MAJOR PROJECT - II

RNN BASED SPOKEN LANGUAGE UNDERSTANDING SYSTEM FOR AIR TRAVEL DOMAIN

Submitted in partial fulfillment of
the requirements for the award of the

Degree of Bachelor of Technology
in
Computer Engineering

Submitted by

2K13/CO/113	SARTHAK JAIN
2K13/CO/119	SHASHANK GARG

Under the supervision of
Dr. Manoj Kumar

Department of Computer Science and Engineering
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi-110042

DECLARATION

I hereby certify that work which is presented in the Major Project entitled **RNN based Spoken Language Understanding System for Air Travel Domain** in fulfillment for the award of the Degree of Bachelor of Technology and submitted to the Department of Computer Engineering, Delhi Technological University (Formerly Delhi College of Engineering), New Delhi is an authentic record of my own, carried out during a period of January,2017-May,2017, under the supervision of Dr. Manoj Kumar.
The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Signature

Sarthak Jain (2K13/CO/113)
Shashank Garg (2K13/CO/119)

Date:

CERTIFICATE

This is to certify that **Sarthak Jain (2K13/CO/113)** and **Shashank Garg (2K13/CO /119)** the bonafide students of **Bachelor of Technology in Computer Engineering** of **Delhi Technological University** (Formerly Delhi College of Engineering), New Delhi of **2013-2017 batch** has completed their major project entitled **RNN based Spoken Language Understanding System for Air Travel Domain** under my supervision.
It is further certified that the work done in this dissertation is a result of candidate's own efforts.

Date:

Dr. Manoj Kumar
(Project Guide)

ACKNOWLEDGEMENTS

We are ecstatic to take this opportunity to thank the people who helped us in carrying out our project successfully. We would like to express a token of gratitude to our guide and mentor Dr. Manoj Kumar, for involving us in such a brilliant project and helping us throughout the process, without which this project couldn't have been possible. He made the entire project experience an exciting and enjoyable one by providing us with a lot of inspiration and motivation.

At the outset, we thank God almighty for making our endeavor a success. We also express our gratitude to our Department of Computer Science and Engineering, Delhi Technological University, for providing us with adequate facilities, ways and means by which we were able to complete this project. We are also grateful to all other teachers and lab assistants of Computer Science and Engineering Department for their invaluable teaching, help and support without which the successful completion of this project would not have been possible.

Lastly, we would like to thank our friends and families for being there for us and supporting us throughout the course of the project and never letting us give up when things got too complicated.

Sarthak Jain (2K13/CO/113)

Shashank Garg (2K13/CO/119)

ABSTRACT

Spoken language understanding (SLU) aims to extract the meaning of the speech utterances. While understanding language is still considered an unsolved problem, in the last decade, a variety of practical goal-oriented conversational understanding systems have been built for limited domains. These systems aim to automatically identify the intent of the user as expressed in natural language, extract associated arguments or slots, and take actions accordingly to satisfy the users requests. In such systems, the speakers utterance is typically recognized using an automatic speech recognizer (ASR). Then the intent of the speaker is identified from the recognized word sequence using an SLU component. Finally, a dialog or task manager (DM) interacts with the user (not necessarily in natural language) and helps the user achieve the task that the system is designed to support.

In this project, we propose an RNN-based online joint SLU model that performs intent detection and slot filling as the input word arrives. In addition, we suggest that the generated intent class and slot labels are useful for next word prediction in online automatic speech recognition (ASR). Therefore, we propose to perform intent detection and slot filling jointly in a conditional RNN model. The proposed joint model can be further extended for belief tracking in dialogue systems when considering the dialogue history beyond the current utterance. Moreover, it can be used as the RNN decoder in an end-to-end trainable sequence-to-sequence speech recognition model.

Evaluation of online SLU model is made on the ATIS benchmarking data set. On SLU tasks, our joint model outperforms the independent task training model on intent detection error rate, with slight degradation on slot filling F1 score. The joint model also shows advantageous performance in the realistic ASR settings with noisy speech input.

CONTENTS

Certificate **i**

Certificate **ii**

Acknowledgements **iii**

Abstract **iv**

1 Introduction **1**

1.1 Overview 1

1.2 Objectives of this work 4

2 Literature Review **6**

2.1 Technical Challenges 8

2.2 Knowledge Based Solutions 9

2.2.1 Semantically Enhanced Syntactic Grammar 9

2.2.2 Semantic Grammars 9

2.3 Data Driven Approaches 10

2.3.1 Generative Models 12

2.3.2 Integrated Models 14

2.3.3 Conditional Models 15

2.3.4 Neural Models 16

3 Techniques used **18**

3.1 Neural Networks 18

3.1.1 Feed-forward neural nets 19

3.2 Recurrent Neural Nets 21
3.2.1 Recurrent architectures 23
3.2.2 Vanishing Gradient & Gradient Explosion Problems 24
3.2.3 Solution to the Exploding & Vanishing Gradients 26
3.2.4 Deep Bidirectional RNNs 28
3.3 Gated Recurrent Units 30
3.4 Long-Short-Term-Memories 31

4 Proposed Work **34**
4.1 Background 34
4.1.1 RNN for Intent Detection and Slot Filling 36
4.2 Method 1 37
4.2.1 Model 37
4.2.2 Next Step Prediction 38
4.3 Method 2 39
4.3.1 Model 39
4.3.2 Objective Functions 41
4.3.3 Training 42
4.3.4 Inference 43

5 Results and Discussion **45**
5.1 Data 45
5.2 Evaluation Metrics 45
5.3 Model Training 46
5.4 Results 46
5.5 Error Analysis 48
5.5.1 Intent Detection Errors 48
5.5.2 Slot Filling Errors 50

6 Conclusion **53**

Bibliography **54**

List of Figures

1.1 High Level Depiction of Spoken Dialog Systems 2

2.1 Frame based SLU in a typical ATIS system 7

3.1 Nonlinear functions used in neural nets. 20

3.2 Applying dropout to a neural network. 21

3.3 A Recurrent Neural Network (RNN). Three time-steps are shown. 22

3.4 The inputs and outputs to a neuron of a RNN 24

3.5 Gradient explosion clipping visualization 27

3.6 A bi-directional RNN model 28

3.7 A deep bi-directional RNN with three RNN layers. 29

3.8 The detailed internals of a GRU 31

3.9 The detailed internals of a LSTM 32

4.1 ATIS corpus sample with intent and slot annotation 35

4.2 (a) RNN language model. (b) RNN intent detection model. The RNN output at last step is used to predict the intent class. (c) RNN slot filling model. 36

4.3 Joint online RNN model for intent detection, slot filling, and next word prediction. 38

4.4 Attention Based RNN Model 40

4.5 Schedule of increasing intent contribution to the context vector along with the growing input sequence 43

List of Tables

5.1 ATIS test set Results 47
5.2 Confusion Matrix for Intent Detection 49

CHAPTER 1

Introduction

In last decades Information Technology has been the main topic in most research centers of the world. Human-Computer interaction is becoming more and more an easy available technology in everyday life. This thanks to impressive advancements in research topics like Machine learning, Automatic Speech Recognition and Understanding, Data-driven stochastic approaches, Text Processing and Speech Synthesis.

One class of applications relying heavily on this technologies is Spoken Dialog Systems. Spoken Dialog Systems allow humans to engage complex dialogs with machines using the most natural communication mean ever known: their voice. In the last decades, this class of applications has been made able to interact with humans and satisfy their needs in such a way that allows hoping that this technology will be soon available to everybody and in any kind of device, from the most powerful desktop PC to the cheapest laptops and mobile phones.

1.1 Overview

Spoken Language Understanding (SLU) is the semantic interpretation of signs conveyed by a speech signal. The goal of SLU is to extract a conceptual representation from spoken sentence transcriptions in a natural language. This task is very complex. Signs to be used to produce conceptual representation are coded in the signal along with other noisy information. Spoken sentences many times dont follow the grammar of a language, they could contain self correction, hesitations, repetitions and many other irregular phenomena

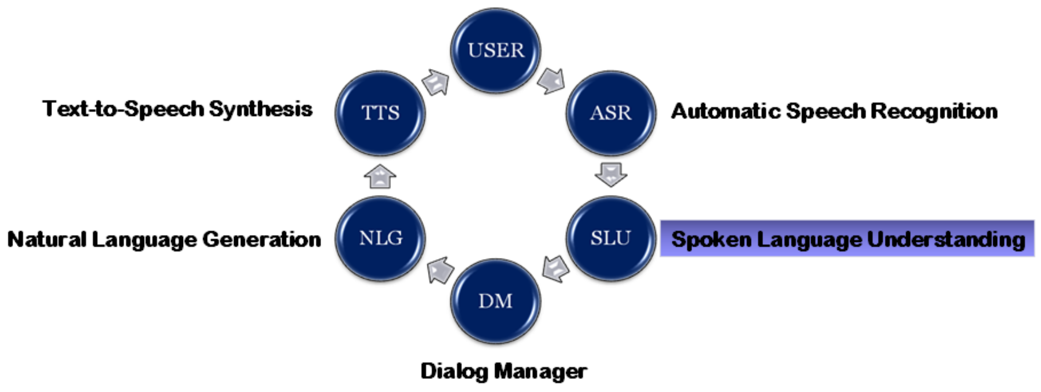

USER
TTS
Text-to-Speech Synthesis
ASR
Automatic Speech Recognition
NLG
Natural Language Generation
SLU
Spoken Language Understanding
DM
Dialog Manager

Figure 1.1: High Level Depiction of Spoken Dialog Systems

due to the spontaneous nature of spoken language. Furthermore SLU systems are applied to the output of an Automatic Speech Recognizer (ASR) so they must be robust to noise introduced by spontaneous events typical of spoken language and errors introduced by ASR. ASR components produce a stream of words with no information about sentence structure, like punctuation and sentence boundaries, so SLU systems must perform text segmentation and understanding at the same time. The level of complexity needed in order to represent the meaning of a spoken utterance depends mainly on the application targeted.

The application we consider is Spoken Dialog, in particular we focus on the understanding module of this class of applications. A high level schema of a Spoken Dialog System application is shown in Figure 2.1. The dialog is initiated by the user as response to an opening prompt from the system. The user utterance is automatically transcribed by the Automatic Speech Recognition (ASR) component. The ASR takes as input a speech signal and produces its transcription in textual format. The Spoken Language Understanding (SLU) module takes as input the output of ASR and generates a meaning representation. Based on the interpretation coming from the SLU module, the Dialog Manager (DM) select the next dialog turn, this is converted into a natural language sentence by the Natural Language Generation (NLG) module. Finally, the Text-To-Speech (TTS) module synthsizes the generated sentence as a speech signal, which is sent back to the user to continue the dialog. The loop depicted in Figure 2.1 is repeated until the application completes the modelled task.

Spoken dialog systems need sophisticated SLU models in order to implement dialog applications that go beyond solving simple tasks like call routing or form filling [35]. Three level of complexity can be defined for dialog applications. The first level involves the translation from words to basic conceptual constituents. The second level includes semantic composition on the basic constituents yielded in the first level. At the third level context-sensitive validation is performed. At this level each utterance is considered as a set of sub-utterances and a broad context is taken into account. The interpretation of a sub-utterance is performed context sensitive to the others. Going from level one to level three, tasks of increasing complexity can be solved, from call routing or classification of utterances [35] to Help-Desk application for hardware and software repairing [31].

SLU is performed as a semantic parsing of spoken sentences. Approaches based on syntactic analysis or directly on semantic analysis have been proposed, in any case semantic constituents are instantiated by one or more words that have a corresponding syntactic constituent. Among understanding modules proposed in the last two decades, first solutions were based on semantic grammars, but as the amount of data available for application development increased, as well as application complexity, stochastic approaches have been preferred.

Many problems of interpretation in SLU systems derive from the fact that many sentences are ungrammatical and ASR hypotheses contain errors, so grammars have a limited coverage. These considerations suggest the use of more specific, but more robust, models. In the early nineties, the DARPA ATIS project started a series of task-dependent SLU systems. Data were collected in the domain of a flight information and reservation service. ATIS project aimed at providing a natural language interface to a travel information database and provided a benchmark for many spoken language understanding systems, one is discussed in [64].

