T și vectorul de context c calculat anterior, se folosește un singur strat pentru a prezice intențiile.

Mai jos în figura 3.4 este descris întreg fluxul de date și operații realizate de decodificatorul responsabil de etichetarea entităților.

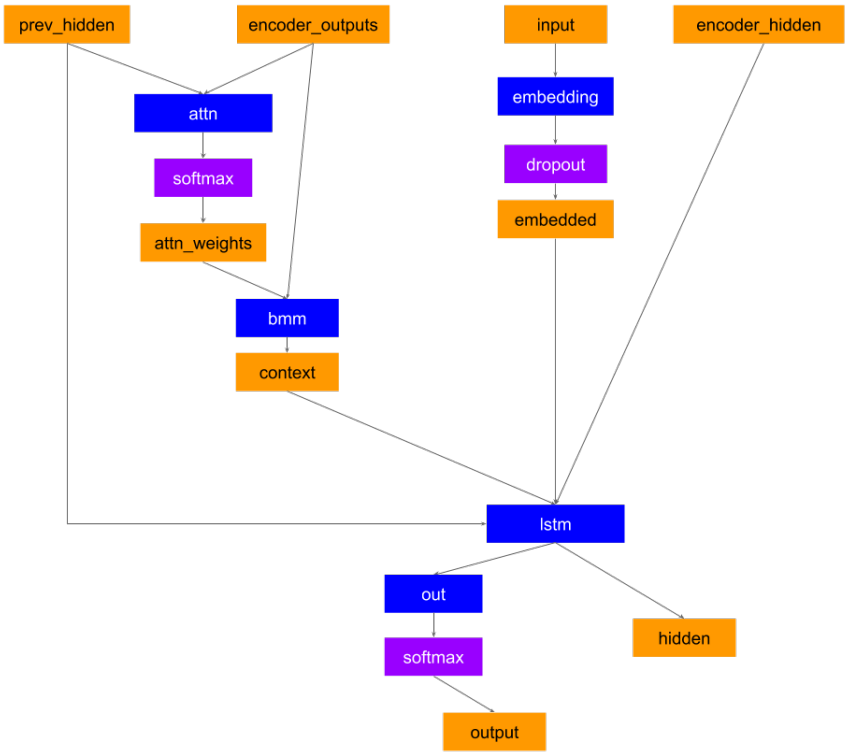

Figura 3.4: Attention Decoder

3.1.3 Mulțimea de antrenare

ATIS (Airline Travel Information Systems) [25] este o colecție de cereri venite din partea clienților referitor la zborurile aeriene de la începutul anilor 90. DARPA este instituția americana ce s-a ocupat de colectarea acestor date. Mulțimea de antrenare conține un număr de aproape 5000 de propoziții etichetate cu intenții si entități. Formatul de etichetare utilizat este IOB (Inside–outside–beginning), care ajuta la etichetarea entitatilor formate din mai multe cuvinte, astfel încât fiecare etichetă este precedata de B-<nume-eticheta> pentru început, iar cu I-<nume-eticheta> pentru următoarele cuvinte care fac parte din entitate. Exemplele sunt date în limba engleză, iar lungimea medie a unei propoziții este de 11 cuvinte.

Tabela 3.1: Exemplu din mulțimea de antrenare

Propoziție	flights	from	New	York	to	Cicago	on	wednesday	morning
Entități	O	O	B-from.loc	I-from.loc	O	B-to.loc	O	B-depart.day-name	B-depart.day-mood
Intenție	flight								

Tabela 3.2: Statistici ATIS

Multime de antrenare	#Train	#Dev	#Test	$\|V\|$	#Intenții	#Entități
ATIS	4978	500	893	722	21	120

3.1.4 Rezultate

Faptul că mulțimea de antrenare descrisă mai sus este folosită în majoritatea lucrărilor ce abordează aceleași probleme ne oferă avantajul de a compara rezultatele experimentelor noastre.

Metode de evaluare

Evaluarea a avut loc pe două metrici: $F1$ și eroarea de misclasare (Err).

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

$$Err = 1 - Accuracy$$

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{TotalPredictedPositive}$$

$$Precision = \frac{TP}{TP + FN} = \frac{TP}{TotalActualPositive}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

TP = True positive; FP = False positive; TN = True negative; FN = False negative

În tabelul 3.3 se face o comparație cu alte abordări. Modelul nostru are patru variante, una în care nu este folosită atenția și încă trei ce folosesc diferite variații ale modelului de atenție descris la 3.1.2.

Observăm re

3.1.5 Analiza erorilor

Modelul folosit ia decizii în principal pe modul cum cuvintele apar într-o propoziție. Aici avem erori în cazurile în care întâlnim în locuri neașteptate cuvinte care nu se află în dicționar. Bineînțeles ele sunt înlocuite cu un token <unknown> dar asta nu e de ajuns deoarece în mulțimea de antrenare acest token apare în text în alte

Model	**Entități(F1)**	**Intenții(F1)**	**Intenții(Err)**
Recursive NN [10]	-	-	4.60
Boosting [11]	-	-	4.38
Boosting + Simplified sentences [12]	-	-	3.02
Attn Enc-Dec NN [4]	-	-	2.02
Attn BiRNN [4]	-	-	2.35
CNN-CRF [13]	94.35	-	-
RNN with Label Sampling [14]	94.89	-	-
Hybrid RNN [18]	95.06	-	-
Deep LSTM [15]	95.08	-	-
RNN-EM [16]	95.25	-	-
Encoder-labeler Deep LSTM [17]	95.66	-	-
Attn Enc-Dec NN [4]	95.78	-	-
Attn BiRNN [4]	95.75	-	-
RecNN [10]	93.22	-	4.60
RecNN+Viterbi [10]	93.96	-	4.60
Attn Enc-Dec NN [4]	95.87	-	1.57
Attn BiRNN [4]	95.98	-	1.79
Joint BERT + CRF [26]	96.1	97.9	-
Bi-model [24]	96.89	-	**1,01**
Seq2SeqX (no - attn)	**97.20**	96.51	3.01
Seq2SeqX (dot - attn)	97.01	**96.90**	**2.78**
Seq2SeqX (general - attn)	96.97	96.31	3.35
Seq2SeqX (concat - attn)	97.18	96.44	3.24

Tabela 3.3: Antrenare simultană pentru detectarea intenției și recunoașterea entităților. Rezultatele pentru abordările anterioare sunt citate din [4]

Model	Entități(F1)	Intenții(F1)	Intenții(Err)
Seq2SeqX (luong - dot - attn)	97,10	96,60	-
Seq2SeqX (luong general - attn)	97.0	96,67	-
Seq2SeqX (luong concat - attn)	97.14	96.41	-
Seq2SeqX (no - attn)	97.14	96.41	3.24
Seq2SeqX (no - attn)	97.34	96.14	3.47
Seq2SeqX (no - attn)	97.34	96.73	2.91
Seq2SeqX (no - attn)	97.39	96.64	3.02
Seq2SeqX (bahdanau - dot - attn)	96.73	96.90	-
Seq2SeqX (bahdanau - dot - attn)	96.96	96.88	2.78
Seq2SeqX (bahdanau - general - attn)	96.86	96.08	-
Seq2SeqX (bahdanau - general - attn)	96.97	96.31	3.35
Seq2SeqX (bahdanau concat - attn)	97.21	96.73	-
Seq2SeqX (bahdanau concat - attn)	97.18	96.44	3.24

Tabela 3.4: Antrenare simultană pentru detectarea intenției și recunoașterea entităților. Rezultatele pentru abordările anterioare sunt citate din [4]

ipostaze. Pentru propoziția `fly to Bucharest on march tenth and arriving on march 12` avem figura 3.5 ce ilustrează eroare apărută în momentul în care folosim un cuvânt care nu aparține dicționarului.

