# A study on integrating distinct classifiers with bidirectional LSTM for Slot Filling task

**DO KHAC PHONG**

School of Information Science
Japan Advanced Institute of Science and Technology
March, 2017

Master's Thesis

# A study on integrating distinct classifiers with bidirectional LSTM for Slot Filling task


**1410220 DO KHAC PHONG**

Supervisor : Associate Professor Nguyen Le Minh

Main Examiner : Associate Professor Nguyen Le Minh

Examiners : Professor Satoshi Tojo

Associate Professor Kiyoaki Shirai

School of Information Science
Japan Advanced Institute of Science and Technology



February, 2017

# Abstract

In spite of being investigated for decades, slot filling task in Spoken Language Understanding has been still challenging and attractive to many researchers. This task is simply perceived as sequential labeling in a specific domain. Previously, features fed into a classifier such as Support Vector Machines (SVMs) or Conditional Random Fields (CRFs), are generated manually, which is relatively expensive and time-consuming. Fortunately, that costly work can be solved with a help of deep learning technique which is able to generate and extract potential features automatically during the training period. For sequential data, Recurrent Neural Network (RNNs) are popular models in order to learn latent representations of data, which are then utilized as input to a classifier *Softmax*.

Our proposed model, in contrast, employed a variant of RNNs, called Long Short-Tem Memory Networks (LSTMs) which, more or less, tackle the downside of RNNs: vanishing gradients. Additionally, apart from using Softmax for classification, we also experimented integrations of LSTMs and other distinct classifiers, e.g. CRFs and SVMs, which are all trained simultaneously in our model. The experimental results show that these combinations are beneficial and worthing on both dataset Airline Travel Information System (ATIS) and DARPA Communicator, compared with the state-of-the-art model.

**Keywords:** LSTMs, CRFs, word embeddings, SVMs

### Declaration

I hereby declare that this whole dissertation is my own work, and that it has not been previously included any other thesis, dissertation or report.
**Student:** Do Khac Phong

# Contents

This dissertation was prepared according to the curriculum for the Collaborative Education Program organized by Japan Advanced Institute of Science and Technology (JAIST) and University of Engineering and Technology, Vietnam National University, Hanoi (VNU-UET).

# List of Figures

# List of Tables

# Acknowledgements

I am forever grateful for the chance of joining Nguyen Laboratory, collaborating and working with many excellent Master and PhD students who have helped me grow up intellectually, and professionally. I would like to express my deepest gratitude to my supervisor, A/Prof. Minh Le Nguyen for teaching me a great deal of research, and supporting me whenever my life in Japan seemed to be hard to cope with. His sharp view, critical thinking, and enthusiastic guidance are influential and keep me on the right track of my research.

My special thanks go to my second supervisor A/Prof. Hirokazu Tanaka for his insightful advice in a series of seminars which encourage me to make clearly my statement and my work. I would like to express my gratitude to Prof. Jianwu Dang, for his tremendous financial support during my first days in Japan.

I would also like to give a big thank to A/Prof. Ha Thanh Le with numerous valuable instructions and advice during the first-year-master period in Vietnam. My sincere thanks to Prof. Satoshi Tojo and A/Prof. Kiyoaki Shirai as committee members of my midterm and final defenses. Their helpful questions and comments help me greatly clarify and improve my research.

This thesis is dedicated to my family for their love, support, and encouragement wherever I am.

# Chapter 1

# Introduction

## 1.1 Research motivation

Unlike automatic speech recognition (ASR), which converts a speaker's spoken utterance into a text string, spoken language understanding (SLU) aims to extract the meaning of the speech utterances. In other words, SLU is the interpretation of signs conveyed by a speech signal, which, along with natural language understanding (NLU), shares the goal of gaining a conceptual representation of natural language sentences [7], [29]. In contrast to NLU that has been studied for the period of 60 years for general domains, SLU concentrates only on particular application domains like airline travel, tourist, restaurant, or so. An example of an on-line shopping system is shown in Figure 1.1 with three principal steps:

- Record a user's voice and convert it into text
- Process text and extract semantics
- Search in database and give answers to a user

In this case study, nonetheless, we concentrate mainly on the second step with respect to text processing and semantic extraction. For a specific domain, myriads of constraints could be integrated into the understanding model [29] with the intention that the machine could perceive a user's purposes easier. Nevertheless, this issue is till complicated and challenging due to the fact that spoken utterances often do not follow rigid grammars of a language. It is obvious that repetitions, hesitations and other irregular phenomena are inevitable in a real conversation. That is the main reason why the spoken language is much more difficult to cope with than the written one.

In general, there are three vital tasks in SLU with respect to semantic extraction of input utterances namely domain detection, intent determination, and slot filling. Typically, whilst the two formers is treated as semantic utterance classification problem, the latter is considered as a sequence classification task in order to assign a semantic concept to each word in a sentence, e.g. person, city, etc. For instance, in the sentence *"I want to fly from Boston to New York today"*, the word *Boston*

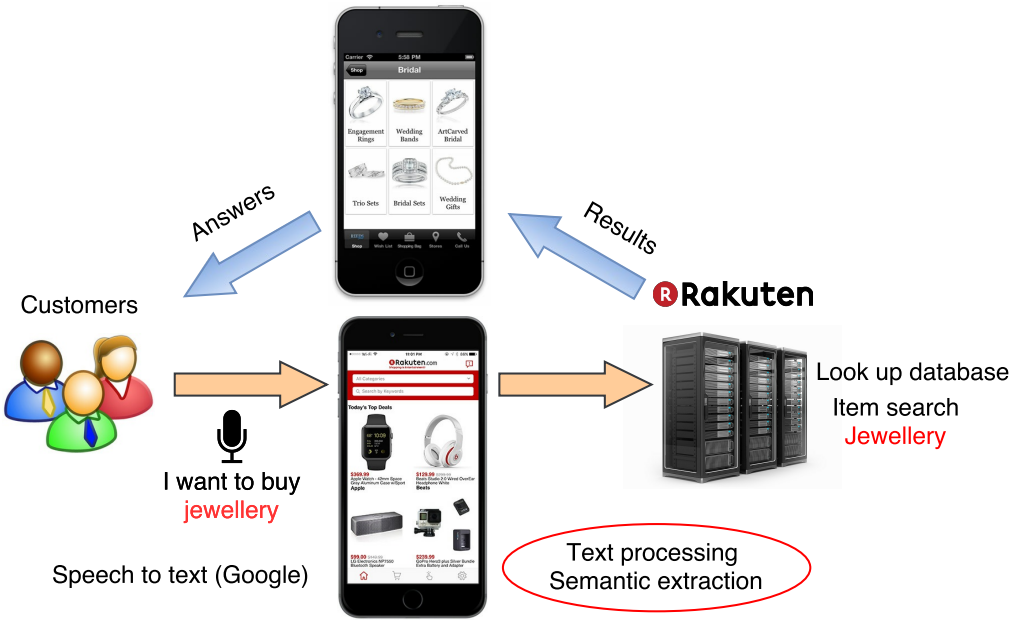



Figure 1.1: An on-line shopping system

should be labeled as the departure-city of a trip, and *New York* as the arrival-city. All the other words, which do not correspond to real slots, are then tagged with a dummy class *O*. For this study, we only take into account slot filling task (SL).

Although slot filling in SLU has been researched for many years, it is still a challenging issue. Fundamental approaches to dealing with slot filling task include generative models such as hidden vector state (HVS) model [10], or discriminative model like Conditional Random Fields (CRFs) [20] and support vector machines (SVMs) [24]. However, the downside of those techniques is that features fed into models need to be defined and prepared in advance. This manual process is relatively expensive and time-consuming. Fortunately, deep learning in recent year can solve this problem due to its capacity to drive new features from a limited set of features contained in a training set. Deep learning with a bunch of variants releases outstanding performance in a wide range of applications ranging from speech recognition [9], computer vision [13], [14] to Natural Language Processing [16], [26], [28]. Following the success of deep learning method, Yao et al. [31] and Mensil et al. [20] investigated in Recurrent Neural Networks (RNNs) and its combination with CRF (R-CRF) which is the state-of-the-art model for SL. In particular, the output of RNN is the input of CRF, which aims to (1) generate automatically features for CRF instead of feature engineering, and (2) make tight constraints among output labels. Nevertheless, in practice, RNNs cannot afford to capture long-term dependencies because the gradients tend to either explode or vanish.

To deal with those daunting problems, in our experiment, a more sophisticated model, Long Short-Term Memory Networks (LSTMs) [11] is utilized for sequence representation. Apart from CRFs merged with LSTM, we also pay our attention to the incorporation of other classifiers to evaluate their potential for SL task. On both

distinct datasets: Airline Travel Information System (ATIS) and DARPA Communicator, our model overcomes the state-of-the-art approach for SL task.

## 1.2 Contributions

This research has two great contributions, including:

First of all, originally DARPA dataset consists of myriads pairs of sentence transcription and abstract semantic annotation. This format makes it difficult for the sequential classification problem. Consequently, we have mapped it into a more convenient format IOB (in/out/begin) for further processes by looking up a database that goes along with DARPA dataset.

Second, the architecture we constructed contains three primary layers: word embeddings, recurrent neural networks, and classification. The word embedding layer was initialized with random uniform values or pre-trained word vectors from other studies. While RNNs have been used in previous research, we have exploited bidirectional LSTM. Additionally, during training and test period, we investigate a variety of classification functions ranging from logistic regression to graphical model. The complete and careful evaluation of our models indicates that there is no an outstanding classification function in both datasets. Our model with CRF classifier achieved the highest performance on both datasets in case of using lexical features. Meanwhile, with additional features on ATIS dataset, *BiLSTM-Softmax* gained the best F1-score.

## 1.3 Thesis outline

The remainder of this thesis is organized as follows:

Chapter 2 describes background of our research with respect to support vector machines (SVMs), recurrent neural networks (RNNs), Long Short-Term Memory Networks (LSTMs) and conditional random fields (CRFs). In fact, SVMs is a supervised learning method which is compatible with classification problem for discrete objects. In contrast, CRFs is the statistical approach, and is really appropriate for sequential tasks in which it discovers the most probable label sequence instead of discrete labels. The strength and weakness of two kinds of neural networks RNNs and LSTMs are also illustrated in this chapter.

In chapter 3, our architecture including four major layers is presented in detail. The first layer - word embedding, takes the place of each word by a unique contiguous vector. The weight of this layer can be uniformly random or using pre-trained vectors. After that, we will describe two LSTM networks, one for forward, the other for backward, that extract advantageous information from these vectors. Eventually, we will give the full particulars of four classification functions and their formula.

The experimental results on two widely used datasets: ATIS and DARPA, are located in chapter 4. Our models are evaluated completely and carefully in terms of context window size, word embedding initialization, and classification function type.

The comparison and error analysis of our models with other previous methods are conducted as well.

Finally, chapter 5 shows our conclusions and future work.

