



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

# LUCRARE DE DISERTAȚIE

Modele de Deep Learning pentru procesare de  
limbaj natural

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

Braşov  
2017

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>3</b>
1.1	Motivația alegerii temei . . . . .	5
1.2	Structura lucrării . . . . .	5
1.3	Recunoaștere . . . . .	5
<b>2</b>	<b>Lucrări similare</b>	<b>6</b>
2.1	Modele de regăsire . . . . .	6
2.1.1	Duolingo . . . . .	7
2.1.2	Facebook Messenger . . . . .	7
2.2	Modele generative . . . . .	8
<b>3</b>	<b>Arhitectura sistemului</b>	<b>9</b>
3.1	Rețele neurale artificiale . . . . .	9
3.2	Rețele neurale artificiale recurente . . . . .	12
3.3	Modele Sequence-to-Sequence . . . . .	14
3.4	Hierarchical Recurrent Encoder-Decoder . . . . .	15
3.4.1	Celula GRU . . . . .	16
3.4.2	HRED bidirecțional . . . . .	17
3.5	Word Embeddings . . . . .	18
3.5.1	Word2vec . . . . .	18
3.5.2	Word embeddings preantrenate . . . . .	19
3.6	Predicție . . . . .	20
3.6.1	Greedy . . . . .	20
3.6.2	Beam search . . . . .	20
<b>4</b>	<b>Experimente și rezultate</b>	<b>22</b>
4.1	Set de date . . . . .	22
4.2	Mediu de lucru/Framework . . . . .	23
4.3	Arhitecturi candidat . . . . .	23
4.3.1	Arhitectura 1.0 . . . . .	23
4.3.2	Arhitectura 1.5 . . . . .	24

4.3.3	Arhitectura 2.0 . . . . .	24
4.3.4	Arhitectura 3.0 . . . . .	24
4.3.5	Arhitectura 3.5 . . . . .	25
4.4	Rezultate . . . . .	26
4.4.1	Word perplexity . . . . .	26
4.4.2	Evaluare umană . . . . .	26
<b>5</b>	<b>Posibile dezvoltări</b>	<b>28</b>
5.1	Învățare prin întărire . . . . .	28
5.2	Mecanisme de atenție . . . . .	30
5.3	Setul de date . . . . .	31
<b>6</b>	<b>Concluzii</b>	<b>33</b>

# Capitolul 1

## Introducere

Procesarea de limbaj natural a reprezentat dintotdeauna una dintre marile provocări ale comunităților științifice de pretutindeni, scopul final fiind realizarea unui sistem suficient de inteligent încât să fie capabil să comunice precum omul. În acest sens, în anul 1950, matematicianul Alan Turing a propus un test care pune la încercare abilitatea unei mașini computaționale de a manifesta inteligență echivalentă cu cea umană [31]. Un evaluator judecă o conversație între un om și o mașină desemnată să genereze răspunsuri cât mai plauzibile, acesta fiind conștient că unul dintre participanții la discuție este o mașină. Conversația este limitată doar la text astfel încât rezultatul să nu depindă de abilitatea calculatorului de a genera sunete. Dacă evaluatorul nu poate diferenția mașina de om, se poate spune că aceasta a trecut testul. Până în prezent, acest test rămâne încă în picioare.

Recent, diverse companii de succes precum Google, Facebook, Microsoft, Apple și alții au realizat un demers semnificativ spre dezvoltarea unor sisteme inteligente de comunicație numite chatbots. La ora actuală, cei mai mulți sunt orientați către suport tehnic [40], scopul lor fiind tocmai de a imita sprijinul pe care o persoană reală îl poate oferi unui client. Un sistem computațional inteligent poate fi folosit pentru a răspunde multor cereri simultane, iar costurile de întreținere sunt mici practic, odată ce sistemul este dezvoltat, este necesară doar expunerea lui, de exemplu ca serviciu web. Cu cât tehnologia și munca de cercetare în această direcție progresează, cu atât sistemele de genul acesta devin din ce în ce mai inteligente și mai apropiate de ceea ce un om este capabil să ofere. Precum automatizarea industrială de producere a vehiculelor reprezintă un standard în era contemporană, asistenții conversaționali vor deveni un standard în anii ce vor urma [35].

Majoritatea modelelor actuale se bazează pe oferirea unor răspunsuri predefinite [41]. Aceste tipuri de sisteme pot returna doar răspunsuri existente, nefiind capabile de a genera text nou. Pe de altă parte, avem sistemele generative bazate pe inteligență artificială [41] care, precum omul, pot emite răspunsuri bazate pe experiențe anterioare. Vorbim deci despre evoluția unor modele de chatbots bazate pe pattern matching către unele bazate pe modele generative. În ultimii ani, ramura inteligenței artificiale numită Machine Learning (învățare automată) a luat amploare în această direcție prin curentul numit Deep Learning [16]. Acest curent a produs o mulțime de rezultate spectaculoase atât în direcția procesării naturale de limbaj cât și a procesării de imagini [14].

Rolul principal al unui chatbot îl reprezintă capabilitatea de a “înțelege” informația receptată de la o persoană pentru a produce un răspuns cât mai plauzibil. Însă cum procesează un calculator o limbă? Pentru a răspunde la această întrebare, se va face o analogie la cum învață un om o limbă. Pornește de la anumite cuvinte de bază iar apoi pe baza acestora, învață cuvinte tot mai complexe. Începe să creeze fraze prin care leagă aceste cuvinte precum și gramatica specifică limbajului. Practic, totul se bazează pe o anumită experiență cumulativă. Conversația devine astfel o modalitate ce facilitează și impulsionează deprinderea limbajului: omul este pus în fața unor contexte de utilizare, fenomen ce întărește deprinderea de utilizare a cuvintelor sau expresiilor individuale. Fiecare cuvânt nou învățat reprezintă o experiență către învățarea în continuare a limbajului. Încă un aspect reprezentativ uman este capabilitatea de a întreține o conversație pe termen lung cu un alt participant în cadrul unui context, lucru dificil de capturat pentru un sistem artificial.

Abordările de chatbot actuale își propun imitarea acestei modalități de învățare pentru un sistem computațional. Modelarea curentă cea mai eficientă pentru o astfel de problemă este oferită de către Deep Learning. Folosind arhitecturi de rețele neurale artificiale mult mai complexe decât cele shallow de la începutul anilor 2000 și plecând de la seturi de instruire masive (în cazul de față, corpusuri de text cu cât mai multe fraze în limba dorită), se pot crea modele generative care pot produce continuarea unor propoziții, fraze etc [46]. Ideea din spatele abordării Deep Learning este că sistemul devine mai performant pe măsură ce volumul de date crește [8]. Pentru a procesa o asemenea cantitate de date de instruire este nevoie de putere computațională pe măsură. Asta presupune pe scurt, capacitate hardware. Milioanele de calcule efectuate pentru modelarea unui limbaj de către sis-

tem nu permit folosirea unui procesor, chiar multicore de ultimă generație, deoarece este considerat a duce la sugrumarea procesului de instruire [38].

În locul microprocesoarelor se preferă folosirea plăcilor grafice (GPU) [7]. Inițial, acestea au fost dezvoltate pentru rulare rapidă de jocuri, dar potențialul lor a fost rapid intuit și exploatat prin programare paralelă. Deoarece placa video conține mult mai multe nuclee de procesare (câteva mii, comparate cu cele 4-8 nuclee tradiționale dintr-un microprocesor actual), este preferată programarea și rularea modelelor computaționale pe GPU. În ultimii ani au fost dezvoltate o multitudine de biblioteci care facilitează unui cercetător dezvoltarea de aplicații de Machine Learning pe GPU: Tensorflow [1], Theano [44], Caffe [20], Keras [10] etc.

## 1.1 Motivația alegerii temei

Testul Turing reprezintă o provocare pe care mulți cercetători din acest domeniu o abordează constant cu noi idei tot mai inovative, având din ce în ce mai mult succes în ultimii ani [45]. Chiar și la nivelul unui model generativ de jucărie, acest subiect reprezintă unul dintre cele mai fierbinți topicuri la ora actuală pentru comunitatea Deep Learning [9]. Acest fapt, precum și cercetarea ideilor state of the art folosite în sistemele actuale reprezintă factorul motivațional principal în alegerea temei de cercetare.