Dacă folosim `New York` în loc de `Bucharest` modelul etichetează cum trebuie. Figura 3.6

3.2 Administrator de dialog

Într-un sistem de dialog au loc mai multe schimburi de informații între utilizator și agent. Acest flux de date nu poate exista fără context, limbaj comun, intenții, entități și nu în ultimul rând acțiuni de ambele părți. Partea responsabilă de înțelegerea limbajului natural ne oferă un cadru semantic compus din intenție și entități la fiecare replică a utilizatorului. Administratorul de dialog este unitatea centrală care coordonează activitatea celorlalte componente, el se folosește de anumite politici pentru a ști ce acțiuni să facă mai departe știind contextul actual.

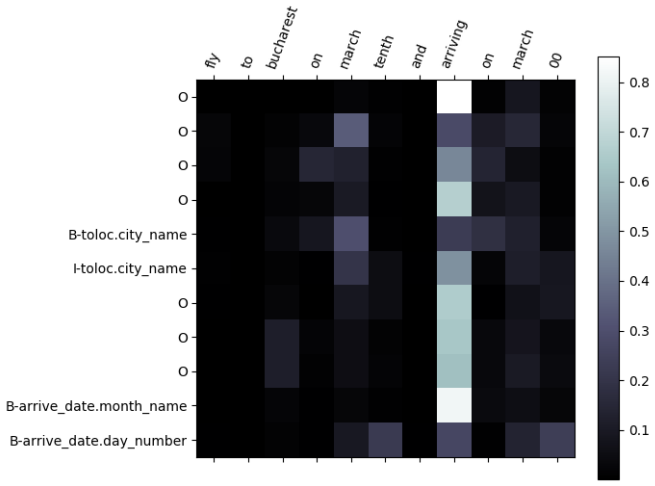

Figura 3.5: Rezultat eronat

Două procese importante au loc aici și anume reprezentarea contextului și inferența responsabilă de următoarea acțiune luată de agent. Ca urmare a acțiunilor sale putem avea interogări ale sistemelor externe, întrebări suplimentare sau notificări privind starea unui anumit obiectiv. De asemenea tot el se ocupă cu situațiile neclare sau cu cazurile aflate în afara competenței sistemului.

3.2.1 Abordări anterioare

De-a lungul timpului s-au propus diferite abordări de a gestiona dialogul, majoritatea reprezintă contextul conversației ca o stare a dialogului ce persistă într-o structură de date și utilizează diferite mecanisme de decizie pentru a selecta acțiunea corectă. În continuare se vor prezenta pe scurt unele dintre cele mai folosite abordări de a gestiona dialogul [27, 28].

Automat finit determinist - Finite State Automata

Această metodă este ce mai simpla și cea mai întâlnită, ea presupune ca stările unui dialog să fie chiar stările unui automat finit, iar tranzițiile să se realizeze pe baza unor condiții stabilite. Avantajul acestei metode il reprezintă în sine determinismul dat de automat, pentru că în majoritatea cazurilor practice se dorește un comportament stabil, a cărui fir de decizie să fie ușor de despicat. Dezavantajul este acela că agentul va avea întotdeauna inițiativa dialogului, lucru ce descrie o conversație nu în tocmai

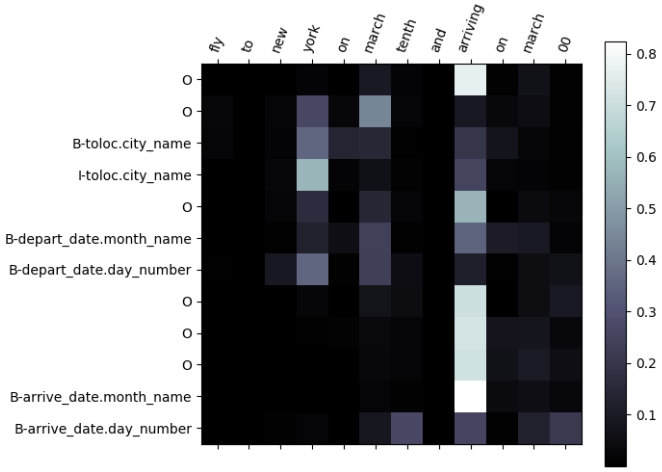

Figura 3.6: Rezultat corect

naturală.

Cadru de informații - Frame based/Slot filling

Metoda impune schematizarea unui scenariu de dialog într-o structură tabelară. Ea mai este întâlnită și sub denumirea de umplere de sloturi. Un slot este o entitate semantică văzută ca un argument ce va fi umplut de modulul de NLU, iar apoi când au fost umplute suficiente sloturi, are loc acțiunea din partea administratorului de dialog. Altfel spus un cadru de informație este compus din sloturi necesare și un obiectiv al utilizatorului. În mare parte această abordare captează strategia de bază al unui agent, scutind munca depusa pentru același comportament descris într-un automat finit, ba mai mult dă o flexibilitate dezvoltării datorită structurii tabelare ce poate fi importată. Un alt avantaj este acela că inițiativa la discurs se găsește într-un echilibru, utilizatorul având posibilitatea să ofere mai multe sloturi în aceeași replică. Faptul că suntem constrânși la un singur scenariu reprezintă un dezavantaj.

Plan

Abordările bazate pe un plan tratează fiecare replică din dialog ca o acțiune din partea utilizatorului spre a-și atinge obiectivul. Agentul detectează intenția, apoi construiește un plan de comunicare pentru a-l ajuta pe interlocutor să își atingă scopul.