1.2 Objectives of this work

Major components in SLU systems include identifying speakers intent and extracting semantic constituents from the natural language query, two tasks that are often referred to as intent detection and slot filling. Intent detection can be treated as a semantic utterance classification problem, and slot filling can be treated as a sequence labeling task. These two tasks are usually processed separately by different models. For intent detection, a number of standard classifiers can be applied, such as support vector machines (SVMs) and convolutional neural networks(CNNs). For slot filling, popular approaches include using sequence models such as maximum entropy Markov models (MEMMs), conditional random fields (CRFs), and recurrent neural networks (RNNs).

Recently, neural network based models that jointly perform intent detection and slot filling have been reported using CRF for joint intent detection and slot filling and recursive neural network (RecNN) that learns hierarchical representations of the input text for the joint task. Such joint models simplify SLU systems, as only one model needs to be trained and deployed.

The previously proposed joint SLU models, however, are unsuitable for online tasks where it is desired to produce outputs as the input sequence arrives. In speech recognition, instead of receiving the transcribed text at the end of the speech, users typically prefer to see the ongoing transcription while speaking. In spoken language understanding, with real time intent identification and semantic constituents extraction, the downstream systems will be able to perform corresponding search or query while the user dictates. The joint SLU models proposed in previous work typically require intent and slot label predictions to be conditioned on the entire transcribed word sequence. This limits the usage of these models in the online setting.

In this paper, we propose an RNN-based online joint SLU model that performs intent detection and slot filling as the input word arrives. Therefore, we propose to perform intent detection, slot filling, and language modeling jointly in a conditional RNN model. The proposed joint model can be further extended for belief tracking in dialogue systems when considering the dialogue history beyond the current utterance. Moreover, it can be used as the RNN decoder in an end-to-end trainable sequence-to-sequence speech recognition model.

CHAPTER 2

Literature Review

In the United States, the study of the frame-based SLU started in the 1970s in the DARPA Speech Understanding Research (SUR) and then the Resource Management (RM) tasks. At this early stage, natural language understanding (NLU) techniques like finite state machine (FSM) and augmented transition networks (ATNs) were applied for SLU [1]. The study of SLU surged in the 90s, with the DARPA sponsored Air Travel Information System (ATIS) evaluations [2]. Multiple research labs from both academia and industry, including AT&T, BBN, Carnegie Mellon University, MIT and SRI, developed systems that attempted to understand users spontaneous spoken queries for air travel information (including flight information, ground transportation information, airport service information, etc.) and then obtain the answers from a standard database. ATIS is an important milestone for the frame-based SLU, largely thanks to its rigorous componentwise and end-to-end evaluation, participated by multiple institutions, with a common test set.

Figure 2.1 shows the role of the frame-based SLU component in a typical ATIS system. While ATIS focused more or less on the understanding of a single-turn utterance, the more recent DARPA Communicator program [3] focused on the rapid and cost-effective development of multi-modal speech enabled dialog systems, in which general infrastructures for dialog systems were developed, where different component systems for ASR, SLU, DM and TTS can be plugged in and evaluated. Naturally, many SLU technologies developed in ATIS were used in the SLU component of the Communicator program. Eight systems from AT&T, BBN, University of Colorado, Carnegie Mellon University, IBM, Lucent Bell Labs, MIT, and SRI participated in the 2001 evaluation [4]. In the

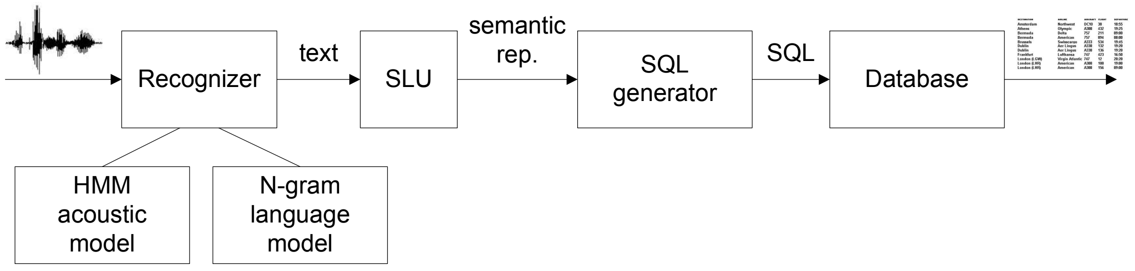


Figure 2.1: Frame-based SLU in a typical ATIS system, which consists of 1) a speech recognizer with both the acoustic model and language model trained with the ATIS specific data; 2) a SLU system that extracts the semantic representation (meaning) from the recognized text; and 3) a SQL generator that automatically generates the database query based on the semantic representation.

mean time, the AI community had separate effort in building a conversational planning agent, such as the TRAINS system [5].

Parallel efforts were made on the other side of the Atlantic. The French EVALDA/MEDIA project aimed at designing and testing the evaluation methodology to compare and diagnose the context-dependent and independent SLU capability in spoken language dialogs. Participants included both academic organizations (IRIT, LIA, LIMSI, LORIA, VALORIA, CLIPS) and industrial institutions (FRANCE TELECOM R&D, TELIP). Like ATIS, the domain of this study was restricted to database queries for tourist and hotel information.

The more recent LUNA project sponsored by the European Union focused on the problem of real-time understanding of spontaneous speech in the context of advanced telecom services. Its major objective is the development of a robust SLU toolkit for dialog systems, which enhances users experience by allowing natural human-machine interactions via spontaneous and unconstrained speech. One special characteristic of the project, which is absent in the similar projects in the US, is its emphasis on multilingual portability of the SLU components.

2.1 Technical Challenges

The frame-based SLU is closely related to natural language understanding (NLU), a field that has been studied for more than half a century. NLU focus mainly on understanding of general domain written texts. Because there is not a specific application domain for the general purposed NLU, the semantics in NLU have to be defined in a broader sense, such as thematic roles (agents, patients, etc.) In contrast, the frame-based SLU has, in the current state of technology, focused only on specific application domains. The semantics are defined very specifically according to the application domain, as illustrated by the above examples of semantic frames. Many domain-specific constraints can be included in the understanding model. Ostensibly, this may make the problem easier to solve. Unfortunately, there are many new challenges for spoken language understanding, including

- Extra-grammaticality spoken languages are not as well-formed as written languages. People are in general less careful with speech than with writings. They often do not comply with rigid syntactic constraints.

- Disfluencies false starts, repairs, and hesitations are pervasive, especially in conversational speech.

- Speech recognition errors Speech recognition technologies are far from perfect. Environment noise, speakers accent, domain specific terminologies, all make speech recognition errors inevitable. It is common to see that a generic speech recognizer has over 30% word error rates on domain specific data.

- Out-of-domain utterances a dialog system can never restrict a user from saying anything out of a specific domain, even in a system-initiated dialog, where users are prompted for answers to specific questions. Because the frame-based SLU focuses on a specific application domain, out-of-domain utterances are not well modeled and can often be confused as an in-domain utterance. Detecting the out-of-domain utterances is not an easy task it is complicated by the extra-grammaticality, disfluencies and ASR errors of in-domain utterances.

2.2 Knowledge Based Solutions

2.2.1 Semantically Enhanced Syntactic Grammar

Many advocates of the knowledge-based approach believe that general linguistic knowledge is helpful in modeling domain specific language. This includes the syntactic constraints as well as some optional rudimentary domain-independent semantic knowledge. However, since the ultimate goal is to extract the domain-dependent semantic information, one major question is how to inject the domain specific semantic constraints into a domain-independent grammar.

MITs TINA system [6] aims at the graceful, seamless interface between syntax and semantics. It uses context free grammar (augmented with a set of features used to enforce several syntactic and semantic constraints via unification). The injection of the domain-dependent semantics is accomplished by replacing the low level syntactic nonterminals with the semantic non-terminals. In doing so, the top level syntactic rules makes the grammar capable of modeling the domain-independent linguistic constraints, such as the Wh-movement/trace-management.

SRIs Gemini system [7] is implemented on top of its Core Language Engine (CLE) [8], a general natural language understanding system that parses an input sentence and generates its semantic representation in the logical forms. Here the unification grammar is also used to model the general syntactic constraints. Unlike TINA that blends syntax and semantics together with the mixed grammar categories in constituent parsing, it clearly separates the domain independent syntax from the domain dependent semantics. Another specialty of Gemini is its adoption of the logical forms instead of the frame-like representation for semantics.

2.2.2 Semantic Grammars

While using the semantically-enhanced syntactic grammars saves grammar developers from the effort to model the general language structures, it requires profound knowledge about the general syntactic grammars. In addition, the knowledge-based approach often requires the exact matching of input sentences to the grammar rules, which makes it not robust to ASR errors, extra-grammaticality and disfluencies in spontaneous speech. Often it has to resort to some kind of semantic-based robust parsing as a backup.

The Phoenix spoken language understanding system [9] directly models the domain dependent semantics with a semantic grammar. It was used by CMU in the ATIS evaluation, and was one of the top performing SLU systems in the evaluation. As we have discussed previously about semantic representation, it uses semantic frames to represent semantic relations the basic type of action for the application. Slots in a frame are filled by matching the input strings (sentences) against the slot-nets, the recursive transition networks (RTNs) that specifies the patterns for filler strings. RTNs are finite state transition networks, where the arcs in the networks can include not only terminal words, but also calls to other networks. They are equivalents of the context free grammars in graph representation.

2.3 Data Driven Approaches

The knowledge-based solution has the advantage of not requiring much labeled data. In addition, almost everyone can start writing a SLU grammar with some basic training. The grammar can be used as both the ASR language model and the SLU model in a single pass speech understanding. However, a knowledge-based system is difficulty and expensive to develop and maintain due to the following reasons:

1. Grammar development is an error-prone process. While it does not take much effort for a common developer to learn the syntax for writing a speech understanding grammar, it requires combined linguistic and engineering expertises, plus the deep knowledge about the application domain, to write a good grammar. Grammar authoring is a balancing act between simplicity and coverage. People talk differently, therefore a good grammar has to account for the different expressions for the same concept, action or request. If a grammar is too simple, it is inadequate to model the linguistic diversity. On the other hand, if a grammar is too complicated, it may not only slow down the parser, but also increase the ambiguities, hence confuses the SLU system and degrades its performance.

Design the structure of a grammar is an art. It takes much experience to have a good design where frequently used concepts, for example, are modeled by separate rules to be shared by other rules at a higher level.

2. It takes multiple rounds to fine tune a grammar. Grammar authoring can hardly be a one-shot deal nobody can write a perfect grammar with a single try. Furthermore, grammars need to evolve over time new features and scenarios may be introduced to an application after its initial deployment. Ideally, an SLU system should be able to automatically adapt to the real data collected after its deployment. On the contrary, knowledge-based systems require an experts involvement, sometimes even the involvement of the original system designer, in the adaptation loop.

3. Grammar authoring is difficult to scale up. It is relatively easy to write a grammar to model a single concept as in a system-initiated dialog system, where the user is prompted to provide a single piece of information (e.g., name, account number, social security number, address, etc.) at a dialog turn. However, if we allow users to volunteer multiple pieces of information in a single utterance, the ways to put together these pieces are combinational. As we have shown previously, for a very restricted domain like ATIS, the semantic grammar already contains 3.2k non-terminals and 13k grammar rules.

SLU based on data-driven statistical learning approaches directly addresses many of the problems associated with the knowledge-based solutions. Statistical SLU systems can automatically learn from example sentences with their corresponding semantics annotated. Compared to the manual grammar authoring, the annotations are much easier to create, without the requirement of the specialized knowledge. The statistical approach can adapt to new data, possibly via unsupervised learning. One disadvantage of such an approach, however, is the data-sparseness problem. The requirement of a large amount of labeled training data is not very practical in real-world applications, which are quite different from a few showcase problems studied in research labs. This is the case especially at the early stage of system development.

2.3.1 Generative Models

In the statistical frame-based SLU, the task is often formalized as a pattern recognition problem. Given the word sequence W, the goal of SLU is to find the semantic representation of the meaning M that has the maximum a posteriori probability $P(M|W)$. And the objective function of a generative model is to maximize the joint probability $P(W, M) = P(W|M)P(M)$ given a training sample of W and its semantic annotation M.

