TECHNICAL UNIVERSITY OF CRETE

MASTER THESIS

Statistical Methods for Dialogue Systems

Author:
Despoina GEORGIADOU

Supervisor:
Professor Vassilis DIGALAKIS

A thesis submitted in fulfillment of the requirements for the degree of M.Sc. in Electrical and Computer Engineering

in the

School of Electrical and Computer Engineering

July 28, 2017

Declaration of Authorship

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

M.Sc. in Electrical and Computer Engineering

Statistical Methods for Dialogue Systems

by Despoina GEORGIADOU

The ideal Dialogue Systems would be natural and error-less human-machine conversational understanding systems. Aiming to this, Spoken Language Understanding (SLU) systems firstly focus on identifying the requests of the users as naturally expressed in their own language. When such a machine serves a user, semantics are automatically extracted by the human utterance, the intention is comprehended and then the appropriate actions are taken in order to satisfy the user's requests. An ubiquitous and challenging task in Spoken Language Understanding is slot filling (SF). By slot we refer to one tag or label for each word. Thus, the slot filling task is the procedure of taking an input as a sequence of words, and resulting in an output as a sequence of slots. This can be considered as sequence classification problem as the slot which corresponds to a word can be considered as the class in which the word is classified. Slots are related not only to the domain, but also to one another, so we should define a sequence of labels jointly for the tested utterance. Originally, the slot filling task was managed with conditional random field (CRF), which were later substituted by neural networks. The latter approach with a wide variety of variants, such as recurrent neural networks (RNNs), significantly outperformed the former approach. In theory, RNNs are really promising when it comes to temporal dependencies. In practice, they cannot memorize long-term dependencies that relate the current-time semantic label prediction to the observations many time instances away, due to limited memory and consequently they tend to be hard to train. To tackle this issue many approaches have been proposed which eventually tend to be more complex. An interesting alternative architecture is the Clockwork RNN (CW-RNN). This variant is less complex and at the same time the overall neural network is faster thanks to the less parameters that should be evaluated. In this study we explore the Clockwork RNN and some variants on the ATIS data resource. In this work all the CW-RNNs are implemented with the use of the publicly available Theano neural network toolkit. Through our experiments we observed that the tested architectures are able to enhance the performance of the simple RNN.

Index Terms: slot filling (SF), spoken language understanding (SLU), clockwork recurrent neural network (CW-RNN)

Acknowledgements

I would like to take some time here and thank the people who made the completion of this work possible.

First of all, I would like to express my sincere thanks to my thesis supervisor, Professor Digalakis Vasilis, for assigning this work to me.

Many thanks also go to Mr Diakoloukas Vasilis and Mr Tsiaras Vasilis who provided their experience and knowledge whenever needed. I deeply appreciate all their guidance and support throughout this work.

Finally, I need to express my deepest gratitude to my best friend for all the patience and mostly for offering me emotional support and encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

SLU	Spoken Language Understanding
SF	Slot Filling
CRFs	Conditional Random Fields
IOB	In/Out/Begin representation
ATIS	Airline Travel Information System
ANNs	Artificial Neural Networks
RNN	Recurrent Neural Network
CW-RNN	ClockWork Recurrent Neural Network
VAD	Voice Activity Detection
ASR	Automatic Speech Recognition
DM	Dialog Manager
NLG	Natural Language Generation
TTS	Text-To-Speech
NLL	Negative Log-Likelihood Loss
SGD	Stochastic Gradient Descent
MSGD	Mini-batch Stochastic Gradient Descent
SVMs	Support Vector Machines
triCRFs	triangular Conditional Random Fields
LSTM	Long Short-Term Memory
RNN-SOP	Structured Output Prediction Recurrent Neural Network
RNN-EM	Recurrent Neural Network-External Memory
RSVM	Recurrent Support Vector Machines
biCW-RNN	Bi-directional ClockWork Recurrent Neural Network
R-biRNN	Bi-directional Recurrent Neural Network with ranking loss

Chapter 1

Introduction

In this chapter we introduce Spoken Language Understanding and we focus on one specific task of it which is the Slot Filling Task. After that, we describe both the objective of this work and how it is organized here.

1.1 Spoken Language Understanding (SLU)

Spoken Language Understanding (SLU) is an emerging field in between speech and language processing, investigating human/machine and human/human communication by leveraging technologies from signal processing, pattern recognition, machine learning and artificial intelligence. SLU aims to extract the meaning of the speech utterances(spoken phrases-we use the word utterance rather than sentence when referring to spoken language [3]) and automatically identify the intent of the user as expressed. The term SLU has largely been coined for understanding of human speech directed at machines and it is widely believed that the number of SLU applications will increase in the future since its applications are vast, from voice search in mobile devices to meeting summarization, attracting interest from both commercial and academic sectors. Several elements that are necessary in human-computer dialog are described in [1].

There are three categories of understanding systems [2]. The first one is about artificial intelligence systems that mimic understanding [Eliza, MIT 1966]. The second category contains systems rooted in artificial intelligence, which are successful for very limited domains, using deeper semantics. The third category is systems where understanding is reduced to a mostly statistical language processing problem and it is our main focus in this work. In the last case, targeted speech understanding tasks are tried to be solved instead of the global machine understanding problem. In targeted speech understanding tasks the problem of understanding a user's utterance is reduced to the problem of extracting specific arguments in a given frame-based semantic representation. Today there are many targeted speech understanding systems.

1.2 The Slot Filling (SF) task

Slot filling in the framework of spoken language understanding (SLU) is a task whose job is to derive semantic components from a human spoken utterance. The slot filling task is different from intent determination which is another significant SLU task. The former aims to define a sequence of labels jointly for the tested utterance, whereas the latter defines one label for the

TABLE 1.1: IOB representation example

I	want	to	fly	from	Chania	to	Athens	next	week
O	O	O	O	O	B-dept	O	B-arr	B-date	I-date

whole utterance and thus it is a standard classification problem.