Învățare de strategii

Datorită recentelor descoperiri în procesarea limbajului natural [29] și a progreselor

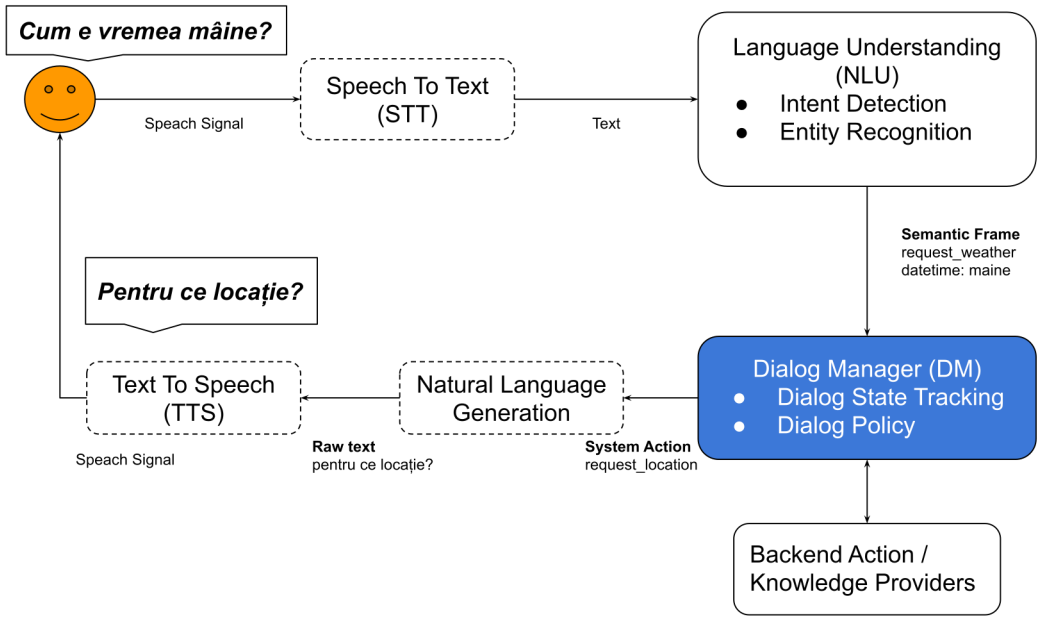

Figura 3.7: Administrator de dialog

din învățarea automată [30], reprezentarea informației și inferența au devenit din ce în ce mai performante. Ceea ce coincide cu necesitățile unui sistem de dialog unde știm ca avem nevoie de o metodă care să poată reprezenta contextul și încă una de a rezona pe baza acestuia. De aici și numărul mare de lucrări științifice în această direcție.

În abordările de acest gen se pleacă de la aceleași principii și anume o mulțime de stări ale conversației și o altă mulțime de acțiuni pe care agentul le poate efectua. Privind conversația ca un proces parțial observat putem atașa un model statistic (partially observable Markov decision process - POMDP), care să ne permită o tranziție din starea A în starea B pe baza unei probabilități.

Avantajele acestor metode sunt legate de automatizarea dezvoltării unui administrator de dialog. Învățarea politicilor de decizie se realizează doar pe baza interacțiunilor anterioare dintre utilizatori și agent. Îmbunătățiri se observă și în calitatea conversației, deoarece antrenarea recompensează conversațiile încheiate cu succes și

care durează mai puțin (ca număr de replici). Un plus este adus și în gestionarea situațiilor neprevăzute, ceea ce nu găsim la metodele deterministe descrise anterior.

Puterea oferită de aceste abordări vine și cu costuri: mulțimi de antrenare mai mari - detaliate la nivel de conversație și antrenarea într-un mediu ce simulează utilizatorul final. Bineînțeles faza de învățare poate avea loc și online, dar iar ne vom lovim de lipsa datelor, mai presus de atât, expunerea unui agent neantrenat riscă să deranjeze majoritatea utilizatorilor care au sosit în sistem pentru o cu totul altă problemă.

3.2.2 Model propus

Scopul dezvoltării agenților conversaționali este de a îmbunătății interacțiunea dintre oameni și tehnologie. Prin caracterul său natural această conexiune impune standarde ridicate în experiența utilizatorului și în eficiența de a rezolva cerințele acestuia. Luând în considerare fragilitatea unui dialog, dar și legătura dintre comportamentul agentului și calitatea conversației, de cele mai multe ori se recurge la implementări bazate pe reguli. Una dintre cela mai populare este cea bazată pe umplerea de sloturi (cadru de informații) care se caracterizează prin următoarele avantaje:

- inițiativa la discurs se găsește într-un echilibru, astfel utilizatorul nu se mai simte ghidat ca într-un formular ci mai degrabă într-o discuție
- utilizatorul are posibilitatea să ofere mai multe sloturi în aceeași replică, ceea ce crește semnificativ experiența cu agentul care de data aceasta recunoaște mai multe entități într-o propoziție, evitând pe viitor să mai întrebe despre ele
- se vede ușor procesul de decizie, fiind un model bazat pe reguli ne dă posibilitatea să mergem pe fir înapoi și să identificăm ipotezele unor anumite acțiuni
- comportament previzibil, o oarecare robustețe este impusă atunci când dorim să avem control asupra sarcinilor gestionate
- ușor de importat scenarii noi

- nu necesită mulțimi de antrenare la fel de bine construite precum modelele statistice, aici fiind nevoie doar de un expert în domeniu care să structureze scenariile de dialog

Dezavataje

- complexitatea structurii de dialog este proporțională cu numărul de scenarii gestionate, lucru ce limitează această abordare la un număr rezonabil de sarcini
- puterea de scalabilitate la domenii diferite este limitată de lipsa și calitatea datelor de antrenare

O îmbunătățire adusă metodei standard de umplere a sloturilor o reprezintă capacitatea de a trata obiective multiple. Pentru dezvoltarea administratorului de dialog, s-a folosit un model bazat pe cadre de informații (frame based/slot filling) ce poate trata mai multe obiective. Pentru a ține evidența intențiilor și entităților se folosește o bază de date care descrie aceste concepte. Ea poartă numele de ontologie a domeniului și este folosită în comun de componentele de NLU și DM pentru a asigura consistența cerută în comunicarea celor două.

Structura dialogului este o altă sursă de date care indică administratorului de dialog logica scenariilor posibile și obiectivele/sarcinile gestionate de către agent. În acest caz sarcină este sinonim cu obiectiv, deoarece ele se referă într-un final la același concept. Astfel un obiectiv al utilizatorului devine o sarcină a sistemului. Deci vorbind din perspectiva consumatorului, un obiectiv este o structură ce pune la o laltă intenția și entitățiile necesare pentru ca acel obiectiv să poată avea loc.

Administratorul de dialog este format din două componente:

Gestionarea stării conversației (Dialog state tracking) care ține istoria sesiunii de dialog grupată în intenții și entități menționate. Conform structurii de dialog această componentă reușește să mențină o evidența a obiectivelor pe parcursul întregii conversații.

Politicile de decizie (Dialog policy) sunt metode/reguli prin care agentul decide

să facă anumite acțiuni, de exemplu să întrebe despre un slot lipsă sau să interogheze sistemul rezolvând o anumită sarcină.

