Universitatea Transilvania din Braşov
Facultatea de Matematică şi Informatică
Specializarea Tehnologii Moderne
în Ingineria Sistemelor Soft

LUCRARE DE DISERTAŢIE

Autor: Muşat Bogdan-Adrian
Coordonator: Conf. dr. Sasu Lucian-Mircea

Braşov
2017

Cuprins

Capitolul 1

Introducere

Procesarea de limbaj natural a reprezentat dintotdeauna una dintre marile provocări ale comunităţiilor ştiinţifice de pretutindeni, scopul final fiind realizarea unui sistem suficient de inteligent încât să fie capabil să comunice precum omul. În acest scop, în anul 1950 a fost lansat testul Turing [3] care presupune o conversaţie om-maşină şi un interlocutor care participă la această discuţie, acesta nefiind capabil să distingă omul de maşină. Acest test încă stă în picioare după aproape 70 de ani, niciun sistem fiind capabil încă să-l doboare.

Recent, diverse companii de succes precum Google, Facebook, Microsoft, Apple şi alţii au realizat un demers semnificativ spre dezvoltarea unor sisteme inteligente de comunicaţie numite chatbots. La ora actuală, cei mai mulţi sunt orientaţi către suport tehnic [5], scopul lor fiind tocmai de a imita sprijinul pe care o persoană reală îl poate oferi unui client. Un sistem computaţional inteligent poate fi folosit pentru a răspunde multor cereri simultane, iar costurile de intreţinere sunt mici practic, odată ce sistemul este dezvoltat, este necesară doar expunerea lui, de exemplu ca serviciu web. Cu cât tehnologia şi munca de cercetare în această direcţie progresează, cu atât sistemele de genul acesta devin din ce în ce mai inteligente şi mai apropiate de ceea ce un om este capabil să ofere. Precum automatizarea industrială de producere a vehiculelor reprezintă un standard în era contemporană, asistenţii conversaţionali vor deveni un standard în anii ce vor urma [4].

Majoritatea modelelor actuale se bazează pe oferirea unor răspunsuri predefinite [6]. Aceste tipuri de sisteme pot returna doar răspunsuri existente, nefiind capabile de a genera text nou. Pe de altă parte, avem sistemele bazate pe inteligenţă artificială generative [6] care, precum omul, pot emite răspunsuri bazate pe experienţe anterioare. Vorbim deci despre evoluţia unor

modele de chatbots bazate pe pattern matching către unele bazate pe modele generative. În ultimii ani, ramura inteligenţei artificiale numită Machine Learning (învăţare automată) a luat amploare în acestă direcţie prin curentul numit Deep Learning [2]. Acest curent a produs o mulţime de rezultate spectaculoase atât în direcţia procesării naturale de limbaj cât şi a procesării de imagini [1].

Rolul principal al unui chatbot îl reprezintă capabilitatea de a "întelege" informaţia primită de la o persoană pentru a produce un raspuns cât mai plauzibil. Însă cum procesează un calculator o limbă? Pentru a răspunde la această întrebare, se va face o analogie la cum învaţă un om o limbă. Porneşte de la anumite cuvinte de bază iar apoi pe baza acestora, învaţă cuvinte tot mai complexe. Începe să creeze fraze prin care leagă aceste cuvinte precum şi gramatica specifică limbajului. Practic, totul se bazează pe o anumită experienţă cumulativă. Conversaţia devine astfel o modalitate ce facilitează şi impulsionează deprinderea limbajului: omul este pus în faţa unor contexte de utilizare, fenomen ce întăreşte deprinderea de utilizare a cuvintelor sau expresiior individuale. Fiecare cuvânt nou învăţat reprezintă o experienţă către învăţarea în continuare a limbajului. Încă un aspect reprezentativ uman este capabilitatea de a întreţine o conversaţie pe termen lung cu un alt participant în intermediul unui context, lucru dificil de capturat pentru un sistem artificial.

Abordările de chatbot actuale îşi propun imitarea acestei modalităţi de învăţare pentru un sistem computaţional. Modelarea curentă cea mai eficientă pentru o astfel de problemă este oferită de către Deep Learning. Folosind arhitecturi de reţele neurale artificiale mult mai complexe decât cele shallow de la începutul anilor 2000 şi plecând de la seturi de instruire masive - în cazul de faţă, corpusuri de text cu cât mai multe fraze în limba dorită se pot crea modele generative care pot produce continuarea unor propoziţii, fraze etc. Ideea din spatele abordării Deep Learning este că sistemul devine mai performant pe măsură ce volumul de date creşte. Pentru a procesa o asemenea cantitate de date de instruire este nevoie de putere computaţională pe măsură. Asta presupune pe scurt, capacitate hardware. Milioanele de calcule efectuate pentru modelarea unui limbaj de către sistem nu permit folosirea unui procesor, chiar multicore de ultimă generaţie, deoarece este considerat a duce la sugrumarea procesului de instruire.

În locul microprocesoarelor se preferă folosirea plăcilor grafice (GPU). Iniţial, acestea au fost dezvoltate pentru rulare rapidă de jocuri, dar potenţialul

lor a fost rapid intuit şi exploatat prin programare paralelă. Deoarece placa video conţine mult mai multe nuclee de procesare (câteva mii, comparate cu cele 4-8 nuclee tradiţionale dintr-un microprocesor actual), este preferată programarea şi rularea modelelor computaţionale pe GPU. În ultimii ani au dezvoltate o multitudine de biblioteci care facilitează unui programator dezvoltarea de aplicaţii de Machine Learning pe GPU: Tensorflow, Theano, Caffe, Keras etc.

1.1 Motivaţia alegerii temei

Testul Turing reprezintă o provocare pe care mulţi cercetători din acest domeniu o abordează constant cu noi idei tot mai inovative, având din ce în ce mai mult succes în ultimii ani. Chiar şi la nivelul unui model generativ de jucărie, acest subiect reprezintă unul dintre cele mai fierbinţi topicuri la ora actuală pentru comunitatea Deep Learning. Acest fapt, precum şi cercetarea ideilor state of the art folosite în sistemele actuale reprezintă factorul motivaţional principal în alegerea temei de cercetare.

1.2 Structura lucrării

TODO

1.3 Recunoaştere

Mulţumiri Universităţii Transilvania din Braşov care a finanţat acest proiect în scopul achiziţionării de hardware necesar pentru realizarea acestuia.

Capitolul 2

Lucrări similare

În anul 1950, matematicianul Alan Turing a propus un test care pune la încercare abilitatea unei maşini computaţionale de a manifesta inteligenţă echivalentă cu cea umană. Un evaluator uman judecă o conversaţie între un om şi o maşină desemnată să genereze răspunsuri cât mai naturale. Evaluatorul este conştient că unul dintre partenerii angrenaţi in discuţie este o maşină. Conversaţia este limitată doar la text astfel încât rezultatul să nu depindă de abilitatea calculatorului de a genera sunete. Dacă evaluatorul nu poate diferenţia maşina de om, putem spune că aceasta a trecut testul. Până în prezent, acest test rămâne încă în picioare.

Există două mari abordări la ora actuală de a genera limbaj cât mai natural. Avem în primul rând modele de tip regăsire, unde răspunsul este stocat într-o sursă de date şi returnat pe baza unor metode de pattern matching. Cea de-a doua variantă o reprezintă modelele generative, care produc un răspuns dinamic folosind diverse metode din teoria probabilităţilor.