## 1.2 Structura lucrării

Capitolul 2 prezintă sisteme deja existente similare cu ceea ce este prezentat în lucrare. În Capitolul 3 este descrisă arhitectura rețelei folosite pentru modelarea limbajului. Capitolul 4 elaborează experimentele realizate, arhitecturi încercate, atât cele reușite cât și cele nereușite și motivarea eșuării lor, precum și rezultatele produse de către sistem și evaluate folosind diverse metrice. Capitolul 5 propune viitoare posibile direcții de dezvoltare care pot îmbunătăți calitatea răspunsurilor produse. Capitolul 6 împachetează lucrarea sub forma unei concluzii despre această activitate de cercetare.

## 1.3 Recunoaștere

Mulțumiri Universității Transilvania din Brașov care a finanțat acest proiect prin achiziționarea de hardware necesar pentru realizarea acestuia.

# Capitolul 2

## Lucrări similare

Fiind un subiect atât de fierbinte care primește o mulțime de atenție din partea comunității științifice, existența a nenumărate abordări și sisteme lingvistice inteligente este un fapt natural. În prezent, există două mari abordări de a genera limbaj natural. În primul rând, cele mai populare sunt modelele de tip regăsire [41], unde răspunsul este stocat într-o sursă de date și returnat pe baza unor metode de pattern matching. Cea de-a doua abordare sunt modelele generative [41], care produc un răspuns dinamic folosind diverse metode din teoria probabilităților.

### 2.1 Modele de regăsire

Modelele de acest tip folosesc o sursă de date care conține numeroase răspunsuri predefinite. Răspunsul este ales folosind o metodă euristică pentru o potrivire cât mai bună cu puțință, luând în considerare intrarea și 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 [47]. Se poate deduce foarte ușor că aceste sisteme nu generează text nou. O problemă uriașă a acestor modele o reprezintă incapacitatea de a reacționa la cazuri nemaîntâlnite pentru care nu există un răspuns potrivit. Acestea au totuși avantajele lor. Datorită sursei de date cu răspunsuri create de oameni, aceste metode nu produc erori gramaticale. În prezent, această abordare este cea mai populară și sigură pentru problemele în care răspunsul este unul sensibil, într-un domeniu precum cel medical, de exemplu, unde răspunsul trebuie să provină de la un specialist.

### 2.1.1 Duolingo

Populara aplicație de învățare a limbilor străine Duolingo [13] (Figura 2.1) folosește o abordare interesantă cu privire la chatbots. Aceasta dorește să își ajute utilizatorii să practice o nouă limbă prin conversații cu chatbots. Având în vedere că o conversație este considerată a fi printre cele mai bune moduri de a învăța o limbă străină, utilizatorii Duolingo pot vorbi cu bot-ul oricât de mult își doresc, iar acesta îi va corecta și le va propune răspunsuri potrivite. Mai mult de atât, poate estima progresul utilizatorului pentru a-și crește nivelul de dificultate, păstrând astfel constantă provocarea.

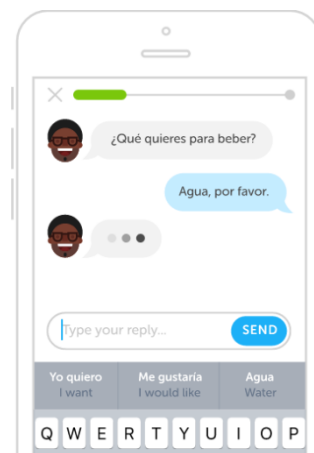


Figura 2.1: Conversație cu chatbot folosind Duolingo

### 2.1.2 Facebook Messenger

Facebook s-a alăturat întru totul afacerii conversaționale, astfel încât și-a transformat aplicația Messenger într-un business de mesagerie [12]. Compania a integrat plățile peer-to-peer în Messenger în anul 2015, urmând să lanseze un API pentru chatbots, astfel încât business-urile să poată crea interacțiuni pentru clienți. Se pot plasa comenzi de flori sau Uber, navigare printre ultimele trend-uri în materie de modă, toate dinăuntrul chat-ului de Messenger (Figura 2.2).



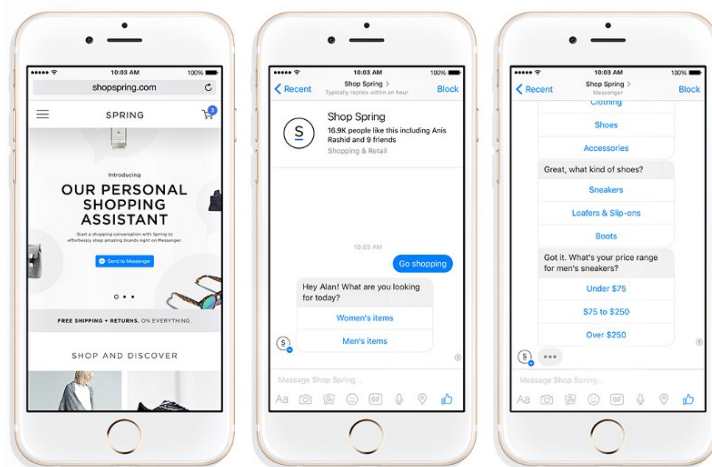


Figura 2.2: Cumpărături de haine folosind chatbot-ul companiei Spring pe Facebook Messenger

## 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 de obicei tehnici din Machine Translation, dar în loc de a traduce dintr-o limbă într-alta, se va realiza o “traduce” de la o intrare la o ieșire (răspuns). Acestea oferă o mai bună impresie de comunicare cu o entitate umană. Totuși, ele sunt dificil de antrenat, predispuse la erori gramaticale (în special unde lungimea propoziției este mai mare) și necesită o cantitate mare de date de instruire [46].

Prezentul este încă sub semnul întrebării pentru acest tip de model, însă în următorii ani, acestea vor căpăta tot mai multă atenție și popularitate devenind în consecință tot mai performante. Dacă un chatbot va doborî testul Turing, șansele 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 producție. În Capitolul 3 vor fi prezentate diferite arhitecturi de rețele neurale, capabile să modeleze conversații.

## Capitolul 3

# Arhitectura sistemului

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

### 3.1 Rețele neurale artificiale

O rețea neurală artificială (RNA) reprezintă o paradigmă bazată pe procesare de informații a cărei inspirație provine din sistemul nervos biologic [2]. 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, învață din exemple. Învățarea în sistemele biologice implică ajustarea conexiunilor sinaptice care există între neuroni. Acest principiu se aplică și acestor rețele.

Ca și modelare matematică propriu-zisă, o RNA poate fi observată în Figura 3.1. Avem o intrare reprezentată în figură de vectorul  $n$ -dimensional  $(x_1, x_2, x_3, x_4)$ . Această intrare reprezintă caracteristicile unui eșantion 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 funcții cu activare neliniară. Ultimul strat se numește strat de ieșire, reprezentând eticheta (clasa) vectorului de intrare.

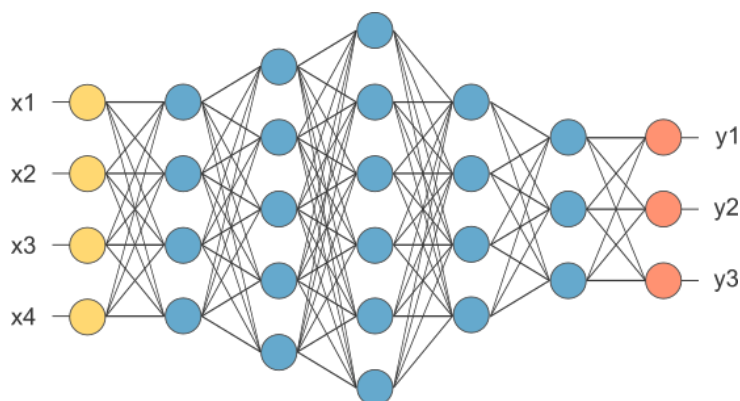


Figura 3.1: Exemplu de rețea neurală artificială

Valorile care creează legăturile dintre straturile rețelei se numesc ponderi. Rolul principal al acestora este să propage informația primită ca intrare, până la stratul de ieșire, astfel încât rețeaua să fie capabilă să modeleze cu succes distribuția setului de instruire. Inițial, valorile acestora sunt alese uniform aleator, mărginite de un anumit interval<sup>1</sup>. Partea actualmente de instruire constă în stabilizarea acestor ponderi până în momentul când rețeaua ajunge la convergență.