Two separate models exist in this generative framework. The semantic prior model $P(M)$ assigns probability to an underlying semantic structure or meaning M. The lexicalization model $P(W|M)$, sometimes called lexical generation or realization model [10], assigns probability to the surface sentence (i.e., word/lexical sequence) W given the semantic structure.

Semantic Priors in Understanding Models

In statistical SLU that models cross-word contextual dependency, each state represents a slot in a semantic frame. For the systems that use the flat concepts for semantic representation, such as AT&Ts CHRONUS [11] and IBMs fertility model [12], the topology of the model is a fully connected network. For models that use hierarchical semantic structures, including BBNs Hidden Understanding Model [10] and Microsoft Researchs HMM/CFG composite model [13], the semantic prior is a natural extension.

Cambridge Universitys Hidden Vector State model [14] uses another way to model the semantic prior with hierarchical structures. Named the hidden vector states, the states in the Markov chain represent the stack status of the pre-terminal nodes (the nodes immediately above the terminal words) in a semantic tree. The hidden vector states encode all the structure information about the tree, so the semantic tree structure (without the terminal words) can be reconstructed from the hidden vector state sequence. The model imposes a hard limit on the maximum depth of the stack, so the number of the states becomes finite, and the prior model becomes the Markov chain in an HMM.

Lexicalisation Models

The first lexicalization model, used by both CHRONUS and the Hidden Understanding Model, assumes a deterministic one-to-one correspondence between model states and the segments, i.e., there is only one segment per state, and the order of the segments follows the order of the states.

[15] introduces a 2+1 SLU model that integrates the normalization process in the lexicalization model. Here the number 2 stands for the semantic prior model and the concept model, which is the traditional lexicalization model that generates the lexical string from the concept. 1 stands for the additional model that treat the normalized attribute values as a hidden variables.

Implementation of Generative Models

Many generative models are implemented with standard toolkits like the stochastic finite state transducers (SFST) [16] or the graphic models [17]. For example, each SLU component can be implemented as a SFST, and the SLU system can be built by composing the component SFSTs. Raymond and Riccardi (2007) shows a SFST implementation of a generative model. The lattice from an ASR is represented by a stochastic finite state machine (SFSM) W . It uses an n-gram as the lexicalization model for concepts, which are slots or a null attribute that models the carrier phrases connecting the slots. N-gram can be represented by a SFSM as well, as described in [18]. An n-gram SFSM for a concept can be turned into a SFST by outputting the accepted words together with the concepts name. The union of all the SFSTs for all concepts forms the lexicalization model w2c that maps words to concepts. Finally, a statistical conceptual language model is used as the semantic prior model. The SFST model CLM is flexible enough to implemented different semantic prior models.

The generative models can also be easily implemented with the general purpose graphic model toolkit. For example, researchers from Universite dAvignon used Dynamic Bayesian Networks (DBNs) [15] to implement generative SLU models.

2.3.2 Integrated Models

One disadvantage of a purely data-driven, statistical SLU approach is the requirement of a large amount of training data. The preprocessing step is not modeled statistically as an integral part of the SLU model. The lack of information about the preprocessing model makes the statistical SLU model unable to predict the words for speech recognition, and it prohibits the model from properly normalizing the probabilities because the actual length of the segment replaced by the superword is unknown to the SLU model. An HMM/CFG composite lexicalization model has been introduced in [19], which we review here, to aim at solving these problems. This model uses the same semantic prior for hierarchical Markov topology. The underlying state corresponding to a slot in the semantic frame is expanded into a preamble-filler-postamble three state sequence. The preamble and postamble serve as the contextual clue for the identity of the slot, while the slot filler decides its value. The lexicalization model follows similar to CHRONUS and the Hidden Understanding model.

The HMM/CFG composite model balances the trade-off between robustness and the constraints on over-generalizations/ambiguities with the different models for the preambles/postambles and the slot fillers. The CFG model imposes a relatively rigid restriction on the slot fillers, which are more crucial for correct understanding and less subject to the disfluencies because they are semantically coherent units. The fillers are often domain specific and can be obtained from the application database, like the city names and airport names in the ATIS domain; or they are common domain-independent concepts like phone number, date, time, which are already modeled in a grammar library; or they can be automatically generated according to some high level description like a regular expression for an alphanumeric concept [19]. The non-slot states serve as the glue that sticks different slot fillers together. This type of inter-concept language is normally domain dependent, hard to pre-build a model for, and subject to more disfluencies. It varies significantly across different speakers. The n-gram model is more robust and thus suitable for this sub-language. Furthermore, the knowledge introduced by the CFG sub-model greatly compensates for the data sparseness problem (e.g., it is very unlikely to see all city names occur in all context in the training data).

2.3.3 Conditional Models

The statistical models for SLU we have introduced so far are all generative models the semantic structure M is first generated according to the semantic prior model $P(M)$, from which the observation W is generated according to $P(W|M)$, which an be modeled by the different lexicalization processes we just described.

Conditional models are non-generative. In a conditional model, the states (which encode the meaning M) are directly conditioned on the observation. For SLU, Conditional Random Fields (CRFs) or Hidden State Conditional Random Fields (HCRFs) are commonly used conditional models. With CRFs or HCRFs, the conditional probability of the entire state (label) sequence y given the observation sequence is modeled as an exponential (log-linear) distribution, with respect to a set of features $f(y, x)$.

[19] compared the use of CRF, perceptron, large margin, and MCE using stochastic gradient descent (SGD) for SF in the ATIS domain. They obtained significantly reduced slot error rates, with best performance achieved by CRF (though it was the slowest to train).

Almost simultaneously [14] proposed the use of CRF, extended by non-local features, which are important to disambiguate the type of the slot. For example, a day can be the arrival day, departure day, or the return day. If the contextual cues disambiguating them are beyond the immediate context, it is not easy for the classifier to choose the correct class. Using non-local trigger features automatically extracted from the training data is shown to improve the performance significantly.

Finally, [16] compared SVM and CRF with generative models for the ATIS task. They concluded that discriminative methods perform significantly better, and furthermore, it is possible to incorporate a-priori information or long distance features easily. For example they added features such as Does this utterance have the verb arrive. This resulted in about 10% relative reduction in slot error rate. The design of such features usually requires domain knowledge.

2.3.4 Neural Models

In comparison to the above described techniques, deep learning uses many layers of neural networks. It has made strong impacts on applications ranging from automatic speech recognition to image recognition.

A distinguishing feature of NLP applications of deep learning is that inputs are symbols from a large vocabulary, which led the initial work on neural language modeling to suggest map words to a learned distributed representation either in the input or output layers (or both), with those embeddings learned jointly with the task. Following this principle, a variety of neural net architectures and training approaches have been successfully applied. Particularly, RNNs are also widely used in NLP. One can represent an input symbol as a one-hot vector, i.e., containing zeros except for one component equal to one, and this weight vector is considered as a low-dimensional continuous valued vector representation of the original input, called word embedding. Critically, in this vector space, similar words that have occurred syntactically and semantically tend to be placed by the learning procedure close to each other, and relationships between words are preserved. Thus, adjusting the model parameters to increase the objective function for a training example which involves a particular word tends to improve performances for similar words in similar context, thereby greatly improving generalization and addressing the curse-of-dimensionality obstacle faced with traditional n-gram non-parametric models [26].

Recently, RNNs [20] and convoluational [21] neural networks (CNNs) have been applied to SLU. The Elman [22] architecture adopted in uses past hidden activities, together with the observations, as the input to the same hidden layer, which in turn applies a nonlinear transformation to convert the inputs to activities. The Jordan architecture exploited in uses past predictions at the output layer instead of the past hidden activities as additional inputs to the hidden layer. CNNs are used similar to that in to extract features through convolving and pooling operations. CNNs achieved comparable performances to RNNs on SLU tasks.

The RNNs in are trained to optimize the frame cross-entropy criterion. More recently, sequence discriminative training is used to train RNNs. Similar work is conducted for CNNs. The main motivation of using sequence discriminative training is to overcome the label biasness problem that is addressed by CRFs. It incorporates dependence between output tags and adds a knowledge source for performance improvements.

CHAPTER 3

Techniques used

This chapter provides an overview of major Natural Language Processing, Machine Learning and Statistical Techniques involved in this project.

3.1 Neural Networks

General idea of artificial neural networks emerged after World War II. Perceptron, a single artificial neuron, was created in 1958 by Frank Rosenblatt, but it became popular only after combination with the backpropagation algorithm. At that time neural nets have not reached massive popularity, not because they do not work, but because small computing power of machines back then, and also the lack of datasets. Recently (after 2000), neural nets became popular again under the name of "deep learning" to emphasize the use of several layers stacked on top of each other to create deep architectures, which are far more practical than shallow ones. During this reinvention, neural nets have been successfully applied in multiple fields like computer vision, speech recognition, and natural language modeling.

Nowadays, various useful architectures, techniques and applications of neural nets are introduced almost every day. As it is not possible to through all of them, in this chapter I will describe only a handful, which are most significant and will be later used in the research. This chapter is divided into three sections, each focusing on different type of neural nets - feed-forward, recurrent, and convolutional. However, do not see this division as strict and separating, tools introduced in one part can and will be used in different types of networks.

3.1.1 Feed-forward neural nets

Feed-forward networks are simplest architecture of neural nets, yet they can solve many real world tasks. Most commonly used in classification problems, feed-forward nets showed very promising results, which later proved to be true. Later, they have been replaced by convolutional nets, which are specific type of complex feed-forward neural net. However, simple architectures still have place for utilization.

In this part, We will cover linear neuron, rectifiers and other nonlinear functions used, and dropout, as they are most important to the following work. Other tools like softmax layer, loss functions, and training algorithms will be skipped.

Linear unit

In this type of neuron, the output of the unit is simply the weighted sum of its inputs added to a bias term, described by equation

$$y = Wx + b. \tag{3.1}$$

A combination of these neurons performs a linear transformation of the input vector. Ability to perform only linear and affine transformations is also its weakness, as some kind of nonlinear function needs to be added to produce more complicated functions. However it is useful at the beginning and end of the network, to emphasize important features of the input or output and change its dimensionality.

This type of unit is the most basic one. It was part of the Rosenblatt's perceptron as well as the boolean function, which later evolved into nonlinear functions, like Rectifier described further.

Rectifier and ReLU

Combination of linear layers in neural network can result only in another linear layer, which is useless for example on problems of nonlinear separation. To break free from limitations induced, we need to introduce some kind of nonlinearity directly into the

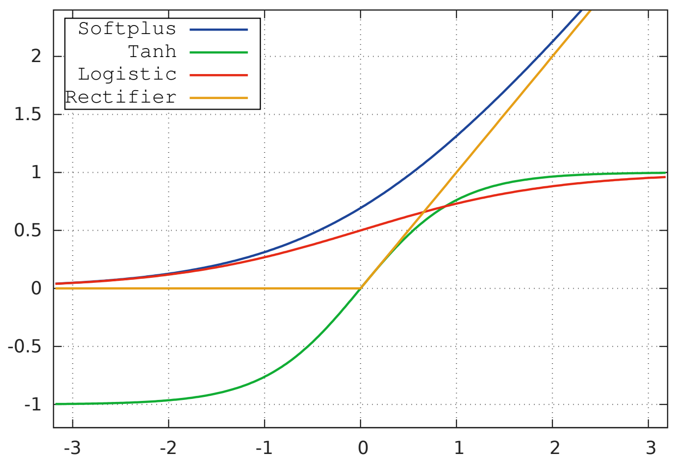


Figure 3.1: Nonlinear functions used in neural nets.

network. Most commonly used method is to apply a nonlinear activation function to the output of a linear neuron. As to which function, there are many suitable options, rectifier nowadays being the most popular one.

In the context of neural networks, the rectifier is an activation function defined as

$$f(x) = max(0, x). \tag{3.2}$$

Rectifier is usually used after a linear unit creating together Rectified Linear Unit (relu), which showed improvements in restricted Boltzmann machines, speech processing, and it is also default option in convolutional networks. This unit has several advantages against other functions – in randomly initialized networks, only about 50% of units are activated. There are no problems with vanishing gradient in large inputs. Computation of the function is also more efficient than other functions. Issue with this function is non-differentiability at zero, however it is differentiable at any point arbitrarily close to 0 and can be replaced with softplus, which is analytic function smoothly approximating rectifier. Currently, more variations of relu were introduced – Leaky relu, parametric relu, etc. and their performance can be even better than vanilla relus.