Nowadays, there is a huge interest in automated systems which support human-machine dialogue. To accomplish such a tremendous achievement a major task in spoken language understanding ought to be addressed. This is the filling of slots embedded in a semantic frame which is actually a really challenging task, since it requires the extraction of complex semantics.

For instance, providing the user's utterance is "I want to fly from Chania to Athens next week" (see TABLE 1.1), we need to determine the domain and the intent as well as fill the slots (give a semantic tag/label for each word). For simplicity and abstraction reasons, the Slot Filling Task transforms the sentence to the IOB format, which is the well-used in/out/begin representation. This results our utterance in becoming "O O O O O B-dept O B-arr B-date I-date" in terms of slots, while the Intent is <Find_Flight> and the domain is <Airline_Travel>.

It can be easily understood that the slots are closely related to the domain. In the example above words such as "want" or "I" are of no particular interest and for this reason no slot is filled or in other words the blank slot is filled. On the other hand, given the domain is <Airline Travel>, words like "Chania", which is a town, are of extremely high significance since the destination and the departure of a route are obviously some of the most essential information to be tagged. Furthermore, the date of the flight that the user attempts to find is a necessary slot to be filled as well. It is also worth noticing that the slots are related not only to the domain, but also to one another. More specifically, the word "next" in our example, is nearly meaningless if seen on its own. However, if we observe the word "next" before the word "week" then it turns to be a part of the tag date and the slot definitely should be filled. However, at the highest level of complexity for slot filling, we should also capture the slot-concept dependencies beyond the context word window that captures short-term dependencies.

Consequently statistical spoken language understanding techniques need to be implemented in order to achieve proper slot filling. In the past, before the extensive use of neural networks, a wide range of methods had been used. Some of the most used models are hidden Markov models, discriminative classification methods such as CRFs, knowledge-based methods, and probabilistic context free grammars. The most successful approach, before the use of neural networks for slot filling was the conditional random fields (CRFs).

1.3 Objective of this work

The slot filling (SF) task, described in previous section, is an essential issue in spoken language understanding (SLU). In recent years, artificial neural networks are on the spotlight of most researches on the SF task [23], thus plenty of neural network technologies have been applied aiming to handle this matter.

In this work, we implemented and compared a variety of recurrent-neural-network architectures. The objective was to investigate the performance of techniques that have never been used on the ATIS benchmark (the airline travel information system data resource). With our work, we tried to study the mystery called neural networks and how these seemingly simple structures manage to deal with so complex tasks such as finding deep dependencies among words in spoken language. We also hope to provide new ideas and motivation for extended research and development of simple, yet effective models.

Starting with some coverage of Spoken Language Understanding (SLU) we focus on Artificial Neural Networks (ANNs) and we provide knowledge on the state-of-the-art about the slot filling task. The novelty of this work is a ClockWork Recurrent Neural Network (CW-RNN) and the variant neural networks that are applied in order to accomplish our classification task. Statistical information about the dataset we used to implement this work is provided. A range of experiments were carried out in order to test the performance of all the algorithms we implemented. The extracted results are offered in tables and figures to show that they are comparative to the state-of-the-art techniques.

1.4 Outlook of the contents

This work is organized in five chapters:

- Chapter 2:
  – Preliminaries: This chapter provides a brief, comprehensive though coverage of Spoken Language Understanding (SLU). Then, focusing on Artificial neural networks, we demonstrate the main architectures on which we based our work. Details about learning, optimization and testing are provided as well.
- Chapter 3:
  – State of the art : Being more specific about the slot filling task, the main established approaches are covered. For each approach, we describe the algorithms used in order to provide the reader with a comprehensive view of the state of the art in this area. In addition, the novelty of this work is described, in order to compare and contrast with the state-of-the-art techniques.
- Chapter 4:
  – Clockwork Recurrent Neural Networks (CW-RNN) for Slot Filling: This chapter shows our proposed approach to the Slot Filling task. The classifier we build is a Clockwork Recurrent Neural Network and, since it results in good performance, we aim to provide details of each stage of the procedure. Consequently, we firstly describe the turning of the training data-set into vectors, analyzing the specific techniques used. Then, we cover extensively the variant neural networks that are applied in order to accomplish our classification task.
- Chapter 5:
  – Datasets and Experiments : This chapter offers all the necessary statistical information about the data-set we used to implement this work and test the performance of all the algorithms. In particular, we used an American-English database. A class description of the data-set is also included. It also focuses on the experiments we carried out in order to test the performance of all the algorithms we implemented. The extracted results as well as comparative tables and figures are offered for a deeper comprehension.
- Chapter 6:
  – Conclusion : This chapter summarizes and features the results and the conclusions from our work. In addition to this, possible future work is proposed.

Chapter 2

Preliminaries

2.1 Spoken Dialogue Systems

A Spoken Dialog System is a system where dialogues are delivered through voice. It denotes a wide range of systems, from weather information systems (for instance MIT Jupiter weather information) to complex problem solving, reasoning, applications. Some applications of spoken language systems are automatic call routing, answering questions about weather or sports, travel planning, tutoring systems, applications within games and many others.

2.1.1 Architecture

The objective of a spoken dialogue system is to let a human interact with a computer-based information system using speech as means of interaction. Because a spoken dialogue system can make errors and user requests can be incomplete or ambiguous, a spoken dialogue system must actively request for missing data and confirm information, resulting in a sequence of interactions between a user and a machine. This turn-based sequence of interactions between a user and a machine is called a dialogue. Figure 2.1 illustrates the architecture of a spoken language system.