Un aspect principal care face posibilă modelarea distribuției datelor îl reprezintă funcțiile cu activare neliniară. Scopul acestora este ca de la strat la strat, informația propagată să creeze abstractizări care surprind caracteristici din ce în ce mai complexe ale intrării (Figura 3.2). După cum se poate observa, rețeaua din Figura 3.2 învață ca la primul strat să detecteze linii, la stratul secund regiuni ale feței, ca în final să fie capabilă să reprezinte chipuri.

---

<sup>1</sup>Inițializare Glorot [15]

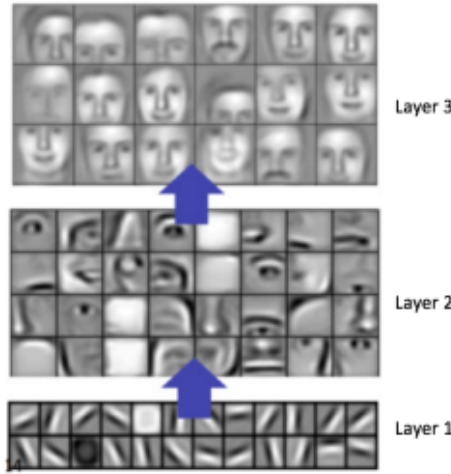


Figura 3.2: Caracteristici învățate pentru straturile unei RNA

Principalele funcții cu activare neliniară folosite în prezent sunt:

- Sigmoidă:  $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tangentă hiperbolică:  $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
- Rectified linear unit (ReLU):  $\text{relu}(x) = \max(0, x)$
- Softmax:  $\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}$

Aceste funcții sunt aplicate straturilor ascunse după ce s-a calculat produsul matriceal dintre ieșirea stratului anterior  $x$  și matricea ponderilor  $W$  pentru stratul curent, la care se adaugă un termen numit bias:  $f(W^t x + b)$ . Important de precizat este faptul că funcția softmax este aplicată întotdeauna doar stratului de ieșire. Scopul acesteia este normalizarea valorilor de ieșire astfel încât acestea să reprezinte un vector de probabilități asupra celor  $K$  posibile clase.

Odată ce informația a fost propagată înainte<sup>2</sup> până la ultimul strat, trebuie definită o funcție de eroare care măsoară diferența dintre distribuția învățată a datelor și cea reală. În acest scop, pentru probleme unde există un număr  $K$  de clase, varianta preferată pentru funcția de eroare este cross entropy. În teoria informației, cross entropy între două distribuții de probabilități  $p$  și  $q$  care stau la baza unui set de evenimente, măsoară numărul de biți necesari

---

<sup>2</sup>Feedforward

pentru a identifica un eveniment produs de către acest set, dacă procedeul de alegere este optimizat pentru distribuția “nenaturală”  $q$  în locul distribuției reale  $p$  [16]. Aceasta este definită ca fiind:

$$H(p, q) = - \sum_x p(x) \log q(x) \quad (3.1)$$

Plecând de la această funcție de eroare, se calculează derivatele parțiale ale fiecărui strat în raport cu aceasta, iar ponderile sunt deplasate cu o anumită magnitudine în sensul invers gradientului obținut, eroarea fiind propagată înapoi<sup>3</sup>. Această idee este cunoscută în literatură drept Gradient Descent [5]. Formula de actualizare a ponderilor este vizibilă în Ecuația 3.2, unde  $\eta$  reprezintă constanta de învățare<sup>4</sup>, iar  $\nabla_{\theta} J(\theta)$  gradientul derivatei parțiale în raport cu funcția de eroare cross entropy.

$$\theta = \theta - \eta \nabla_{\theta} J(\theta) \quad (3.2)$$

Astfel, instruirea unei RNA se rezumă la următoarele etape: propagarea informației de la stratul de intrare către cel de ieșire, calcularea valorii funcției de eroare, propagarea înapoi a erorii și actualizarea ponderilor. Acest principiu se aplică și arhitecturilor ulterior prezentate în acest capitol.

## 3.2 Rețele 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. Soluția pentru această problemă este oferită de rețelele neurale artificiale recurente (RNAR) [48]. Acestea sunt folosite pentru a modela secvențe unde există o dependență temporală (frazе, serii de timp). Fiecare intrare curentă din secvență este condiționată de cele precedente. O astfel de rețea se poate observa în Figura 3.3. 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 și de ieșirea stratului ascuns anterior. Această modelare poate fi

---

<sup>3</sup>Backpropagation

<sup>4</sup>Magnitudinea cu care ponderile se deplasează în direcția gradientului

considerată o probabilitate condiționată  $P(x_n|x_1, x_2, \dots, x_{n-1})$ , unde în cazul unei fraze,  $x_1, x_2, \dots, x_n$  reprezintă cuvintele acesteia.

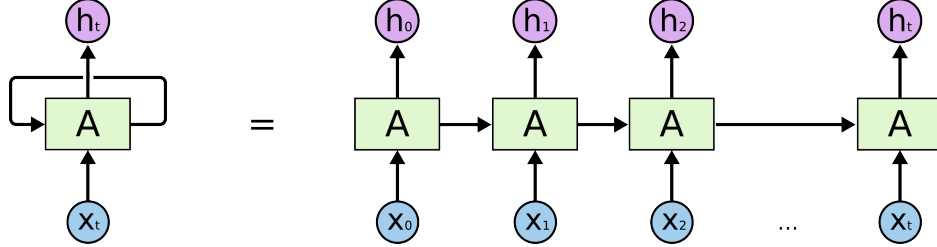


Figura 3.3: Exemplu de rețea neurală artificială recurentă

Propagarea înapoi a erorii depinde acum de mai multe secvențe temporale<sup>5</sup> [29]. O problemă serioasă care a apărut odată cu introducerea acestor rețele se numește anulara gradientilor<sup>6</sup>. În cazul în care secvența de intrare are o lungime mare, informația nu reușește să se propage în timp deoarece gradientul ponderilor tinde să devină 0 [18]. Mecanismul care înlătură această problemă se numește Long Short Term Memory (LSTM) [19]. Principiul este ca prin folosirea unor porți de transmitere a informației, gradientul să fie stabilizat. Această descoperire a reprezentat un avans spectaculos pentru Deep Learning, permițând modelarea secvențială cu reținere a informației pe o perioadă îndelungată de timp. Ecuațiile 3.3 descriu calculele pentru porțile LSTM.

$$\begin{aligned}
 f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
 i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
 o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
 c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
 h_t &= o_t \circ \tanh(c_t)
 \end{aligned} \tag{3.3}$$

Poarta  $f_t$  se numește poartă de uitare (forget gate). Rolul ei este să stabilizească câtă informație se va uita din trecut. Poarta  $i_t$  este poarta de intrare (input gate). Aceasta delimitează care este cantitatea de informație care se păstrează din intrarea curentă. Poarta  $o_t$  este cea de ieșire (output gate). Ea controlează ce informație va fi transmisă către ieșire.  $c_t$  se numește starea internă a celulei LSTM. Aceasta este calculată ca o combinație între starea

<sup>5</sup>Backpropagation through time

<sup>6</sup>Vanishing gradients

anterioară a celulei, poarta de intrare și cea de ieșire.  $h_t$  reprezintă ieșirea curentă a rețelei și depinde de poarta de ieșire și starea celulei LSTM.

O RNAN aduce mai aproape ideea de modelare lingvistică, 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

Traducerea a reprezentat mereu un punct de interes în mediul procesării naturale de limbaj, constituind de altfel o provocare uriașă de-a lungul ultimilor ani. Până în anul 2014, majoritatea modelelor de traducere se bazau pe lanțuri Markov ascunse (Hidden Markov Models - HMM) [32], 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) [42] și urma să aducă îmbunătățiri majore atât pe partea de traducere cât și pe cea conversațională. Modelul este vizibil în Figura 3.4.