# Chapter 2

# Background

## 2.1 Slot Filling in Spoken Language Understanding

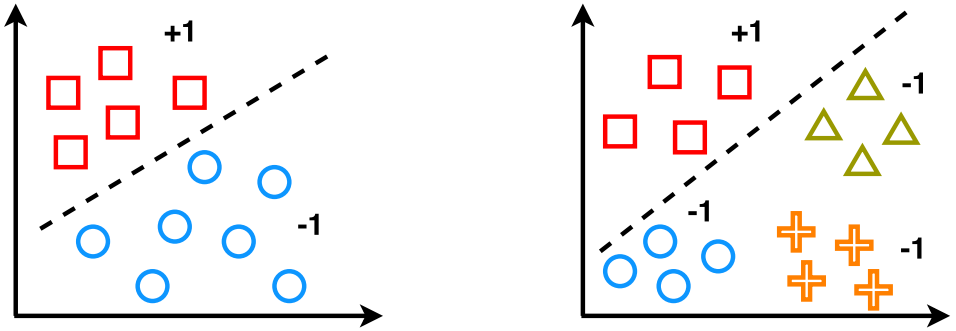
In order to achieve a goal in a human-machine conversational understanding system, automatically filling in a set of arguments or slots embedded in a semantic frame plays a vital role in SLU. This is considered as semantic-concepts extractions as well. The popular representation for extraction problem is IOB (in/out/begin) tags. An example sentence is illustrated in Table 2.1, along with domain, slot annotations, and special domain-independent named entity. It shows that *Boston* is the departure city with a label *B-fromloc.city_name* whilst the arrival city is *New York* with two associated labels *B-toloc.city_name* and *I-toloc.city_name* that correspond to two words *New* and *York*; which reflect the user's intention. The other words in a sentence are assigned a dummy tag *O*.

Table 2.1: IOB representation

| **Domain** | Airline Travel | |
|---|---|---|
| **Sentence** | **Slot Label** | **Named Entity** |
| show | O | O |
| flights | O | O |
| from | O | O |
| Boston | B-fromloc.city_name | B-city_name |
| to | O | O |
| New | B-toloc.city_name | B-city_name |
| York | I-toloc.city_name | I-city_name |
| today | O | O |

## 2.2 Support Vector Machines

SVM is a supervised learning algorithm which is mostly used for classification problems, and its goal is to find the optimal separating hyperplane which maximizes the margin of the training data. SVMs have been applied successfully by many researchers in a wide range of applications, e.g. face detection, object recognition or image retrieval [4]; text categorization [12]; or sequential labeling [24], [33].




(a) Binary classification (b) Multi-class classification

Figure 2.1: SVMs classification

Inherently, SVM is formulated for binary classification (Figure 2.1a). When the two classes are linearly separable in $R^D$, among the infinite number of possible hyperplanes, a separating hyperplane which releases the smallest generalization error would be chosen. In other words, that optimal hyperplane has maximal margin, in particular, the maximal sum of the distances from the hyperplane to the closest data points of each class. However, for practical application with real data, the two classes are not perfectly separable, therefore, a positive slack variable $\xi$ which penalizes data points which violate the margin requirements is introduced. Given $n$ training data and its corresponding labels $(x_i, y_i)$, $x_i \in R^D$, $y_i \in \{-1, 1\}$ for all $i = 1, \ldots, n$; SVMs learning optimize the following constraints:

$$\min_{w,\xi_i} \quad \frac{1}{2}\|w\|^2 + \lambda \sum_{i=1}^{n} \xi_i \tag{2.1}$$

$$s.t. \quad w^T x_i y_i \geq 1 - \xi_i \quad \forall i$$
$$\xi_i \geq 0 \quad \forall i$$

where $\lambda$ is a parameter that controls the trade-off between the margin and the misclassification errors.

Many real applications require SVMs' extension to multiclass to classify into more than two classes. One basic strategy for this is One-versus-All (OVA) or One-to-Others, in which all the training samples with the same label are considered as one class while the others as the remaining class. It then becomes a binary classification

problem (see Figure 2.1b). For sophisticated issues that are hard to be solved be a linear classifier, the data will be mapped from the input space into a higher dimensional vector space by a nonlinear transformation. Notwithstanding, in our study, we only focus on standard linear SVMs with recurrent neural networks for the SL task.

SVMs are widely used as an alternative of softmax for classification in a disjoint classifier. More precisely, the first step is using deep neural nets to learn good invariant hidden latent representations which then are fed into linear or kernels SVM as input [5], [17]. Nonetheless, those hidden representations are not fine-tuned with respect to SVMs' objective [27]. Other related works combined SVMs and deep neural networks into a single model to identify individual objects [22], [27]. In our research, we would like to extend this kind of model for sequence labeling problem beyond particular objects.

## 2.3 Recurrent Neural Networks

### 2.3.1 Recurrent Neural Networks

It is obvious that in traditional neural networks, all inputs and outputs are assumed independent of each other. Nevertheless, this assumption is not flexible in sequential processing, e.g translation, text generation. A classic example for this is that if we want to predict the next word in a sentence, it is worth understanding preceding words or context. Undoubtedly, the idea behind RNNs is to exploit sequential information, with the current output being dependent on the previous computations. To do this, RNNs have a "memory" which is able to capture information about what has already been calculated. Therefore, RNNs are popular models that release great promise in myriads of NLP tasks [19], [20], [30], [31].

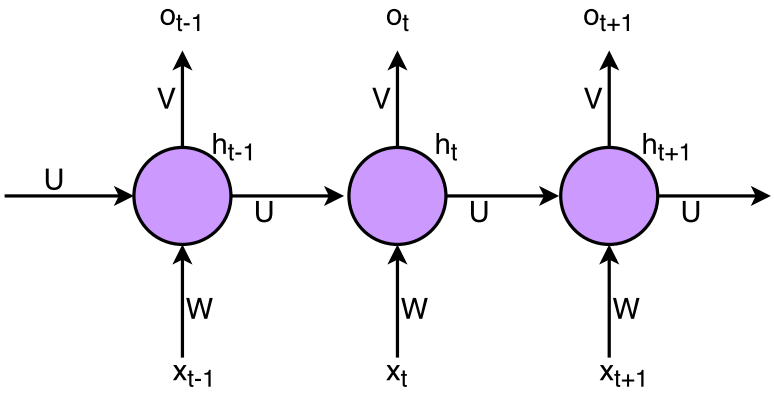



Figure 2.2: Recurrent Neural Networks

Figure 2.2 demonstrates the RNNs architecture. Given a sequence of vectors $X = (x_1, x_2, ..., x_n)$ in which $x_i$ corresponds to the $i^{th}$ word of a sentence, the recurrent hidden state $h_t$ is calculated based on the previous hidden state $h_{t-1}$ and the current

input $x_t$ at time $t$ as follows:

$$h_t = f(Wx_t + Uh_{t-1}) \tag{2.2}$$
$$y_t = softmax(Vh_t) \tag{2.3}$$

where $W, U$ are transformation matrices of the input and preceding hidden state respectively; the function $f$ could be a nonlinearity such as *tanh*. After that, $V$ is a matrix which transforms the current hidden state $h_t$ into the output $y_t$ under an activation function (*softmax* in this case).

Our goal is to find the parameters $U, V$ and $W$ that minimize the loss (error) function for our training data. The most common approach for this is Stochastic Gradient Descent (SGD). The idea behind SGD is that for each training sample, all parameters are adjusted smaller or greater whichever direction decreases error. These directions are given by the gradients on the loss, and is computed by Backpropagation through time algorithm which is described more detail in the following parts.

### 2.3.2 Backpropagation through time

Basically, backpropagation is a popular and powerful technique for calculating derivatives quickly. In other words, it is perceived as a primary algorithm that is capable of training deep models efficiently. In feed-forward networks, backpropagation moves backward from the final error through the outputs, weights, and inputs of each hidden layer [3]. During the backpropagation period, those weights are assigned responsibility for a portion of the error by computing their partial derivatives, or the relationship between their rates of change. Afterward, in order to adjust the weights up or down towards error decrease, those derivatives are used by our learning rule and gradient descent. RNNs rely on an extension of backpropagation named backpropagation through time (BPTT) [1]. Apparently, the parameters are shared by all time step in the network, the gradient at each output, therefore, is affected by not only the current time step, but also the preceding ones.

First, let define the loss function to be cross entropy loss for one training sample $X$:

$$L = \sum_t E_t(y_t, \hat{y}_t) = \sum_t -y_t \log \hat{y}_t \tag{2.4}$$

where $y_t, \hat{y}_t$ correspond to the correct and forecast label at time $t$. The partial derivatives with respect to those weights $U, V, W$ in Equation 2.2 and 2.3 is summarized at each time step for one training sample.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W} \tag{2.5}$$

For each time $t$, the chain rule of differentiation is exploited to calculate the gradient. Fundamentally, the gradient for $V$ relies on values at current time step,

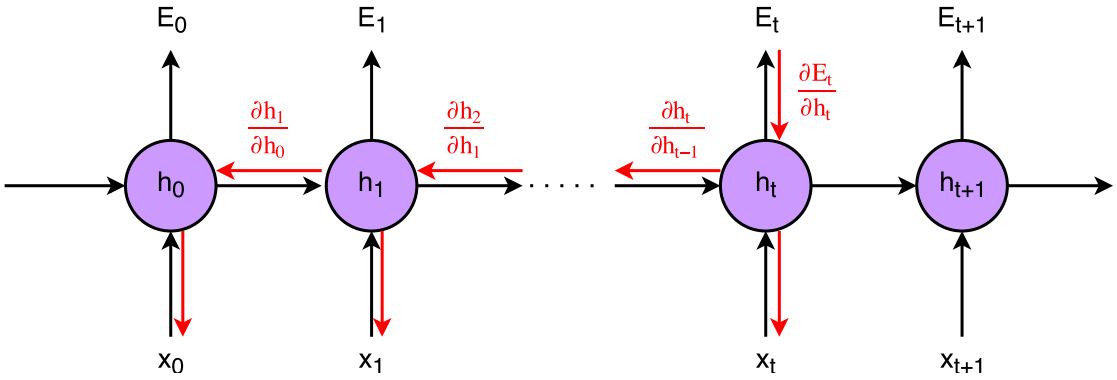



Figure 2.3: Backpropagation through time

and is calculated by simple matrix multiplication as follows:

$$\begin{aligned}\frac{\partial E_t}{\partial V} &= \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial V} \\ &= \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial V} \\ &= (\hat{y}_t - y_t) \otimes h_t\end{aligned} \tag{2.6}$$

where $z_t = Vh_t$ and $\otimes$ is the outer product of two vectors. In contrast, it is relatively sophisticated to obtain the gradient in terms of $U$ and $W$. The chain rule is applied as well:

$$\frac{\partial E_t}{\partial U} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial U} \tag{2.7}$$

Due to the dependency of $h_t = \tanh(Wx_t + Uh_{t-1})$ on $h_{t-1}$, which depends on $U$ and $h_{t-2}$, and so on, the Equation 2.7 is rewritten in a form:

$$\frac{\partial E_t}{\partial U} = \sum_{\tau=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial h_\tau}\frac{\partial h_\tau}{\partial U} \tag{2.8}$$

Since the weight $U$ and $W$ are shared all the time in RNNs, the gradient at time $t$ with regard to those weights is a summation of the contributions at each time step back to $t = 0$ as shown in Figure 2.3. Since the layers and time steps of RNNs relate to each other through multiplication, derivatives are susceptible to vanishing with small values in the matrix or exploding with large ones [1]. While exploding gradients can be tackled relatively easily by truncation or squash, vanishing gradients is a more sophisticated issue that makes our networks learn harder. That is a reason why though RNNs, in theory, can deal with long sequences, they face with a major obstacle that only "remember" a few preceding steps. In order to overcome this issue, a special kind of RNNs, Long Short-Term Memory Networks (LSTMs for short) was introduced by *Hochreiter et al.* [11], which are capable of looking back a plenty of steps. The more detail of LSTMs is described in the following section.