2.1 Modele de regăsire

Modelele de acest tip folosesc o sursă de date care conţine numeroase răspunsuri predefinite. Răspunsul este ales folosind o metodă euristică pentru o potrivire cât mai bună cu putinţă, luând în considerare intrarea şi contextul. Tipul de euristică folosit poate fi ceva simplu precum o expresie bazată pe o regulă de potrivire sau ceva mai complex cum ar fi un clasificator de Machine Learning. Se poate deduce foarte uşor că aceste sisteme nu generează text nou. O problemă uriaşă a acestor modele o reprezintă incapacitatea de a reacţiona la cazuri nemaiîntalnite pentru care nu există un răspuns potrivit. Acestea au totuşi avantajele lor. Datorită sursei de date cu răspunsuri create

de oameni, aceste metode nu produc erori gramaticale. În prezent, acestea reprezintă abordarea sigură pentru problemele în care răspunsul este unul sensibil, într-un domeniu precum cel medical, de exemplu.

2.1.1 Duolingo

Populara aplicaţie de învăţare a limbilor străine Duolingo (Figura 2.1) foloseşte o abordare interesantă cu privire la chatbots. Aceasta doreşte să îşi ajute utilizatorii să practice o nouă limbă prin conversaţii cu chatbots. Având în vedere că o conversaţie este considerată a fi printre cele mai bune moduri de a învăţa o limbă străină, utilizatorii Duolingo pot vorbi cu bot-ul oricât de mult îşi doresc, iar acesta îi va corecta şi le va propune răspunsuri potrivite. Mai mult de atât, poate estima progresul utilizatorului pentru a-şi creşte nivelul de dificultate, păstrând astfel constantă provocarea.

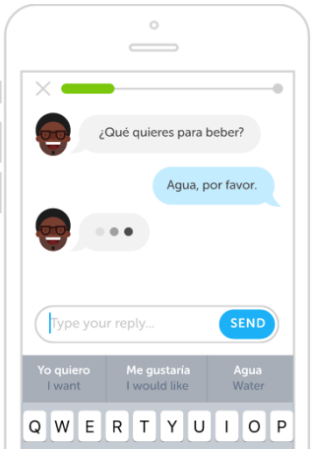

Figura 2.1: Conversaţie cu chatbot folosind Duolingo

2.1.2 Facebook Messenger

Facebook s-a alăturat întru totul afacerii conversaţionale, astfel încât şi-a transformat aplicaţia Messenger într-un business de mesagerie. Compania a integrat plaţile peer-to-peer în Messenger în anul 2015, apoi urmând să lanseze un API pentru chatbots, astfel încât business-urile să poată crea interacţiuni pentru clienţi. Poţi comanda flori, să navighezi printre ultimele trend-uri în materie de modă, să comanzi Uber, toate dinăuntrul chat-ului de Messenger (Figura 2.2).

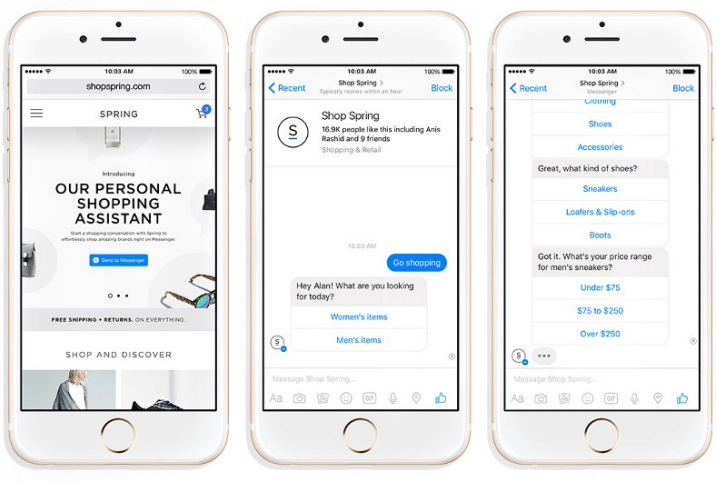

Figura 2.2: Cumpărături de haine folosind chatbot-ul companiei Spring pe Facebook Messenger

2.2 Modele generative

Spre deosebire de modelele anterioare, cele generative nu se bazează pe răspunsuri predefinite ci generează noi răspunsuri pornind de la zero. Modelele generative folosesc deobicei tehnici din Machine Translation, dar în loc de a traduce dintr-o limbă într-alta, vom "traduce" de la o intrare la o ieşire (răspuns). Acestea oferă o mai bună impresie de comunicare cu un om real. Totuşi, ele sunt extrem de greu de antrenat, sunt predispuse la erori gramaticale (în special unde lungimea propoziţiei este mai mare) şi necesită o cantitate mare de date de antrenare.

Prezentul este încă sub semnul întrebării pentru acest tip de model, însă în următorii ani, acestea vor căpăta tot mai multă atenţie şi popularitate devenind tot mai performante. Dacă un chatbot va doborî testul Turing, şansele ca acesta să fie generativ sunt destul de mari. Deoarece modelele generative reprezintă o arie de cercetare încă nefinisată, acestea nu sunt folosite momentan în producţie. În capitolul 3 vor fi prezentate diferite arhitecturi de reţele neurale, capabile să modeleze conversaţii.

Capitolul 3

Arhitectura

Modelarea unui limbaj reprezintă o sarcină extrem de dificilă pentru un calculator. Recente progrese în aria Deep Learning au făcut posibile diverse dezvoltări în această direcţie, însă lucrurile sunt departe de a fi rezolvate. Arhitectura folosită pentru construcţia chatbot-ului prezentat în această lucrare va fi expusă precum un bloc de construcţii, plecând de la noţiunile de bază, până la arhitectura finală.

3.1 Reţele neurale artificiale

O reţea neurală artificială (RNA) reprezintă o paradigmă bazată pe procesare de informaţii a cărei inspiraţie provine din sistemul nervos biologic. Precum creierul uman, o RNA este compusă dintr-un număr mare de elemente de procesare interconectate (neuroni) lucrând la unison pentru a rezolva diverse probleme. RNA, precum oamenii, învaţă din exemple. Învăţarea în sistemele biologice implică ajustarea conexiunilor sinaptice care există între neuroni. Acest principiu se aplică şi acestor reţele.

Ca şi modelare matematică propriu-zisă, o RNA poate fi observată în Figura 3.1. Avem o intrare reprezentată în figură de vectorul n-dimensional (x1, x2, x3, x4). Această intrare reprezintă caracteristicile unui eşantion care face parte dintr-un set de date. Straturile intermediare se numesc straturi ascunse, iar rolul lor este să producă o abstractizare cât mai complexă a datelor de intrare, folosind diverse funcţii cu activare neliniară. Ultimul strat se numeşte strat de ieşire, reprezentând eticheta (clasa) vectorului de intrare.

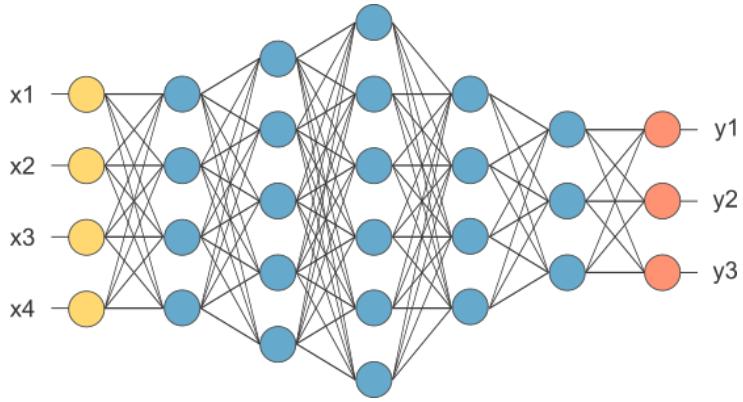

Figura 3.1: Exemplu de reţea neurală artificială

Ceea ce de fapt acest tip de reţele incearcă să inveţe sunt ponderile dintre straturile sale. Se pleacă de la un set de ponderi alese aleator[1] şi se ajustează folosind algoritmul de propagare înapoi a erorii până când reţeaua se stabilizează.