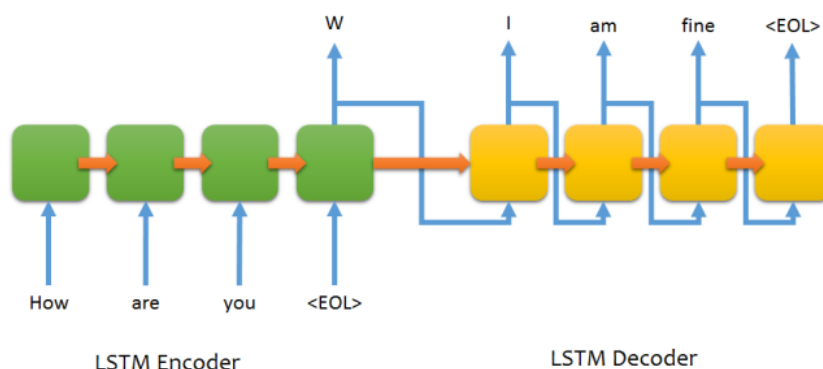


Figura 3.4: Modelul Sequence-to-Sequence

Aceast tip de arhitectură este împărțită în două jumătăți: codor (encoder) și decodor (decoder). Sarcina codorului este să primească o frază cu un număr variabil de cuvinte ca intrare iar unica sa ieșire<sup>7</sup> să fie o captură a întregii intrări (un vector de lungime fixă). Această captură poate fi privită ca o sumarizare a întregii fraze. Sarcina decodorului este de a învăța fraze pornind de la ieșirea codorului, care va deveni prima intrare din decodor, și restul intrărilor precedente. Intrarea curentă  $w_n$  în decodor este ieșirea de

<sup>7</sup>Ieșirea ultimului element din secvență

la timpul anterior,  $w_{n-1}$ . Modelarea se transformă astfel într-o probabilitate condiționată:  $P(w_n|w_1, w_2, \dots, w_{n-1}, c)$ , unde  $c$  reprezintă ieșirea codorului, deseori întâlnit sub numele de context în literatura de specialitate.

Intrările în acest model sunt de forma întrebare-răspuns. Original folosit ca model de translație, 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 ușor pentru a modela o conversație [33]. Spre deosebire de traducere unde totul se întâmplă punctual iar răspunsul nu depinde decât de intrarea curentă, o conversație este foarte dependentă de un context. Fără acest context, partenerul angrenat în discuție ar răspunde mereu luând în considerare doar ce i s-a spus în momentul de față, ignorând orice replică anterioară. Acesta nu este un comportament dorit în cadrul unei conversații și de aceea modelul seq2seq nu este destul de puternic de sine stătător pentru a modela o discuție.

### 3.4 Hierarchical Recurrent Encoder-Decoder

Modelarea contextului discuției reprezintă una dintre principalele nevoi în ceea ce privește o conversație care se dorește să pară cât mai reală. Până recent, cea mai apropiată arhitectură care realiza acest lucru era modelul seq2seq, însă contextul era reținut doar la nivelul unei singure fraze. În anul 2016, Iulian Șerban et Al. a introdus rețeaua Hierarchical Recurrent Encoder-Decoder (HRED - Figura 3.5) [37]. Ea poate fi văzută ca o extensie peste seq2seq. Avantajul acesteia este că poate reține contextul discuției.



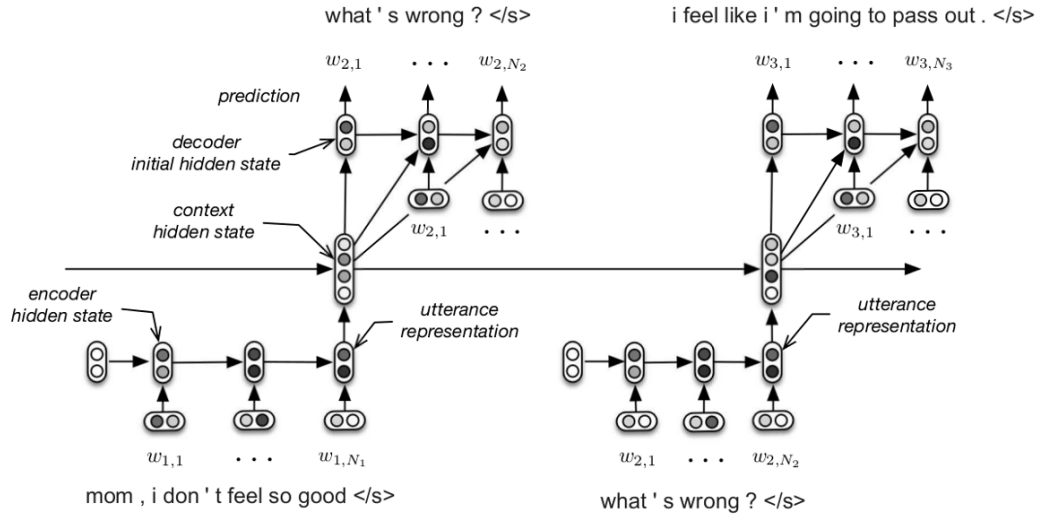


Figura 3.5: Hierarchical Recurrent Encoder-Decoder

După cum se poate observa, modelul este construit prin alăturarea mai multor rețele seq2seq legate între ele printr-o RNAR contextuală. După cum sugerează și numele, rolul acestei RNAR este de a reține contextul discuției de-a lungul mai multor fraze. Spre deosebire de seq2seq, ieșirea codorului nu mai este oferită direct ca și primă intrare pentru decodor, aceasta trecând mai întâi prin RNAR contextuală. Pentru a fi mai ușor de înțeles ce se întâmplă, ne putem imagina toate cele  $n$  codoare și decodoare ca fiind simple unități de intrare, respectiv ieșire într-o RNAR obișnuită. Precum sunt într-o RNAR ieșirile dependente atât de intrarea curentă, cât și de cele precedente, prin analogie putem deduce că într-o rețea HRED, fiecare ieșire depinde de fraza curentă și de contextul discuției (frazele precedente).

### 3.4.1 Celula GRU

Gated Recurrent Unit sau GRU [11] este un tip de celulă pentru o RNAR, menită să o înlocuiască pe cea LSTM. Ecuațiile 3.4 din spatele celulei sunt asemănătoare cu cele LSTM.

$$\begin{aligned}
 z &= \sigma(x_t U^z + s_{t-1} W^z) \\
 r &= \sigma(x_t U^r + s_{t-1} W^r) \\
 h &= \tanh(x_t U^h + (s_{t-1} \circ r) W^h) \\
 s_t &= (1 - z) \circ h + z \circ s_{t-1}
 \end{aligned} \tag{3.4}$$

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

- GRU are doar două porți, pe când LSTM trei.
- GRU nu posedă o memorie internă  $c_t$  care să fie diferită de starea ascunsă expusă. Nu conține poarta de ieșire care este prezentă pentru LSTM.
- Poarta de intrare și cea de ieșire sunt cuplate de poarta de actualizare, iar poarta de resetare este aplicată direct stării ascunse anterioare. Astfel, responsabilitatea porții de resetare din LSTM este împărțită între cea de resetare și cea de actualizare de la GRU.
- Nu se aplică o a doua neliniaritate atunci când se calculează ieșirea.

Nu există o arhitectură câștigătoare clară între LSTM și GRU. Având mai puțini parametri ( $U$  și  $W$ ), GRU se antrenează mai rapid și are nevoie de mai puține date pentru a generaliza. Pe de altă parte, o cantitate mare de date exprimă o putere mai mare de modelare din partea LSTM și astfel se pot obține rezultate mai bune. Pentru realizarea acestei aplicații a fost aleasă o arhitectură de tip GRU.

### 3.4.2 HRED bidirecțional

Rolul codorului, precum a fost menționat mai devreme, este să captureze informația unei fraze într-un vector de lungime fixă. În orice limbaj, sensul unui cuvânt nu este stabilit doar de cuvintele precedente ci și de cele ce vor urma. Deoarece RNAR modelează cuvintele dintr-o frază pornind de la cuvântul  $w_1$  la  $w_n$ , înțelesul din viitor al acestora este ignorat, iar captura frazei poate să nu fie îndeajuns de reprezentativă. Astfel, este propusă folosirea unei RNAR bidirecționale [36] (Figura 3.6) pentru codor. Acest tip de rețea folosește două parcurgeri ale secvenței de intrare: cea înainte, care se desfășoară în mod obișnuit și cea inversă (de la  $w_n$  la  $w_1$ ), pentru capturarea înțelesului din viitor al cuvintelor. Ieșirea pentru parcurgerea înainte va fi la timpul  $n$  iar cea pentru parcurgerea inversă va fi la timpul inițial. Vectorul de lungime fixă va fi format din concatenarea celor două ieșiri ale rețelei bidirecționale.