Pentru a valida această abordare am construit o ontologie (anexa 6.1) bazată pe mulțimea de antrenare folosită la modulul de NLU și am creat o structură de dialog (anexa 6.2) specifică domeniul de zboruri în care se încadrează datele.

Pe baza dialogului din 3.5 vom putea analiza procesul de decizie din spatele administratorului de dialog. Utilizatorul formulează o cerere către sistem: "I need a flight from Denver to Philadelphia". În acest moment gestionarea dialogului este inițializată de cadrul semantic venit de la NLU în forma <intenție>(entități).

Conform structurii de dialog (anexa 6.2) avem următorul tabel 3.6. După prima replică el este completat parțial, iar următoarele acțiuni vor veni în sprijinul umplerii sloturilor necesare, așadar se va alege la întâmplare o entitate nemenționată și se va întreba despre ea (în 6.2). În cazul în care se specifică mai multe entități ele se completează în tabel fără a mai întreba în viitor de ele, iar când toate informațiile necesare devin disponibile agentul răspunde cererii utilizatorului. De menționat aici că replicile agentului sunt descrise într-o formă de șablon, de exemplu `"Done! That's your flights for {depart_date.month_name} {depart_date.day_number} flying from {fromloc.city_name} to {toloc.city_name}"`, permițând răspunsuri bazate pe datele cunoscute.

	Replică	**Cadru semantic**
Sistem:	Hello! I am your flights assistant. How can I help you?	`welcome()`
Utilizator:	I need a flight from Denver to Philadelphia	`flight(fromloc.city_name=Denver, toloc.city_name=Philadelphia)`
Sistem:	When do you want to arrive?	`request(arrival_date)`
Utilizator:	arrive on monday, june fifth	`flight(arrive_date.day_name=monday, arrive_date.month_name=june, arrive_date.day_number=fifth)`
Sistem:	When would you like to leave?	`request(depart_date)`
Utilizator:	on june first in the evening	`flight(depart_date.month_name=june, depart_date.day_number=first, depart_time.period_of_day=evening)`
Sistem:	Done! That's your flights for **june first** flying from **Denver** to **Philadelphia** <system_call>	`inform(fromloc.city_name=Denver, toloc.city_name=Philadelphia, depart_date.month_name=june, depart_date.day_number=first)`

Tabela 3.5: Conversație cu agentul rezultat

Intenție	**Entități necesare**
`flight`	`fromloc.city_name=Denver`
	`toloc.city_name=Philadelphia`
	`depart_date.month_name=None`
	`depart_date.day_number=None`
	`arrive_date.month_name=None`
	`arrive_date.day_number=None`

Tabela 3.6: Cadru de informații pentru obiectivul atis_flight

Capitolul 4

Noțiuni teoretice

4.1 Rețele Neurale Recurente

Rețelele neuronale recurente (RNN - Recurent Neural Networks) sunt o arhitectura aparte de rețele neuronale, ce le face atat de speciale este faptul ca ele reușesc sa capteze secvențialitatea datelor. Ele sunt folosite in special în procesarea limbajului natural, dar și în procesarea imaginilor, a seriilor de timp, a recomandarilor de produse. Cu alte cuvinte oricând vine vorba de succesiunea anumitor evenimente, ele reprezintă un candidat bun în captarea acestor modele in date.

Figura 4.1 prezintă arhitectura unei rețele recurente, unde fiecare dreptunghi ține locul stratului ascuns de la pasul, t. Fiecare strat ascuns este format din perceptroni care execută operația de înmulțire intre parametrii și input, urmată de o operație nonlineară (ex.$tanh$). La fiecare pas din timp, ieșirea de la pasul anterior, împreună cu vectorul următorului cuvânt, x_t, sunt intrări în stratul ascuns care produce pentru pasul următor, ieșirea y, și vectorul de caracteristici al stratului ascuns, h.

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}) \tag{4.1}$$

$$y_t = softmax(W^{(S)}h_t) \tag{4.2}$$

Descrierea parametrilor din rețea:

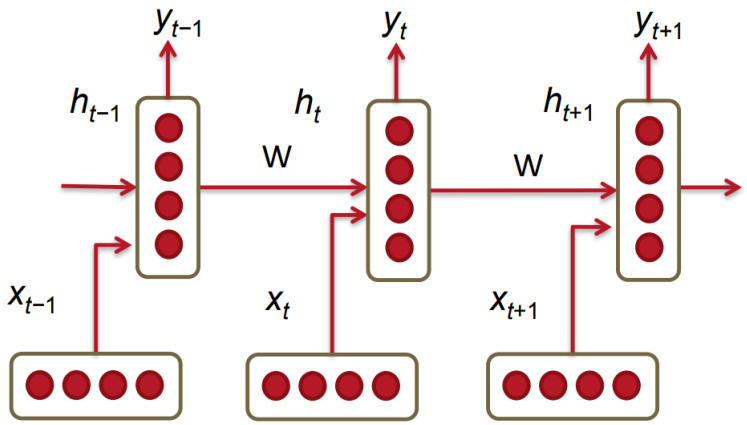

Figura 4.1: Arhitectura RNN [1]

- $x_1, x_2, ..., x_T$ vectorii cuvintelor dintr-o secvență de lungime T
- $h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]})$: formula care descrie calculul vectorului de caracteristici, h_t, la fiecare pas t:
  - $x_t \in \mathbb{R}^d$ vectorul cuvântului t
  - $W^{hx} \in \mathbb{R}^{D_h \times d}$ matricea ponderilor utilizată la condiționarea vectorului de intrare, x_t
  - $W^{hh} \in \mathbb{R}^{D_h \times D_h}$ matricea ponderilor utilizată la condiționarea ieșirii de la pasul anterior, h_{t-1}
  - $h_{t-1} \in \mathbb{R}^{D_h}$ ieșirea funcției nonlineare de la pasul anterior, $t-1$, iar $h_0 \in \mathbb{R}^{D_h}$ este vectorul de inițializare pentru stratul ascuns, la pasul $t = 0$
  - $\sigma()$ funcția nonlineară (sigmoid in acest exemplu)
- $y_t = softmax(W^{(S)}h_t)$ ieșirea care reprezintă o distribuție de probabilitate peste vocabular la fiecare pas t. În esență, y_t este următorul cuvânt prezis, dându-se contextul de până acum (h_{t-1}) și ultimul cuvânt observat (x_t). Avem $W^S \in \mathbb{R}^{|V| \times D_h}$ și $y \in \mathbb{R}^{|V|}$, unde $|V|$ reprezintă cardinalitatea mulțimii V adică a vocabularului de cuvinte.

Funcția de cost utilizată in rețelele neuronale recurente este de obicei CE (cross entropy error).