TO CONTINUE

3.2 Reţele neurale artificiale recurente

După cum se poate observa, o RNA este utilă atunci când intrarea este formată dintr-un singur element, de exemplu o imagine. Modelarea unui limbaj însă presupune ca intrările să fie fraze. O frază este formată din mai multe elemente (cuvinte), deci o RNA clasică nu poate modela o astfel de intrare. Soluţia pentru această problemă este oferită de reţelele neurale artificiale recurente (RNAR). Acestea sunt folosite pentru a modela secvenţe unde există o dependenţă temporală (fraze, serii de timp). Fiecare intrare curentă din secvenţă este condiţionată de cele precedente. O astfel de reţea se poate observa în Figura 3.2. Straturile ascunse într-o RNAR au rolul de a păstra o captură a tot ceea ce s-a oferit ca intrare până în momentul curent. Valoarea fiecărui strat ascuns curent depinde astfel atât de intrarea curentă cât şi de ieşirea stratului ascuns anterior. Putem considera toată această modelare ca pe o probabilitate condiţionată $P(x_n|x_1, x_2, ..., x_{n-1})$, unde în cazul unei fraze $x_1, x_2, ..., x_n$ reprezintă cuvintele acesteia.

[1]Valorile ponderilor sunt deobicei subunitare iar iniţializarea este făcută urmând diverse principii matematice bine definite. Pentru cazuri simpliste, iniţializarea poate fi totuşi facută folosind o distribuţie uniformă.

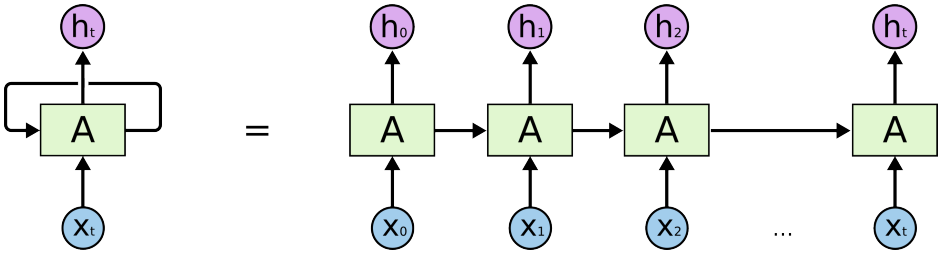

Figura 3.2: Exemplu de reţea neurală artificială recurentă

Algoritmul de reglare a ponderilor se numeşte propagarea înapoi în timp a erorii. O problemă serioasă care a apărut odată cu introducerea acestor reţele se numeşte anularea gradienţilor[2]. În cazul în care secvenţa de intrare are o lungime mare, informaţia nu reuşeşte să se propage în timp deoarece gradientul ponderilor tinde să devină 0. Mecanismul care înlătură această problemă se numeşte Long Short Term Memory (LSTM). Principiul este ca prin folosirea unor porţi de transmitere a informaţiei, gradientul să fie stabilizat. Acest mecanism a reprezentat un avans spectaculos pentru Deep Learning, permiţând modelarea secvenţială cu reţinere a informaţiei pe o perioadă îndelungată de timp. Ecuaţiile 3.1 descriu calculele pentru porţile LSTM.

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c) \\ h_t &= o_t \circ \tanh(c_t) \end{aligned} \tag{3.1}$$

Poarta f_t se numeşte poartă de uitare (forget gate). Rolul ei este să stabilească câtă informaţie se va uita din trecut. Poarta i_t este poarta de intrare (input gate). Aceasta delimitează care este cantitatea de informaţie care se păstrează din intrarea curentă. Poarta o_t este cea de ieşire (output gate). Ea controlează ce informaţie va fi transmisă către ieşire. c_t se numeşte starea internă a celulei LSTM. Aceasta este calculată ca o combinaţie între starea anterioară a celulei, poarta de intrare şi cea de ieşire. h_t reprezintă ieşirea curentă a reţelei şi depinde de poarta de ieşire şi starea celulei LSTM.

[2]vanishing gradients

O RNAN aduce mai aproape ideea de modelare lingvistica, însă se poate observa că aceasta nu permite ca intrare decât o frază pe rând. Modelarea dorită în această lucrare este o pereche de forma întrebare-răspuns.

3.3 Modele Sequence-to-Sequence

Translaţia a reprezentat mereu un punct de interes în mediul procesării naturale de limbaj, constituind de altfel o provocare uriaşă de-a lungul ultimilor ani. Până în anul 2014, majoritatea modelelor de translaţie se bazau pe lanţuri Markov ascunse (Hidden Markov Models - HMM), totul urmând a se schimba odată cu introducerea unei noi arhitecturi în acel an de către Cho et Al. Noul tip de arhitectură se numea Sequence-to-Sequence (seq2seq) şi urma să aducă imbunătăţiri spectaculoase atât pe partea de translaţie cât şi pe partea conversaţională. Modelul este vizibil în Figura 3.3.

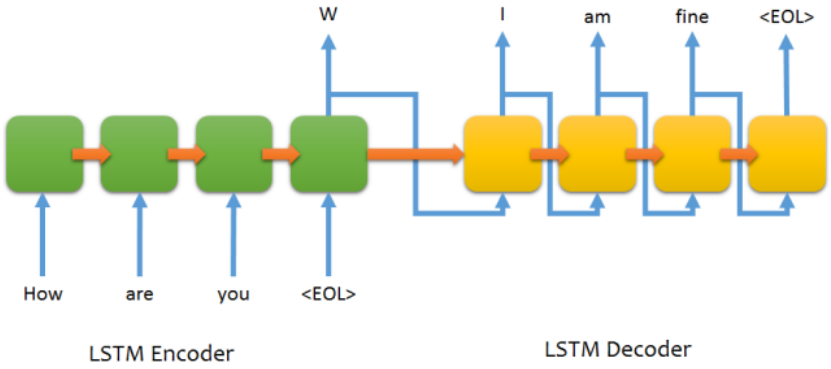

Figura 3.3: Modelul Sequence-to-Sequence

Aceast tip de arhitectură este împarţită în două jumătăţi: codor (encoder) şi decodor (decoder). Sarcina codorului este să primească o frază cu un număr variabil de cuvinte ca intrare iar unica sa ieşire [3] să fie o captură a întregii intrari (un vector de lungime fixă). Această captură poate fi privită ca o sumarizare a întregii fraze. Sarcina decodorului este de a învăţa fraze pornind de la ieşirea codorului, care va deveni prima intrare din decodor, şi restul intrarilor precedente. Intrarea curentă w_n în decodor este ieşirea de la timpul anterior, w_{n-1}. Modelarea se transformă astfel într-o probabilitate condiţionată: $P(w_n|w_1, w_2, ..., w_{n-1}, c)$, unde c reprezintă ieşirea codorului, deseori întâlnit sub numele de context în literatura de specialitate.