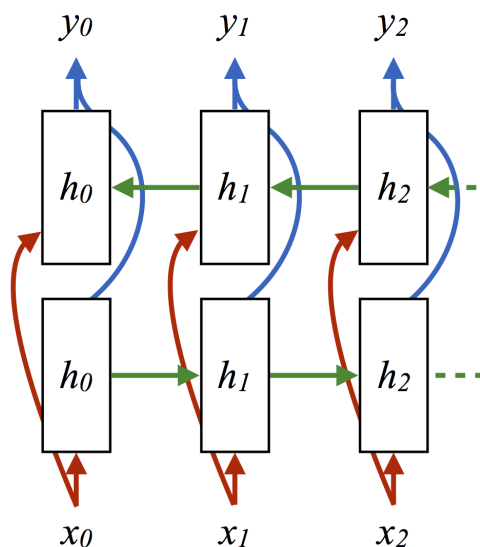


Figura 3.6: RNAR bidirecțională

## 3.5 Word Embeddings

Buna reprezentare a caracteristicilor unei intrări conduce la rezultate comparativ mai bune față de situațiile 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 poziția pe care se află cuvântul va fi marcată cu 1. Dacă lungimea vocabularului este mare, vectorul va deveni rar (sparse), instalându-se efectul denumit blestemul dimensionalității<sup>8</sup> [26]. 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 numește word2vec [28]. În prezent, atunci când vine vorba despre modelare de limbaj natural, word2vec este soluția cea mai abordată în prac-

<sup>8</sup>Dimensionalitatea vectorului de intrare este extrem de mare iar arhitectura rețelei nu este capabilă să modeleze distribuția de probabilități a datelor

tică. Aceasta propune folosirea unor vectori de lungime fixă, compactă care totodată să captureze sensul cuvintelor. Ideea de bază este ca în spațiul  $n$ -dimensional folosit, cuvintele cu semantică asemănătoare să aibă un scor de similaritate cât mai mare. Ca și metrică de similaritate preferată pentru cuantificarea asemănării între reprezentarea a două cuvinte, câștigătoare este de cele mai multe ori similaritatea cosinus (Ecuția 3.5). Cel mai popular exemplu produs pentru a descrie puterea de modelare folosită de word2vec este  $king - man + woman \approx queen$ . Figura 3.7 capturează ideea de word2vec vizualizat într-un spațiu bidimensional.

$$s(w_1, w_2) = \frac{\sum_{i=1}^n w_{1i} w_{2i}}{\sqrt{\sum_{i=1}^n w_{1i}^2} \sqrt{\sum_{i=1}^n w_{2i}^2}} \quad (3.5)$$

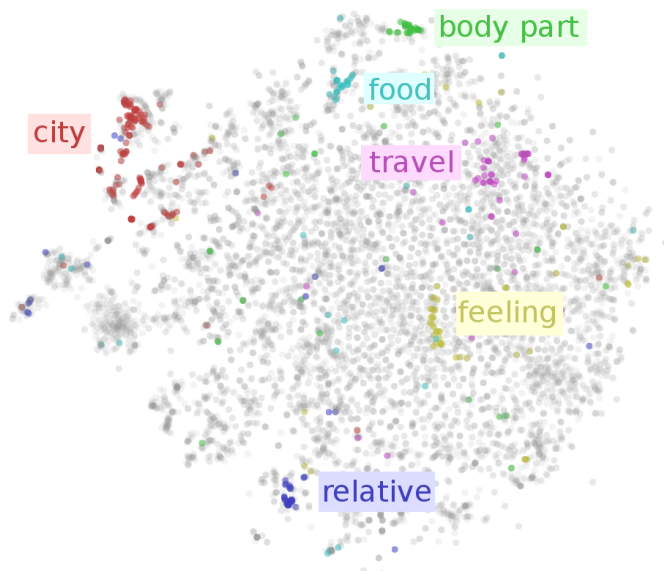


Figura 3.7: Word2vec vizualizat în spațiul 2D [34]

### 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, recunoaștere vocală etc. La fel este cazul și pentru word embeddings. Google a produs o antrenare pe un set de date care constă din miliarde de articole de știri, rezultând în urma acesteia word embeddings pentru aproximativ trei miliarde de cuvinte [27]. Dacă se dorește antrenarea unei rețele care să învețe 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 și nu un subset al acesteia. Din acest motiv, este preferată folosirea modelului preantrenat de word embeddings pus la dispoziție de Google.

## 3.6 Predicție

Având modelul antrenat, acesta trebuie folosit pentru predicție, adică generarea unor răspunsuri adecvate pentru contextul discuției. Generarea cuvintelor este condiționată atât de context, cât și de cuvintele anterior generate. Spre deosebire de antrenare, în momentul predicției, secvența de ieșire este inițial goală (un vector de zerouri). Cuvintele sunt generate secvențial, unul câte unul și adăugate pe poziția  $t$  la care a ajuns secvența. Deoarece stratul softmax al secvenței de ieșire returnează un vector de probabilități 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 și 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 față, optimul local reprezintă cuvântul cu probabilitatea de apariție cea mai mare. Prin această metodă nu este garantat la final că fraza produsă este cea mai bună cu putință, deoarece o altă combinație de cuvinte poate produce oricând o frază cu un scor mai bun<sup>9</sup>. Astfel, singurul avantaj al acestei metode este viteza de predicție rapidă. În practică, se preferă folosirea unor algoritmi mai complecși, capabili să aleagă fraza cea mai potrivită dintr-un set de fraze candidat.

### 3.6.2 Beam search

Algoritmul beam search [30] folosește ideea de arbore de căutare pentru a genera fraze (Figura 3.8). În locul alegerii cuvântului cu probabilitatea cea mai mare la fiecare pas, se vor alege top  $k$  cuvinte cu cele mai mari probabilități. În literatură,  $k$  se mai numește și beam size. Fiecare nod (cuvânt) ales va produce un număr  $n_c$  de copii. În felul acesta este garantată generarea mai multor fraze candidat din care se va alege. Fiecare frază are un scor

---

<sup>9</sup>De regulă se utilizează media aritmetică a sumei din logaritmul probabilităților cuvintelor frazei

$S(f)$ , care este de forma Ecuației 3.6, unde  $P(w_i)$  reprezintă probabilitatea cuvântului ales. O frază devine candidat atunci când cuvântul curent generat este simbolul de sfârșit de propoziție. În momentul în care algoritmul a terminat de generat toate frazele candidat, cea mai bună este determinată ca fiind cea cu scorul  $S(f)$  cel mai mare.

$$S(f) = \frac{1}{n} \sum_{i=0}^n \log P(w_i) \quad (3.6)$$

Pentru ca acest arbore de căutare să nu capete o creștere exponențială, la fiecare pas se păstrează un număr egal cu beam size cele mai bune fraze iar restul sunt decartate. Acest algoritm nu asigură de departe optimul global, însă spațiul căutării oferit de folosirea unui arbore este mult mai bine dezvoltat și ales decât la metoda greedy. Cu cât beam size și numărul de copii generați sunt mai mari, cu atât algoritmul oferă soluții cât mai apropiate de optimul global. De asemenea, odată cu creșterea acestor parametri, crește și timpul de căutare și generare în arbore. De aceea, trebuie să existe un compromis între calitatea predicției și viteza de execuție a algoritmului.

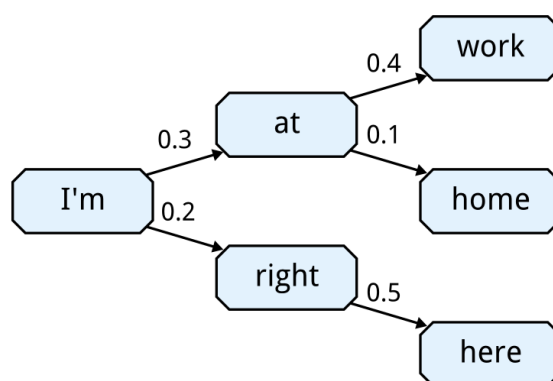


Figura 3.8: Arbore de căutare pentru algoritmul beam search

# Capitolul 4

## Experimente și rezultate

Acest capitol prezintă configurațiile arhitecturale încercate, cele care au avut succes precum și rezultatele finale obținute de către sistem. Pentru validarea modelului obținut 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 puține exemple de intrare, modelul să fie capabil să învețe 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 răspunsuri monotone. Metricile de evaluare principale ale sistemului au fost cea umană și word perplexity [6].