Before relu, popular functions were hyperbolic tangent and standard logistic function. However, these functions are costly to compute, even though they can be replaced with polynomials. Hyperbolic tangent was preferred as better version of logistic function. See how the discussed functions look at Figure 3.1.

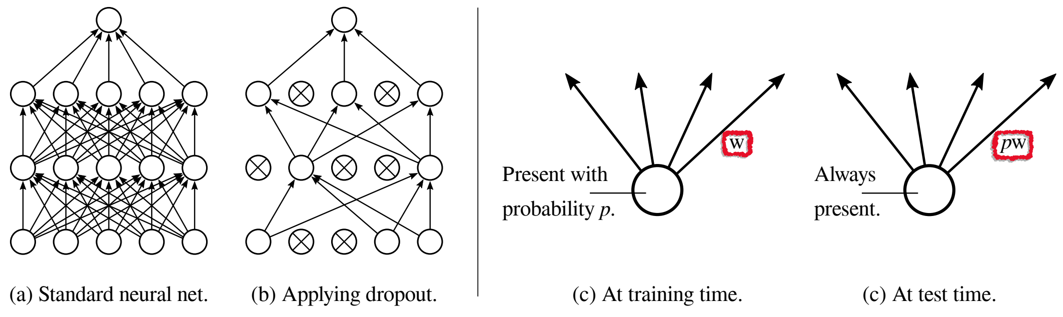


Figure 3.2: Applying dropout to a neural network.

Dropout

Dropout can be considered as one of the biggest recent inventions in the field of neural networks. It is extremely simple and effective technique addressing the problem of overfitting. It can be seen as type of regularization, together with techniques like L1 and L2 regularization, and constraining maximum value of weights.

Dropout works with the idea of "dropping out" some of the unit activations in a layer, that is setting them to zero, during training. This can be interpreted as sampling a neural network from the full neural network, and only updating the parameters of the sampled network for the given data. Visualisation is on Figure 3.2, parts *a* and *b*. Dropout behaves differently during sampling phase – all the units are present, but their outputs are multiplied by the same probability used before for dropping them out. See the part *c* and *d* of Figure 3.2.

This technique should prevent complex co-adaptations, in which unit is only helpful in the context of several other specific units. Each neuron instead learns to detect a feature generally useful for computing the answer.

3.2 Recurrent Neural Nets

Feedforward neural nets are extremely powerful models, but they can be only applied to problems with inputs and outputs of fixed dimensionality. This is a serious drawback, as many of the real-world problems are defined as sequences with lengths that are unknown to us beforehand. Recurrent neural networks were introduced soon after feed-forward

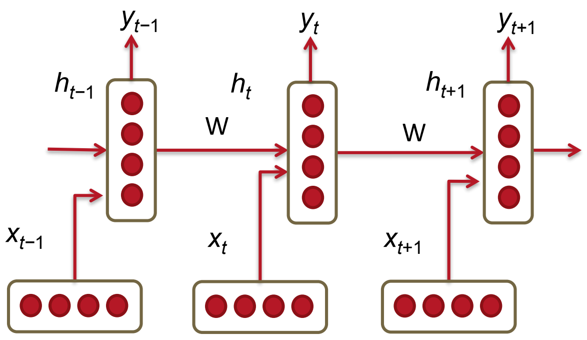


Figure 3.3: A Recurrent Neural Network (RNN). Three time-steps are shown.

nets and they proved to be very useful in this kind of a task. There is vast amount of recurrent neural network types, many not suitable for sequential tasks, like Hopfield networks, which are very successful in what they do, but nevertheless not useful for us now.

Apart from classification, which can be more precise when using sequences, one of the most important tasks is next value prediction. This core task can be then extended very simply to predict arbitrary number of future values. Prediction problems are all around us, from the weather forecast and stock market prediction to the autocomplete in smartphones or web browsers.

Figure 3.3 introduces the RNN architecture where rectangular box is a hidden layer at a time-step, t. Each such layer holds a number of neurons, each of which performing a linear matrix operation on its inputs followed by a non-linear operation (e.g. tanh()). At each time-step, the output of the previous step along with the next word vector in the document, x_t, are inputs to the hidden layer to produce a prediction output $\hat{y}$ and output features h_t (Equations 3.3 and 3.4). The inputs and outputs of each single neuron are illustrated in Figure 3.4.

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}) \tag{3.3}$$

$$\hat{y}_t = softmax(W^{(S)}h_t) \tag{3.4}$$

Below are the details associated with each parameter in the network:

- $x_1, ..., x_{t-1}, x_t, x_{t+1}, ...x_T$: the word vectors corresponding to a corpus with T words.

- $h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$: the relationship to compute the hidden layer output features at each time-step t
  - $x_t \in \mathbb{R}^d$: input word vector at time t.
  - $W^{hx} \in \mathbb{R}^{D_h \times d}$: weights matrix used to condition the input word vector, x_t
  - $W^{hh} \in \mathbb{R}^{D_h \times D_h}$: weights matrix used to condition the output of the previous time-step, h_{t-1}
  - $h_{t-1} \in \mathbb{R}^{D_h}$: output of the non-linear function at the previous time-step, $t-1$. $h_0 \in \mathbb{R}^{D_h}$ is an initialization vector for the hidden layer at time-step $t = 0$.
  - $\sigma()$: the non-linearity function (sigmoid here)
- $\hat{y}_t = softmax(W^{(S)}h_t)$: the output probability distribution over the vocabulary at each time-step t. Essentially, $\hat{y}_t$ is the next predicted word given the document context score so far (i.e. h_{t-1}) and the last observed word vector $x^{(t)}$. Here, $W^{(S)} \in \mathbb{R}^{|V| \times D_h}$ and $\hat{y} \in \mathbb{R}^{|V|}$ where $|V|$ is the vocabulary.

The loss function used in RNNs is often the cross entropy error introduced in earlier notes. Equation 3.5 shows this function as the sum over the entire vocabulary at time-step t.

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times log(\hat{y}_{t,j}) \tag{3.5}$$

The cross entropy error over a corpus of size T is:

$$J = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \times log(\hat{y}_{t,j}) \tag{3.6}$$

Long Short-Term Memory unit and other architectures commonly used in rnns are discussed in following section 3.2.1.

3.2.1 Recurrent architectures

RNNs have many different architectures, however, most of them are derived from the basic fully recurrent network. This network do not have units separated into layers, as each of them has a directed connection to every other unit. Rest of the architectures are

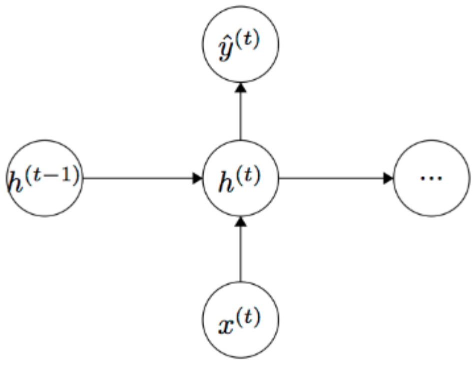


Figure 3.4: The inputs and outputs to a neuron of a RNN

special cases of this one, as they group neurons into layers and implement only a subset of the connections. Examples of these architectures can be Hopfield and Elman networks, and Restricted Boltzmann Machines. Different architectures are trying to connect rnn with an external memory resource, which can be a tape in case of Neural Turing Machines, a stack in Neural network Pushdown Automata, etc. During training rnn unrolling can be applied to these architectures, although training can be quite difficult, as explained earlier.

From here on in, We will focus on an architecture called Long Short-Term Memory and architectures derived from it, as they are very powerful and dominating the current field.These units are carefully designed with the vanishing gradient problem in mind and perform better than most of the other architectures.

3.2.2 Vanishing Gradient & Gradient Explosion Problems

Recurrent neural networks propagate weight matrices from one time-step to the next. Recall the goal of a RNN implementation is to enable propagating context information through faraway time-steps. For example, consider the following two sentences:

"Jane walked into the room. John walked in too. Jane said hi to ___"

"Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to ___"

In both sentences, given their context, one can tell the answer to both blank spots is most likely "John". It is important that the RNN predicts the next word as "John", the second person who has appeared several time-steps back in both contexts. Ideally, this should be possible given what we know about RNNs so far. In practice, however, it turns out RNNs are more likely to correctly predict the blank spot in Sentence 1 than in Sentence 2. This is because during the back-propagation phase, the contribution of gradient values gradually vanishes as they propagate to earlier time-steps. Thus, for long sentences, the probability that "John" would be recognized as the next word reduces with the size of the context. Below, we discuss the mathematical reasoning behind the vanishing gradient problem.

At a time-step t; to compute the RNN error, dE/dW, we sum the error at each time-step. That is, dE_t/dW for every time-step, t, is computed and accumulated.

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \tag{3.7}$$

The error for each time-step is computed through applying the chain rule differentiation to Equations 3.4 and 3.3; Equation 3.8 shows the corresponding differentiation. Notice dh_t/dh_k refers to the partial derivative of h_t with respect to *all* previous k time-steps.

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W} \tag{3.8}$$

Equation 3.9 shows the relationship to compute each dh_t/dh_k; this is simply a chain rule differentiation over all hidden layers within the $[k, t]$ time interval.

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \times diag[f'(j_{j-1})] \tag{3.9}$$

Because $h \in \mathbb{R}^{D_n}$, each $\partial h_j / \partial h_{j-1}$ is the Jacobian matrix for h:

$$\frac{\partial h_j}{\partial h_{j-1}} = [\frac{\partial h_j}{\partial h_{j-1,1}} \cdots \frac{\partial h_j}{\partial h_{j-1,D_n}}] = \begin{bmatrix} \frac{\partial h_{j,1}}{\partial h_{j-1,1}} & . \; . \; . & \frac{\partial h_{j,1}}{\partial h_{j-1,D_n}} \\ . & . & . \\ . & . & . \\ . & . & . \\ \frac{\partial h_{j,D_n}}{\partial h_{j-1,1}} & . \; . \; . & \frac{\partial h_{j,D_n}}{\partial h_{j-1,D_n}} \end{bmatrix} \tag{3.10}$$

Putting Equations 3.7, 3.8, 3.9 together, we have the following relationship.

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}})\frac{\partial h_k}{\partial W} \tag{3.11}$$

Equation 3.12 shows the norm of the Jacobian matrix relationship in Equation 3.10. Here, β_W and β_h represent the upper bound values for the two matrix norms. The norm of the partial gradient at each time-step, t, is therefore, calculated through the relationship shown in Equation 3.12.

$$\| \frac{\partial h_j}{\partial h_{j-1}} \| \leq \| W^T \| \| diag[f'(h_{j-1})] \| \leq \beta_W \beta_h \tag{3.12}$$

The norm of both matrices is calculated through taking their L2-norm. The norm of $f'(h_{j-1})$ can only be as large as 1 given the sigmoid non-linearity function.

$$\| \frac{\partial h_t}{\partial h_k} \| = \| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \| \leq (\beta_W \beta_h)^{t-k} \tag{3.13}$$

The exponential term $(\beta_W \beta_h)^{t-k}$ can easily become a very small or large number when $\beta_W \beta_h$ is much smaller or larger than 1 and $t-k$ is sufficiently large. Recall a large $t-k$ evaluates the cross entropy error due to faraway words. The contribution of faraway words to predicting the next word at time-step t diminishes when the gradient vanishes early on.

During experimentation, once the gradient value grows extremely large, it causes an overflow (i.e. NaN) which is easily detectable at runtime; this issue is called the *Gradient Explosion Problem.* When the gradient value goes to zero, however, it can go undetected while drastically reducing the learning quality of the model for far-away words in the corpus; this issue is called the *Vanishing Gradient Problem.*

3.2.3 Solution to the Exploding & Vanishing Gradients

Now that we gained intuition about the nature of the vanishing gradients problem and how it manifests itself in deep neural networks, let us focus on a simple and practical heuristic to solve these problems.

To solve the problem of exploding gradients, Thomas Mikolov first introduced a simple heuristic solution that *clips* gradients to a small number whenever they explode. That is, whenever they reach a certain threshold, they are set back to a small number as shown in Algorithm 1.