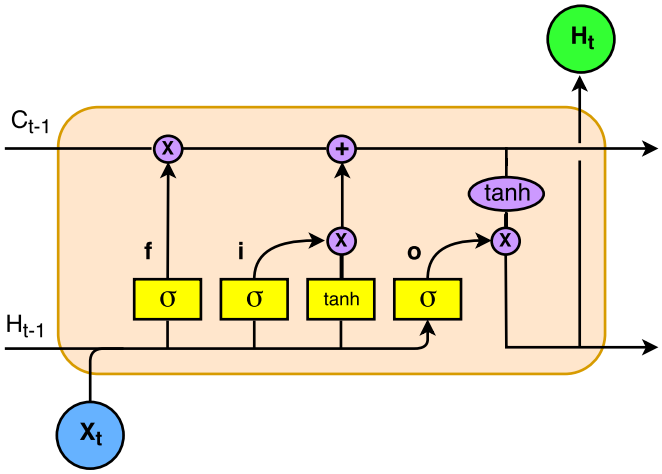



Figure 2.4: Long Short-Term Memory Network

## 2.4 Long Short-Term Memory Networks

LSTMs have been designed to combat the vanishing gradient obstacle by incorporating a memory-cell. Practically, LSTMs is one of the most popular networks which is applied successfully in a wide range of applications such as speech recognition [9], machine translation [26], image captioning [28], to name but a few.

Given a sequence of vectors $(x_1, x_2, ..., x_n)$, the hidden state $h_t$ of LSTM at time t is calculated as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.9}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.10}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.11}$$
$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_C) \tag{2.12}$$
$$C_t = i_t \odot \tilde{C}_t + f_t \odot C_{t-1} \tag{2.13}$$
$$h_t = o_t \odot tanh(C_t) \tag{2.14}$$

where $x_t$ is the input at time $t$; $i, f, o$ are input gate, forget gate and output gate respectively; $W$ and $U$ are transformation matrices; $\sigma$ is the element-wise sigmoid function; and $\odot$ denotes the element-wise product of two vectors. For simplification, the above equations are abbreviated to $h_t = LSTM(x_t, h_{t-1})$.

LSTMs contains information apart from the usual flow of the recurrent network in a gated cell [2]. The cell makes a decision about what to store, and what to keep and erase via its three gates that open or close [1]. These operations are controlled by *sigmoid* functions, which are all in the range of $[0, 1]$. Firstly, the forget gate decides which information is eliminated from the cell state (Equation 2.9). It observes $h_{t-1}$ and $x_t$, and outputs a filter between 0 and 1 for each number in the cell state $C_{t-1}$. Secondly, new information is handled towards its storage in the cell state. This step consists of two parts. The input gate (Equation 2.10) makes a decision which values are updated, while a *tanh* layer generates a vector describing a new candidate

values, $\tilde{C}_t$, that could be added to the state (Equation 2.12). The combination of these two produces an update to the state. Then, the old cell state $C_{t-1}$ is updated into the new cell state $C_t$. This is implemented by multiplying the old state by $f_t$ and adding $i_t \odot \tilde{C}_t$, resulting in a new candidate value which scales how much each state value is updated (Equation 2.13). The final step is to decide what information is released. The output gate (Equation 2.11) makes a decision which parts of the cell state is emitted. It is worth putting the cell state $C_t$ through *tanh*, followed by a production with $o_t$ (Equation 2.14). On the other hand, the weighted matrices $W$ and $U$ of each gate and hidden state are adjusted during the recurrent network learning process.

In the transformation of input, addition and multiplication have different roles given by LSTMs memory cells. The secrets of LSTMs is the central **plus sign** $\oplus$ in Figure 2.4, which is capable of preserving a constant error when it has to be propagated back at depth. Indeed, the subsequent cell state is a summation of its current state and new input instead of multiplying them.

## 2.5 Conditional Random Fields

### 2.5.1 Statistical Approaches for Slot Filling task

For the slot filling task, many researchers have employed statistical approaches including generative models such as Hidden Markov Models (HMMs), Hidden State Vector model (HVS) [10] or discriminative classification methods like CRFs [20], [32]. Given an input sentence consisting of a sequence of word $X = (x_1, x_2, ..., x_n)$ and the output label for each word is $y = (y_1, y_2, ..., y_n)$, the goal of SLU system is to find the semantic representation of the slot sequence $y$ that has a maximum posterior probability $P(y|X)$:

$$y^* = \arg\max_y P(y|X) = \arg\max_y P(X|y)P(y)$$

The Bayes rule is applied in the generative model, so the objective function is to maximize the joint probability $P(X|y)P(y) = P(X, y)$. Nevertheless, in this day and age, discriminative approaches have become more popular, and Conditional Random Fields (CRFs) [15] are applied successfully in a plenty of circumstances. As shown in Figure 2.5, the linear-chain CRF models the conditional probability $P(y|X)$ based on features $h_m(y_{t-1}, y_t, x_{t-d}^{t+d})$ extracted from the current and preceding states $y_{t-1}$ and $y_t$ respectively; along with a window of $2d$ surrounding words of the current word $x_t$ as follows:

$$P(y|X) = \frac{1}{Z} \prod_{t=1}^{n} \exp\left\{ H(y_{t-1}, y_t, x_{t-d}^{t+d}) \right\} \tag{2.15}$$

where

$$H(y_{t-1}, y_t, x_{t-d}^{t+d}) = \sum_{m=1}^{M} \lambda_m h_m(y_{t-1}, y_t, x_{t-d}^{t+d}) \tag{2.16}$$

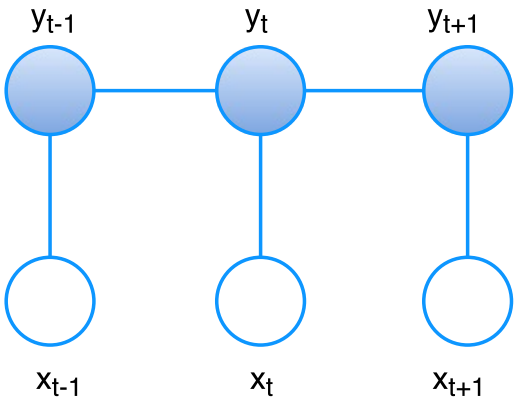



Figure 2.5: CRF.

$$Z = \sum_{\tilde{y}} \prod_{t=1}^{n} \exp\left\{ H(\tilde{y}_{t-1}, \tilde{y}_t, x_{t-d}^{t+d}) \right\} \tag{2.17}$$

In the rest of this section, let $H(y_{t-1}, y_t) = H(y_{t-1}, y_t, x_{t-d}^{t+d})$ to simplify notation. The main disadvantage of these models is good feature engineering which requires numerous manual work. As a result, one promising direction towards deep learning networks is incorporating both feature design and classification into learning procedure.

### 2.5.2 Forward-backward algorithm for CRFs

During learning procedure, direct calculation of Equation 2.17 is impractical due to exponential amounts of all possible sequences $y$. Consequently, forward-backward algorithm is applied in order to compute $Z$. Let $G_t(y_1, y_2) = \exp\{H(y_{t-1}, y_t)\}$, the Equation 2.17 is rewritten as follows:

$$Z = \sum_{\tilde{y}} \prod_{t=1}^{n} G_t(\tilde{y}_{t-1}, \tilde{y}_t) \tag{2.18}$$

The forward partial sums up to time t, denoted $\alpha(y, t)$, represent the total score of all sequences ending at label $y$ at time $t$:

$$\alpha(y, t) = \sum_{\tilde{y}} \left( \prod_{\tau=1}^{t-1} G_\tau(\tilde{y}_{\tau-1}, \tilde{y}_\tau) G_t(\tilde{y}_{t-1}, y) \right) \tag{2.19}$$

and can be updated by the following forward recursion:

$$\alpha(y, t) = \sum_{\tilde{y}} \alpha(\tilde{y}, t-1) G_t(\tilde{y}, y)) \tag{2.20}$$

Similarly, the backward partial sums $\beta(y,t)$ demonstrate the total score of all sequences starting at label $y$ at time $t$:

$$\beta(y,t) = \sum_{\tilde{y}} G_{t+1}(y, \tilde{y}_{t+1}) \prod_{\tau=t+1}^{n} G_{\tau+1}(\tilde{y}_\tau, \tilde{y}_{\tau+1}) \tag{2.21}$$

and can be updated with the backward recursion as well:

$$\beta(y,t) = \sum_{\tilde{y}} \beta(\tilde{y}, t+1) G_{t+1}(y, \tilde{y}) \tag{2.22}$$

Then the normalization quantity $Z$ is rewritten by:

$$Z = \sum_{\tilde{y}} \alpha(\tilde{y}, n) = \sum_{\tilde{y}} \beta(\tilde{y}, 1) \tag{2.23}$$

As usual, two special labels *start* and *stop* are added to avoid summation:

$$Z = \alpha(stop, n+1) = \beta(start, 0) \tag{2.24}$$

Apart from $Z$, the forward and backward partial sums can be utilized to estimate the following marginal posterior probabilities:

$$p(y_t|X) = \frac{\alpha(y_t,t)\beta(y_t,t)}{Z} \tag{2.25}$$

$$p(y_{t-1}, y_t|X) = \frac{\alpha(y_{t-1}, t-1) G_t(y_{t-1}, y_t) \beta(y_t,t)}{Z} \tag{2.26}$$

### 2.5.3 Viterbi algorithm for CRFs

In theory, we can find the most probable label sequence by listing all possible ones and calculating the probability of the label sequence for each of the combinations. Afterward, the most likely label sequence is a combination that achieves the maximal probability:

$$y^* = \arg\max_{\tilde{y}} p(\tilde{y}|X;\lambda) \tag{2.27}$$

This approach is viable, however, it is computationally expensive due to exponential numbers of all possible $\tilde{y}$ as when computing the quantity $Z$ in Equation 2.17. As a result, to reduce complexity, recursion is a wise choice. First, it is worth rewriting 2.27 by expanding the features from Equation 2.15 and 2.16 in a form:

$$y^* = \arg\max_{\tilde{y}} \prod_{t=1}^{n} \exp\left\{H(\tilde{y}_{t-1}, \tilde{y}_t)\right\} \tag{2.28}$$

$$= \arg\max_{\tilde{y}} \sum_{t=1}^{n} H(\tilde{y}_{t-1}, \tilde{y}_t) \tag{2.29}$$

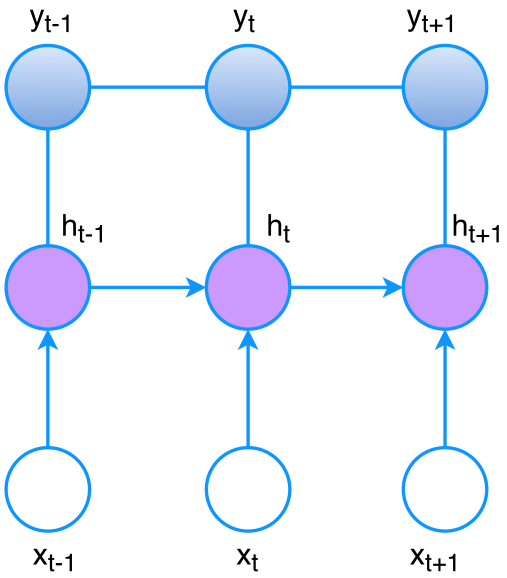



Figure 2.6: R-CRF.