Pentru pasul t:

$$J_{(t)}(\theta) = -\sum_{j=1}^{|V|} ytrue_{t,j} \times \log(y_{t,j})$$

Pentru toată secvența de lungime T funcția de cost devine:

$$J = -\frac{1}{T}\sum_{t=1}^{T} J_{(t)}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} ytrue_{t,j} \times \log(y_{t,j})$$

[1]

4.1.1 Problema diminuării si a exagerării gradientului

O RNN împarte aceeași matrice a parametrilor pentru toată secvența de intrare. Țelul unei astfel de arhitecturi este acela de a capta contextul și în cazul secvențelor de lungimi mai mari.

Considerăm ecuațiile 4.1 și 4.2 de mai sus la pasul t. Pentru a calcula eroarea unei RNN, dE/dW, vom însuma erorile de la fiecare pas de timp.

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \tag{4.3}$$

Eroarea la fiecare pas de timp este calculată prin diferențierea ecuațiilor 4.1 și 4.2:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W} \tag{4.4}$$

Unde $\frac{dh_t}{dh_k}$ este derivata parțială a stării ascunse h_t în raport cu toți k pași de timp anteriori.

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \times diag[f'(j_{j-1})] \tag{4.5}$$

Deoarece $h \in \mathbb{R}^{D_n}$, fiecare $\partial h_j/\partial h_{j-1}$ este matricea Jacobian pentru h:

$$\frac{\partial h_j}{\partial h_{j-1}} = [\frac{\partial h_j}{\partial h_{j-1,1}} ... \frac{\partial h_j}{\partial h_{j-1,D_n}}] = \begin{bmatrix} \frac{\partial h_j,1}{\partial h_{j-1,1}} & \cdots & \frac{\partial h_j,1}{\partial h_{j-1,D_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_j,D_n}{\partial h_{j-1,1}} & \cdots & \frac{\partial h_j,D_n}{\partial h_{j-1,D_n}} \end{bmatrix} \tag{4.6}$$

Din 4.4, 4.5, 4.6 avem următoarea relație:

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} (\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}) \frac{\partial h_k}{\partial W} \tag{4.7}$$

Inecuația 4.8 arată norma matricei Jacobian, unde β_w și β_h reprezintă limitele superioare ale celor doua norme ale matricelor. Norma gradientului de la fiecare pas t este calculată în funcție de inecuația de mai jos.

$$||\frac{\partial h_j}{\partial h_{j-1}}|| \leq ||W^T|| ||diag[f'(h_{j-1})]|| \leq \beta_w \beta_h \tag{4.8}$$

Norma L2 este folosită în aceste inecuații. Iar norma derivatei $f'(h_{j-1})$ nu poate depăși valoarea 1, deoarece $f = sigmoid$.

$$||\frac{\partial h_j}{\partial h_k}|| = || \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}|| \leq (\beta_w \beta_h)^{t-k} \tag{4.9}$$

Termenul $(\beta_w \beta_h)^{t-k}$ poate foarte ușor să devină foarte mic sau foarte mare, atunci când $\beta_w \beta_h$ este mai mare ca 1 și $t - k$ suficient de mare. Obținem valori mari pentru $t-k$ în cazul in care se dorește evaluarea funcției de criteriu pentru cuvintele (intrările) mai îndepărtate. Așadar contribuția intrărilor de la pașii inițiali se diminuează cu cât ne îndepărtăm în timp.

În timpul experimentelor s-a observat că odată ce valoare gradientului crește foarte mult, poate cauză erori de memorie (valoarea nu încape în tipul de date specificat) ceea ce rezultă în valori nedefinite (exemplu NaN (not a number)), lucru ușor detectabil la timpul rulării. Această problemă poartă denumirea de exagerarea valorilor gradientului.

Referitor la problema diminuării gradientului, ea privește cazul în care valorile pentru gradient devin apropiate de zero, de multe ori acest comportament trece nedetectat, rezultând astfel o calitate mai slabă a învățării cuvintelor îndepărtate.

4.1.2 Soluții pentru dispariția/exagerarea gradientului

Referitor la problema exagerării valorilor gradientului, Thomas Mikolov, introduce pentru prima dată o euristică simplă - când gradientul tinde să aibă valori exagerate, ele se vor reseta la valori mici.

Cu privire la problema dispariției gradientului, se folosesc de regulă două metode. Una din ele abordează modul de inițializare al parametriilor astfel că în locul unei inițializări aleatoare se folosește matricea identitate. Iar cea de-a doua tehnică propune folosirea funcției de activare ReLu (Rectified Linear Units) în detrimentul celei sigmoid. Motivul este acela că derivata funcției relu este fie 1 fie 0, iar la propagarea erorii înapoi vor fi afectați doar neuronii a căror derivată este 1.

4.1.3 Seq2Seq

Modelul de arhitectură seq2seq se referă la translatarea unei secvențe în alta, de aceea ele sunt cu succes folosite în machine translation. Acest model are două componente importante: cea care face codificarea (captează înțelesul secvenței de intrare) și cea de decodificare înțelesului într-o secvență de ieșire. În funcție de specificul problemei, vocabularul de intrare și de ieșire pot să difere.

Ecuația 4.10 descrie operațiile necesare pentru a codifica în h_t înțelesul secvenței de intrare. Apoi decodificare se realizează cu ajutorul ecuațiilor 4.11 și 4.12 pentru a scoate o distribuție de probabilitate peste cuvintele din vocabularul de ieșire la fiecare pas t.

$$h_t = \phi(h_{t-1}, x_t) = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t) \tag{4.10}$$

$$h_t = \phi(h_{t-1}) = f(W^{(hh)}h_{t-1}) \tag{4.11}$$

$$y_t = softmax(W^{(S)}h_t) \tag{4.12}$$

$$max_\theta \frac{1}{N} \sum_{n=1}^{N} \times log(p_\theta(y^{(n)}|x^{(n)})) \tag{4.13}$$

Pentru ajunge la o performanță crescută, pe lângă funcția de cost 4.13 se adaugă mai multe extensii.

- Extensia 1: se vor folosi ponderi diferite pentru fiecare rețea în parte, adică în ecuația 4.11 în loc de $W^{(hh)}$ vom avea o matrice diferită.
- Extensia 2: fiecare stare ascunsă din rețeaua care decodifică va fi compusă din trei intrări diferite:
  - starea anterioară h_{t-1}
  - ultima stare din encoder ($c = h_T$)
  - ultima ieșire prezisă, y_{t-1}.
- Extensia 3: creșterea numărului de straturi ascunse din encoder și decoder
- Extensia 4: antrenarea unui encoder bidirecțional
- Extensia 5: antrenarea folosind la secvența de intrare în ordine inversă

4.1.4 GRU

Pe lângă extensiile aduse arhitecturii de codare-decodare, o îmbunătățire majoră are loc atunci când modelului matematic din interiorul celulei RNN i se adaugă mai multe porți ce simulează comportamentul unei memorii. (numim celulă a unei rețele neuronale recurente, partea recurentă a modelului care folosește aceleași ponderi de-a lungul secvenței).