Algorithm 1 Psudo-code for norm clipping in the gradients whenever they explode

```
$\hat{g} \leftarrow \frac{\partial E}{\partial W}$
**if** $\parallel \hat{g} \parallel \geq threshold$ **then**
    $\hat{g} \leftarrow \frac{threshold}{\parallel \hat{g} \parallel} \hat{g}$
**end if**
```

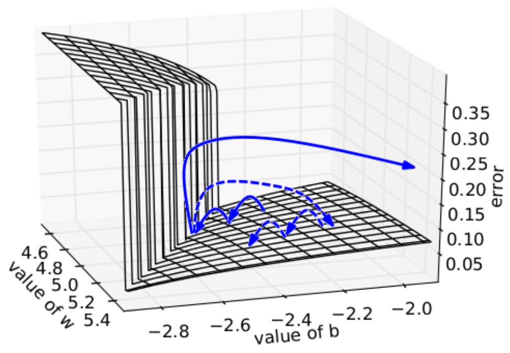


Figure 3.5: Gradient explosion clipping visualization

Figure 3.5 visualizes the effect of gradient clipping. It shows the decision surface of a small recurrent neural network with respect to its W matrix and its bias terms, b. The model consists of a single unit of recurrent neural network running through a small number of time-steps; the solid arrows illustrate the training progress on each gradient descent step. When the gradient descent model hits the high error wall in the objective function, the gradient is pushed off to a far-away location on the decision surface. The clipping model produces the dashed line where it instead pulls back the error gradient to somewhere close to the original gradient landscape.

To solve the problem of vanishing gradients, we introduce two techniques. The first technique is that instead of initializing $W^{(hh)}$ randomly, start off from an identify matrix initialization.

The second technique is to use the Rectified Linear Units (ReLU) instead of the sigmoid function. The derivative for the ReLU is either 0 or 1. This way, gradients would flow through the neurons whose derivative is 1 without getting attenuated while propagating back through time-steps.

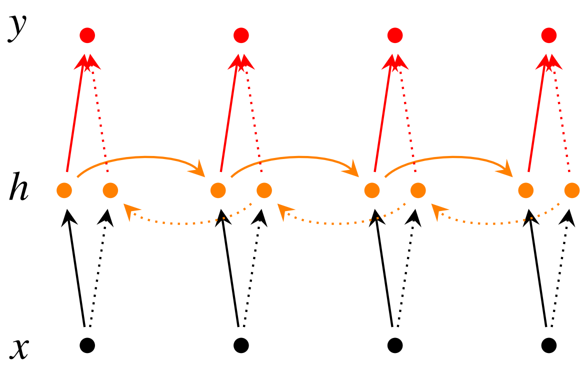


Figure 3.6: A bi-directional RNN model

3.2.4 Deep Bidirectional RNNs

So far, we have focused on RNNs that look into the past words to predict the next word in the sequence. It is possible to make predictions based on future words by having the RNN model read through the corpus backwards. Irsoy et al. shows a bi-directional deep neural network; at each time-step, t, this network maintains two hidden layers, one for the left-to-right propagation and another for the right-to-left propagation. To maintain two hidden layers at any time, this network consumes twice as much memory space for its weight and bias parameters. The final classification result, $\hat{y}_t$, is generated through combining the score results produced by both RNN hidden layers. Figure 3.6 shows the bi-directional network architecture, and Equations 3.14 and 3.15 show the mathematical formulation behind setting up the bi-directional RNN hidden layer. The only difference between these two relationships is in the direction of recursing through the corpus. Equation 3.16 shows the classification relationship used for predicting the next word via summarizing past and future word representations.

$$\overrightarrow{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b}) \tag{3.14}$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b}) \tag{3.15}$$

$$\hat{y}_t = g(Uh_t + c) = g(U[\overrightarrow{h}_t; \overleftarrow{h}_t] + c) \tag{3.16}$$

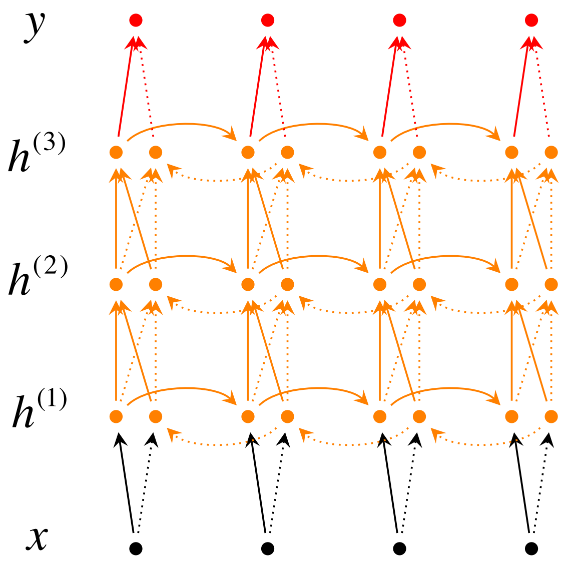


Figure 3.7: A deep bi-directional RNN with three RNN layers.

Figure 3.7 shows a multi-layer bi-directional RNN where each lower layer feeds the next layer. As shown in this figure, in this network architecture, at time-step t each intermediate neuron receives one set of parameters from the previous time-step (in the same RNN layer), and two sets of parameters from the previous RNN hidden layer; one input comes from the left-to-right RNN and the other from the right-to-left RNN.

To construct a Deep RNN with L layers, the above relationships are modified to the relationships in Equations 3.17 and 3.18 where the input to each intermediate neuron at level i is the output of the RNN at layer $i-1$ at the same time-step, t. The output, $\hat{y}$, at each time-step is the result of propagating input parameters through all hidden layers (Equation 3.19).

$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)}) \tag{3.17}$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)}) \tag{3.18}$$

$$\hat{y}_t = g(Uh_t + c) = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c) \tag{3.19}$$

3.3 Gated Recurrent Units

Beyond the extensions discussed so far, RNNs have been found to perform better with the use of more complex units for activation. So far, we have discussed methods that transition from hidden state h_{t-1} to h_t using an affine transformation and a point-wise nonlinearity. Here, we discuss the use of a gated activation function thereby modifying the RNN architecture. What motivates this? Well, although RNNs can theoretically capture long-term dependencies, they are very hard to actually train to do this. Gated recurrent units are designed in a manner to have more persistent memory thereby making it easier for RNNs to capture long-term dependencies. Let us see mathematically how a GRU uses h_{t-1} and x_t to generate the next hidden state h_t. We will then dive into the intuition of this architecture.

$$\begin{aligned}
z_t &= \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) && \text{(Update gate)}\\
r_t &= \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) && \text{(Reset gate)}\\
\tilde{h}_t &= \tanh(r_t \circ Uh_{t-1} + Wx_t) && \text{(New memory)}\\
h_t &= (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1} && \text{(Hidden state)}
\end{aligned}$$

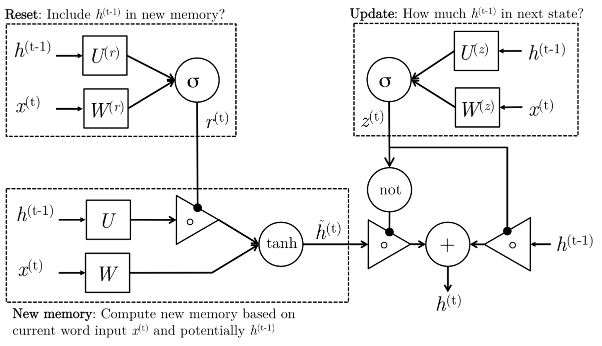
The above equations can be thought of a GRU's four fundamental operational stages and they have intuitive interpretations that make this model much more intellectually satisfying (see Figure 3.8):

1. **New memory generation:** A new memory $\tilde{h}_t$ is the consolidation of a new input word x_t with the past hidden state h_{t-1}. Anthropomorphically, this stage is the one who knows the recipe of combining a newly observed word with the past hidden state h_{t-1} to summarize this new word in light of the contextual past as the vector $\tilde{h}_t$.

2. **Reset Gate:** The reset signal r_t is responsible for determining how important h_{t-1} is to the summarization $\tilde{h}_t$. The reset gate has the ability to completely diminish past hidden state if it finds that h_{t-1} is irrelevant to the computation of the new memory.

3. **Update Gate:** The update signal z_t is responsible for determining how much of h_{t-1} should be carried forward to the next state. For instance, if $z_t \approx 1$, then h_{t-1} is almost entirely copied out to h_t. Conversely, if $z_t \approx 0$, then mostly the new memory $\tilde{h}_t$ is forwarded to the next hidden state.

4. **Hidden state:** The hidden state h_t is finally generated using the past hidden input h_{t-1} and the new memory generated $\tilde{h}_t$ with the advice of the update gate.



Reset: Include $h^{(t-1)}$ in new memory?
$h^{(t-1)}$
$U^{(r)}$
$x^{(t)}$
$W^{(r)}$
σ
$r^{(t)}$
Update: How much $h^{(t-1)}$ in next state?
$U^{(z)}$
$W^{(z)}$
$z^{(t)}$
not
U
W
tanh
$\tilde{h}^{(t)}$
+
$h^{(t)}$
New memory: Compute new memory based on current word input $x^{(t)}$ and potentially $h^{(t-1)}$

Figure 3.8: The detailed internals of a GRU

It is important to note that to train a GRU, we need to learn all the different parameters: $W, U, W^{(r)}, U^{(r)}, W^{(z)}, U^{(z)}$. These follow the same backpropagation procedure we have seen in the past.

3.4 Long-Short-Term-Memories

Long-Short-Term-Memories are another type of complex activation unit that differ a little from GRUs. The motivation for using these is similar to those for GRUs however the architecture of such units does differ. Let us first take a look at the mathematical formulation of LSTM units before diving into the intuition behind this design:

$$\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) && \text{(Input gate)}\\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) && \text{(Forget gate)}\\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) && \text{(Output/Exposure gate)}\\
\tilde{c}_t &= \tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) && \text{(New memory cell)}\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t && \text{(Final memory cell)}\\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}$$

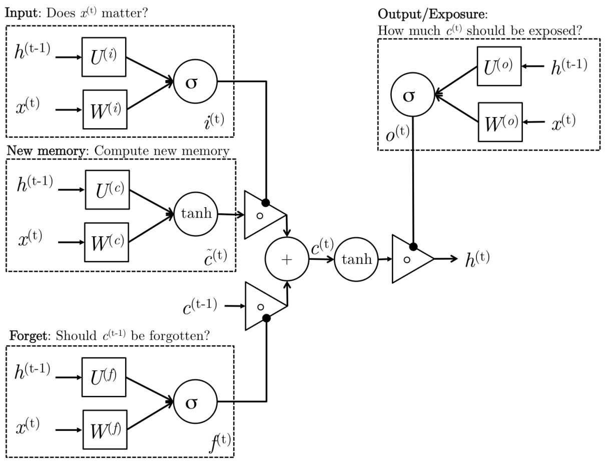


Figure 3.9: The detailed internals of a LSTM

We can gain intuition of the structure of an LSTM by thinking of its architecture as the following stages:

1. **New memory generation:** This stage is analogous to the new memory generation stage we saw in GRUs. We essentially use the input word x_t and the past hidden state h_{t-1} to generate a new memory $\tilde{c}_t$ which includes aspects of the new word $x^{(t)}$.

2. **Input Gate:** We see that the new memory generation stage doesn't check if the new word is even important before generating the new memory – this is exactly the input gate's function. The input gate uses the input word and the past hidden state to determine whether or not the input is worth preserving and thus is used to gate the new memory. It thus produces i_t as an indicator of this information.

3. **Forget Gate:** This gate is similar to the input gate except that it does not make a determination of usefulness of the input word – instead it makes an assessment on whether the past memory cell is useful for the computation of the current memory cell. Thus, the forget gate looks at the input word and the past hidden state and produces f_t.

4. **Final memory generation:** This stage first takes the advice of the forget gate f_t and accordingly forgets the past memory c_{t-1}. Similarly, it takes the advice of the input gate i_t and accordingly gates the new memory $\tilde{c}_t$. It then sums these two results to produce the final memory c_t.

5. **Output/Exposure Gate:** This is a gate that does not explicitly exist in GRUs. It's purpose is to separate the final memory from the hidden state. The final memory c_t contains a lot of information that is not necessarily required to be saved in the hidden state. Hidden states are used in every single gate of an LSTM and thus, this gate makes the assessment regarding what parts of the memory c_t needs to be exposed/present in the hidden state h_t. The signal it produces to indicate this is o_t and this is used to gate the point-wise tanh of the memory.