[3] Ieşirea ultimului element din secvenţă

Intrările în acest model sunt de forma întrebare-răspuns. Original folosit ca model de translaţie, unde intrarea pentru codor era fraza într-o limbă iar intrarea în decodor era fraza tradusă în limba dorită, acest principiu se poate aplica la fel de uşor pentru a modela o conversaţie. Spre deosebire de traducere unde totul se întâmpla punctual iar răspunsul nu depinde decât de intrarea curentă, o conversaţie este foarte dependentă de un context. Fără acest context, partenerul angrenat în discuţie ar răspunde mereu luând în considerare doar ce i s-a spus în momentul de faţă, ignorând orice replică anterioară. Acesta nu este un comportament dorit în cadrul unei conversaţii şi de aceea modelul seq2seq nu este destul de puternic de sine stătător pentru a modela o discuţie.

3.4 Hierarchical Recurrent Encoder-Decoder

Modelarea contextului discuţiei reprezintă una dintre principalele nevoi în ceea ce priveşte o conversaţie care doreşte să pară cât mai reală. Până recent, cea mai apropiată arhitectură care realiza acest lucru era modelul seq2seq, însă contextul era reţinut doar la nivelul unei singure fraze. În anul 2016, Iulian Şerban et Al. a introdus reţeaua Hierarchical Recurrent Encoder-Decoder (HRED - Figura 3.4). Ea poate fi vazută ca o extensie peste seq2seq. Avantajul acesteia este că poate reţine contextul discuţiei.

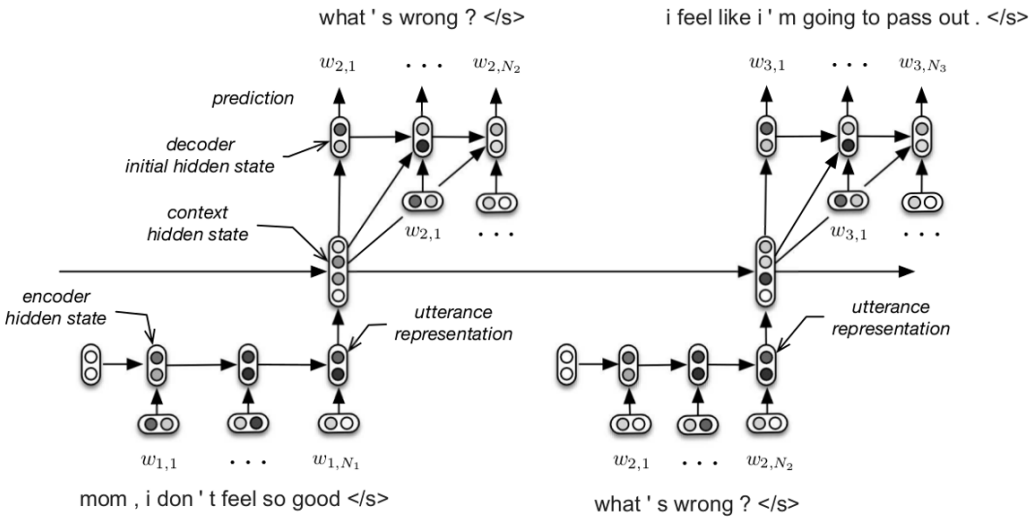

Figura 3.4: Hierarchical Recurrent Encoder-Decoder

După cum se poate observa, modelul este construit prin alăturarea mai multor reţele seq2seq legate între ele printr-o RNAR contextuală. După cum

sugerează şi numele, rolul acestei RNAR este de a reţine contextul discuţiei de-a lungul mai multor fraze. Spre deosebire de seq2seq, ieşirea codorului nu mai este oferită direct ca şi primă intrare pentru decodor, aceasta trecând mai întai prin RNAR contextuală. Pentru a fi mai uşor de înteles ce se întamplă, ne putem imagina toate cele n codoare şi decodoare ca fiind simple unităţi de intrare, respectiv ieşire într-o RNAR obişnuită. Precum sunt într-o RNAR ieşirile dependente atât de intrarea curentă, cât şi de cele precedente, prin analogie putem deduce că într-o reţea HRED, fiecare ieşire depinde de fraza curentă şi de contextul discuţiei (frazele precedente).

3.4.1 Celula GRU

Gated Recurrent Unit sau GRU este un tip de celulă pentru o RNAR, menită să o înlocuiască pe cea LSTM. Ecuaţiile din spatele celulei(3.2) sunt asemănătoare cu cele LSTM.

$$\begin{aligned} z &= \sigma(x_t U^z + s_{t-1} W^z) \\ r &= \sigma(x_t U^r + s_{t-1} W^r) \\ h &= \tanh(x_t U^h + (s_{t-1} \circ r) W^h) \\ s_t &= (1 - z) \circ h + z \circ s_{t-1} \end{aligned} \tag{3.2}$$

GRU are doar două porţi: poarta r de resetare (reset gate) şi poarta z de actualizare (update gate). Intuitiv, poarta de resetare determină cum se va combina noua intrare cu memoria precedentă iar cea de actualizare defineşte cât de multă memorie anterioară se păstrează. Principalele diferenţe între GRU şi LSTM sunt:

- GRU are doar două porţi, pe când LSTM trei.
- GRU nu posedă o memorie internă c_t care să fie diferită de starea ascunsă expusă. Nu conţine poarta de ieşire care este prezentă pentru LSTM.
- Poarta de intrare şi cea de ieşire sunt cuplate de poarta de actualizare, iar poarta de resetare este aplicată direct stării ascunse anterioare. Astfel, responsabilitatea porţii de resetare din LSTM este împărţită între cea de resetare şi cea de actualizare de la GRU.
- Nu se aplică o a doua neliaritate atunci când se calculează ieşirea.

Nu există o arhitectură câştigătoare clară între LSTM şi GRU. Având mai puţini parametri (U şi W), GRU se antrenează mai rapid şi are nevoie

de mai puţine date pentru a generaliza. Pe de altă parte, o cantitate mare de date exprimă o putere mai mare de modelare din partea LSTM şi astfel se pot obţine rezultate mai bune. Pentru realizarea acestei aplicaţii a fost aleasă o arhitectură de tip GRU.

3.4.2 HRED bidirecţional

Rolul codorului, precum a fost menţionat mai devreme, este să captureze informaţia unei fraze într-un vector de lungime fixă. În orice limbaj, sensul unui cuvânt nu este stabilit doar de cuvintele precedente ci şi de cele ce vor urma. Deoarece RNAR modelează cuvintele dintr-o frază pornind de la cuvântul w_1 la w_n, înţelesul din viitor al acestora este ignorat, iar captura frazei poate să nu fie îndeajuns de reprezentativă. Astfel, este propusă folosirea unei RNAR bidirecţionale (Figura 3.5) pentru codor. Acest tip de reţea foloseşte două parcurgeri ale secvenţei de intrare: cea înainte, care se desfăşoară în mod obişnuit şi cea inversă (de la w_n la w_1), pentru capturarea înţelesului din viitor al cuvintelor. Ieşirea pentru pargurgerea înainte va fi la timpul n iar cea pentru parcurgerea inversă va fi la timpul iniţial. Vectorul de lungime fixă va fi format din concatenarea celor două ieşiri ale reţelei bidirecţionale.