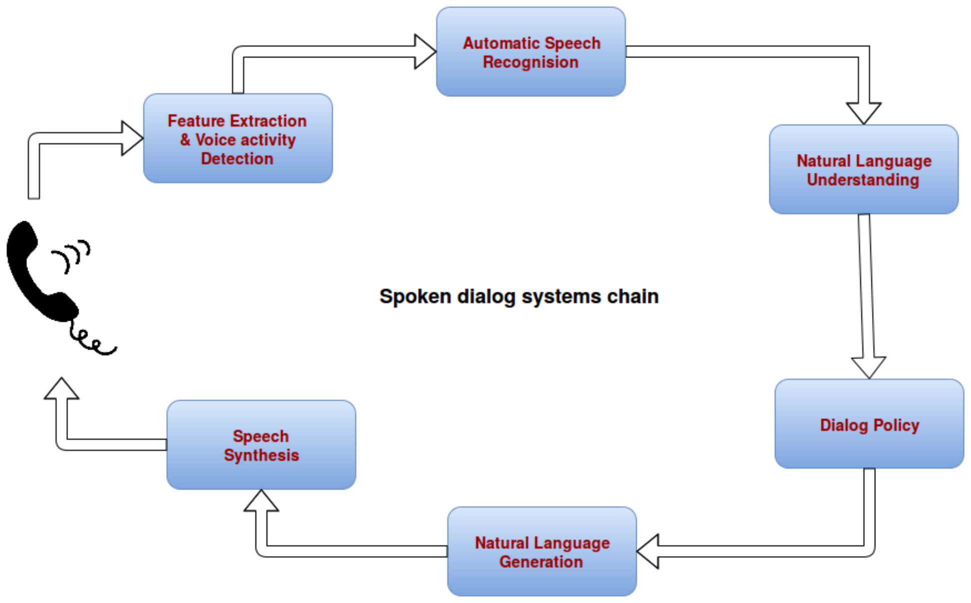
Automatic Speech Recognition
Feature Extraction & Voice activity Detection
Natural Language Understanding
Spoken dialog systems chain
Dialog Policy
Speech Synthesis
Natural Language Generation

FIGURE 2.1: Spoken dialog systems chain

One example of a dialogue between a human (user) and a machine (system) could be a telephone directory assistance task. In this example, the user can ask for telephone numbers, fax numbers, and email addresses of persons, departments, and companies. After each user input, the system must select an appropriate dialogue action and respond to the user in a cooperative way such that finally the user's request can be answered. Spoken dialog systems divide the complex task of conversing with the user into more specific subtasks handled by specialized components: voice activity detection, speech recognition, natural language understanding, dialog management, natural language generation, and speech synthesis. These components are usually organized in a pipeline as shown in Figure 2.1, where each component processes the result of the preceding one and sends its result to the next one. The following sections give a brief overview of each component and the typical issues they face.

In Figure 2.1, a user s spoken utterance is taken and transformed into a textual hypothesis of the utterance. This hypothesis is parsed and a semantic representation of the utterance is generated. This representation is then handled by the dialogue policy and generates a response. The language generation component then generates a surface representation of the utterance, often in some textual form, and passes it to a text-to- speech synthesis which generates the audio output to the user.

2.1.2 Components

Spoken dialog systems divide the complex task of conversing with the user into more specific subtasks handled by specialized components. These components are usually organized in a pipeline as shown in Figure 2.1, where each component processes the result of the preceding one and sends its result to the next one. These may be the most important components of most dialogue systems [1]:

1. Feature Extraction and Voice Activity Detection
2. Speech Recognition
3. Natural Language Understanding
4. Dialog Policy
5. Natural Language Generation
6. Speech Synthesis

The following sections give a brief overview of each component and the typical issues they face.

Feature Extraction and Voice Activity Detection

Voice activity detection (VAD) is the problem of detecting in the incoming audio signal when the user speaks and when he/she does not. Features from the audio signal are extracted and classified as speech or non-speech. This apparently easy problem can in fact be extremely hard to solve accurately in noisy conditions and has been the focus of much research, particularly in the signal processing community.

Speech Recognition

The automatic speech recognition (ASR) module takes the speech audio data segmented by the VAD and generates its word-level transcription. In addition, the generated hypothesis is sometimes annotated at the word- or utterance-level with confidence scores. ASR engines rely on three models an acoustic model, which describes the mapping between audio data and phonemes, a lexicon, which describes the mapping between phoneme sequences and words, and a language model, which describes the possible (or likely) sequences of words in a language. The acoustic model needs to be trained on a corpus of transcribed utterances. The lexicon can be either trained as a set of letter-to-sound rules from a corpus of words and their pronunciation, or, more often, it can be written by hand. The language model can be either a hand-written grammar, or a statistical language model trained on a corpus of in-domain data. Most ASR engines are designed to process full utterances, where the definition of an utterance corresponds to a phrase or sentence. However, they usually perform recognition incrementally, as the user is speaking, and therefore can provide partial recognition results at any time.

Natural Language Understanding

The natural language understanding (NLU) module takes the sequence of words output by the ASR and generates a semantic representation of it. NLU can be performed by providing a hand-written grammar that captures semantic relationships (either directly from the words, or via a syntactic analysis). Another approach to NLU is to train a statistical parser on a corpus of sentences annotated with their semantic representation. Most NLU modules assume that they are given a full utterance. Again the definition of an utterance varies and depends on the grammar or corpus on which the module was built.

Dialog Policy

The dialog policy or dialog manager (DM) takes the semantic representation of the user input generated by the NLU and outputs the semantic representation of the system's response. While there are many approaches to dialog management, the DM generally performs (at least) the following three tasks: interpreting user input in the current dialog context, updating the dialog context based on user input, generating relevant system responses. The DM also exploits knowledge about the domain and task at hand, which are usually provided by some back end module such as a database or an expert system.

Natural Language Generation

The natural language generation module (NLG) takes the semantic representation of the system response and outputs a natural language expression of it. Simple and common approaches to NLG include canned text when there is little variation in system prompts and templates. More advanced approaches have been proposed, either based on linguistic concepts such as discourse structure [Wilcock and Jokinen, 2003] or using statistical mapping between the semantic representation and the surface form [Oh and Rudnicky, 2000]. The NLG can optionally annotate the surface form with mark up tags destined to help speech synthesis using a speech synthesis mark up language such as SSML or JSAPI. Such tags can indicate prosodic patterns such as rising or falling pitch, pauses, or emphasis.