CHAPTER 4

Proposed Work

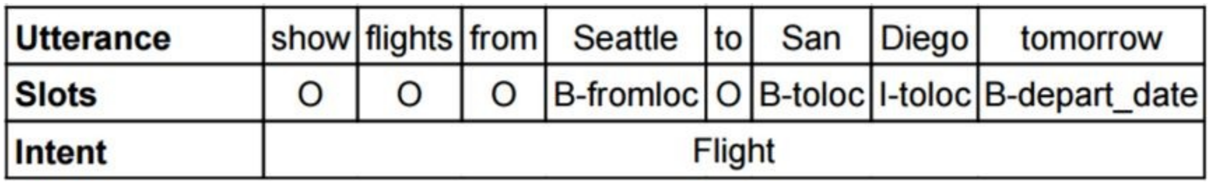
4.1 Background

Intent Detection

Intent detection can be treated as a semantic utterance classification problem, where the input to the classification model is a sequence of words and the output is the speaker intent class. Given an utterance with a sequence of words $w = (w1, w2, ..., wT)$, the goal of intent detection is to assign an intent class c from a pre-defined finite set of intent classes, such that:

$$\tilde{c} = arg \max_{c} P(c|w) \tag{4.1}$$

Recent neural network based intent classification models involve using neural bag-of-words (NBoW) or bag-of-n-grams, where words or ngrams are mapped to high dimensional vector space and then combined component-wise by summation or average before being sent to the classifier. More structured neural network approaches for utterance classification include using recursive neural network (RecNN) (Guo et al., 2014), recurrent neural network (Ravuri and Stolcke, 2015), and convolutional neural network models (Collobert and Weston, 2008; Kim, 2014). Comparing to basic NBoW methods, these models can better capture the structural patterns in the word sequence.



Utterance	show	flights	from	Seattle	to	San	Diego	tomorrow
Slots	O	O	O	B-fromloc	O	B-toloc	I-toloc	B-depart_date
Intent	Flight							

Figure 4.1: ATIS corpus sample with intent and slot annotation

Slot Filling

A major task in spoken language understanding (SLU) is to extract semantic constituents by searching input text to fill in values for predefined slots in a semantic frame (Mesnil et al.,2015), which is often referred to as slot filling. The slot filling task can also be viewed as assigning an appropriate semantic label to each word in the given input text. In the below example from ATIS (Hemphill et al., 1990) corpus following the popular in/out/begin (IOB) annotation method, Seattle and San Diego are the from and to locations respectively according to the slot labels, and tomorrow is the departure date. Other words in the example utterance that carry no semantic meaning are assigned O label.

Given an utterance consisting of a sequence of words $w = (w1, w2, ..., wT)$, the goal of slot filling is to find a sequence of semantic labels $s = (s1, s2, ..., sT)$, one for each word in the utterance, such that:

$$\tilde{s} = arg \max_{s} P(s|w) \tag{4.2}$$

Slot filling is typically treated as a sequence labeling problem. Sequence models including conditional random fields (Raymond and Riccardi, 2007) and RNN models (Yao et al., 2014; Mesnil et al., 2015; Liu and Lane, 2015) are among the most popular methods for sequence labeling tasks.

RNN Language Model

A language model assigns a probability to a sequence of words w = (w1, w2, ..., wT) following probability distribution. In language modeling, w0 and wT +1 are added to the

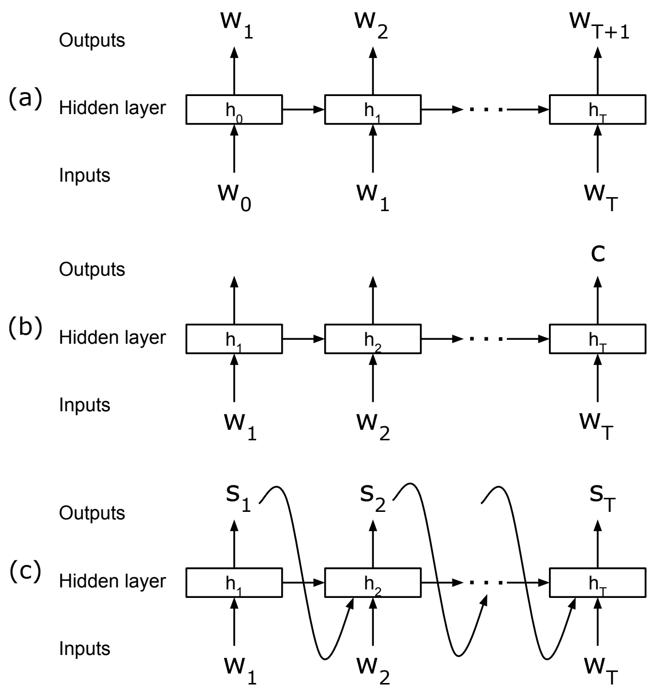


Figure 4.2: (a) RNN language model. (b) RNN intent detection model. The RNN output at last step is used to predict the intent class. (c) RNN slot filling model.

word sequence representing the beginning-of-sentence token and end-of-sentence token. Using the chain rule, the likelihood of a word sequence can be factorized as:

$$P(w) = \sum_{t=1}^{T} P(w_t|w_0, ..., w_{t-1}) \tag{4.3}$$

RNN-based language models (Mikolov et al.,2011), and the variant (Sundermeyer et al., 2012) using long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) have shown superior performance comparing to traditional n-gram based models. In this work, we use an LSTM cell as the basic RNN unit for its stronger capability in capturing long-range dependencies in word sequence.

4.1.1 RNN for Intent Detection and Slot Filling

As illustrated in Figure, RNN intent detection model uses the last RNN output to predict the utterance intent class. This last RNN output can be seen as a representation or embedding of the entire utterance. Alternatively, the utterance embedding can be obtained by taking mean of the RNN outputs over the sequence. This utterance embedding is then used as input to the multinomial logistic regression for the intent class prediction. RNN slot filling model takes word as input and the corresponding slot label as output at each time step. The posterior probability for each slot label is calculated using the softmax function over the RNN output. Slot label dependencies can be modeled by feeding the output label from the previous time step to the current step hidden state (Figure 2(c)). During model training, true label from previous time step can be fed to current hidden state. During inference, only the predicted label can be used. To bridge the gap between training and inference, scheduled sampling method (Bengio et al., 2015) can be applied. Instead of only using previous true label, using sample from previous predicted label distribution in model training makes the model more robust by forcing it to learn to handle its own prediction mistakes (Liu and Lane, 2015).

4.2 Method 1

In this section we describe the joint SLU-LM model in detail.

4.2.1 Model

Let $w = (w0, w1, w2, ..., wT + 1)$ represent the input word sequence, with w0 and wT +1 being the beginning-of-sentence (*bos*) and end-of-sentence (*eos*) tokens. Let $c = (c0, c1, c2, ..., cT)$ be the sequence of intent class outputs at each time step. Similarly, let $s = (s0, s1, s2, ..., sT)$ be the slot label sequence, where s0 is a padded slot label that maps to the beginning-of-sentence token *bos*.

Referring to the joint SLU-LM model shown in Figure , for the intent model, instead of predicting the intent only after seeing the entire utterance as in the independent training intent model, in the joint model we output intent at each time step as input word sequence arrives. The intent generated at the last step is used as the final utterance intent prediction. The intent output from each time step is fed back to the RNN state, and thus the entire intent output history are modeled and can be used as context to other tasks. It is not hard to see that during inference, intent classes that are predicted during the first few time steps are of lower confidence due to the limited information available.

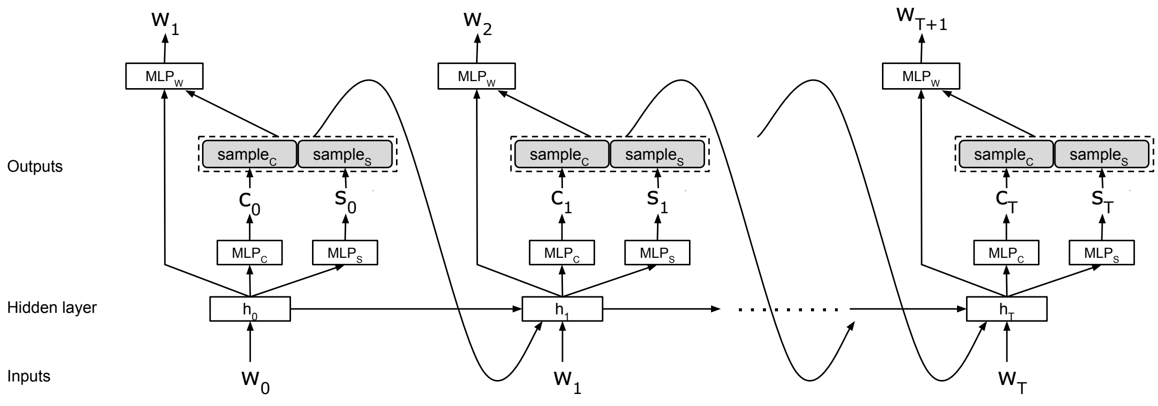


Figure 4.3: Joint online RNN model for intent detection, slot filling, and next word prediction.

We describe the techniques that can be used to ameliorate this effect in section 3.3 below. For the intent model, with both intent and slot label connections to the RNN state, we have:

$$P(c_T|w) = P(c_T|w \leq T, c < T, s < T) \tag{4.4}$$

For the slot filling model, at each step t along the input word sequence, we want to model the slot label output st as a conditional distribution over the previous intents $c < t$, previous slot labels $s < t$, and the input word sequence up to step t. Using the chain rule, we have:

$$P(s|w) = P(s0|w0) \sum_{t=1}^{T} P(s_t|wt, c < t, s < t) \tag{4.5}$$

4.2.2 Next Step Prediction

Following the model architecture in Figure 3, at time step t, input to the system is the word at index t of the utterance, and outputs are the intent class, the slot label, and the next word prediction. The RNN state ht encodes the information of all the words, intents, and slot labels seen previously. The neural network model computes the outputs through the following sequence of steps:

$$h_t = LSTM(h_{t1}, [w_t, c_{t1}, s_{t1}]) \tag{4.6}$$

$$P(c_t|wt, c < t, s < t) = IntentDist(h_t) \tag{4.7}$$

$$P(s_t|wt, c < t, s < t) = SlotLabelDist(h_t) \tag{4.8}$$

where LSTM is the recurrent neural network function that computes the hidden state h_t at a step using the previous hidden state h_{t1}, the embeddings of the previous intent output c_{t1} and slot label output s_{t1}, and the embedding of current input word w_t. IntentDist, SlotLabelDist are multilayer perceptrons (MLPs) with softmax outputs over intents, slot labels, and words respectively. Each of these three MLPs has its own set of parameters. The intent and slot label distributions are generated by the MLPs with input being the RNN cell output. The next word distribution is produced by conditioning on current step RNN cell output together with the embeddings of the sampled intent and sampled slot label.