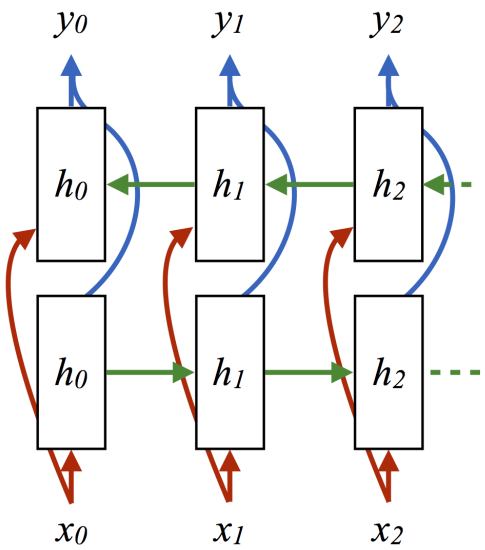

Figura 3.5: RNAR bidirecţională

3.5 Word Embeddings

Buna reprezentare a caracteristicilor unei intrări conduce la rezultate comparativ mai bune faţă de situaţiile unde acestea sunt înfăptuite manual. Când vine vorba despre procesarea naturală de limbaj, buna reprezentare a cuvintelor din vocabular (word embeddings) înseamnă un factor decisiv pentru succesul final. O reprezentare evidentă ar fi folosirea codificării de tipul one-hot care presupune un vector de lungimea vocabularului, umplut cu elemente 0 iar poziţia pe care se află cuvântul va fi marcată cu 1. Dacă lungimea vocabularului este mare, vectorul va deveni rar (sparse) şi avem de a face cu noţiunea de blestemul dimensionalităţii[4]. Astfel, este necesară o codificare compactă, mai inteligentă, care să fie de asemenea semnificativă pentru semantica cuvintelor.

3.5.1 Word2vec

În anul 2013, Mikolov et al. a propus un model pentru word embeddings care se numeşte word2vec. În prezent, atunci când vine vorba despre modelare de limbaj natural, word2vec este soluţia cea mai abordată în practică. Aceasta propune folosirea unor vectori de lungime fixă, compactă care totodată să captureze sensul cuvintelor. Ideea de bază este ca în spaţiul n-dimensional folosit, cuvintele cu semantică asemănatoare să aibă un scor de similaritate cât mai mare. Ca şi metrică de similaritate preferată pentru cuantificarea asemănării între reprezentarea a două cuvinte, câstigătoarea este de cele mai multe ori similaritatea cosinus (ecuaţia 3.3). Cel mai popular exemplu produs pentru a descrie puterea de modelare folosită de word2vec este $king - man + woman \approx queen$. Figura 3.6 capturează ideea de word2vec vizualizat într-un spaţiu bidimensional.

$$s(w_1, w_2) = \frac{\sum_{i=1}^{n} w_{1i} w_{2i}}{\sqrt{\sum_{i=1}^{n} w_{1i}^2} \sqrt{\sum_{i=1}^{n} w_{2i}^2}} \tag{3.3}$$

[4]Dimensionalitatea vectorului de intrare este extrem de mare iar arhitectura reţelei nu este capabilă să modeleze distribuţia de probabilităţi a datelor

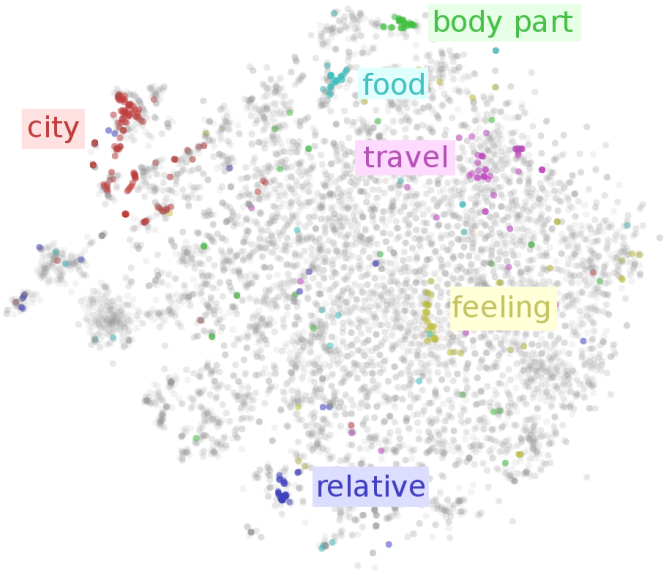

Figura 3.6: Word2vec vizualizat în spaţiu 2D

3.5.2 Word embeddings preantrenate

În machine learning, de cele mai multe ori există arhitecturi deja antrenate pentru diverse sarcini precum clasificare de imagini, recunoaştere vocală etc. La fel este cazul şi pentru word embeddings. Google a produs o antrenare pe un set de date care constă din miliarde de articole de ştiri, rezultând în urma acesteia word embeddings pentru aproximativ trei miliarde de cuvinte. Dacă se doreşte antrenarea unei reţele care să înveţe word embeddings pornind de la zero este posibil, însă sarcina este destul de dificilă iar rezultatul produs este dependent de natura cuvintelor din setul de date disponibil. Modelarea ideală ar fi ca aceste reprezentări ale cuvintelor să exprime cât mai bine realitatea înconjurătoare şi nu un subset al acesteia. Din acest motiv, este preferată folosirea modelului preantrenat de word embeddings pusă la dispoziţie de Google.

3.6 Predicţie

Având modelul antrenat, acesta trebuie folosit pentru predicţie, adică generarea unor răspunsuri adecvate pentru contextul discuţiei. Generarea cuvintelor este condiţionată atât de context, cât şi de cuvintele anterior generate. Spre deosebire de antrenare, în momentul predicţiei, secveţa de ieşire este goală iniţial (un vector de zerouri). Cuvintele sunt generate secvenţial,

unul câte unul şi adăugate pe poziţia t la care a ajuns secvenţa. Deoarece stratul softmax al secvenţei de ieşire returnează un vector de probabilităţi peste toate cuvintele din vocabular, pentru a returna cuvântul dorit se alege cel cu probabilitatea cea mai mare. Pornind de la această idee, există două variante de generare: metoda greedy şi beam search.

3.6.1 Greedy

Precum în teoria clasică a algoritmicii, scopul metodelor greedy este să selecteze la fiecare pas optimul local. În cazul de faţă, optimul local reprezintă cuvântul cu probabilitatea de apariţie cea mai mare. Prin acestă metodă nu este garantat la final că fraza produsă este cea mai bună cu putinţă, deoarece o altă combinaţie de cuvinte poate produce oricând o frază cu un scor mai bun[5]. Astfel, singurul avantaj al acestei metode este viteza de predicţie rapidă. În practică, se preferă folosirea unor algoritmi mai complexi, capabili să aleagă fraza cea mai potrivită dintr-un set de fraze canditat.

3.6.2 Beam search

Algoritmul beam search foloseşte ideea de arbore de căutare pentru a genera fraze (Figura 3.6). În locul alegerii cuvântului cu probabilitatea cea mai mare la fiecare pas, se vor alege top k cuvinte cu cele mai mari probabilităţi. În literatură, k se mai numeşte şi beam size. Fiecare nod (cuvânt) ales va produce un număr n_c de copii. În felul acesta este garantată generarea mai multor fraze canditat din care se va alege. Fiecare frază are un scor $S(f)$, care este de forma ecuaţiei 3.3, unde $P(w_i)$ reprezintă probabilitatea cuvântului ales. O frază devine canditat atunci când cuvântul curent generat este simbolul de sfârşit de propoziţie. În momentul în care algoritmul a terminat de generat toate frazele canditat, cea mai bună este determinată ca fiind cea cu scorul $S(f)$ cel mai mare.

$$S(f) = \frac{1}{n} \sum_{i=0}^{n} \log P(w_i) \tag{3.4}$$

Pentru ca acest arbore de căutare să nu capete o creştere exponenţială, la fiecare pas se păstrează un numar egal cu beam size cele mai bune fraze

[5]De regulă se utilizează media aritmetică a sumei din logaritmul probilităţilor cuvintelor frazei

iar restul sunt decartate. Acest algoritm nu asigură de departe optimul global, însă spaţiul căutării oferit de folosirea unui arbore este mult mai bine dezvoltat şi ales decât la metoda greedy. Cu cât beam size si numărul de copii generaţi sunt mai mari, cu atât algoritmul oferă soluţii cât mai apropiate de optimul global. De asemenea, odată cu creşterea acestor parametri, creşte şi timpul de căuture şi generare în arbore. De aceea, trebuie să existe un compromis între calitatea predicţiei şi viteza de execuţie a algoritmului.