The partial maximum $V(y,t)$ describes the highest score of all sequences obtaining label $y$ at time $t$ and is defined as:

$$V(y,t) = \max_{\tilde{y}} \left( \sum_{\tau=1}^{t-1} H_\tau(\tilde{y}_{\tau-1}, \tilde{y}_\tau) + H_t(\tilde{y}_{t-1}, y) \right) \tag{2.30}$$

and it is can be computed in a recursive form:

$$V(y,t) = \max_{\tilde{y}} (V(\tilde{y}, t-1) + H_t(\tilde{y}, y)) \tag{2.31}$$

Eventually, in order to obtain the most likely label sequence $y^*$, we can trace back from $V(y^*, T)$ in the trellis.

## 2.6 Recurrent Conditional Random Field

In order to avoid label bias problem in RNNs which release a position-by-position distribution over output labels, the input features for a CRF is generated by an RNN in the Recurrent Conditional Random Field (R-CRF) model [31], as shown in Figure 2.6. In this scheme, RNN and transition probabilities parameters are jointly trained according to the objective function of a CRF [20]. By this way, benefits of the sequence-level discrimination ability of the CRF and the feature learning of the RNN was taken in the R-CRF model. Even though the objective function of the R-CRF is the same as Equation 2.15, the features $h_m(y_{t-1}, y_t, x_{t-d}^{t+d})$ are generated by the RNN and learned by employing error back-propagation to obtain gradients. In contrast, in a CRF, only the weight $\lambda_m$ is tuned, whereas features are unchanged and are usually a binary vector. The features $h_m(y_{t-1}, y_t, x_{t-d}^{t+d})$ now can be decomposed

into transition feature $h_m(y_{t-1}, y_t)$ and tag-specific feature $h_m(y_t, x_{t-d}^{t+d})$ output from the RNN. Since features extracted from RNN are sensitive to all words from the beginning of a sentence [20], the Equation 2.16 is modified as follows:

$$\begin{aligned} H(y_{t-1}, y_t) &= \sum_{m=1}^{M} \lambda_m h_m(y_{t-1}, y_t, x_0^{t+d}) \\ &= \sum_{p=1}^{P} \lambda_p h_p(y_{t-1}, y_t) + \sum_{q=1}^{Q} \lambda_q h_q(y_t, x_0^{t+d}) \end{aligned} \tag{2.32}$$

Note that, the features $h_m(y_t, x_0^{t+d})$ that R-CRF used are un-normalized before *softmax* activation function with the intention that it is able to prevent the label bias problem that motivated CRFs. To find the best sequence of label from R-CRF, Viterbi algorithm is applied. Those work have inspired us to integrate CRFs with bidirectional LSTMs to solve the label bias problem.

# Chapter 3

# Proposed Model

## 3.1 Overview

An overview of our architecture is shown in Figure 3.1. The input of a network is a sentence $S$ with length $n$, which consists of a sequence of words $w_1, w_2, ..., w_n$. Four main layers in our model have distinct purposes:

- Word embedding layer: maps word feature indices to feature vectors which are fed into two LSTM layers.
- Forward LSTM layer: extracts beneficial information of each word in a sen-

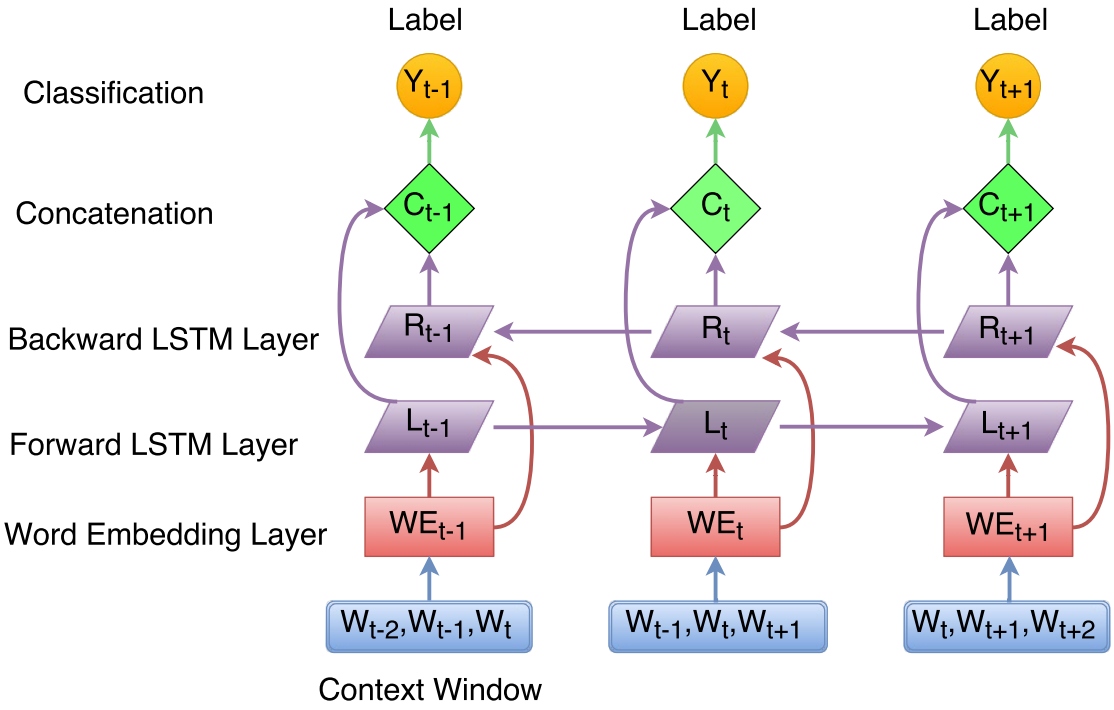



Figure 3.1: The main architecture of our model

tence from left to right.

- Backward LSTM layer: extracts beneficial information of each word in a sentence from right to left.
- Classification layer: labels word by word using one of four classification functions, e.g. CRFs, SVMs, based on extracted information from forward and backward LSTM layers.

In the rest of this chapter, we describe more detail about functionality and responsibility of each layer in our model.

## 3.2 Word Embedding layer

The traditional representation of the input word is a one-hot vector which, in fact, explores a dimension in case of large vocabulary size. If the corpus has $V$ words, there will be a matrix with a size of $V \times V$ to represent all words as seen in Figure 3.2. An alternative approach for word representation is utilizing word embeddings as a first-layer weight matrix which identifies a unique weighted vector for each input word (Figure 3.3). If there are $V$ words in our corpus and each word is represented by a M-dimensional vector in a continuous space, the size of word embedding matrix is $V \times M$. In practical, $V$ is huge and much larger than $M$.

These word embeddings usually are trained in advance on a large external data with various models, e.g. *Mikolov et al.* [21] exploited continuous skip-gram model

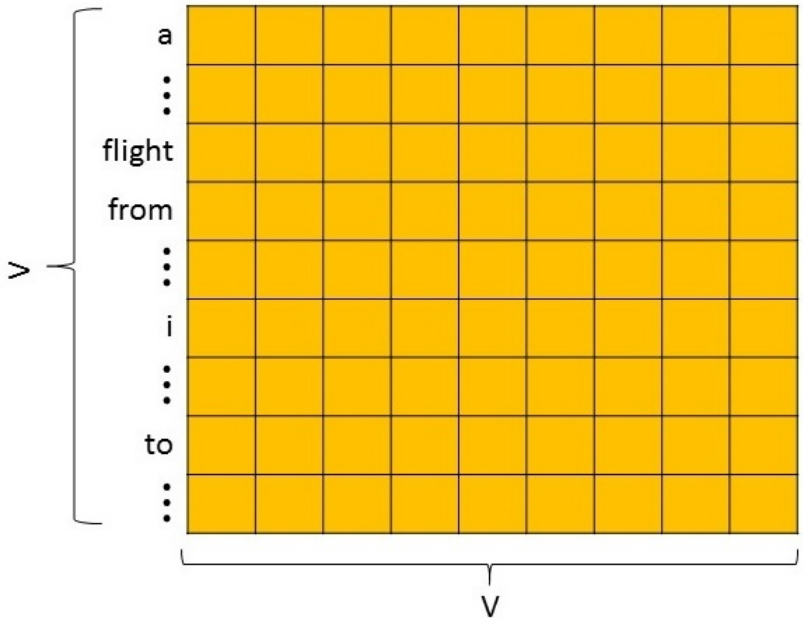



Figure 3.2: One-hot vector presentation matrix with size $V \times V$

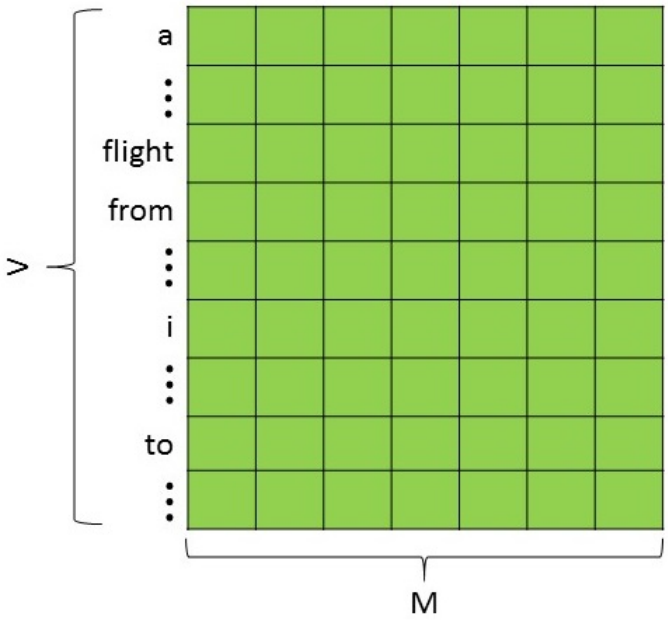



Figure 3.3: Word embedding layer for vector presentations of words. The size of this layer is $V \times M$

from large amounts of unstructured data; *Pennington et al.* [23] used global log-bilinear regression model; while SENNA word embeddings were trained by convolutional neural network [6]. Nevertheless, *Mesnil et al.* [20] state that word embeddings can be also learned from initialized random values, which exerts same performance compared to pre-trained word embeddings. Therefore, in our experiments as in [8], the initial values for the weights of word embeddings, along with other layers, were uniformly sampled from a symmetric interval $\left[ -\sqrt{\frac{6}{f_{in}+f_{out}}}, \sqrt{\frac{6}{f_{in}+f_{out}}} \right]$, where $f_{in}$ and $f_{out}$ are the input and output dimension of a corresponding layer.