Bineînțeles în teorie RNN captează dependințele de lungă durată de-a lungul unei secvențe, dar un asemenea model devine foarte greu de antrenat când vine vorba de cazuri practice. Așadar prin introducerea acestor unități recurente cu porți, rețeaua neuronală recurentă va căpăta o memorie mult mai persistentă fiind mult mai ușor de captat dependințele de lungă durată.

Prima arhitectură de acest gen este Gated Recurrent Units (GRU)

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \tag{4.14}$$

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \tag{4.15}$$

$$\widetilde{h_t} = \tanh(r_t \circ Uh_{t-1} + Wx_t) \tag{4.16}$$

$$h_t = (1 - z_t) \circ \widetilde{h_t} + z_t \circ h_{t-1} \tag{4.17}$$

1. Generarea unei noi memorii: ecuația 4.16 surprinde consolidarea intrării x_t cu starea anterioară. Acest pas este răspunzător de îmbogățirea contextului cu noua intrare venită.

2. Poarta de resetare: semnalul r_t din 4.15 determină cât de important este h_{t-1} pentru a fi luat în calcul de $\widetilde{h_t}$ care calculează noua memorie.

3. Poarta de actualizare: semnalul z_t din 4.14 este responsabil de propagarea contextului surprins anterior de h_{t-1}. De exemplu dacă $z_t \approx 1$ atunci h_{t-1} este aproape copiat în întregime în h_t, altfel dacă $z_t \approx 0$ noua memorie $\widetilde{h_t}$ este transmisă mai departe în h_t

4. Starea ascunsă: h_t este generată folosind ca intrare starea de la pasul $t-1$ și noua memorie în concordanță cu semnalul de actualizare.

4.1.5 LSTM

Long-Short-Term-Memories este un alt timp de celulă recurentă, care folosește aceeași filosofie a porților, dar într-un mod puțin mai diferit față de GRU.

$$i_t = \sigma(W^{(i)} x_t + U^{(i)} h_{t-1}) \tag{4.18}$$

$$f_t = \sigma(W^{(f)} x_t + U^{(f)} h_{t-1}) \tag{4.19}$$

$$o_t = \sigma(W^{(o)} x_t + U^{(o)} h_{t-1}) \tag{4.20}$$

$$\widetilde{c_t} = \tanh(W^{(c)} x_t + U^{(c)} h_{t-1}) \tag{4.21}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t} \tag{4.22}$$

$$h_t = o_t \circ \tanh(c_t) \tag{4.23}$$

1. Generarea unei noi memorii: ecuația 4.21 captează noi aspecte introduse de intrarea x_t

2. Poarta de intrare: se asigură că intrarea de la pasul t merită să fie luată în calcul pentru calculul memoriei curente: ecuația 4.18

3. Poarta de uitare: această poartă este similară cu cea de intrare, diferența este că ea determină utilitatea memoriei precedente în calculul celei curente.

4. Generarea memoriei finale: acest pas este descris de ecuația 4.22 care agregă sub semnul operației de adunare deciziile celor doua porți, de uitare și de intrare cu privire la memoria precedentă $c_{t-1}r$, respectiv noua memorie $\widetilde{c_t}$.

5. Poarta de ieșire/expunere: ecuațiile 4.20, 4.23. Rolul acestei porți este de a separa memoria finală c_t de starea ascunsă, h_t. c_t conține informații care nu sunt neapărat esențiale pentru calculul stării ascunse. Semnalul produs de această poartă, o_t este folosit la ponderarea utilizării memoriei finale în calculul stării, h_t.

Capitolul 5

Expunerea tehnologiei

5.1 Serviciul NLU

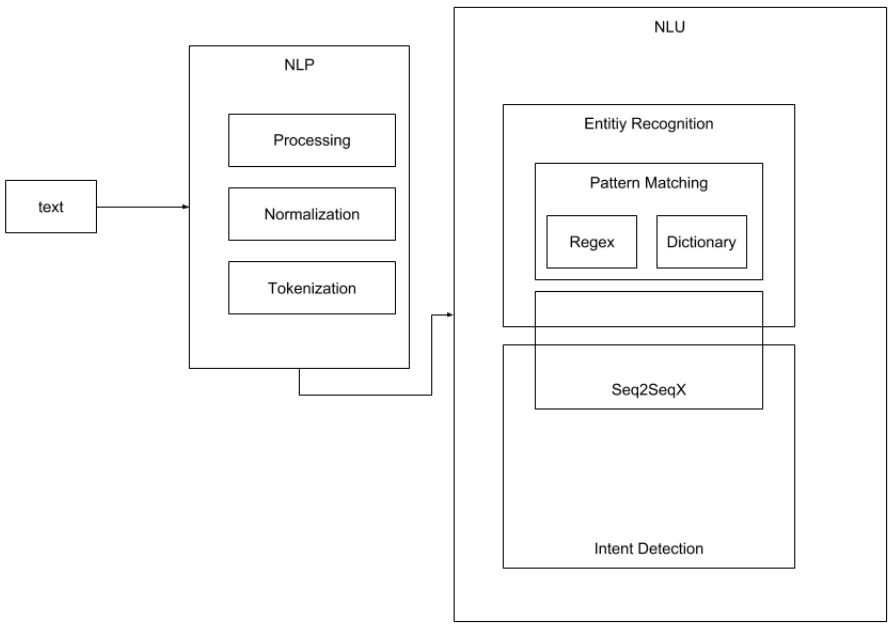

Figura 5.1: Arhitectura serviciului de NLU

5.2 Tehnologii folosite

Desemnarea tehnologiilor open source folosite ca dependințe poate fi văzută la prima vedere o alegere ușoară, însă este nevoie de o analiză mult mai amănunțită în ceea ce privește specificul proiectului. Pentru această lucrare am luat în considerare următorii factori: complexitatea de a descrie o rețea neuronală, modul de expunere (la nivel de API) al proceselor matematice din spate și flexibilitatea de a putea jongla cu diferite arhitecturi de rețele. De asemenea eficiența și posibilitatea de a rula pe diferite unități de calcul (CPU/GPU).

5.2.1 Python

Limbajul de programare utilizat în implementarea modelelor propuse este Python. El este folosit de la dezvoltare de servicii web (django, flask) până la biblioteci software de învățare automată (pytorch, tensorflow). Datorită popularității sale el se bucură din plin de licențierea open source, având o comunitate impresionantă de oameni care contribuie la dezvoltarea acestui ecosistem, cel puțin în partea de inteligență artificială el este instrumentul cel mai folosit și are cele mai multe pachete care îi extind funcționalitatea.

Python a fost creat la începutul anilor 1990 de Guido van Rossum la Stichting Mathematisch Centrum (CWI) în Olanda ca un succesor al limbajului ABC. [31]. Este un limbaj orientat pe obiecte ce vine însoțit de structuri de date la nivel înalt (dicționare, mulțimi, liste), uneori împrumutând concepte din programarea funcțională (list comprehention, map/reduce), alteori din programarea paralelă (apeluri asincrone).