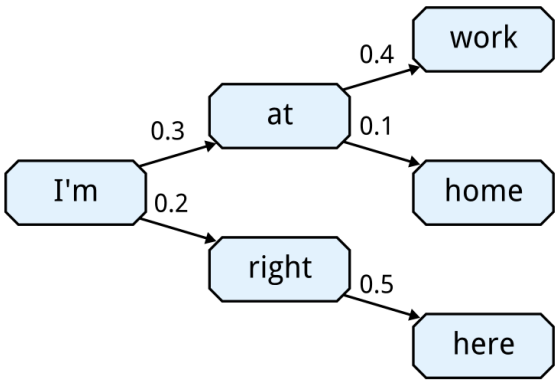

Figura 3.7: Arbore de căutare pentru algoritmul beam search

Capitolul 4

Experimente şi rezultate

Acest capitol prezintă configuraţiile arhitecturale încercate, cele care au avut succes precum şi rezultatele finale obţinute de către sistem. Pentru validarea modelului obţinut s-a realizat o scalare progresivă a numărului de date de intrare, acesta fiind supus la două teste principale. Unul dintre teste a fost ca pentru puţine exemple de intrare, modelul să fie capabil să înveţe mot a mot să producă răspunsuri precum în setul de antrenare (overfitting). Un alt test a fost ca atunci când numărul de intrări este crescut considerabil, sistemul să nu producă întotdeauna răsupunsuri monotone. Metrica de evaluare principală a sistemului a fost cea umană, ca şi în cazul testului Turing.

4.1 Set de date

Setul de date folosit pentru experimente se numeşte Movie Triples şi, după cum indică numele, reprezintă un set de date care foloseşte subtitrări din filme. Deoarece chatbot-ul descris în această lucrare îşi propune să fie unul open-topic, tipul conversaţiilor din filme alcătuiesc setul de date ideal pentru acest obiectiv. Desigur, setul de date nu este din categoria celor foarte mari, conţinând aproximativ 200.000 de intrări, dar suficiente pentru a modela un limbaj de o anvergură mai mică. Vocabularul setului de date conţine 10.000 de cuvinte; cuvintele din afara acestui vocabular sunt marcate cu simbolul unk (unknown). Intrările sunt de forma triplete de fraze (F_1, F_2, F_3), capturând astfel contextul discuţiei pe o întindere de trei replici. În cazul în care ar exista o pereche de n fraze pentru o intrare, HRED este capabil să captureze contextul de-a lungul acestora.

4.2 Mediu de lucru/Framework

Pentru dezvoltarea modelului s-a folosit framework-ul Keras cu backend de Tensorflow în limbajul de programare Python. Keras este un framework pentru Deep Learning, preferat de majoritatea comunităţii ştiinţifice de Deep Learning, care a fost gândit şi construit pentru uşurarea muncii de cercetare şi prototipizarea rapidă a modelelor. Astfel, efortul de concentrare este orientat spre descoperirea arhitecturii potrivite pentru modelarea problemei necesare şi foarte puţin pe latura de software development. De asemenea, framework-ul este dezvoltat astfel încât să profite de capacitatea de procesare GPU pe care sistemul hardware o pune la dispoziţie, procesul de instruire fiind accelerat de zeci de ori.

4.3 Arhitecturi canditat

4.3.1 Arhitectura 1.0

Arhitectura iniţială presupunea folosirea unei dimensiuni maxime pentru o frază de 75 de cuvinte. Deoarece frazele au dimensiuni variabile de lungime, iar o RNAR acceptă dimensiuni fixe, pentru a completa fraza până la lungimea maximă aleasă, se umpleau spaţiile lipsă cu vectori 0, procedeu denumit şi padding în literatură. Fiecare frază se sfârşeşte cu un simbol special numit end token iar apoi urmează padding-ul. Fiecare strat de ieşire RNAR din HRED (codor, context şi decodor) avea o dimensiune de 256 de neuroni. De remarcat este faptul că fiecare unitate structurală din HRED, reţelele recurente folosite pentru codor şi decodor, foloseau ponderi shared[1]. Stratul de embedding pentru cuvinte pornea cu învăţarea de la 0, nefolosindu-se word embeddings provenite de la word2vec. Ieşirea din RNAR contextuală era concatenată cu reprezentarea fiecărui cuvânt din decodor. Ieşirea era mai apoi procesată de un strat ascuns cu activare tanh pentru expandarea dimensiunii şi rezultatul era concatenat cu restul secvenţei de intrare pentru decodor. Algoritmul pentru învăţarea ponderilor folosit a fost rmsprop. Pentru prediţie s-a folosită varianta greedy descrisă în Capitolul 3. Această versiune performa bine atunci când erau date puţine, însă odată cu creşterea volumului, modelul nu reuşea să scaleze.

[1]Fiecare codor din HRED folosea acelaşi set de ponderi, precum şi în cazul decodorului

4.3.2 Arhitectura 1.5

Principala deosebire faţă de versiunea precedentă a fost folosirea word embeddings pentru reprezentarea cuvintelor şi ingheţarea stratului respectiv pentru a nu modifica ponderile preantrenate. S-a dovedit că această alegere a îmbunătăţit atât timpul de antrenare, deoarece numărul parametrilor a fost redus considerabil, cât şi performanţa modelului, acesta fiind capabil de a captura un număr mai mare de date, dar totuşi insuficient. Algoritmul pentru optimizarea ponderilor folosit a fost înlocuit cu adam folosind constanta de învăţare implică (0.001), care se presupune că obţine o convergenţă mai bună faţă de rmsprop. De asemenea s-a încercat folosirea simbolului special de început de frază denumit şi start token, însă s-a observat că ieşirea contextuală era întotdeauna înclinată către acest simbol în timpul predicţiei, sistemul nemaifiind capabil să modeleze fraza dorită ca răspuns.

4.3.3 Arhitectura 2.0

Deoarece modelul eşua în a scala atunci când numărul de date creştea, presupunerea a fost că acesta nu era îndeajuns de complex pentru a reprezenta distribuţia setului de antrenare. De aceea, s-a încercat folosirea unor straturi suplimentare alipite vertical[2] pentru codor şi decodor deoarece teoria spune că de la strat la strat, modelul capturează reprezentări tot mai complexe ale intrării. Procesul de instruire a devenit astfel foarte lent, deoarece complexitatea arhitecturii a crescut, însă rezultatul final era nesatisfăcător.