## 3.3 Context Window Size

In order to improve the performance of RNNs, a context word window which incorporates several surrounding words to represent a word of interest is introduced and used as input for our models. The reason for this procedure is that it can enable to capture short-term temporal dependencies [20]. In specific, each word is mapped to an embedding vector. Afterward, the $d$-context window $x_t^d$ takes into account $d$ preceding words and $d$ following words of a current word $x_t$ to describe $x_t$ as a concatenation of $(2d + 1)$ word embedding vectors as follows:

$$x_t^d = [e(w_{t-d}), ..., e(w_t), ..., e(w_{t+d})] \tag{3.1}$$

where $e(w_t)$ is an embedding vector of word $w_t$. The length of vector representation for $x_t^d$ is now $(2d + 1) \times M$. For the first or last words in sentences, to avoid

border effect problem, one special token was used to pad the beginning and ending of sentences.

## 3.4 Bidirectional Long Short-Term Memory Networks

In sequence labeling, it is possible to extract beneficial information from not only the past but also the future simultaneously, e.g bidirectional LSTM[16], bidirectional Jordan RNN [19]. Subsequently, a bidirectional variant is utilized in our experiments to describe the hidden layer.

First, we define the forward $\overrightarrow{h}_t$ and the backward $\overleftarrow{h}_t$ hidden layers:

$$\overrightarrow{h}_t = LSTM(x_t, \overrightarrow{h}_{t-1}) \tag{3.2}$$

$$\overleftarrow{h}_t = LSTM(x_t, \overleftarrow{h}_{t+1}) \tag{3.3}$$

where $\leftarrow$ and $\rightarrow$ denote the forward and backward pass respectively. It is noticeable that the initial hidden state for each network $\overrightarrow{h}_0$ and $\overleftarrow{h}_0$ was initialized uniformly as discuss in section 3.2.

The bidirectional hidden layer $\overleftrightarrow{h}_t$ at time $t$ then takes the forward and backward hidden layers as input which is fed to further classification procedure:

$$\overleftrightarrow{h}_t = \left[\overrightarrow{h}_t, \overleftarrow{h}_t\right] \tag{3.4}$$

It is importantly noted that our models are easy to adjust in case of having additional features. One possible solution is to concatenate them with hidden layers $\overleftarrow{h}_t$ and $\overrightarrow{h}_t$ to get a bigger and richer representation like:

$$\overleftrightarrow{h}_t = \left[\overrightarrow{h}_t, \overleftarrow{h}_t, a_t\right]$$

where $a_t$ is an additional feature vector which supports for determining a unique label for a particular word at time $t$.

## 3.5 Classification functions

### 3.5.1 Softmax activation function

A conventional approach for classification problem is using softmax activation function. Given an input sentence $X = (w_1, w_2, .., w_n)$, considering a matrix of scores $P$ output by the bidirectional LSTM network, the size of P is $n \times k$, where $k$ is the number of diverse labels; and $P_{i,j}$ represents the score of the $j^{th}$ label of the $i^{th}$ word in a sentence. The probability to assign label $j$ for each word $w_t$ is calculated by

$$p(y_t = j | w_1, ..., w_{t+d}) = \frac{e^{P_{t,j}}}{\sum_{u=1}^{k} e^{P_{t,u}}} \tag{3.5}$$

Due to using context window, the input of our model at time $t$ is $(t + d)$ words from the beginning of the sentence. During training, we use stochastic gradient descent with the parameters updated after computing the gradient for each segment of a sentence $S$ in the training set $D$, towards minimizing the following negative log-likelihood:

$$L = -\sum_{(S,W)\in D}\sum_{t=1}^{T} log(p(y_t|w_1, ..., w_{t+d})) \tag{3.6}$$

where $T$ is the length of each sentence, and it can be flexible.

The predicted label for word $t^{th}$ in a sentence is

$$y_t^* = \arg\max_j p(y_t = j|w_1, ..., w_{t+d}) \tag{3.7}$$

### 3.5.2 Sigmoid activation function

Sigmoid function is considered as a simple case of CRFs without any constraints among output labels like softmax function. However, in contrast to softmax function which focuses on a single label, sigmoid function takes into account multiple top-score labels [25]. We therefore replace the exponential function in equation (3.5) by logistic sigmoid function $\sigma$

$$p(y_t = j|w_1, ..., w_{t+d}) = \frac{\sigma(P_{t,j})}{\sum_{u=1}^{k}\sigma(P_{t,u})} \tag{3.8}$$

The objective function is also negative log-likelihood as in equation (3.6). Meanwhile, an output label of a word is forecast as in equation (3.7)

### 3.5.3 Conditional Random Fields

The LSTM, the same as RNN, makes independent labeling decisions for single word position $y_t$, which is more prone to the label bias problem [15]. The reason is that the labels in a sentence are mutually dependent on each other (e.g., I-fromloc.city cannot follow B-time). Therefore, Conditional Random Fields (CRFs) were used to model labeling decisions jointly [16].

Suppose that $A$ is a transition matrix of scores in which $A_{i,j}$ corresponds to the scores of a transition from label $i$ to label $j$. These scores are then fine-tuned during the training time. It is noticeable that two special labels $y_o$ and $y_{n+1}$ added to each sentence are considered as the *start* and *end* labels which have already described in section 2.5.2. Thus, A is a square matrix of size $k + 2$.

The score of a predicted sequential labels $y = (y_1, y_2, .., y_n)$, denoted as $s(X, y)$, consists of two elements: a score at each position $i$ corresponding to label $y_i$; and a transition score of two neighboring labels $y_i$ and $y_{i+1}$. The score $s(X, y)$ is calculated as following:

$$s(X, y) = \sum_{i=0}^{n} A_{y_i,y_{i+1}} + \sum_{i=1}^{n} P_{i,y_i} \tag{3.9}$$

After obtaining score of all possible sequential label, a probability of each sequence $y$ is yielded using a softmax function:

$$p(y|X) = \frac{e^{s(X,y)}}{\sum_{\tilde{y} \in Y_X} e^{s(X,\tilde{y})}} \tag{3.10}$$

The log-probability of the correct label sequence is maximized during training:

$$log(p(y|X)) = s(X,y) - log\left(\sum_{\tilde{y} \in Y_X} e^{s(X,\tilde{y})}\right) \tag{3.11}$$

where $Y_X$ denotes all possible label sequences given a sentence X.

Decoding from LSTM-CRF uses the Viterbi algorithm described in section 2.5.3, and we anticipate the output sequence that releases the maximum score by:

$$y^* = \arg\max_{\tilde{y} \in Y_X} s(X, \tilde{y}) \tag{3.12}$$

### 3.5.4 Support Vector Machines - One versus all

Tang [27] demonstrated a small but consistent advantage of replacing the top softmax layer with a linear support vector machine. For $k$ class problems, $k$ linear SVMs will be trained independently, where the label from other classes form the negative cases. In our experiment, we implemented the L1-SVM form for multiclass problems. Firstly, we define a label matrix $C$ with size $n \times k$ as follow:

$$C_{i,j} = \begin{cases} 1 & \text{if} \quad y_i = j \\ -1 & \text{otherwise} \end{cases} \tag{3.13}$$

During training, the following loss objective function is minimized

$$L_1 = \sum_{i=1}^{n} \sum_{j=1}^{k} \max(1 - P_{i,j} C_{i,j}, 0) \tag{3.14}$$

The predicted label for word $i^{th}$ in a sentence is

$$y_i^* = \arg\max_{j} P_{i,j} \tag{3.15}$$

# Chapter 4

# Experimental results

## 4.1 Datasets

### 4.1.1 ATIS

ATIS is a well-known dataset which was used extensively for SLU research. There are several variants of this dataset, and in our experiment, we used the ATIS corpus[1] utilized in [19] as well. The training set has 4978 sentences in the ATIS-2 and ATIS-3 corpora, whilst the test set consists of 893 sentences from the ATIS-3 Nov93 and Dec94 datasets. The number of distinct labels and named entities is 128 and 141 respectively.

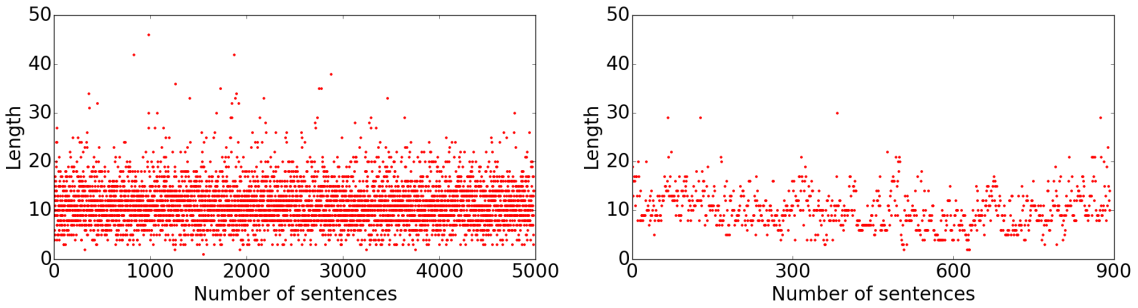
We preprocessed the data as in [30]. In detail, a special symbol <UNK> was used as an alternative of all words with only one occurrence in the training set. This symbol was also utilized to represent those unseen words in the test set. Besides, all sequences of numbers is converted to the string DIGIT, e.g. "320" is described by "DIGIT*3".

In ATIS dataset, additional features "Named Entity" (NE) are often used to obtain better performance. Those features are marked via look-up table such as city, dates, etc,.. which nearly determine the slot label. In our model, the NE information feature is a one-hot vector which is concatenated with the output of bidirectional LSTM and the classification layers.