Speech Synthesis

The speech synthesis, or text-to-speech module (TTS) takes the natural language output of the NLG (potentially augmented with mark up tags) and generates an audio waveform corresponding to its spoken version. The simplest way to perform TTS, and also the one that leads to the highest naturalness, is to use pre-recorded prompts. As for canned-text NLG, this can be used when there is little variation in the system prompts so that they can all be covered by a voice talent. General speech synthesis allows the system to say any text, even potentially unplanned ones (which is necessary when the system retrieves variable information from, say, a web site). The synthesized utterance is played back to the user through the output audio device.

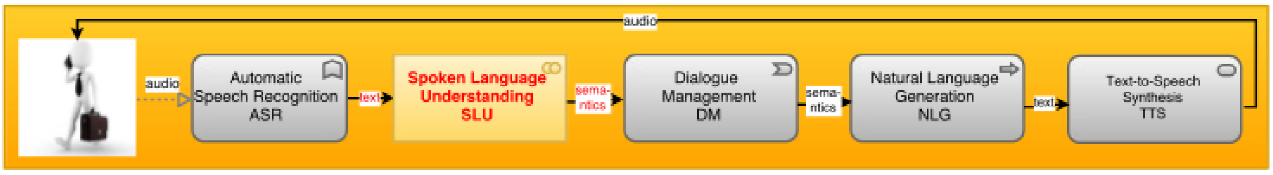
audio
audio
Automatic Speech Recognition ASR
text
Spoken Language Understanding SLU
sema-ntics
Dialogue Management DM
sema-ntics
Natural Language Generation NLG
text
Text-to-Speech Synthesis TTS

FIGURE 2.2: Dialogue System Architecture

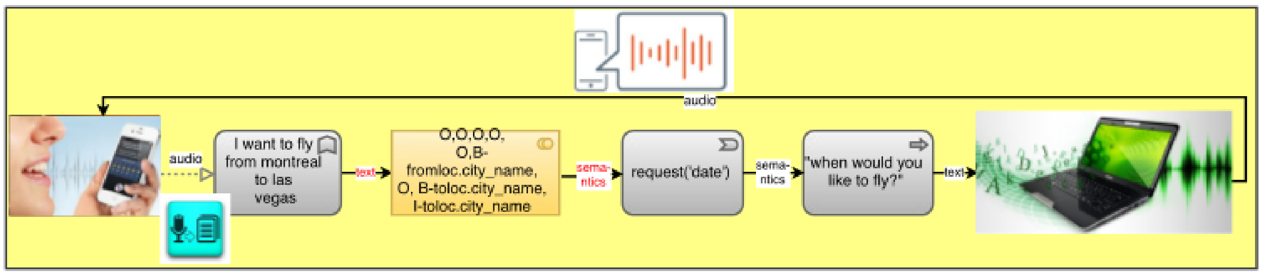
audio
audio
I want to fly from montreal to las vegas
text
O,O,O,O, O,B-fromloc.city_name, O, B-toloc.city_name, I-toloc.city_name
sema-ntics
request('date')
sema-ntics
"when would you like to fly?"
text

FIGURE 2.3: Dialogue System Architecture-example

2.2 Artificial neural networks

In this section we briefly introduce the Artificial neural networks [6]. A connection of simple processing units is called Artificial Neural Network or Neural Net or ANN. These processing units are called neurons. They take inputs, they are connected to other neurons to form the network and finally give the outputs. Every neuron performs a simple mathematical task, but as a whole these processors give a neural network the ability to achieve more complex accomplishments. Not all the connections are equally important. Each input is weighted by a weight w which represents the strength of the connection. The sum of the inputs and their weights is called the activation S of the neuron. Early neural nets used to result in binary outputs. However, it was soon discovered that a continuous output was more accurate and flexible. In this case the neurons obtained the ability to express uncertainty. The sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ is one of the most popular continuous functions used and it is called Activation Function. Other such functions are the linear, the logarithmic and the tangential functions.

The neural networks started to be used for pattern recognition in the late 1950s. Experiments that were conducted on neural networks demonstrated that they have the ability to generalize and at the same time they were extremely noise tolerant. These two things are the most essential facts about Neural Networks.

A typical neural network can learn or in other words it can be trained by changing the weights. The most usual and popular way to do so is the Back-propagation learning algorithm. The main idea is to adjust the weights depending on the output. For instance, if the output is what we want, then the weights remain unchanged. But if the output is too high or too low then the weights should be decreased or increased respectively. In order to decrease or increase the weights, a constant is added to the formula of each weight. This constant is called learning rate and its job is to speed up or slow down the learning process when needed.

During the training stage, it should be taken into consideration that the network should have the ability to recognize noisy or even corrupted patterns. To manage this, one idea is to add noisy versions of the input data in the training set. However, there is another interesting and better way. We still need the noisy versions of the input data but we do not add them in the training set. Instead we keep the apart as a different set called Validation set. Each time after training we use this validation set to calculate the error. When the error becomes low, the network stops. This stops the problem of over-training which means that the networks becomes too accurate on the data it is trained to. In case of over-training the network will not handle the noisy data well and the performance will consequently drop.

2.2.1 Feed-forward Neural Networks

A feed-forward neural network is a classification algorithm and it is the simplest form of neural networks. A number of simple units (neurons or nodes) are organized in layers and these units are connected with all the other units in the previous layer. The data enters the input layer moving forward to the output layer. Since there is no feedback between layers such a network is called feed-forward neural network. Usually, a feed-forward neural network has two main operations which are learning and classification. In Figure we provide a simple example of a 3-layered network.