4.3.4 Arhitectura 3.0

Arhitectura 3.0 a adus cu sine un progres important spre atingerea obiectivului final. S-a renunţat astfel la straturi stacked deoarece nu rezolvau problema scalabilităţii. Noua configuraţie pentru fiecare ieşire RNAR era 300 de neuroni pentru codor, 300 de neuroni pentru context şi 300 pentru decodor. S-a renunţat de asemenea la concatenarea ieşirii contextuale cu fiecare reprezentare a cuvintelor din decodor şi trecerea acesteia prin stratul ascuns cu activare tanh, deoarece nu mai era nevoie de expandarea dimensiunii. Astfel, ieşirea contextuală era folosită acum doar ca primă intrare T_1 pentru secvenţa decodorului, presupunerea fiind că influenţa contextuală se propagă oricum prin RNAR a decodorului. Încă un considerent pentru această decizie a fost faptul că ieşirea contextuală şi reprezentarea word2vec a cuvintelor provin din distribuţii diferite iar împreunarea lor nu reprezenta un factor corect din punct de vedere matematic. Desigur, mecanismele de atenţie descrise în

[2]Stacked

Capitolul 5 posedă o putere de modelare sporită faţă de simpla utilizare a contextului ca primă intrare pentru decodor. Încă o observaţie realizată care încetinea procesul de antrenare era folosirea unei fraze cu o lungime maximă de 75 de cuvinte. În urma analizei amanunţite a setului de date, s-a observat că majoritatea frazelor aveau o lungime scurtă. Astfel, s-a calculat valoarea de 80% (80 percentile) pentru lungimea maximă a frazelor şi s-a determinat că aceasta este aproximativ 25. Sacrificiul realizat pentru a ignora cuvintele frazelor care depăşesc lungimea de 25 de cuvinte a produs o scădere considerabilă a timpului de instruire. Odată cu schimbarea către această configuraţie, s-a adoptat algoritmul beam search de predicţie. Deoarece spaţiul de căutare devenea mai larg, răspunsurile au devenit şi ele mai diversificate. Valorile pentru numărul de copii generaţi de fiecare nod şi beam size au fost alese 2, respectiv 5. În teorie, cu cât aceste numere sunt mai mari, spaţiul explorat creşte şi răspunsul devine mai calitativ. Însă creşterea considerabilă a acestor doi parametri aduce cu sine o viteză computaţională lentă, de aceea trebuie făcut un compromis la nivelul acestor valori. În pofida acestor îmbunătăţiri, sistemul încă nu scala în mod aşteptat odată cu creşterea datelor.

4.3.5 Arhitectura 3.5

Deoarece setul de date conţine subtitrări din filme, este un lucru obişnuit ca majoritatea răspunsurilor să fie monotone. Astfel, se creează o tendinţă pentru anumite cuvinte care apar foarte des, cele rare fiind aproape ignorate de către sistem. De aceea, sistemul eşuează în a furniza răspunsuri diversificate, care să conţină şi cuvinte rare. Acest neajuns se poate rezolva prin folosirea ponderilor pentru cuvinte în momentul când este calculată valoarea funcţiei de eroare. Ponderea pentru fiecare cuvânt va fi numărul de apariţii / numărul total de cuvinte. Prin această metodă se asigură că sistemul consideră influenţa egală a tuturor cuvintelor nemaiavând o predispoziţie către cele care apar des. Dimensiunea ieşirilor codor, context, decodor au fost alese ca fiind 500, 300, 500 pentru creşterea uşoară a complexităţii modelului. În urma antrenării se putea observa că modelul rămâne blocat în anumite regiuni de minim local. Acest aspect a fost înlăturat prin reducerea progresivă a constantei de învăţare de-a lungul instruirii, atunci când valoarea funcţiei de eroare nu scădea pe parcursul a cinci epoci consecutive cu mai mult de o valoare ϵ aleasă. Factorul de reducere a fost ales ca fiind 75%. Această combinaţie de ponderi împreună cu reducerea treptată a constantei de învăţare au reprezentat succesul care au produs convergenţa modelului curent şi totodată apariţia rezultatelor satisfăcătoare.

Capitolul 5

Posibile dezvoltări

5.1 Învăţare prin întărire

Învăţarea prin întărire sau Reinforcement learning (RL) reprezintă o arie machine learning inspirată de psihologia comportamentală, a cărei principale preocupări este cum poate un agent software să îndeplinească anumite acţiuni într-un mediu înconjurător astfel încât să maximizeze o răsplată cumulativă. Tipul acesta de învăţare diferă de metodele standard supervizate prin faptul că sistemului nu îi este prezentat niciodată perechi de forma intrare-ieşire, precum nici rezolvări ale unor acţiuni suboptimale. Focusul principal este asupra performanţei on-line care presupune găsirea balanţei ideale între explorare (dobândirea de cunoştinţe noi) şi exploatare (folosirea cunoştinţelor curente).

Modelul de bază RL presupune:

- Un set de stări pentru mediu şi agent S
- Un set de acţiuni A ale agentului
- O politică[1] de tranziţie de la stări la acţiuni
- Reguli care determină răsplata scalară imediată a tranziţiilor
- Reguli care descriu observaţiile agentului

[1] policy

Un agent RL interacţionează cu mediul în paşi de timpi discreţi. La fiecare pas t, agentul primeşte o obsersvaţie o_t, care deobicei include o răsplată r_t. Apoi alege o acţiune a_t din setul de acţiuni posibile, care este trimisă mai departe către mediu. Mediul realizează o tranziţie către starea nouă s_{t+1}, iar recompensa r_{t+1} asociată cu tranziţia (s_t, a_t, s_{t+1}) este determinată (Figura 5.1).

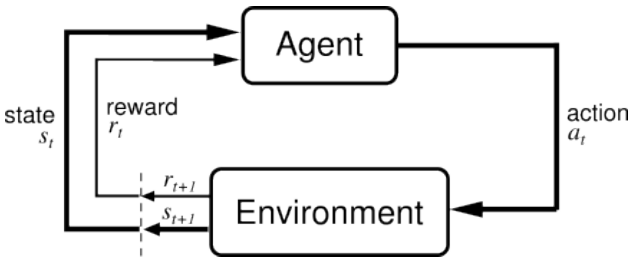

Figura 5.1: Tranziţia stărilor în RL

Pentru a acţiona optim, agentul trebuie să ia în considerare dependenţa pe termen lung a acţiunilor sale (să maximizeze recompensa viitoare), chiar dacă răsplata imediată este asociată cu o valoare negativă. Astfel, RL este potrivit pentru probleme unde există un compromis care include o recompensă între o dependenţă pe termen lung şi una pe termen scurt. A fost aplicat cu succes în diverse situaţii precum controlul automat de roboţi, telecomunicaţii, jocuri ATARI şi recent jocul GO.

Jiwei Li et al. propun în articolul lor, Deep Reinforcement Learning for Dialogue Generation, utilizarea metodelor de RL pentru învăţare de limbaj. Plecând de la aceeaşi arhitectură HRED, descrisă în capitolul 3, se poate ca prin folosirea unor reguli euristice de determinare a recompensei, pe baza răspunsurilor oferite de către chatbot, reţeaua să fie capabilă să ofere răspunsuri mai clare, contextuale şi corect gramaticale. Astfel, articolul propune trei metode de recompensă pentru cuantificarea calităţii unui răspuns:

- r_1, care recompensează modelul pentru răspunsuri care nu fac parte dintr-un subset de răspunsuri manual alese ca fiind monotone (I'm sorry, I don't know, Yes, No, etc.)
- r_2, care recompensează sistemul pentru nerepetarea aceluiaşi răspuns consecutiv de mai multe ori la rând
- r_3, care recompensează corectitudinea gramaticală şi coerenţa răspunsului

Răsplata finală va fi o medie ponderată între cele trei recompense:

$$\lambda_1 r_1 + \lambda_2 r_2 + \lambda_3 r_3$$

unde $\lambda_1 + \lambda_2 + \lambda_3 = 1$, iar $\lambda_1 = 0.25, \lambda_2 = 0.25, \lambda_3 = 0.5$

Folosind RL, autorii respectivi demonstrează că rezultatele pot fi considerabil îmbunătăţite. De aceea, extinderea modelului curent pentru a folosi RL reprezintă una dintre principalele posibilele dezvoltări dorite.