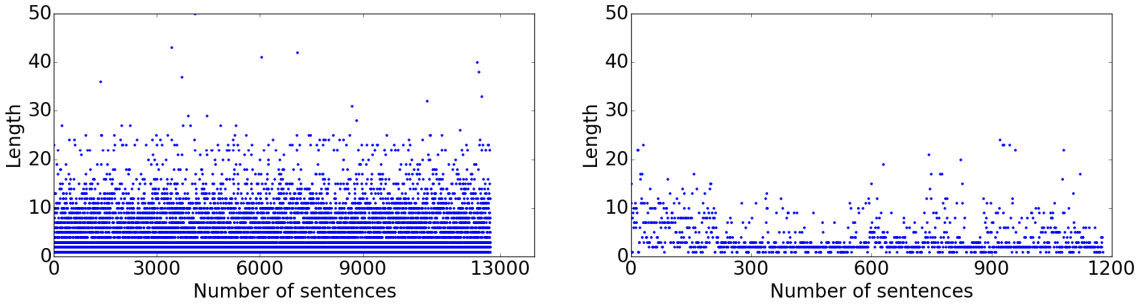
### 4.1.2 DARPA Communicator

We also have evaluated the performance of proposed model on another more complex dataset DARPA Communicator. Whilst the training set consists of 12702 utterances, the test set contains 1178 utterances. As can be seen from figure 4.1, ATIS dataset contains more long sentences, in particular, the average length of all sentences in the training set and test set are approximate 11.37 and 10.3 respectively. In contrast, the overwhelming minority in DARPA are short sentences, in which the average length is around 4.36 for the training set and 4.04 for the test set.

[1] https://github.com/mesnilgr/is13




(a) ATIS training set (left) and test set (right)




(b) DARPA training set (left) and test set (right)

Figure 4.1: Length of sentences in two datasets

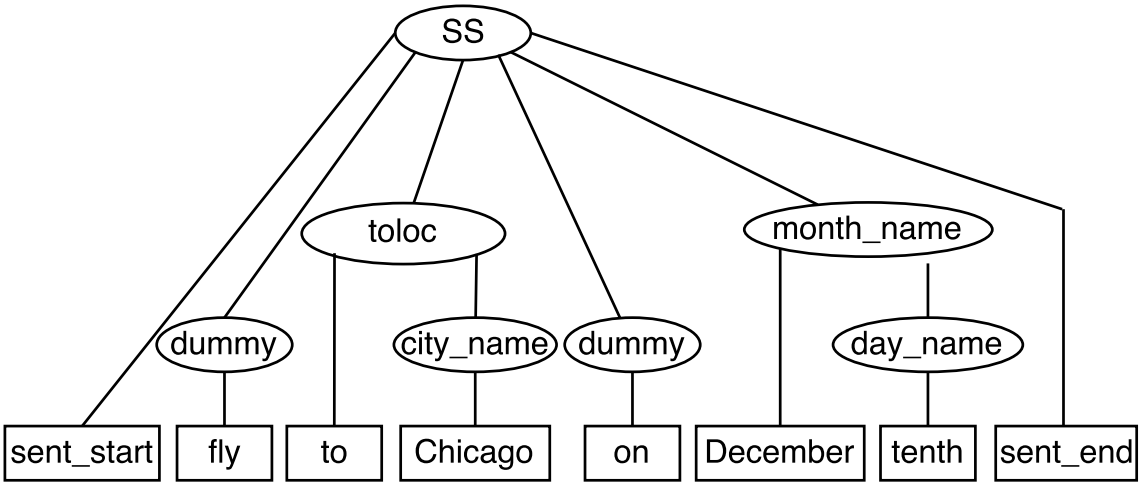



Figure 4.2: The Phoenix parser

Another noticeable distinction between two datasets is that while ATIS data is tagged greatly in IOB form, the DARPA data includes utterance transcriptions and the semantic parse results from the rule-based Phoenix parser. It, therefore, has to be converted into IOB tags in advance. For example, the Phoenix parser tree of the sentence "**fly to Chicago on december tenth**" and its abstract annotation is **TOLOC(CITY_NAME) MONTH_NAME(DAY_NUMBER)**, is demonstrated in Figure 4.2. *Chicago* will then be marked as *B-toloc.city*; and *december* and *tenth* will be labeled as *B-month_name* and *B-day_number* respectively. This

was carried out by using look-up database which comes up with DARPA Communicator dataset. Further processes were the same as dealing with ATIS dataset.

## 4.2 Experimental Settings

In this experiment, we have implemented our models with PYTHON code and its supported library THEANO[2]. Our models with various settings then have been run on High Performance Parallel Computing Servers CX250[3].

As in several preceding work [19], [20], [30], [31], the dimension of word embedding layer and forward/backward recurrent hidden states are both 100. Other parameters are set as follows:

- Learning rate: 0.05 for lexical feature only, 0.1 for both lexical and named entity features
- Context Window Size: 1, 3, 5, 7, 9

## 4.3 Evaluation Metrics

In statistical analysis of binary classification, the F1-score is a measure of accuracy for a particular test set, and it controls the trade-off between *precision* and *recall*. The F1-score can also be interpreted as a weighted average of the *precision* and *recall*:

$$F1 = 2\frac{precision * recall}{precision + recall}$$

where *precision* is the fraction of the number of correct positive results and the number of all positive results; *recall* is the number of correct positive results and the number of positive results that should be predicted. In case of multi-class classification problem, this is the weighted average of the F1-score of each class.

| | |
|---|---|
| Show me flights from Boston to New York. | |
| Frame: | FLIGHT |
| Slot/value: | FROMLOC.CITY = Boston |
| | TOLOC.CITY = New York |

For ATIS dataset, the F1-score was used as evaluation metrics, and it was calculated using CoNLL script[4]. A slot is perceived as a correction in case its range and type are correct. For DARPA dataset, performance was measured in terms of F1-score on slot/value pairs as in [33]. This is slightly different from CoNLL script in a way that after obtaining IOB tags, slot/value pairs will be filled in advance.

[2]http://deeplearning.net/software/theano/
[3]https://www.jaist.ac.jp/iscenter/en/mpc/cx250-cluster/
[4]http://www.cnts.ua.ac.be/conll2000/chunking/output.html

## 4.4 Baselines

For ATIS dataset, our model was compared to the following baselines:

**SVM:** *Raymond and Riccardi* [24] utilized heuristic combinations of forward and backward moving sequential SVMs classifiers for slot filling.

**CRF:** A CRF used n-gram as input and is considered as a baseline in [20]

**RNN:** Two RNNs models (Elman and Jordan) were employed in [20]. Here we only report the best result of outstanding Elman model.

**R-CRF:** R-CRF was proposed in [31]. However, we use the result in [20] due to the same configuration of the embedding layer.

For DARPA dataset, the comparison of our models to the following baselines:

**HVS:** *He and Young* [10] used HVS to model embedded structural context in sentences.

**CRF:** *Zhou and He* [32] proposed an iterative learning approach based on expectation maximization to train the CRFs from abstract semantic annotations.

**HM-SVMs:** *Zhou and He* [33] trained HM-SVMs model taking as input a training set of sentences labeled with abstract semantic annotations to map sentences to semantic meaning representations.

## 4.5 Results and Discussion

### 4.5.1 Distinct classifiers with and without Local Context Window

Table 4.1 illustrates the performance of distinct architectures with a variety of context-window size and without local context window. Overall, a model using a context window performs considerably better than the model does not, ranging frequently from 1 to 2.6% on both datasets. Undoubtedly, an effective representation of a current word could derive from not only its embedding vector but also vector representations of preceding and following words, which reflects the context a word immersing in. As a consequence, this information is beneficial on label forecast of the word of interest.

On the other hand, there is a small gap in the performance of four classifiers. While *BiLSTM-CRF* shows the highest F1-score for ATIS dataset (95.29), *BiLSTM-SVM* with 94.86 F1-score outperforms other models with different classification functions for DARPA dataset. We believe the reason is that the DARPA dataset has a great number of short sentences, hence SVMs are able to determine easily a precise label of each word. For longer and more complex sentences, CRFs shows its potential when taking into account tight constraints between contiguous labels in order to eliminate false cases.

Table 4.1: F1-score of various classification functions with and without context window. The best context window size hyper-parameter is reported as the number in the round brackets. Only lexical features were used.

| **Model** | **Local Context Window** | **Datasets** | |
|---|---|---|---|
| | | **ATIS** | **DARPA** |
| BiLSTM, sigmoid | NO | 92.62 | 93.14 |
| | YES | 94.53(7) | 94.36(3) |
| BiLSTM, softmax | NO | 93.28 | 92.9 |
| | YES | 94.98(7) | 94.5(7) |
| BiLSTM, SVM | NO | 93.45 | 92.23 |
| | YES | 94.89(7) | **94.86**(7) |
| BiLSTM, CRF | NO | 95.08 | 94.54 |
| | YES | **95.29**(3) | 94.77(9) |

### 4.5.2 Pre-trained and Randomly Initial Word Embedding

In our experiment, we deployed a variety of initial word embeddings, ranging from random values to pretrained vectors Senna, Glove or Structured Skipngram (S-Skip)[18]. As can be seen from Table 4.2 and Table 4.3, while SENNA was trained and released 130000-word vectors with 50 dimensions, Glove and S-Skip published 100-dimensional vector representations on larger vocabulary, 400000 and 243003 respectively. Generally speaking, all models in which the word embedding parameters were tuned during the training period exert better results in contrast to those with static parameters.

On ATIS dataset, as demonstrated in Table 4.2, the Glove corpus shares the most words with the ATIS corpus with only 10 Out-Of-Vocabulary (OOV) words, slightly less than SENNA one (20 OOV words). There are, however, 98 OOV words in S-Skip. Interestingly, the more words are OOV, the higher performance is. Obviously, directly learning the embedding vectors initialized from random values leads to higher results on the ATIS dataset (about 96.51 F1-score), even though pretrained word embeddings were tuned or static. Notwithstanding, such pre-trained vector representations are advantageous on DARPA dataset as illustrated in Table 4.3. In fact, the performance enhanced steadily and reached to the highest F1-score, approximately 95.17, when it comes to Glove corpus with the least OOV words (50). This outcome is slightly greater than the counterpart initializing with SENNA and S-Skip corpus: 95.05 and 94.96 respectively. These two corpora contain more OOV words, especially, 406 OOV words in S-Skip. Meanwhile, the word embedding weights initializing with uniformly random values gives the lowest F1-score, at 94.77.

Table 4.2: Pretrained Word Embedding with and without tuning. Model BiLSTM-CRF used lexical and named entity features on ATIS dataset. The best performances were reported

| **Word Embedding** | **Dimension** | **Vocabulary** | **OOV** | **F1-score** |
|---|---|---|---|---|
| Senna – tuning | 50 | 130000 | 20 | 96.24 |
| Senna – static | | | | 96.2 |
| Glove – tuning | 100 | 400000 | 10 | 96.22 |
| Glove – static | | | | 96.27 |
| S-Skip – tuning | 100 | 243003 | 98 | 96.39 |
| S-Skip – static | | | | 96.41 |
| Random – tuning | 100 | 572 | – | **96.51** |

Consequently, in our experiment, the word embedding weights were initialized with uniformly random values on ATIS dataset, and with pre-trained Glove vectors on DARPA dataset.