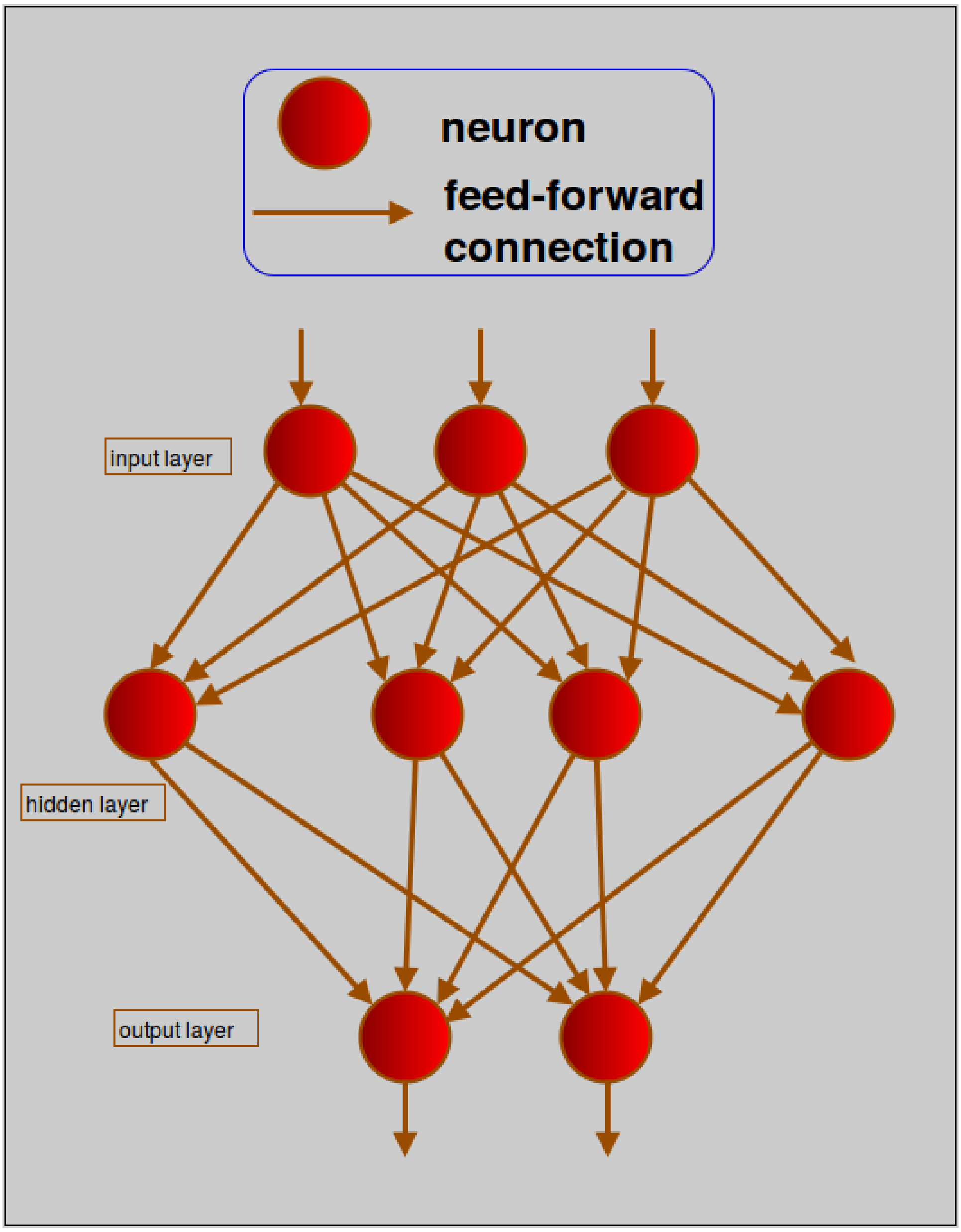

FIGURE 2.4: A Feed-forward Neural Network

In the feed-forward neural network shown in Figure 2.4, the input layer consists of 3 nodes, the hidden layer consists of 4 nodes and the output layer of 2 nodes. In the figure all the feed-forward connections are depicted.

2.2.2 Recurrent Neural Networks

Feedback or Recurrent Neural Network (RNNs) have been investigated quite a lot in spoken language understanding (SLU) the last few years. The variants that have been tested are of high interest and provide great results. It is an undeniable fact that recurrent neural networks outweigh the traditional feed-forward neural networks. Actually, the former seem to differ from the latter on the fact that they have the ability to capture and exploit long-term dependencies since it connects past and current entries. Of course, there are common points as well. In both feed-forward and feed-back neural networks, the words are transformed into high-dimensional real-valued vector. So, we instead of working with words, we work in the vector space. As logically expected, in the vector space the similar semantically words are close. Thus, it is feasible to develop a continuous space language model that exploits the relationship of such words. Training the model on large datasets and searching for the most suitable parameters for this model which maximizes the likelihood of one word in a context, then the likelihood of similar words in similar contexts is maximized as well. In language understanding and in this work particularly the focus is on the slot filling task. The last few years, a really popular technique to do so, used to be Conditional Random Fields (CRFs). This technique deploys an exponential model which has been widely used for sequence tagging. However, further techniques such as Support Vector Machines or Finite State Transducers have been examined for the same purpose.

2.2.2.1 Elman-type neural networks

One successful recurrent architecture is the Elman-type [4]. The main goal of this architecture is to introduce memory to the neural networks (see Figure 2.5). Like the conventional feed-forward networks, the recurrent neural networks have three basic parts called units or layers. These are the input, the hidden and the output layer. In feed-forward networks the input units activate the hidden units and then the hidden units activate the output units. The difference in the Elman RNN architecture is that except for the mentioned activations, the hidden units activate themselves through some other hidden units called context units. To sum up, not only the input units but also the context units activate the hidden units and the hidden units feed both forward to activate the output units and back to activate the context units. The hidden units map the input and the previous hidden units' state to the output. To achieve this, the units are connected with weights which are continually adjusting in order to map the input to the output in the best possible way.