5.2 Mecanisme de atenţie

Mecanismele de atenţie pentru o RNA sunt puternic bazate pe cele vizuale existente la om. Atenţia vizuală umană, deşi este bine studiată şi există diverse modele, toate se rezumă la a fi capabile să se concentreze asupra unei anumite regiuni dintr-o imagine cu o rezoluţie sporită, pe când imaginea de fundal este percepută cu o rezoluţie mai scazută, punctul focal fiind ajustat de-a lungul timpului.

Pentru a observa utilitatea acestora, se pleacă de la arhitectura standard seq2seq ca exemplu. Precum a fost descris în capitolul 3, codorul are rolul de a captura înţelesul întregii fraze de intrare într-un vector de lungime fixă, care mai apoi va fi folosit pentru inferenţă. Pare nerezonabil ca întreaga informaţie să fie stocată într-un singur vector, în special dacă lungimea frazei este mare. LSTM se presupune că rezolvă cu succes problema dependenţei temporale a unei secvenţe, însă în practică aceste neajunsuri încă apar atunci când lungimea acesteia este prea mare. O rezolvare plauzibilă poate fi oferită prin folosirea RNAR bidirecţionale, care produc rezultate semnificativ mai bune.

Aici intervin mecanismele de atenţie. Acestea nu mai capturează întreaga informaţie a codorului într-un singur vector fix ci permit decodorului să observe diferite părţi ale frazei sursă la fiecare pas al generării. Ceea ce este important este că modelul învaţă singur ce informaţie trebuie selectată pe baza a ceea ce a fost produs până în acel moment şi al intrării (Figura 5.2).

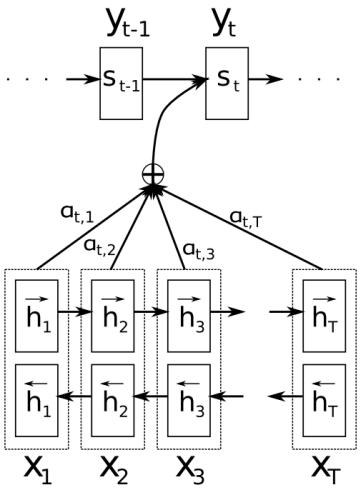

Figura 5.2: Mecanism de atenţie

În figura de mai sus, y reprezintă ieşirea generată de către decodor iar x cuvintele corespunzătoare frazei de intrare. Aspectul cel mai important este că fiecare ieşire y_t depinde acum de o combinaţie ponderată (valorile a) a tuturor stărilor din codor, şi nu doar de ultima. De exemplu, dacă $a_{3,2}$ este un număr mare, asta înseamnă că decodorul este foarte atent la cea de-a doua stare de intrare atunci când este produs al treilea cuvânt. Ponderile a sunt deobicei normalizate astfel încât suma lor să fie egală cu 1[2].

Costul ataşat acestor mecanisme este destul de mare. Pentru fiecare combinaţie intrare-ieşire trebuie calculată o valoare de atenţie şi stocată. Acest lucru este contraintuitiv, deoarece atenţia umană se presupune că economiseşte resurse computaţionale focalizând doar ce este important. Modelul prezentat analizează totul în detaliu înainte de a decide ce este important. Totuşi, acest lucru nu a împiedicat ca mecanismele de atenţie să devină atât de populare şi să producă rezultate bune pentru o multitudine de sarcini.

5.3 Setul de date

Unul dintre aspectele esenţiale de care puterea de procesare Deep Learning profită este cantitatea mare de date. Dacă modelele tradiţionale de Machine Learning nu reuşeau să exploateze creşterea volumului de date, modelele Deep Learning nu suferă de aceast neajuns (Figura 5.3). Acest fapt

[2]Distribuţie a stărilor de intrare

este explicabil prin simplul fapt că modelele tradiţionale nu erau suficient de complexe pentru a captura distribuţia datelor de intrare. Datorită complexităţii sporite a modelelor deep learning, care au avut parte de o explozie de popularitate odată cu creşterea capacităţii computaţionale, a memoriei pentru stocarea datelor şi folosirea procesorului grafic pentru a paraleliza calculele pe mii de nuclee, acestea pot surprinde cu uşurinţă distribuţia unui set de date cu multe şabloane. Având acest beneficiu la dispoziţie, modelul actual poate fi îmbunătăţit dacă este antrenat luând în considerare o cantitate mai mare de triplete de fraze.

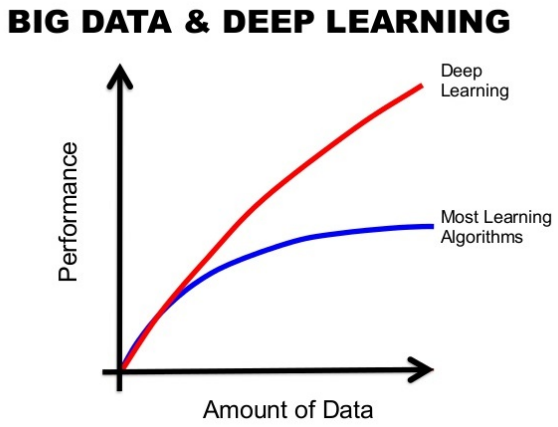

Figura 5.3: Scalabilitatea folosind Machine Learning vs Deep Learning

Desigur, o cantitate mai mare de date nu înseamnă întotdeauna că rezultatul final este mai bun. În urma experimentelor realizate, se poate observa o tendinţă a chatbot-ului de a fi înclinat către răspunsuri care se regăsesc în setul de antrenare de un numar mare de ori, precum "I'm sorry", "I don't know" etc. Acest fenomen este explicat de faptul că statistic vorbind aceste răspunsuri se apropie foarte mult de media distribuţiei datelor de intrare. Acest lucru ridică problema diversităţii cât şi a calităţii datelor care sunt furnizate sistemului. O soluţie plauzibilă este antrenarea folosind diverse seturi de date cunoscute ca fiind de referinţă pentru sarcini de procesare de limbaj natural, precum The Ubuntu Dialogue Corpus, şi compararea cu modelul curent.

Capitolul 6

Concluzii

TODO

Bibliografie

[1] Forbes. Advances in deep learning will lead to high-tech product breakthroughs. https://www.forbes.com/sites/quora/2016/10/25/advances-in-deep-learning-will-lead-to-high-tech-product-breakthroughs/#30a52b465195.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Graham Oppy and David Dowe. The turing test. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2016 edition, 2016.

[4] Gerardo Salandra. Does your business need a chatbot? https://rocketbots.io/blog/business-need-chatbot/.

[5] Tom Simonite. Customer service chatbots are about to become frighteningly realistic. https://www.technologyreview.com/s/603895/customer-service-chatbots-are-about-to-become-frighteningly-realistic/.

[6] Pavel Surmenok. Chatbot architecture. http://pavel.surmenok.com/2016/09/11/chatbot-architecture/.