5.2.2 Numpy

Pentru lucrul cu matrice și alte operații ce impun operații algebrice vectoriale se folosește de obicei Numpy. El este un acronim pentru ”Numeric Python” sau ”Numerical Python”. Este un pachet fundamental pentru calculul științific in Python, ce furnizează funcții precompilate care se execută rapid cu scopul de a efectua operațiile matematice de rutină. Mai mult decât atât, NumPy îmbogățește limbajul de pro-

gramare cu structuri puternice de date pentru calculul eficient de vectori si matrice, implementarea sa suportând chiar și dimensiuni uriașe.

5.2.3 PyTorch

PyTorch [32] este o bibliotecă software ce oferă un cadru de lucru cu algoritmi de invățare automată. Se prezintă ca o variantă de Numpy care poate rula pe placa video, având tot odată și capacitatea de autodiferențiere atunci când este nevoie să antrenăm, spre exemplu folosind metoda gradientului descendent.

Diferențiere Automată

Componenta cheie a rețelelor neuronale din PyTorch este pachetul *autograd*. El oferă diferențierea automată pentru toate operațiile cu *tensori*. Este un cadru de definire a operațiilor (forward dar și backward) la momentul execuției, ceea ce înseamnă că pasul de backpropagation este definit de modul în care este rulat codul.

Tensor

torch.Tensor este clasa centrală a pachetului. Dacă se setează atributul `.requires_grad` ca `True`, se va începe urmărirea tuturor operațiilor în care acesta intervine. După ce se termină calculul, se poate apela `backward()` pentru a calcula automat toate derivatele, iar gradientul pentru acest tensor va fi acumulat în atributul `.grad`.

Pentru a opri tensorul din istoricul de urmărire, se apelează `.detach()` care detașează tensorul de istoricul de calcul și care împiedica urmărirea viitoarelor calcule.

Mai există încă o clasă care este foarte importantă pentru implementarea autodiferențierii - și anume *Function*.

Tensorul și funcția sunt interconectate și construiesc un graf aciclic, care codifică un istoric complet al calculelor. Fiecare tensor are un atribut `.grad_fn` care se referă la o funcție care a creat tensorul (cu excepția tensorurilor creați de utilizator unde - `.grad_fn = None`).

Capitolul 6

Concluzii

Vorbind dintr-o perspectivă academică, rezultatele obținute reușesc să convingă de adevărata putere a acestei arhitecturi. Datorită rețelelor recurente se reușește captarea contextului, fapt ce duce la o putere mare de generalizare. Capacitatea modelului de a prezice crește atunci când in joc intervine și atenția, acest strat care ajută estimatorul să se concentreze pe anumite cuvinte atunci când decide asupra unei etichete.

Din perspectiva comercializării unei astfel de soluții există puține impedimente, dar totuși notabile și anume: în faza de antrenare este nevoie de un număr mare de exemple etichetate. Un alt factor care stă în calea scalării numărului de clienți este nevoia unui expert în domeniu care să concentreze în mulțimea de antrenare intenții și exemple relevante. Construirea unei astfel de mulțimi de antrenare necesită și implicarea dezvoltatorului (sau a unei persoanei care cunoaște cum funcționează tehnologia), astfel încât să se asigure că exemplele construite de expertul în domeniu, au o oarecare consistență, spre exemplu: dacă s-a dat propoziția: ”vreau să imi resetez parola din aplicația Saturn” etichetat cu entitatea: app: Saturn, să existe și contra exemple în care să se specifice și celalalt sens al cuvântului ”Saturn”, astfel modulul de NLU să poată identifica cu succes atunci când este vorba de aplicație sau despre planetă.

Legat de componenta care ține contextul întregii conversații, este foarte ușor de urmărit întregul fir al discuției, datorită modelului bazat pe umplerea de sloturi (entități necesare în cadrul unei sarcini).

Bibliografie

[1] Richard Socher Milad Mohammadi, Rohit Mundra. Cs 224d: Deep learning for nlp, lecture notes: Part iv, 2015. URL http://cs224d.stanford.edu/lecture_notes/notes4.pdf. [Online, accesat la: 27.03.2019].

[2] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 2018.

[3] Bing Liu and Ian Lane. Joint online spoken language understanding and language modeling with recurrent neural networks, 2016.

[4] Bing Liu and Ian Lane. Attention-based recurrent neural network models for joint intent detection and slot filling, 2016.

[5] J Weizenbaum. Eliza. *Communications of the ACM, 9(1), 36–45*, 1966.

[6] Bing Liu and Ian Lane. Adversarial learning of task-oriented neural dialog models, 2018.

[7] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system, 2016.

[8] Xuesong Yang, Yun-Nung Chen, Dilek Hakkani-Tur, Paul Crook, Xiujun Li, Jianfeng Gao, and Li Deng. End-to-end joint learning of natural language understanding and dialogue manager, 2016.

[9] Mihail Eric and Christopher D. Manning. Key-value retrieval networks for task-oriented dialogue, 2017.

[10] W.-t. Yih D. Guo, G. Tur and G. Zweig. Joint semantic utterance classification and slot filling with recursive neural networks, 2014.

[11] D. Hakkani-Tur G. Tur and L. Heck. What is left to be understood in atis? *Spoken Language Technology Workshop (SLT), 2010 IEEE.*, page 19–24, 2010.

[12] L. Heck G. Tur, D. Hakkani-Tur and S. Parthasarathy. Sentence simplification for spoken language understanding. *Acoustics, Speech and Signal Processing (ICASSP)*, page 5628–5631, 2011.

[13] P. Xu and R. Sarikaya. Convolutional neural network based triangular crf for joint intent detection and slot filling. *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on.*, page 78–83, 2013.

[14] Bing Liu and Ian Lane. Recurrent neural network structured output prediction for spoken language understanding, 2015.

[15] Y. Zhang D. Yu G. Zweig K. Yao, B. Peng and Y. Shi. Spoken language understanding using long short-term memory neural networks. *Spoken Language Technology Workshop (SLT), 2014 IEEE.*, page 189–194, 2014.

[16] B. Peng and K. Yao. Recurrent neural networks with external memory for language understanding. 2015.

[17] Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. Leveraging sentence-level information with encoder lstm for semantic slot filling, 2016.

[18] K. Yao Y. Bengio L. Deng D. HakkaniTur X. He L. Heck G. Tur D. Yu et al. G. Mesnil, Y. Dauphin. Using recurrent neural networks for slot filling in spoken language understanding. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on, vol. 23, no. 3,*, page 530–539, 2015.

[19] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[20] Ludwig Wittgenstein. *Philosophical Investigations.* Translated by Anscombe, G.E.M. Blackwell, 1953.

[21] Alina Pătrunjel. Aspecte privind clasificarea actelor de vorbire din limba română. *Buletin de Lingvistică*, pages 15–16, 2015.