### 4.1 Set de date

Setul de date utilizat pentru experimente se numește Movie Triples [37], obținut prin expandarea și preprocesarea setului de date inițial numit Movie-DiC [4] și, după cum indică numele, reprezintă un set de date care folosește subtitrări din filme<sup>1</sup>. Deoarece chatbot-ul descris în această lucrare își propune să fie unul open-topic, tipul conversațiilor din filme alcătuiesc setul de date ideal pentru acest obiectiv. Desigur, setul de date nu este din categoria celor foarte mari, conținând aproximativ 200.000 de intrări, dar suficiente pentru a modela un limbaj de o anvergură mai mică. Vocabularul setului de date conține 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 discuției 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.

---

<sup>1</sup>Setul de date a fost obținut în urma cererii către autorul articolului original, Iulian Șerban

## 4.2 Mediu de lucru/Framework

Pentru dezvoltarea modelului s-a folosit framework-ul Keras [10] cu backend de Tensorflow [1] în limbajul de programare Python. Keras este un framework pentru Deep Learning, preferat de majoritatea comunității științifice de Deep Learning, care a fost gândit și construit pentru ușurarea muncii de cercetare și prototipizarea rapidă a modelelor. Astfel, efortul de concentrare este orientat spre descoperirea arhitecturii potrivite pentru modelarea problemei necesare și foarte puțin 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 dispoziție, procesul de instruire fiind accelerat de zeci de ori.

## 4.3 Arhitecturi candidat

### 4.3.1 Arhitectura 1.0

Arhitectura inițială 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 spațiile lipsă cu vectori 0, procedeu denumit și padding în literatură. Fiecare frază se sfârșește cu un simbol special numit end token iar apoi urmează padding-ul. Fiecare strat de ieșire RNAR din HRED (codor, context și decodor) avea o dimensiune de 256 de neuroni. De remarcat este faptul că fiecare unitate structurală din HRED, rețelele recurente folosite pentru codor și decodor, foloseau ponderi shared<sup>2</sup>. Stratul de embedding pentru cuvinte pornea cu învățarea de la 0, nefolosindu-se word embeddings provenite de la word2vec. Ieșirea din RNAR contextuală era concatenată cu reprezentarea fiecărui cuvânt din decodor. Ieșirea era mai apoi procesată de un strat ascuns cu activare tanh pentru expandarea dimensiunii și rezultatul era concatenat cu restul secvenței de intrare pentru decodor. Algoritmul pentru învățarea ponderilor folosit a fost RMSprop [17]. Pentru predicție s-a folosită varianta greedy descrisă în Capitolul 3. Această versiune performa bine atunci când erau date puține, însă odată cu creșterea volumului, modelul nu reușea să scaleze.

---

<sup>2</sup>Fiecare codor din HRED folosea același set de ponderi, precum și în cazul decodurului



### 4.3.2 Arhitectura 1.5

Principala deosebire față de versiunea precedentă a fost folosirea word embeddings pentru reprezentarea cuvintelor și înghețarea stratului respectiv pentru a nu modifica ponderile preantrenate. S-a dovedit că această alegere a îmbunătățit atât timpul de antrenare, deoarece numărul parametrilor a fost redus considerabil, cât și performanța modelului, acesta fiind capabil de a captura un număr mai mare de date, totuși insuficient. Algoritmul pentru optimizarea ponderilor folosit a fost înlocuit cu Adam [23] folosind constanta de învățare implicită (0.001), care se presupune că obține o convergență mai bună față de RMSprop. De asemenea s-a încercat folosirea simbolului special de început de frază denumit și start token, însă s-a observat că ieșirea contextuală era întotdeauna înclinată către acest simbol în timpul predicției, sistemul nemaifiind capabil să modeleze fraza dorită ca răspuns.

### 4.3.3 Arhitectura 2.0

Deoarece modelul eșua în a scala atunci când numărul de date creștea, presupunerea a fost că acesta nu era îndeajuns de complex pentru a reprezenta distribuția setului de antrenare. De aceea, s-a încercat folosirea unor straturi suplimentare alipite vertical<sup>3</sup> pentru codor și decodor deoarece teoria spune că de la strat la strat, modelul capturează reprezentări tot mai complexe ale intrării [22]. 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 renunțat astfel la straturi stacked deoarece nu rezolvau problema scalabilității. Noua configurație pentru fiecare ieșire RNAR era 300 de neuroni pentru codor, 300 de neuroni pentru context și 300 pentru decodor. S-a renunțat de asemenea la concatenarea ieșirii contextuale cu fiecare reprezentare a cuvintelor din decodor și trecerea acestuia prin stratul ascuns cu activare tanh, deoarece nu mai era nevoie de expandarea dimensiunii. Astfel, ieșirea contextuală era folosită acum doar ca primă intrare  $T_1$  pentru secvența decodurului, presupunerea fiind că influența contextuală se propagă oricum prin RNAR a decodurului. Încă un considerent pentru această decizie a fost faptul că ieșirea contextuală și reprezentarea word2vec a cuvintelor provin din distribuții diferite iar împreunarea lor nu reprezenta un factor corect din punct de vedere matematic. Desigur, mecanismele de atenție descrise

---

<sup>3</sup>Stacked

în Capitolul 5 posedă o putere de modelare sporită față de simpla utilizare a contextului ca primă intrare pentru decodor. Încă o observație realizată care încetinește procesul de antrenare era folosirea unei fraze cu o lungime maximă de 75 de cuvinte. În urma analizei amănunțite a setului de date, s-a observat că majoritatea frazelor aveau o lungime scurtă. Astfel, s-a calculat valoarea de 80% (80 percentile) [21] pentru lungimea maximă a frazelor și s-a determinat că aceasta este aproximativ 25. Sacrificiul realizat pentru a ignora cuvintele frazelor care depășesc lungimea de 25 de cuvinte a produs o scădere considerabilă a timpului de instruire. Odată cu schimbarea către această configurație, s-a adoptat algoritmul beam search de predicție. Deoarece spațiul de căutare devenea mai larg, răspunsurile au devenit și ele mai diversificate. Valorile pentru numărul de copii generați de fiecare nod și beam size au fost alese 2, respectiv 5. În teorie, cu cât aceste numere sunt mai mari, spațiul explorat crește și răspunsul devine mai calitativ. Însă creșterea considerabilă a acestor doi parametri aduce cu sine o viteză computațională lentă, de aceea trebuie făcut un compromis la nivelul acestor valori. În pofida acestor îmbunătățiri, sistemul încă nu scala în mod așteptat odată cu creșterea datelor.

#### 4.3.5 Arhitectura 3.5

Deoarece setul de date conține subtitrări din filme, este un lucru obișnuit ca majoritatea răspunsurilor să fie monotone. Astfel, se creează o tendință pentru anumite cuvinte care apar foarte des, cele rare fiind aproape ignorate de către sistem. De aceea, sistemul eșuează în a furniza răspunsuri diversificate, care să conțină și cuvinte rare. Acest neajuns se poate rezolva prin folosirea ponderilor pentru cuvinte în momentul când este calculată valoarea funcției de eroare. Ponderea pentru fiecare cuvânt va fi numărul de apariții / numărul total de cuvinte. Prin această metodă se asigură că sistemul consideră influența egală a tuturor cuvintelor nemaiaivând o predispoziție către cele care apar des. Dimensiunea ieșirilor codor, context, decodor au fost alese ca fiind 500, 300, 500 pentru creșterea ușoară a complexității 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 învățare de-a lungul instruirii, atunci când valoarea funcției de eroare nu scădea pe parcursul a cinci epoci consecutive cu mai mult de o valoare  $\epsilon$  aleasă. Factorul de reducere a fost ales ca fiind 75%. Această combinație de ponderi împreună cu reducerea treptată a constantei de învățare au reprezentat succesul care au produs convergența modelului curent și totodată apariția rezultatelor satisfăcătoare.