4.3 Method 2

4.3.1 Model

The idea of introducing attention to the alignment-based RNN sequence labeling model is motivated by the use of attention mechanism in encoder-decoder models. In bidirectional RNN for sequence labeling, the hidden state at each time step carries information of the whole sequence, but information may gradually lose along the forward and backward propagation. Thus, when making slot label prediction, instead of only utilizing the aligned hidden state hi at each step, we would like to see whether the use of context vector ci gives us any additional supporting information, especially those require longer term dependencies that is not being fully captured by the hidden state. In the proposed model, a bidirectional RNN (BiRNN) reads the source sequence in both forward and backward directions. We use LSTM cell for the basic RNN unit. Slot label dependencies are modeled in the forward RNN. Similar to the encoder module in the above described encoder-decoder architecture, the hidden state hi at each step is a concatenation of the

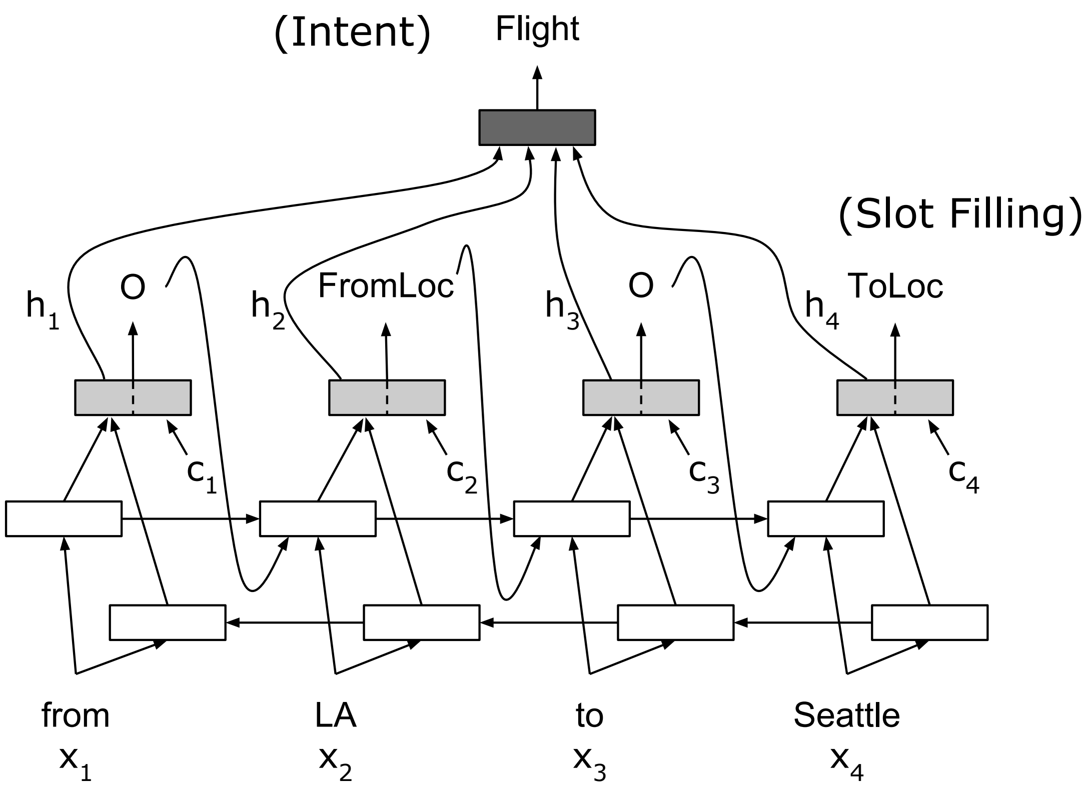


Figure 4.4: Attention-based RNN model for joint intent detection and slot filling. The bidirectional RNN reads the source sequence forward and backward. Slot label dependency is modeled in the forward RNN. At each time step, the concatenated forward and backward hidden states is used to predict the slot label. If attention is enabled, the context vector ci provides information from parts of the input sequence that is used together with the time aligned hidden state hi for slot label prediction.

forward state fhi and backward state bhi, hi = [fhi, bhi]. Each hidden state hi contains information of the whole input word sequence, with strong focus on the parts surrounding the word at step i. This hidden state hi is then combined with the context vector ci to produce the label distribution, where the context vector ci is calculated as a weighted average of the RNN hidden states h = (h1, ..., hT).

For joint modeling of intent detection and slot filling, we reuse the pre-computed hidden states h of the bidirectional RNN to produce intent class distribution. If attention is not used, we apply mean-pooling [17] over time on the hidden states h followed by logistic regression to perform the intent classification. If attention is enabled, we instead take the weighted average of the hidden states h over time. Comparing to the attention-based encoder-decoder model that utilizes explicit aligned inputs, the attention-based RNN model is more computational efficient. During model training, the encoder-decoder slot filling model reads through the input sequence twice, while the attention-based RNN model reads through the input sequence only once.

4.3.2 Objective Functions

Cross entropy

Most approaches use a logistic regression classifier with the softmax activation function in the final layer. The objective function which is mainly used in this case is based on cross entropy:

$$L = \sum_{c} y_c log(s_\theta(x)c)(4) \tag{4.9}$$

In this equation, c iterates over all classes, y_c is the correct value for class c and s is the score the network assigned to class c given the current data point x.

Ranking

Instead of using the softmax activation function, we train a matrix W_{class} whose columns contain vector representation of the different classes. Therefore, the score for each class c can be computed by using the product

$$s_\theta(x)c = h_t^x[W_{class}]c \tag{4.10}$$

We use a ranking loss function to train the RNN. It learns to maximize the distance between the true label y+ and the best competitive label c- given a data point x. The objective function is

$$L = log(1 + exp(\gamma(m + -s_\theta(x)y+))) + log(1 + exp(\gamma(m - +s_\theta(x)c))) \tag{4.11}$$

This function was proposed by Dos Santos et al. [23] to train convolution neural networks for relation classification. The parameter γ controls the penalization of the prediction errors and m+ and m- are margins for the correct and incorrect classes. γ, m+ and m- are hyperparameters which can be tuned on the development set. For the class O, we only calculate the second summand of equation. By doing this, we do not learn a pattern for class O but nevertheless increase its difference to the best competitive label.

During testing, the model will predict class O if the score for all the other classes is lower than 0. One of the advantages of this loss function over the softmax function is efficiency. Since only two classes are computed at every training iteration, the network can be trained quite fast even with a large number of classes. Furthermore, a ranking loss function is suitable for tasks like slot filling because it does not force the network to learn a pattern for the O class which in fact may not exist.

4.3.3 Training

The network is trained to find the parameters θ that minimise the cross-entropy or ranking of the predicted and true distributions for intent class, slot label, and next word jointly. The objective function also includes an L2 regularization term $R(\theta)$ over the weights and biases of the three MLPs. This equalizes to finding the parameters θ that maximize the below objective function:

$$\max_\theta \sum_{t=1}^{T} \alpha_c \log P(c^*|w \leq t, c < t, s < t; \theta) + \alpha_s \log P(s_t^*|w \leq t, c < t, s < t; \theta) - \lambda R(\theta) \tag{4.12}$$

where c* is the true intent class and and s* is the true slot label at time step t. α_c and α_s are the linear interpolation weights for the true intent, slot label. During model training, c_t can either be the true intent or mixture of true and predicted intent. During

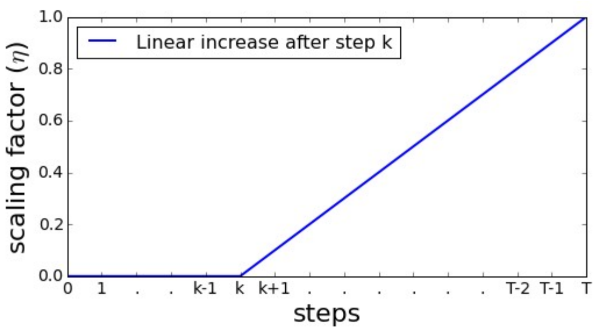


Figure 4.5: Schedule of increasing intent contribution to the context vector along with the growing input sequence

inference, however, only predicted intent can be used. Confidence of the predicted intent during the first few time steps is likely to be low due to the limited information available, and the confidence level is likely to increase with the newly arriving words. Conditioning on incorrect intent for next word prediction is not desirable. To mitigate this effect, we propose to use a schedule to increase the intent contribution to the context vector along the growing input word sequence. Specifically, during the first k time steps, we disable the intent context completely by setting the values in the intent vector to zeros. From step k + 1 till the last step of the input word sequence, we gradually increase the intent context by applying a linearly growing scaling factor η from 0 to 1 to the intent vector.

4.3.4 Inference

For online inference, we simply take the greedy path of our conditional model without doing search. The model emits best intent class and slot label at each time step conditioning on all previous emitted symbols:

$$\tilde{c}_t = arg \max_{c_t} P(c_t | w \leq t, \tilde{c} < t, \tilde{s} < t) \tag{4.13}$$

$$\tilde{s}_t = arg \max_{s_t} P(s_t | w \leq t, \tilde{c} < t, \tilde{c} < t) \tag{4.14}$$

Many applications can benefit from this greedy inference approach comparing to search based inference methods, especially those running on embedded platforms that without GPUs and with limited computational capacity. Alternatively, one can do left-to-right beam search (Sutskever et al., 2014; Chan et al., 2015) by maintaining a set of β best partial hypotheses at each step. Efficient beam search method for the joint conditional model is left to explore in our future work.

CHAPTER 5

Results and Discussion

5.1 Data

To compare our work with previously studied methods, we report results on the widely used ATIS dataset [24, 25]. This dataset is from the air travel domain and consists of audio recordings of speakers making travel reservations. All the words are labeled with a semantic label in a BIO format (B: begin, I: inside, O: outside), e.g. New York contains two words New and York and is therefore labeled with B-fromloc.city name and I-fromloc.city name respectively. Words which do not have semantic labels are tagged with O. In total, the number of semantic labels is 127, including the label of the class O. The training data consists of 4,978 sentences and 56,590 words. The test set contains 893 sentences and 9,198 words. To evaluate our models, we used the script provided in the text chunking CoNLL shared task 20001 in line with other related work.

5.2 Evaluation Metrics

The most commonly used metrics for Intent Detection(ID) and Slot Filling(SF) are class (or slot) error rate (ER) and F-Measure. The simpler metric ER for ID can be computed as:

$$ER_{ID} = \frac{\#\text{ misclassified utterances}}{\#\text{ utterances}} \tag{5.1}$$

Note that one utterance can have more than one intent. A typical example is ”Can you tell me my balance? I need to make a transfer.” In most cases, where the second intent is generic (a greeting, small talk with the human agent) or vague, it is ignored. If none of the true classes is selected, it is counted as a misclassification.

For SF, the error rate can be computed in two ways: The more common metric is the F-measure using the slots as units. This metric is similar to what is being used for other sequence classification tasks in the natural language processing community, such as parsing and named entity extraction. In this technique, usually the IOB schema is adopted, where each of the words are tagged with their position in the slot: beginning (B), in (I) or other (O). Then, recall and precision values are computed for each of the slots. A slot is considered to be correct if its range and type are correct. The F-Measure is defined as the harmonic mean of recall and precision:

$$F1 - Measure = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{5.2}$$

$$Recall = \frac{\text{\# correct slots found}}{\text{\# true slots}} \tag{5.3}$$

$$Precision = \frac{\text{\# correct slots found}}{\text{\# found slots}} \tag{5.4}$$

5.3 Model Training

We used LSTM cell as the basic RNN unit, following the LSTM design in (Zaremba et al., 2014). The default forget gate bias was set to 1. We used single layer uni-directional LSTM in the proposed joint online SLU-LM model. Deeper models by stacking the LSTM layers are to be explored in future work. Word embeddings of size 300 were randomly initialized and fine-tuned during model training. We conducted mini-batch training (with batch size 16) using Adam optimization method following the suggested parameter setup in (Kingma and Ba, 2014). Maximum norm for gradient clipping was set to 5. During model training, we applied dropout (dropout rate 0.5) to the non-recurrent connections (Zaremba et al., 2014) of RNN and the hidden layers of MLPs, and applied L2 regularization ($\gamma = 10^{-4}$) on the parameters of MLPs.

5.4 Results

Table 1 summarises the result our method and its variations on ATIS dataset. The basic

Model	Intent Error	F1 Score
RecNN (Guo et al., 2014)	4.60	93.22
RecNN+Viterbi (Guo et al., 2014)	4.60	93.96
Independent training RNN intent model	2.13	-
Independent training RNN slot filling model	-	94.91
Basic joint training model	2.02	94.15
Joint model with recurrent intent context	1.90	94.16
Joint model with recurrent slot label context	1.79	94.64
Joint model with recurrent intent + slot label context	1.57	94.47
Independent Attention BiRNN	-	95.4
Joint Attention BiRNN	1.60	95.54
Joint Attention BiRNN (with Ranking Loss)	1.61	95.75

Table 5.1: ATIS Test set results on intent detection error, slot filling F1 score, and language modeling perplexity. Related joint models: RecNN: Joint intent detection and slot filling model using recursive neural network (Guo et al., 2014). RecNN+Viterbi: Joint intent detection and slot filling model using recursive neural network with Viterbi sequence optimization for slot filling (Guo et al., 2014).

joint model uses a shared representation for all the three tasks. It gives slightly better performance on intent detection, with some degradation on slot filling F1 score. If the RNN output h_t is connected to each task output directly via linear projection without using MLP, performance drops for intent classification and slot filling. Thus, we believe the extra discriminative power introduced by the additional model parameters and non-linearity from MLP is useful for the joint model.