For a mathematical representation of the input-output relation we have to denote some details. Let $in(t)$ and $o(t)$ the input and output vector respectively. The output vector $o(t)$ has a dimensionality equal to the number of possible slots. The hidden layer $h(t)$ represents of the memory of the network.

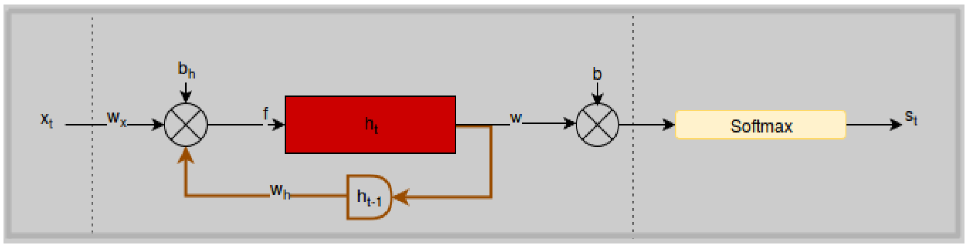

FIGURE 2.5: An Elman-type Neural Network

Then the hidden and output layers are:

$$h(t) = f\Big(W_x x(t) + W_h h(t-1) + b_h\Big)$$

$$s(t) = softmax\{Wh(t) + b\}$$

where W_{in}, W_h and W are the matrices with the weights of the network's connections.

We also use the sigmoid function at the hidden layer: $f(x) = \frac{1}{1+e^{-x}}$

and a softmax function at the output layer: $softmax(x_i) = \frac{e^{x_i}}{\sum\limits_{k=1}^{K} e^{x_k}}$

One popular way to train this model is back-propagation which maximizes conditional likelihood of the data.

2.2.2.2 Jordan-type neural networks

The Jordan RNN [5] is likewise an architecture with recurrent connections (see Figure 2.6). The different part is that the hidden layers are fed with the output posterior probabilities:

$$h(t) = f\Big(W_{in} in(t) + W_o o(t-1)\Big)$$

$$o(t) = softmax\{Wh(t)\}$$

2.2.2.3 Hybrid neural networks

A combination of the Jordan and the Elman architectures (see Figure 2.7) results in the hybrid model [9]:

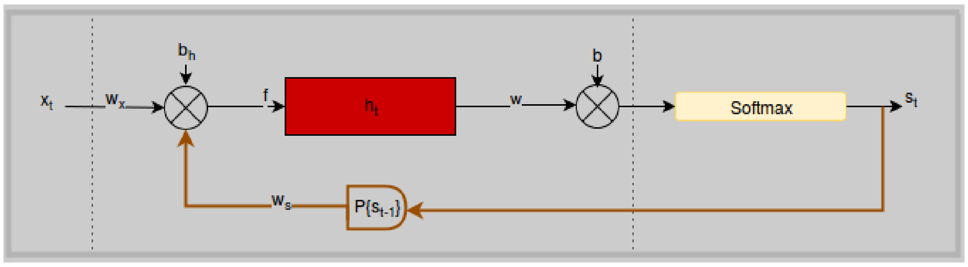

FIGURE 2.6: A Jordan-type Neural Network

$$h(t) = f\Big(W_{in}in(t) + W_h h(t-1) + W_o P\{o(t-1)\}\Big)$$

$$o(t) = softmax\{Wh(t)\}$$

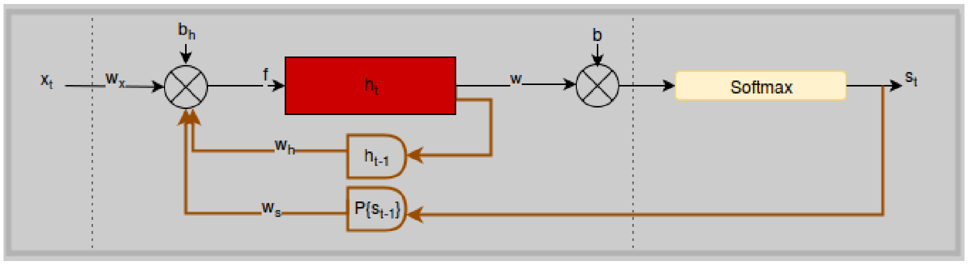

FIGURE 2.7: An Elman-Jordan Hybrid Neural Network

2.2.3 Learning, optimization and testing

2.2.3.1 Gradient descent

During training our neural network, we try to minimize the number of errors on unseen examples by optimizing the network's parameters θ. A not computationally expensive way is to minimize the Negative Log-Likelihood Loss (NLL) as the loss function, which means that we maximize the log-likelihood of our classifier given all the labels $y^(i)$ in a training dataset. The gradient for each one of the sentences in our training set D is used as a supervised learning signal for deep learning of a classifier. The negative log-likelihood is:

$$NLL(\theta, D) = -\sum_{i=0}^{|D|} \log P\Big(Y = y^{(i)}|x^{(i)}, \theta\Big)$$

2.2.3.1.1 Stochastic gradient descent (SGD)

Gradient descent is the most common way to optimize neural networks. In this algorithm, we start with our loss function and some parameters and we work in a loop. In each repetition we compute the gradient and based on that we compute the parameters. The loop is terminated when some stopping conditions are met. To speed up this procedure we can estimate the gradient using some of the training data, not the entire training set. This form of Gradient descent is called Stochastic gradient descent (SGD) and in its purest form, it takes as input only one single example at a time to estimate the gradient.

2.2.3.1.2 Mini-batch stochastic gradient descent (MSGD)

There are many variants of the stochastic gradient descent used in training a classifier. One of them is called minibatch stochastic gradient descent (MSGD). In this algorithm, we use more than one training data to make each estimate of the gradient. This technique reduces variance in the estimate of the gradient, and often makes better use of the hierarchical memory organization in modern computers. Generally, mini-batches are usually better. They need less computations and are more efficient than true stochastic gradient descent. This variant seems like a trade-off between computing the true gradient and the gradient at one only example. It may also result in smoother convergence of the algorithm, as the gradient which is estimated at each step uses more training data.