## 4.4 Rezultate

### 4.4.1 Word perplexity

Evaluarea precisă a modelelor lingvistice care nu sunt bazate pe un topic restrâns reprezintă o problemă deschisă. Nu există o metrică bine stabilită pentru evaluarea automată, iar evaluarea umană este costisitoare. Cu toate acestea, pentru modele probabilistice lingvistice, word perplexity [6] este o metrică recunoscută, fiind sugerată și cu alte ocazii pentru modele de dialog generative. Word perplexity este definit ca:

$$\text{perplexity} = e^{-\sum_x \tilde{p}(x) \log q(x)} \quad (4.1)$$

Exponentul ecuației 4.1 reprezintă valoarea de cross entropy a modelului evaluat pe setul de date de test. O valoare perplexity mai mică este indicația unui model mai bun. Metrica măsoară abilitatea modelului de a lua în considerare structura sintactică a dialogului (schimbul de replici), precum și structura sintactică a fiecărei fraze (semnele de punctuație). În generarea de dialog, distribuția cuvintelor din fraza următoare este multi-modală (există o mulțime de răspunsuri potrivite), aspect ce recomandă folosirea acestei metrici, deoarece aceasta va măsura întotdeauna probabilitatea de regenerare a exact frazei de referință. Modelul curent a obținut o valoare word perplexity de **118.74** pentru setul de test. Acest număr se poate traduce prin faptul că modelul este confuz ca și cum ar avea de ales uniform și independent aproximativ 119 posibilități pentru fiecare cuvânt.

Desigur, word perplexity nu reprezintă metrica ideală de evaluare a unui model lingvistic, deoarece un model cu o valoare mică obținută pe un set de test prezintă riscul de a nu performa la fel de bine într-o aplicație reală a cărei date nu provin din aceeași distribuție ca a setului de test. Totuși, în lipsa unei metrici eficiente de a măsura performanța unui sistem dialog, word perplexity oferă o alternativă suficient de bună pentru moment.

### 4.4.2 Evaluare umană

Deși costisitoare din punct de vedere al timpului, evaluarea umană reprezintă poate cea mai bună metrică curentă de evaluare a unui model generativ lingvistic, încadrându-se în același timp cu ideea din spatele testului Turing. În urma generării mai multor conversații, s-au ales următoarele rezultate:

Persoană reală	Chatbot
Hi!	Hi.
Are you ok?	Yes
Anything else?	Uh. Yeah.
What is it?	Well, I'd have to go back to New York anyway. I spent so much money on television.

Persoană reală	Chatbot
You are a shame.	I've never known a long way to do it anyway.
Are you proud of yourself?	I want you to go back to the hotel. I want you to leave. For me.
I will probably do that.	Well, I'm in town for a few years now. I'd like to go to sleep.

Persoană reală	Chatbot
Do you know this guy?	He'll be here. I'm sorry for nothing to worry about. I'll survive some days.
I hope I can trust you.	You don't understand. I don't believe you.
I am a man of my word.	You know what I'm gonna do.

Persoană reală	Chatbot
Are you a robot?	What are you doing?
Do you have feelings?	I don't know what you are talking about.
Tell me about yourself.	I know. It's not logical, it's not what I can do. Forget it.

După cum se poate observa, răspunsurile produse de către chatbot prezintă o corectitudine gramaticală și sintactică suficient de bună, însă acestea suferă la capitolul coerență, putându-se observa că anumite replici nu se încadrează neapărat în contextul discuției. Capitolul următor descrie câteva posibile îmbunătățiri care ar putea suprima aceste probleme.

# Capitolul 5

## Posibile dezvoltări

### 5.1 Învățare prin întărire

Învățarea 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 acțiuni într-un mediu înconjurător astfel încât să maximizeze o răsplată cumulativă [43]. Tipul acesta de învățare diferă de metodele standard supervizate prin faptul că sistemului nu îi este prezentat niciodată perechi de forma intrare-ieșire, precum nici rezolvări ale unor acțiuni suboptimale. Focusul principal este asupra performanței on-line care presupune găsirea balanței ideale între explorare (dobândirea de cunoștințe noi) și exploatare (folosirea cunoștințelor curente).

Modelul de bază RL presupune:

- Un set de stări pentru mediu și agent  $S$
- Un set de acțiuni  $A$  ale agentului
- O politică<sup>1</sup> de tranziție de la stări la acțiuni
- Reguli care determină răsplata scalară imediată a tranzițiilor
- Reguli care descriu observațiile agentului

---

<sup>1</sup>policy

Un agent RL interacționează cu mediul în pași de timpi discreți. La fiecare pas  $t$ , agentul primește o observație  $o_t$ , care de obicei include o răsplată  $r_t$ . Apoi alege o acțiune  $a_t$  din setul de acțiuni posibile, care este trimisă mai departe către mediul. Mediul realizează o tranziție către starea nouă  $s_{t+1}$ , iar recompensa  $r_{t+1}$  asociată cu tranziția  $(s_t, a_t, s_{t+1})$  este determinată (Figura 5.1).

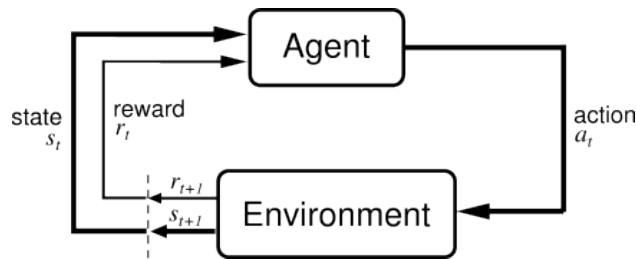


Figura 5.1: Tranziția stărilor în RL

Pentru a acționa optim, agentul trebuie să ia în considerare dependența pe termen lung a acțiunilor 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 dependență pe termen lung și una pe termen scurt. A fost aplicat cu succes în diverse situații precum controlul automat de roboți, telecomunicații, jocuri ATARI și recent jocul Go [39].

Jiwei Li et al. propun în articolul lor, Deep Reinforcement Learning for Dialogue Generation [24], utilizarea metodelor de RL pentru învățare de limbaj. Plecând de la aceeași 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, rețeaua să fie capabilă să ofere răspunsuri mai clare, contextuale și corect gramaticale. Astfel, articolul propune trei metode de recompensă pentru cuantificarea calității 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 aceluiași răspuns consecutiv de mai multe ori la rând
- $r_3$ , care recompensează corectitudinea gramaticală și coerența 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 îmbunătățite. De aceea, extinderea modelului curent pentru a folosi RL reprezintă una dintre principalele posibilele dezvoltări dorite.

## 5.2 Mecanisme de atenție

Mecanismele de atenție [3] pentru o RNA sunt puternic bazate pe cele vizuale existente la om. Atenția vizuală umană, deși este bine studiată și există diverse modele, toate se rezumă la a fi capabile să se concentreze asupra unei anumite regiuni dintr-o imagine cu o rezoluție sporită, pe când imaginea de fundal este percepută cu o rezoluție mai scăzută, 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 înțelesul întregii fraze de intrare într-un vector de lungime fixă, care mai apoi va fi folosit pentru inferență. Pare nerezonabil ca întreaga informație să fie stocată într-un singur vector, în special dacă lungimea frazei este mare. LSTM se presupune că rezolvă cu succes problema dependenței temporale a unei secvențe, însă în practică aceste neajunsuri încă apar atunci când lungimea acesteia este prea mare. O rezolvare plauzibilă poate fi oferită prin folosirea RNAR bidirecționale, care produc rezultate semnificativ mai bune.

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

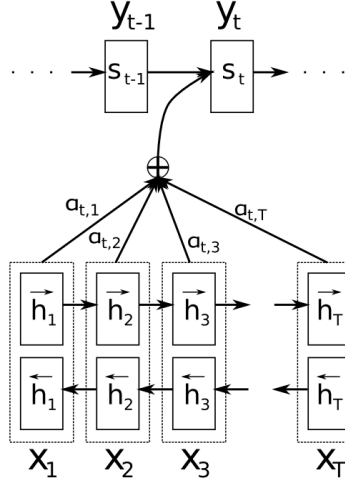


Figura 5.2: Mecanism de atenție

În figura de mai sus,  $y$  reprezintă ieșirea generată de către decodor iar  $x$  cuvintele corespunzătoare frazei de intrare. Aspectul cel mai important este că fiecare ieșire  $y_t$  depinde acum de o combinație ponderată (valorile  $a$ ) a tuturor stărilor din codor, și 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 de obicei normalizate astfel încât suma lor să fie egală cu  $1^2$ .