As can be seen from the results, the joint model that utilizes two types of recurrent context maintains the benefits of both, namely, the benefit of applying recurrent intent context to intent detection, and the benefit of applying recurrent slot label context to slot filling. Another observation is that once recurrent context is applied, the benefit of adding local context for next word prediction is limited. It might hint that the most useful information for the next word prediction can be well captured in the RNN state, and thus adding explicit dependencies on local intent class and slot label is not very helpful.

For the attention-based bidirectional RNN architecture, the joint training model achieves 0.23% absolute gain on slot filling over the independent training models. The attention-based RNN model seems to benefit more from the joint training. Results from both of our joint training approaches outperform the best reported joint modeling results. Instead of applying a softmax layer and training the network with the cross entropy loss function, we also used the ranking loss function. The hyper-parameters of the ranking loss function are optimized with a 5-fold cross validation. The best parameters are: γ = 2, m+ = 3, m- = 0.5. We obtain 0.21% absolute improvement compared to the cross entropy loss function

5.5 Error Analysis

5.5.1 Intent Detection Errors

The problem is mostly the non-Flight utterances erroneously classified as Flight. While one cause of these errors is the unbalanced intent distribution, we have manually checked each error and clustered them into 6 categories:

1. **Prepositional phrases embedded in noun phrases:** These errors involve phrases such as Capacity of the flight from Boston to Orlando, where the prepositional phrase suggests flight information, whereas the destination category is mainly determined by the head word of the noun phrase (capacity in this case). Since classifier has no syntactic features, such sentences are usually classified erroneously. Using features from a syntactic parser can alleviate this problem.

2. **Wrong functional arguments of utterances:** This category is similar to the first category but the difference is that, instead of a prepositional phrase, the confused phrase is a semantic argument of the utterance. Consider the example utterance What day of the week does the flight from Boston to Orlando fly? These are errors that can be solved by using either a syntactic parser that identifies functions of

Correct-Estimated	a	b	b	d	e	f	g	h	i	j	k	l	m	n	o	p	q
a. Abbreviation	30									2							
b. Aircraft		6								3							
c. Airfare			64							1							
d. Airline				37						2							
e. Airport					15					2					1		
f. Capacity	1	5				13				2							
g. City							3			2							
h. Day Name										2							
i. Distance									9	1							
j. Flight		1	1				1			623							
k. Flight No										2	6						
l. Flight Time										1							
m. Ground Fare			1										3	3			
n. Ground Service														36			
o. Meal										5							
p. Quantity																8	
q. Restriction	1																

Table 5.2: Confusion Matrix for Intent Detection

phrases or a semantic role labeler.

3. **Annotation errors:** These are utterances that were assigned the wrong category during manual annotation.

4. **Utterances with multiple sentences:** These are utterances with more than one sentence. In such cases, the intent is usually in the last sentence, whereas the classification output is biased by the other sentence.

5. **Other:** These include several infrequent error types such as ambiguous utterances, ill-formulated queries, and preprocessing/tokenization issues:

- Ambiguous utterances: These errors involve utterances where the destination category is not clear in the utterance. An example from the ATIS test set is list Los Angeles. In this utterance, the speaker intent could either be to find cities that have flights from Los Angeles or flights to Los Angeles.
- Ill-formulated queries: These are utterances which include a phrase that may mislead the classification or understanding. An example from the ATIS test set is: Whats the airfare for a taxi to the Denver airport? In this case, the word airfare implies a destination category of Airfare, whereas what is meant is Ground transportation fare. These type of errors are easier for humans to handle, but it is not presently clear how they can be resolved in automatic processing.
- Preprocessing/Tokenization issues: These are errors that could be resolved by using a domain ontology or special pre-processing or tokenization related to the domain. Some domain specific abbreviations and restriction codes are examples of this category.

6. **Difficult Cases:** These are utterances that include words or phrases that were previously unseen in the training data. For the example utterance Are snack served on Tower Air?, none of the content words and phrases appear with the Meal category in the training data.

5.5.2 Slot Filling Errors

Analyzing the SF decisions, the model found 2,614 of 2,837 slots with the correct type and span for the input out of 9,164 words. We manually checked each of the 223 erroneous cases and clustered them into 8 categories:

1. **Long distance dependencies:** These are slots where the disambiguating tokens are out of the current n-gram context. For example, in the utterance Find flights to New York arriving in no later than next Saturday, a 6-gram context is required to resolve that Saturday is the arrival date.

2. **Partially correct slot value annotations:** These are slots assigned a category that is partially correct; either the category or the sub-category matches the manual annotation. For example, the word tomorrow can either be a Depart Date.Relative or Arrive Date.Relative for the utterance flights arriving in Boston tomorrow. Note that these can overlap with other error types.

3. **Previously unseen sequences:** While this category requires further analysis, the most common reason is the mismatch between the training and test sets. For example, meal related slots are missed by the model (8.0% of all errors) because there are no similar cases in the training set. This is also the case for the aircraft models (10.0%), and traveling to states instead of cities (3.3%), etc.

4. **Annotation errors:** These are the slots that were assigned the wrong category during manual annotation.

5. **Other:** These include several infrequent error types such as ambiguous utterances, ill-formulated queries, and preprocessing/tokenization issues:

   - Ill-formulated queries: These errors usually involve an ungrammatical phrase that may mislead the interpretation of the slot value or there is insufficient context to disambiguate the value of the slot. For example, in the utterance Find a flight from Memphis to Tacoma dinner, it is not clear if the word dinner refers to the description of the flight meal.
   - Ambiguous utterances: These are utterances where the slot category is not explicit given the utterance. For example, in the utterance I would like to have the airline that flies Toronto, Detroit and Orlando, it is not clear if the speaker is searching for airlines that have flights from Toronto to Detroit and Orlando or from some other location to Toronto, Detroit and Orlando.
   - Preprocessing/Tokenization issues: These are errors that could be resolved using a domain ontology or special pre-processing or tokenization related to the domain. For example, in the utterance What airline is AS, it would be helpful to know AS is a domain specific abbreviation.

- Ambiguous part-of-speech tag-related errors: These are errors that could be resolved if the part-of-speech tags were resolved. For example, the word arriving can be a verb or an adjective, as in the utterance I want to find the earliest arriving flight to Boston. In this case, the slot category for the words earliest arriving is Flight-Mod, but since the word arriving is very frequently seen as a verb in this corpus, it is assigned no slot category.

CHAPTER 6

Conclusion

Leveraging recent improvements in machine learning and spoken language processing, the performance of the SLU systems for the ATIS domain has improved dramatically. Around 5% error rate for the SLU task implies a solved problem. It is clear, however, that the problem of SLU is far from being solved, especially for more realistic, naturally-spoken utterances of a variety of speakers from tasks more complex than simple flight information requests.

In this work, we studied using RNNs for SLU slot filling task, with particular attention on modeling output sequence dependencies. Modeling structured output is vital to many sequence learning tasks. We propose a conditional RNN model that can be used to jointly perform online spoken language understanding. We show that by continuously modeling intent variation and slot label dependencies along with the arrival of new words, the joint training model achieves advantageous performance in intent detection and language modeling with slight degradation on slot filling comparing to the independent training models.

We explored strategies in utilizing explicit alignment information in the attention-based encoder-decoder neural network models. We further proposed an attention-based bidirectional RNN model for joint intent detection and slot filling. Using a joint model for the two SLU tasks simplifies the dialog system, as only one model needs to be trained and deployed. Our independent training models achieved state-of-the-art performance for both intent detection and slot filling on the benchmark ATIS task. The proposed joint training models improved the intent detection accuracy and slot filling F1 score further over the independent training models.

Even with such low error rates, the ATIS test set includes many example categories and sequences unseen in the training data, and the error rates have not converged yet. In that respect, more data from just the ATIS domain may be useful for SLU research. The error analysis on the ATIS domain shows the primary weaknesses of the current modeling approaches: The local context overrides the global, the model has no domain knowledge to make any inferences, and it tries to fit any utterance into some known sample, hence not really robust to any out-of-domain utterances. One possible research direction consists of employing longer distance syntactically or semantically motivated features, while preserving the robustness of the system to the noise introduced by the speech recognizer and variance due to natural language.

Bibliography

[1] Woods WA. *Language processing for speech understanding.* NJ, 1983.

[2] Brown M Fisher W Hunicke-Smith K Pallett D Pao C Rudnicky A Dahl DA, Bates M and Shriberg E. Expanding the scope of the atis task: the atis-3 corpus. In *Proceedings of the Human Language TechnologyWorkshop.* 1994.

[3] M Walker, J Aberdeen, J Boland, E Bratt, J Garofolo, L Hirschman, Lee S Le_A, S Narayanan, K Papineni, Polifroni J PellomB, A Potamianos, P Prabhu, Rudnicky A, and Sanders G. Darpa communicator dialog travelplanning systems: The june 2000 data collection. In *Proceedings of the Eurospeech Conference.* 2001.

[4] Walker MA, A Rudnicky, R Prasad, J Aberdeen, Bratt EO, J Garofolo, H Hastie, Pellom B Le_A, A Potamianos, R Passonneau, S Roukos, G S, S Seneff, and D Stallard. Darpa communicator: cross-system results for the 2001 evaluation. In *Proceedings of the International Conference on Spoken Language Processing,*, pages 269–272. 2002.

[5] Allen JF, Miller BW, Ringger EK, and T Sikorski. A robust system for natural spoken dialogue. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics,*, pages 62–70. 1996.

[6] S Seneff. Tina: A natural language system for spoken language applications. *Computational Linguistics*, 1992.

[7] J Dowding, Gawron JM, D Appelt, J Bear, L Cherny, R Moore, and D Moran. Gemini: A natural languagesystem for spoken-language understanding. In *Proceedings of the 31st Annual Meeting of the Association forComputational Linguistics,*, pages 54–61. Columbus, Ohio, 1993.

[8] H Alshawi. *The Core Language Engine.* MIT Press,, Cambridge, MA, USA, 1992.

[9] W Ward. Understanding spontaneous speech: the phoenix system. In *Proceedings of the IEEE InternationalConference on Acoustics, Speech, and Signal Processing,*, pages 365–367. Toronto, Canada, 1991.

[10] S Miller, R Bobrow, R Ingria, and R Schwartz. Hidden understanding models of natural language. In *Proceedingsof the 31st Annual Meeting of the Association for Computational Linguistics,*. New, 1994.

[11] R Pieraccini and E Levin. A learning approach to natural language understanding. In *NATO ASI SummerSchool New Advances and Trends in Speech Recognition and Coding.* Springer-Verlag,, Bubion, Spain, 1993.

[12] Della S Pietra, M Epstein, S Roukos, and T Ward. Fertility models for statistical natural language. In *understandingProceedings of the 35th Annual Meeting of the Association for Computational Linguistics,*, pages 168–173. Madrid,Spain, 1997.

[13] Wang YY and Acero A. Combination of cfg and n-gram modeling in semantic grammar learning. In *Proceedings of the Eurospeech Conference,*, pages 2809–2812. 2003.

[14] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014.

[15] F Lef‘evre. Dynamic bayesian networks and discriminative classifiers for multi-stage semantic interpretation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,*, pages 13–16. 2007.

[16] C Raymond and G Riccardi. Generative and discriminative algorithms for spoken language understanding. In *Proceedings of INTERSPEECH*, page 16051608. 2007.

[17] J Bilmes and G Zweig. The graphical models toolkit: An open source software

system for speech and timeseriesprocessing. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,pp*, pages 3916–3919. 2002.

[18] G Riccardi, R Pieraccini, and E Bocchieri. Stochastic automata for language modeling. *Computer Speech andLanguage*, pages 265–293, 1996.

[19] Wang YY and Acero A. Rapid development of spoken language understanding grammars. In *Speech Communication*, pages 390–416. 2006.

[20] Dilek Hakkani-Tr, Gokhan Tur, Asli Celikyilmaz, Yun-Nung Vivian Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. In *INTERSPEECH 2016*. ISCA, 2016.

[21] P. Xu and R. Sarikaya. Convolutional neural network based triangular CRF for joint detection and slot filling In *ASRU*. 2013.

[22] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.