### 4.5.3 Compare to other methods

The results on ATIS dataset are summarized in Table 4.4. Obviously, CRF outperforms SVM due to its objective function is compatible with sequence labeling task. Notwithstanding, RNN releases better performance than CRF's due to (1) the ability of capturing long-term dependencies, and (2) automatically generating features instead of feature engineering in CRF. The state-of-the-art model is the combination of RNN and CRF, in which the output of RNN is used as features for CRF, whilst CRF makes solid constraints among final labels. Our model *BiLSTM-CRF* beats R-CRF due to the fact that LSTM can solve the vanishing gradient problem of RNN. Other classification functions (sigmoid, SVM, and softmax) obtain lower results than RNN (95.06 F1-score) in [20] since that RNN variant used information from the previously predicted label when forecasting the label of the current word. It looks like a way to create a constraint between the labels of contiguous words. When named entity features were integrated into all models, generally speaking, the performances increase significantly due to the high relevance of named entity and slot labels. Our model with a bunch of different classification functions outperforms the state-of-the-art model R-CRF (94.46), in particular, *BiLSTM-Softmax* gains the highest F1-score (96.62).

For the DARPA Communicator dataset, only lexical features were used as input to our model and word embedding weights were initialized with Glove pre-trained vectors. To the best of our knowledge, HM-SVMs is the state-of-the-art model

Table 4.3: Pretrained Word Embedding with and without tuning. Model BiLSTM-CRF used lexical feature on DARPA dataset. The best performances were reported

| **Word Embedding** | **Dimension** | **Vocabulary** | **OOV** | **F1-score** |
|---|---|---|---|---|
| Senna – tuning | 50 | 130000 | 58 | 95.05 |
| Senna – static | | | | 94.43 |
| Glove – tuning | 100 | 400000 | 50 | **95.17** |
| Glove – static | | | | 94.74 |
| S-Skip – tuning | 100 | 243003 | 406 | 94.96 |
| S-Skip – static | | | | 94.59 |
| Random – tuning | 100 | 1069 | – | 94.77 |

Table 4.4: Comparison to previous methods on ATIS dataset

| **Model** | | **Feature** | **F1-score** |
|---|---|---|---|
| SVM | | W | 89.76 |
| | | W + NE | – |
| CRF | | W | 92.94 |
| | | W + NE | 95.16 |
| RNN | | W | 95.06 |
| | | W + NE | 96.24 |
| R-CRF | | W | – |
| | | W + NE | 96.46 |
| BiLSTM | sigmoid | W | 94.53 |
| | | W + NE | 96.5 |
| | softmax | W | 94.98 |
| | | W + NE | **96.62** |
| | SVM | W | 94.87 |
| | | W + NE | 96.56 |
| | CRF | W | **95.29** |
| | | W + NE | 96.51 |

Table 4.5: Comparison to previous methods on DARPA dataset

| **Model** | | **F1-score** |
|---|---|---|
| HVS | | 87.97 |
| CRF | | 92.37 |
| HM-SVMs | | 93.18 |
| BiLSTM | sigmoid | 94.29 |
| | softmax | 94.61 |
| | SVM | 94.83 |
| | CRF | **95.17** |

on this dataset with 93.18 F1-score (see Table 4.5). It is obvious that all of our architectures gains considerably higher results ranging from 1.11% to nearly 2%, in which *BiLSTM-CRF* yields the best performance, 95.17. Using SVMs as a classifier at the final layer exerts the second highest F1-score with 94.83. Such results prove that deep learning, particularly, bidirectional LSTM in our model, outperforms other conventional approaches in SL task with a significant improvement.

## 4.6 Error Analysis

### 4.6.1 Classification function errors

Given an input sentence *"Please list the ground transportation from <UNK> into New York City"*, Table 4.6 demonstrates the output of four distinct classification functions. All methods predicted a label *B-fromloc.airport_code* for a word <UNK>, even though the ground truth label is *B-airport_ code*. However, we believe that since its previous word is *from*, the predicted label *B-fromloc.airport_code* is more reasonable than the ground truth one. About three continuous words *New York City*, only *Softmax* and *SVM* could classify correctly, whereas *Sigmoid* predicted definitely incorrect. The forecast labels of CRF, from our point of view, are also compatible with this context because the passenger want to travel to New York city. Personally, the labels for three words *New York city* should contain an indicator *toloc*. Consequently, this ambiguity influences more or less on the performance of the CRF model (see Table 4.4).

### 4.6.2 RNNs and Bidectional LSTM

As mentioned before, the advantage of LSTMs is that it is able to capture long dependency. Table 4.7 illustrates an instance in which BiLSTM outperform RNNs on ATIS dataset. Given a sentence "*Find me a flight from Cincinnati to any airport in*

Table 4.6: Forecast labels of four classification functions. Both lexical and named entity features were used in this case.

| **Sentence** | **Ground truth** | **Classification functions** | | | |
|---|---|---|---|---|---|
| | | **Sigmoid** | **Softmax** | **SVM** | **CRF** |
| Please | O | O | O | O | O |
| list | O | O | O | O | O |
| the | O | O | O | O | O |
| ground | O | O | O | O | O |
| transportation | O | O | O | O | O |
| from | O | O | O | O | O |
| <UNK> | B-airport_code | B-fromloc. airport_code | B-fromloc. airport_code | B-fromloc. airport_code | B-fromloc. airport_code |
| into | O | O | O | O | O |
| new | B-city_name | B-fromloc. city_name | B-city_name | B-city_name | B-toloc. city_name |
| york | I-city_name | I-fromloc. city_name | I-city_name | I-city_name | I-toloc. city_name |
| city | I-city_name | I-fromloc. city_name | I-city_name | I-city_name | I-toloc. city_name |

*the New York city area*", our models and R-CRF are all forecast precisely *Cincinnati* as a departure city with the label *B-fromloc.city_name*. It is a straightforward circumstance because the word *Cincinnati* appears immediately after the word *from*. However, there is an enormous difference in predicted labels for the arrival city *New York city*. Whilst R-CRF model anticipates incorrectly *New York city* as a departure city, our models using Softmax and CRF classification can recognize it exactly as an arrival city. Intuitively, the word *to* is far from the phrase *New York city* so that it is a complex case for R-CRF. On the other hand, our models using LSTMs can tackle that situation and give proper anticipations.

### 4.6.3 DARPA dataset errors

In contrast to ATIS dataset which is labeled greatly for both training and test set, DARPA dataset is more complicated. The label of a training utterance is in hierarchical semantic relationship form [10]. Although IOB annotation is simply associating the appropriate semantics with each training utterance and does not require any linguistic skills, some of the relationships have not matched as two

Table 4.7: Comparison of R-CRF and our models on ATIS dataset. Lexical and named entity were used.

| **Sentence** | **Ground truth** | **Methods** | | |
|---|---|---|---|---|
| | | **R-CRF** | **BiLSTM** | |
| | | | **Softmax** | **CRF** |
| find | O | O | O | O |
| me | O | O | O | O |
| a | O | O | O | O |
| flight | O | O | O | O |
| from | O | O | O | O |
| cincinnati | B-fromloc. city_name | B-fromloc. city_name | B-fromloc. city_name | B-fromloc. city_name |
| to | O | O | O | O |
| any | O | O | O | O |
| airport | O | O | O | O |
| in | O | O | O | O |
| the | O | O | O | O |
| new | B-toloc. city_name | B-fromloc. city_name | B-toloc. city_name | B-toloc. city_name |
| york | I-toloc. city_name | I-fromloc. city_name | I-toloc. city_name | I-toloc. city_name |
| city | I-toloc. city_name | I-fromloc. city_name | I-toloc. city_name | I-toloc. city_name |
| area | O | O | O | O |

instances shown in Table 4.8. For instance, the $11084^{th}$ abstract annotation has an only *DAY_NUMBER* label whilst there are three possible tokens corresponding to this label such as second, twenty or twenty second. In our case, we only recognize the first word *second* as *B-DAY_NUMBER*. For this reason, the IOB tags for several training utterances were not annotated perfectly. However, we still kept them in training set to ensure the objectivity.

Table 4.8: Examples of a mismatch between transcriptions and annotations

| **Index** | **Sentence (upper) and Abstract (lower)** |
|---|---|
| 4261 | i would like to fly from denver to san diego and then from san diego to new york and from new york back to denver starting from on the |
| | FROMLOC(CITY_NAME) TOLOC(CITY_NAME CITY_NAME) CITY_NAME FROMLOC(CITY_NAME) TOLOC(CITY_NAME) |
| 11084 | november second twenty second |
| | MONTH_NAME(DAY_NUMBER) |

# Chapter 5

# Conclusion and future work

## 5.1 Conclusion

In this paper, we have proposed LSTMs which can be merged successfully with a set of classification functions for SLU slot filling task. Word Embedding is the first layer of our model, and its purpose is to map each word in a sample sentence into a contiguous representation in contrast with a traditional one-hot vector. Afterward, bidirectional LSTMs are utilized for capturing implicit features of that sentence in both forward and backward path. The reason for using LSTMs is that they can tackle the detrimental issue in RNNs: vanishing gradients. Finally, the top classification function namely Sigmoid, Softmax, SVMs or CRFs, takes those features as its input in order to anticipate all sequential words accordingly.

We carried out the experiments on two common datasets ATIS and DARPA Communicator. In contrast to the relatively good labeled ATIS dataset in IOB tags, DARPA data is in a more sophisticated form (utterance transcriptions and the semantic parse). It is, hence, essential to transform it into a simpler format IOB before tagging. Besides, the word embedding parameters were initialized with pre-trained word vectors on the latter dataset and with uniformly random values on the former one.

The results indicate that our model architectures obtain higher performance than that of the state-of-the-art model though there is a small difference among four classification functions. In particular, BiLSTM-CRF yields the highest performance on both datasets if using only lexical features. This reflects the benefits of the sequence-level discrimination ability of CRF and the feature engineering of bidirectional LSTM networks. On the other hand, auxiliary features in ATIS dataset contributes greatly to performance since those features nearly identify a label for an individual word. In this case, BiLSTM-Softmax is the best model.

## 5.2 Future work

For future work, it is worth taking into account several errors when converting to IOB tasks for the training set, which is more likely to improve performance considerably. Furthermore, beyond sequence tagging for all words in discrete sentences, we would focus more on slot filling task in a dialog system, which is, without doubt, more appealing and challenging. In that system, the content of a conversation and the intent of customers are two vital points which should be captured precisely. To do this, it is essential to investigate other crucial aspects of NLP accordingly such as textual entailment, texture extraction. Besides, other supported techniques namely Part-Of-Speech Tagging, chunking, to name but a few, more or less could accelerate the performance of our model.