Costul atașat acestor mecanisme este destul de mare. Pentru fiecare combinație intrare-ieșire trebuie calculată o valoare de atenție și stocată. Acest lucru este contraintuitiv, deoarece atenția umană se presupune că economisește resurse computaționale focalizând doar ce este important. Modelul prezentat analizează totul în detaliu înainte de a decide ce este important. Totuși, acest lucru nu a împiedicat ca mecanismele de atenție să devină atât de populare și să producă rezultate bune pentru o multitudine de sarcini.

### 5.3 Setul de date

Unul dintre aspectele esențiale de care puterea de procesare Deep Learning profită este cantitatea mare de date. Dacă modelele tradiționale de Machine Learning nu reușeau să exploateze creșterea volumului de date, modelele Deep Learning nu suferă de această neajuns (Figura 5.3). Acest fapt

<sup>2</sup>Distribuție a stărilor de intrare



este explicabil prin simplul fapt că modelele tradiționale nu erau suficient de complexe pentru a captura distribuția datelor de intrare. Datorită complexității sporite a modelelor deep learning, care au avut parte de o explozie de popularitate odată cu creșterea capacității computaționale, a memoriei pentru stocarea datelor și folosirea procesorului grafic pentru a paraleliza calculele pe mii de nuclee, acestea pot surprinde cu ușurință distribuția unui set de date cu multe șabloane. Având acest beneficiu la dispoziție, modelul actual poate fi îmbunătățit dacă este antrenat luând în considerare o cantitate mai mare de triplete de fraze.

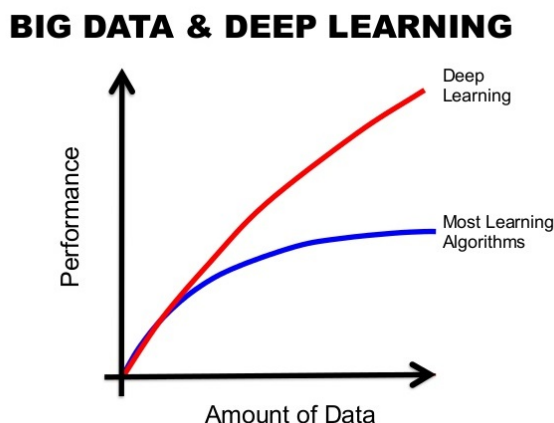


Figura 5.3: Scalabilitatea folosind Machine Learning vs Deep Learning

Desigur, o cantitate mai mare de date nu înseamnă întotdeauna că rezultatul final este mai bun. În urma experimentelor realizate, se poate observa o tendință a chatbot-ului de a fi înclinat către răspunsuri care se regăsesc în setul de antrenare de un număr 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 distribuției datelor de intrare. Acest lucru ridică problema diversității cât și a calității datelor care sunt furnizate sistemului. O soluție plauzibilă este antrenarea folosind diverse seturi de date cunoscute ca fiind de referință pentru sarcini de procesare de limbaj natural, precum The Ubuntu Dialogue Corpus [25], și compararea cu modelul curent.

# Capitolul 6

## Concluzii

În această lucrare s-a prezentat un model generativ pentru procesare naturală de limbaj, capabil să întrețină contextul discuției folosind o arhitectură de rețea neurală considerată state of the art. Folosind un set de date care conține replici din filme, s-a arătat că este posibilă modelarea unui sistem care poate produce răspunsuri open-topic generative. Deși rezultatele sunt departe de a oferi impresia unui partener uman angajat într-o conversație, acest experiment de cercetare realizează un pas important către dezvoltarea unor modele lingvistice cu adevărat inteligente.

Dacă prezentul este încă sub semnul întrebării în privința acestor sisteme conversaționale, eforturile uriașe investite de către marile grupuri de cercetare ale lumii nu pot decât să dovedească interesul major pentru această arie, rezultatele urmând să apară fără îndoială în anii ce vor urma, pentru ca în final un astfel de sistem să doboare testul vechi de aproape 70 de ani.

# Bibliografie

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.
- [2] Abu-Mostafa. Neural networks - biological inspiration. <https://work.caltech.edu/library/103.html>.
- [3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.
- [4] Rafael E. Banchs. Movie-dic: A movie dialogue corpus for research and development. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, pages 203–207, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.
- [5] Mokhtar S. Bazaraa. *Nonlinear Programming: Theory and Algorithms*. Wiley Publishing, 3rd edition, 2013.
- [6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

- [7] Dimitrios Bizopoulos. Gpu vs cpu in convolutional neural networks using tensorflow. <https://relinklabs.com/gpu-vs-cpu-in-convolutional-neural-networks-using-tensorflow>.
- [8] Jason Brownlee. How to improve deep learning performance. <http://machinelearningmastery.com/improve-deep-learning-performance/>.
- [9] Ellie Burns. From the turing test to deep learning: Artificial intelligence goes mainstream. <http://www.cbronline.com/4th-revolution/from-the-turing-test-to-deep-learning-artificial-intelligence-goes-mainstream/>.
- [10] François Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [11] Junyoung Chung, Çağlar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
- [12] Josh Costine. Facebook will launch group chatbots at f8. <https://techcrunch.com/2017/03/29/facebook-group-bots/>.
- [13] Jill Duffy. Duolingo. <http://uk.pcmag.com/duolingo/18817/review/duolingo>.
- [14] 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>.
- [15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 249–256, May 2010.
- [16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [17] Geoffrey Hinton. Overview of mini-batch gradient descent. [http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf).
- [18] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998.
- [19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

- [20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- [21] R. Johnson and P. Kuby. *Elementary Statistics, Enhanced Review Edition*. Cengage Learning, 2007.
- [22] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. <http://cs231n.github.io/convolutional-networks/>.
- [23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [24] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *CoRR*, abs/1606.01541, 2016.
- [25] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *CoRR*, abs/1506.08909, 2015.
- [26] R. B. MARIMONT and M. B. SHAPIRO. Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics*, 24(1):59, 1979.
- [27] Chris McCormick. Google’s trained word2vec model in python. <http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/>.
- [28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.
- [29] Michael C. Mozer. Backpropagation. chapter A Focused Backpropagation Algorithm for Temporal Pattern Recognition, pages 137–169. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.
- [30] P. Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common LISP*. Artificial intelligence programming languages. Morgan Kaufman Publishers, 1992.
- [31] 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.

- [32] Lawrence R. Rabiner. Readings in speech recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
- [33] Suriyadeepan Ram. Chatbots with seq2seq. <http://suriyadeepan.github.io/2016-06-28-easy-seq2seq/>.
- [34] Sebastian Ruder. On word embeddings - part 1. <http://sebastianruder.com/word-embeddings-1/>.
- [35] Gerardo Salandra. Does your business need a chatbot? <https://rocketbots.io/blog/business-need-chatbot/>.
- [36] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.
- [37] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. Hierarchical neural network generative models for movie dialogues. *CoRR*, abs/1507.04808, 2015.
- [38] Faizan Shaikh. Why are gpus necessary for training deep learning models? <https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/>.
- [39] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- [40] 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/>.
- [41] Pavel Surmenok. Chatbot architecture. <http://pavel.surmenok.com/2016/09/11/chatbot-architecture/>.
- [42] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.
- [43] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

- [44] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.
- [45] theguardian. Computer simulating 13-year-old boy becomes first to pass turing test. <https://www.theguardian.com/technology/2014/jun/08/super-computer-simulates-13-year-old-boy-passes-turing-test>.
- [46] Wildml. Deep learning for chatbots, part 1 introduction. <http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/>.
- [47] Wildml. Deep learning for chatbots, part 2 implementing a retrieval-based model in tensorflow. <http://www.wildml.com/2016/07/deep-learning-for-chatbots-2-retrieval-based-model-tensorflow/>.
- [48] Wildml. Recurrent neural networks tutorial, part 1 - introduction to rnns. <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>.