2.2.3.1.3 Sentence-level instead of word-level gradient descents

During the training procedure of a recurrent neural network, we have to make a decision about the updates. One option would be to do on-line updates, which means that the updates are done on word-level. This would mean that we would have to estimate and update the model parameters of the network after predicting every single word of every single sentence. Taking into consideration the fact that in our input data the length of a sentence could be long, it is easy to realize that this is a complex thing to do. This is due to the fact that when updating after the prediction of the i-th slot, we should then re-compute all the parameters from the beginning of this sentence if we want predictions consistent with the current model parameters.

So, a way to deal with this is to work in sentence level instead of word level. More specifically, the whole sentence can be considered as a mini-batch. For each mini-batch we compute the mean loss. Afterwards, we perform a gradient update for the this sentence. This is the basic idea of mini-batch gradient descent . This approach can be applied irrespectively the network architecture and there is no issue that the mini-batches are of different size.

2.2.3.2 Regularization

When it comes to neural networks and generally classification tasks, optimization is not the only issue to deal with. When we train our model we should make sure that our classifier can treat unseen entries, not only the ones it has already seen in the training data. The mini-batch stochastic gradient descent algorithm does not take this into consideration, and this may lead in over-fitting the training data. One way to handle this situation and avoid over-fitting is regularization. There are several techniques for regularization but in this work we used early-stopping and dropConnect. A brief description of these two methods follows.

2.2.3.2.1 Early-Stopping

The first regularization technique to overcome over-fitting is called early-stopping. This method is widely used because it is simple to understand and implement and has been reported to be superior to regularization methods in many cases. The main idea is to monitor the performance of the model on a validation set, not on the training set. A validation set is a set of data that are not part either of the training or the test set and thus it is considered to be representative of future test set. When the performance of the model improves sufficiently on the validation set, then the model implemented here gives up on further optimization.

The procedure start by splitting the training data into a training set and a validation set. An efficient way to separate these two sets is to consider as training set the 80 percent of the training data and the rest 20 percent would be the validation set. Then, the model trains on the training set and evaluates the validation set after every epoch which is a single pass through the whole dataset. When the error on the validation set is higher than it was the last time it was checked, we have early-stopping. This means that we stop the training procedure before the maximum number of epochs is reached. Finally, we use the last parameters of the network on our test data. This approach uses the validation set assuming that the error on the test set will be similar.

2.2.3.2.2 Dropout and DropConnect regularization

Another form of regularization we used is called DropConnect and it is variant of the Dropout regularization. In Dropout we set some activations in each layer to zero during the training stage. In this way, over-fitting is reduced and the performance is improved. The main idea is that omitting the effect of a part of the training examples the model does not learn the training data exclusively.

The variant we used in this work is DropConnect. DropConnect is a generalization of Dropout. It randomly drops the weights, which means that it sets the weights to zero, rather than the activations. Both Dropout and DropConnect are suitable for fully connected layers. A brief description of Dropout and DropConnect follows.

Let's suppose a fully connected layer of a neural network with:

- input $v = [v_1, v_2, ..., v_n]$
- weight parameters W (of size $d \times n$) and
- output $r = [r_1, r_2, ..., r_d]$

The output r is computed as a matrix multiply between the input vector and the weight matrix followed by a non-linear activation function, a:

$r = a(Wv)$ (where biases [37] are omitted for simplicity)

Dropout

Each element of a layer's output is kept with probability p, otherwise being set to 0 with probability (1-p). Applying Dropout on the outputs of a layer, we have

$r = ma(Wv)$

where m is a binary mask vector of size d with each element, j, drawn independently from $m_j \sim$ Bernoulli(p).

Many activation functions such as tanh, centered sigmoid and relu have the property that $a(0) = 0$. Thus, we have $r = a(mWv)$, where Dropout is applied at the inputs to the activation function.

DropConnect

Instead of dropping an activation, one can drop some weights connecting the neurons of a layer. Each connection is set to zero with probability $1 - p$ and with probability p it is set to one. Applying DropConnect on a layer on the training stage, the output is

$r = a((MW)v)$

where M is a binary matrix encoding the connection information and $M(i, j)$ $\sim Bernoulli(p)$.

Each element of the mask M is drawn independently for each example during training. The biases [37] are also masked out during training. An example is shown in Figure 2.8.

2.2.3.3 Testing

We partition our dataset into three sets, the training set, the validation set and the test. First we use our training set. We start with a random initialization and we apply the stochastic gradient descent algorithm to calculate the parameters of the model. Then, at each epoch we use the validation set to check if the model is working properly. If a model results in a good performance for the validation set, we save it. In case we use the early-stopping regularization, we keep doing this procedure for a specific number of loops. If it has been a long time since the last time we found a good model, we stop our loop. As the final parameters of the model, we keep the best parameters we had saved. Finally, we use these parameters for evaluation on the test set. The performance of the model is the evaluation on the test set, using the best parameters for the validation set and this is the performance we expect on unseen data.