[22] A. M. Turing. Computing machinery and intelligence, 1950.

[23] Kaplan R. M. Kay M. Norman D. A. Thompson H. Bobrow, D. G. and T. Winograd. Gus, a frame driven dialog system. *Artificial Intelligence, 8*, page 155–173, 1950.

[24] Yu Wang, Yilin Shen, and Hongxia Jin. A bi-model based rnn semantic frame parsing model for intent detection and slot filling, 2018.

[25] J. J. Godfrey C. T. Hemphill and G. R. Doddington. The atis spoken language systems pilot corpus. *Proceedings, DARPA speech and natural language workshop*, pages 96–101, 1990.

[26] Qian Chen, Zhu Zhuo, and Wen Wang. Bert for joint intent classification and slot filling, 2019.

[27] Andre Niklasson. Dialogue systems using web-based language tools, 2017.

[28] Jungyun Seo1 Youngmin Park, Sangwoo Kang. An efficient framework for development of task-oriented dialog systems in a smart home environment, 2018.

[29] K. Chen G. Corrado J. Dean T. Mikolov, I. Sutskever. Distributed representations of words and phrases and their compositionality. *In Proceedings of NIPS 2013*, 2013.

[30] Jimmy Lei Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *Published as a conference paper at ICLR 2015*, 2015. URL http://arxiv.org/pdf/1412.6980v8.pdf.

[31] Python Software Fundation. History and license. URL https://docs.python.org/3/license.html. Online, accesat la: 11.06.2016.

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Anexe

Ontologie pentru ATIS

```
{
  "entities": {
    "aircraft_code": [],
    "airline_code": [],
    "airline_name": [],
    "class_type": [],
    "connect": [],
    "cost_relative": [],
    "days_code": [],
    "economy": [],
    "fare_amount": [],
    "fare_basis_code": [],
    "flight_days": [],
    "flight_mod": [],
    "flight_number": [],
    "flight_stop": [],
    "flight_time": [],
    "meal": [],
    "meal_description": [],
    "mod": [],
    "or": [],
    "restriction_code": [],
    "round_trip": [],
    "transport_type": [],
    "date_relative": [],
    "day_name": [],
    "day_number": [],
    "month_name": [],
    "today_relative": [],
    "year": [],
    "end_time": [],
    "period_mod": [],
```

```
    "period_of_day": [],
    "start_time": [],
    "time": [],
    "time_relative": [],
    "airport_code": [],
    "airport_name": [],
    "city_name": [],
    "state_code": [],
    "state_name": [],
    "arrive_date": ["date_relative", "day_name", "day_number", "
    month_name", "today_relative", "year"],
    "depart_date": ["date_relative", "day_name", "day_number", "
    month_name", "today_relative", "year"],
    "return_date": ["date_relative", "day_name", "day_number", "
    month_name", "today_relative", "year"],
    "arrive_time": ["end_time", "period_mod", "period_of_day", "
    start_time", "time", "time_relative"],
    "depart_time": ["end_time", "period_mod", "period_of_day", "
    start_time", "time", "time_relative"],
    "return_time": ["end_time", "period_mod", "period_of_day", "
    start_time", "time", "time_relative"],
    "fromloc": ["airport_code", "airport_name", "city_name", "state_code"
    , "state_name"],
    "stoploc": ["airport_code", "airport_name", "city_name", "state_code"
    , "state_name"],
    "toloc": ["airport_code", "airport_name", "city_name", "state_code",
    "state_name"]
  },

  "intents": {
    "atis_abbreviation": [],
    "atis_aircraft": [],
    "atis_airfare": [],
    "atis_airline": [],
    "atis_airport": [],
    "atis_aircraft#atis_flight#atis_flight_no": [],
    "atis_airfare#atis_flight_time": [],
    "atis_capacity": [],
    "atis_city": [],
```

```
    "atis_cheapest": [],
    "atis_distance": [],
    "atis_flight": [],
    "atis_flight_time": [],
    "atis_flight#atis_airfare": [],
    "atis_ground_fare": [],
    "atis_quantity": [],
    "atis_restriction": [],
    "atis_airline#atis_flight_no": [],
    "atis_flight_no": [],
    "atis_ground_service": [],
    "atis_ground_service#atis_ground_fare": [],
    "atis_meal": []
  }
}
```

Listing 6.1: Ontologie pentru ATIS

Structura dialogului pentru ATIS

```
{
  "domain": "flights_callcenter",
  "fallback_confidence_score": 0.3,
  "entities": {
    "fromloc.city_name": {
      "template_text": ["Where are you leaving from?", "What city are you leaving from?"]
    },
    "toloc.city_name": {
      "template_text": ["What is the destination city?", "Where are you going?"]
    },
    "depart_date.month_name": {
      "template_text": ["Please let me know the depart date", "When would you like to leave?"]
    },
    "depart_date.day_number": {
      "template_text": ["When would you like to leave?", "Can your repeat the depart date and specify the day?"]
    },
    "depart_date.day_name": {
      "template_text": ["Please specify the week day when you want to depart"]
    },
    "arrive_date.month_name": {
      "template_text": ["Please let me know the arrival date", "When would you like to land?",  "When do you want to arrive?"]
    },
    "arrive_date.day_number": {
      "template_text": ["When do you want to arrive?", "Can your repeat the arrival date and specify the day"]
    },
    "arrive_date.day_name": {
      "template_text": ["Please specify the week day when you want to arrive"]
    }
  },
  "intents": {
```

```
31      "welcome": {
32        "template_text": ["Hello! I am your flights assistant. How can I
        help you?"]
33      },
34      "fallback": {
35        "template_text": ["Sorry, I 'didnt get that. Can you be more
        specific?"]
36      },
37      "atis_flight": {
38        "template_text": ["Done! That's your flights for {depart_date.
        month_name} {depart_date.day_number} flying from {fromloc.city_name}
        to {toloc.city_name} \n<system_call>"],
39        "entities": [
40          "fromloc.city_name",
41          "toloc.city_name",
42          "depart_date.month_name",
43          "depart_date.day_number",
44          "arrive_date.month_name",
45          "arrive_date.day_number"
46        ]
47      },
48      "atis_quantity": {
49        "template_text": ["Here is the quantity from {fromloc.city_name} to
        {toloc.city_name} \n<system_call>"],
50        "entities": [
51          "fromloc.city_name",
52          "toloc.city_name"
53        ]
54      },
55      "atis_distance": {
56        "template_text": ["Here is the distance from {fromloc.city_name} to
        {toloc.city_name} \n<system_call>"],
57        "entities": [
58          "fromloc.city_name",
59          "toloc.city_name"
60        ]
61      },
62      "atis_abbreviation": {
63        "template_text": ["This code {flight_number}. \n<system_call>"],
```

```
64        "entities": [
65          "flight_number"
66        ]
67      }
68    }
69  }
```

Listing 6.2: Structura dialogului pentru ATIS