# References


[1] A beginners guide to recurrent networks and lstms. URL `https://deeplearning4j.org/lstm`.

[2] Understanding lstm networks, 2015. URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[3] Recurrent neural networks tutorial, part 3 - backpropagation through time and vanishing gradients, 2015. URL `http://www.wildml.com/2015/10/`.

[4] Hyeran Byun and Seong-Whan Lee. Applications of support vector machines for pattern recognition: A survey. In *Pattern recognition with support vector machines*, pages 213–236. Springer, 2002.

[5] Adam Coates, Honglak Lee, and Andrew Y Ng. An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor*, 1001(48109):2, 2010.

[6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[7] Renato De Mori, Frédéric Bechet, Dilek Hakkani-Tur, Michael McTear, Giuseppe Riccardi, and Gokhan Tur. Spoken language understanding. *IEEE Signal Processing Magazine*, 25(3):50–58, 2008.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[9] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[10] Yulan He and Steve Young. Semantic processing using the hidden vector state model. *Computer speech & language*, 19(1):85–106, 2005.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[13] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[15] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289, 2001.

[16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[17] Quoc V. Le, Jiquan Ngiam, Zhenghao Chen, Pang We i Koh Daniel Chia, and Andrew Y. Ng. Tiled convolutional neural networks. In *Advances in neural information processing systems*, pages 1279–1287, 2010.

[18] Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, 2015.

[19] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*, pages 3771–3775, 2013.

[20] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3): 530–539, 2015.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[22] Jawad Nagi, Gianni A Di Caro, Alessandro Giusti, Farrukh Nagi, and Luca M Gambardella. Convolutional neural support vector machines: hybrid visual

pattern classifiers for multi-robot systems. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 1, pages 27–32. IEEE, 2012.

[23] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43, 2014.

[24] Christian Raymond and Giuseppe Riccardi. Generative and discriminative algorithms for spoken language understanding. In *INTERSPEECH*, pages 1605–1608, 2007.

[25] Sheng-syun Shen and Hung-yi Lee. Neural attention models for sequence classification: Analysis and application to key term extraction and dialogue act detection. *arXiv preprint arXiv:1604.00077*, 2016.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[27] Yichuan Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[28] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[29] Ye-Yi Wang, Li Deng, and Alex Acero. Spoken language understanding. *IEEE Signal Processing Magazine*, 22(5):16–31, 2005.

[30] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. Recurrent neural networks for language understanding. In *INTERSPEECH*, pages 2524–2528, 2013.

[31] Kaisheng Yao, Baolin Peng, Geoffrey Zweig, Dong Yu, Xiaolong Li, and Feng Gao. Recurrent conditional random field for language understanding. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4077–4081. IEEE, 2014.

[32] Deyu Zhou and Yulan He. Learning conditional random fields from unaligned data for natural language understanding. In *European Conference on Information Retrieval*, pages 283–288. Springer, 2011.

[33] Deyu Zhou and Yulan He. A novel framework of training hidden markov support vector machines from lightly-annotated data. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2025–2028. ACM, 2011.

# Publications

### Minor research

[1] **Do, K. P.**, Nguyen, B. T., Nguyen, X. T., Bui, Q. H., Tran, N. L., Nguyen, T. N. T., Vuong, V. Q., Nguyen, H. L. and Le, T. H., 2015. Spatial Interpolation and Assimilation Methods for Satellite and Ground Meteorological Data in Vietnam. Journal of Information Processing Systems. 11, 4, 556–572.

[2] Nguyen, X. T., Nguyen, B. T., **Do, K. P**., Bui, Q. H., Nguyen, T. N. T., Vuong, V. Q. and Le, T. H., 2015. Spatial Interpolation of Meteorological Variables in Vietnam using the Kriging Method. Journal of Information Processing Systems. 11, 1, 134–147.

### Other research

[3] **Do, P.K.**, Nguyen, H.T. , Tran, C.X., Nguyen, M.T., and Nguyen, M.L. (2016). Legal Question Answering using Learning to Rank and Deep Learning. Accepted at International Workshop on Juris-informatics (JURISIN)

[4] Nguyen, M.T., Lai, V.D., **Do, P.K.**, Tran, D.V., and Nguyen M.L. (2016). VSoLSCSum: Building a Vietnamese Sentence-Comment Dataset for Social Context Summarization . Accepted at the $26^{th}$ International Conference on Computational Linguistics (COLING)

# Appendix

Our model is modified greatly from this source[1] and this source[2]. In this section, we provide codes of three main layers implemented in THEANO: a word embedding layer, a hidden layer and a LSTM layer. Intuitively, it is easy to construct a model by stacking such layers.

```
import theano
import theano.tensor as T

def shared(shape, name):
    """
    Create a shared object of a numpy array.
    """
    if len(shape) == 1:
        value = np.zeros(shape) # bias are initialized with zeros
    else:
        drange = np.sqrt(6. / (np.sum(shape)))
        value = drange * np.random.uniform(low=-1.0, high=1.0, size=shape)
    return theano.shared(value=value.astype(theano.config.floatX),
        name=name)

class EmbeddingLayer(object):
    """
    Embedding layer: word embeddings representations
    Input: tensor of dimension (dim*) with values in range(0, input_dim)
    Output: tensor of dimension (dim*, output_dim)
    """
    def __init__(self, input_dim, output_dim, name='embedding_layer'):
        """
        Typically, input_dim is the vocabulary size,
        and output_dim the embedding dimension.
        """
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.name = name

        # Randomly generate weights
        self.embeddings = shared((input_dim, output_dim), self.name +
            '__embeddings')
        # Define parameters
        self.params = [self.embeddings]
    def link(self, input):
        """
        Return the embeddings of the given indexes.
```

[1] https://github.com/glample/tagger
[2] https://github.com/mesnilgr/is13

```
        Input: tensor of shape (dim*)
        Output: tensor of shape (dim*, output_dim)
        """
        self.input = input
        self.output = self.embeddings[self.input]
        return self.output
```

---

```
class HiddenLayer(object):
    """
    Hidden layer with or without bias.
    Input: tensor of dimension (dims*, input_dim)
    Output: tensor of dimension (dims*, output_dim)
    """
    def __init__(self, input_dim, output_dim, bias=True, activation=
        'sigmoid', name='hidden_layer'):
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.bias = bias
        self.name = name
        if activation is None:
            self.activation = None
        elif activation == 'tanh':
            self.activation = T.tanh
        elif activation == 'sigmoid':
            self.activation = T.nnet.sigmoid
        elif activation == 'softmax':
            self.activation = T.nnet.softmax
        else:
            raise Exception("Unknown activation function: " % activation)

        # Initialize weights and bias
        self.weights = shared((input_dim, output_dim), name + '__weights')
        self.bias = shared((output_dim,), name + '__bias')

        # Define parameters
        if self.bias:
            self.params = [self.weights, self.bias]
        else:
            self.params = [self.weights]
    def link(self, input):
        """
        The input has to be a tensor with the right
        most dimension equal to input_dim.
        """
        self.input = input
        self.linear_output = T.dot(self.input, self.weights)
```

```
        if self.bias:
            self.linear_output = self.linear_output + self.bias
        if self.activation is None:
            self.output = self.linear_output
        else:
            self.output = self.activation(self.linear_output)
        return self.output
```

---

```
class LSTM(object):
    """
    Long short-term memory (LSTM). Can be used with or without batches.
    Without batches:
    Input: matrix of dimension (sequence_length, input_dim)
    Output: vector of dimension (output_dim)
    With batches:
    Input: tensor3 of dimension (batch_size, sequence_length, input_dim)
    Output: matrix of dimension (batch_size, output_dim)
    """
    def __init__(self, input_dim, hidden_dim, with_batch=True,
        name='LSTM'):
        """
        Initialize neural network.
        """
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.with_batch = with_batch
        self.name = name

        # Input gate weights
        self.w_xi = shared((input_dim, hidden_dim), name + '__w_xi')
        self.w_hi = shared((hidden_dim, hidden_dim), name + '__w_hi')

        # Forget gate weights
        self.w_xf = shared((input_dim, hidden_dim), name + '__w_xf')
        self.w_hf = shared((hidden_dim, hidden_dim), name + '__w_hf')
        # Output gate weights
        self.w_xo = shared((input_dim, hidden_dim), name + '__w_xo')
        self.w_ho = shared((hidden_dim, hidden_dim), name + '__w_ho')

        # Cell weights
        self.w_xc = shared((input_dim, hidden_dim), name + '__w_xc')
        self.w_hc = shared((hidden_dim, hidden_dim), name + '__w_hc')

        # Initialize the bias vectors, c_0 and h_0 to zero vectors
        self.b_i = shared((hidden_dim,), name + '__b_i')
        self.b_f = shared((hidden_dim,), name + '__b_f')
        self.b_c = shared((hidden_dim,), name + '__b_c')
```

```
        self.b_o = shared((hidden_dim,), name + '__b_o')
        self.c_0 = shared((hidden_dim,), name + '__c_0')
        self.h_0 = shared((hidden_dim,), name + '__h_0')

        # Define parameters
        self.params = [self.w_xi, self.w_hi,
        self.w_xf, self.w_hf,
        self.w_xo, self.w_ho,
        self.w_xc, self.w_hc,
        self.b_i, self.b_c, self.b_o, self.b_f,
        self.c_0, self.h_0]

    def link(self, input):
        """
        Propagate the input through the network and return the last hidden
        vector. The whole sequence is also accessible via self.h, but
        where self.h of shape (sequence_length, batch_size, output_dim)
        """
        def recurrence(x_t, c_tm1, h_tm1):
            i_t = T.nnet.sigmoid(T.dot(x_t, self.w_xi) + T.dot(h_tm1,
                self.w_hi) + self.b_i)
            f_t = T.nnet.sigmoid(T.dot(x_t, self.w_xf) + T.dot(h_tm1,
                self.w_hf) + self.b_f)
            o_t = T.nnet.sigmoid(T.dot(x_t, self.w_xo) + T.dot(h_tm1,
                self.w_ho) + self.b_o)
            Ce_t = T.tanh(T.dot(x_t, self.w_xc) + T.dot(h_tm1, self.w_hc)
                + self.b_c)
            c_t = i_t * Ce_t + f_t * c_tm1
            h_t = o_t * T.tanh(c_t)
            return [c_t, h_t]

        # If we use batches, we have to permute the first and second
            dimension.
        if self.with_batch:
            self.input = input.dimshuffle(1, 0, 2)
            outputs_info = [T.alloc(x, self.input.shape[1],
                self.hidden_dim)
                           for x in [self.c_0, self.h_0]]
        else:
            self.input = input
            outputs_info = [self.c_0, self.h_0]

        [_, h], _ = theano.scan(
                               fn=recurrence,
                               sequences=self.input,
                               outputs_info=outputs_info,
                               n_steps=self.input.shape[0])
```

```
        self.h = h
        self.output = h[-1]
        return self.output
```