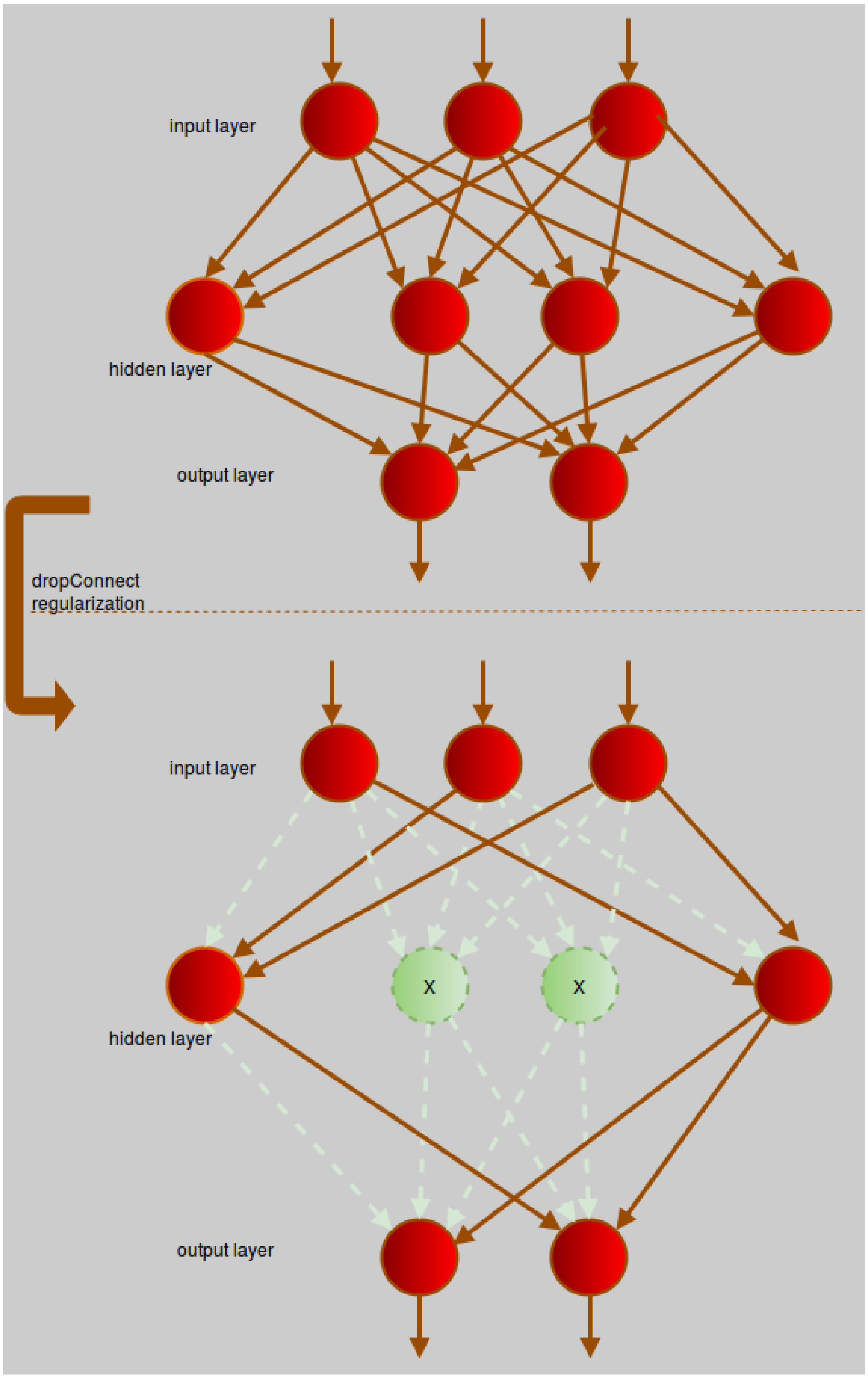

FIGURE 2.8: DropConnect regularization example

Chapter 3

State-of-the-art

3.1 Slot filling techniques

For a long time the predominant models for the SF task [2] in SLU were the popular CRFs [19] and SVMs [12]. In 2013, simple RNNs [14] were investigated for the SF task and outperformed all previous approaches. Since then, several RNN variants were proposed, which either experiment with different connection types inside the network such as the Bi-directional [9, 22], the hybrid [9] and the RNN-SOP [25] or add extra memory such as LSTM [15] and RNN-EM [17]. Many other combinations have been investigated such as the encoder-labeler LSTM [18], Bi-directional RNN-LSTM [39] and the Bi-directional RNN using a ranking loss function [21] which showed great performance. Recently, attention-based RNN [40] and knowledge-guided structural attention networks [41] have shown benefits on the language understanding task.

3.1.1 Conditional Random Fields (CRF)

A discriminative method which has been extensively used in spoken language understanding for the slot filling task is the Conditional Random Fields (CRFs). CRFs were introduced by Raymond and Riccardi [10] for the slot filling task. Several studies have been conducted and they have shown that CRFs outperform conventional generative models. This technique maximizes the log posterior probability of training data label sequences given the observation which is usually achieved by applying gradient-based optimization like stochastic gradient decent. More information and experiments using CRFs are in [10][11].

3.1.2 Support Vector Machines (SVM)

Support Vector Machines (SVMs) is a technique used in [12] to identify English base phrases. This technique is popular due to its ability to generalize. Moreover, it is considered to perform well even with data of high dimensionality. Considering a binary classification task, the training data are represented by a feature vector. The purpose is to find an optimal hyperplane. SVMs are able to do so and successfully manage to separate the training data by finding the hyperplane that maximizes its margin. SVMs are also used for non-linear classification (Vapnik, 1998). Kernel SVMs have been used in order to handle non-linear hypotheses building optimal separating hyperplanes which take into account all combinations of features. Some of the commonly used kernels are linear, polynomial and radial basis functions. A detailed discussion on SVMs can be found in [12],[13].

3.1.3 Neural Networks

3.1.3.1 RNN

For a long time the predominant models for the slot filling (SF) task in spoken language understanding were the Conditional Random Fields (CRFs) and the Support Vector Machines (SVMs). In 2013 an adaptation of the Recurrent Neural Networks (RNN) [14] was proposed for the SF task (see Figure 3.1). That basic RNN outperformed the previous CRF results. In the same work it was also demonstrated that the model is able to learn task appropriate word-embedding. This basic RNN architecture consists of an input layer, a hidden layer and an output layer. The input is connected to a set of hidden nodes in the hidden layer. All hidden nodes are fully connected with recurrent connections among the hidden nodes. These hidden nodes are also connected with a set of output nodes. For language understanding, the RNN is trained using the semantic labels of the words rather than the words themselves. The proposed RNN produced state-of-the-art results and for more advanced state-of-the-art bag-of-words, word embedding, named-entity, syntactic and word- class features used as well.

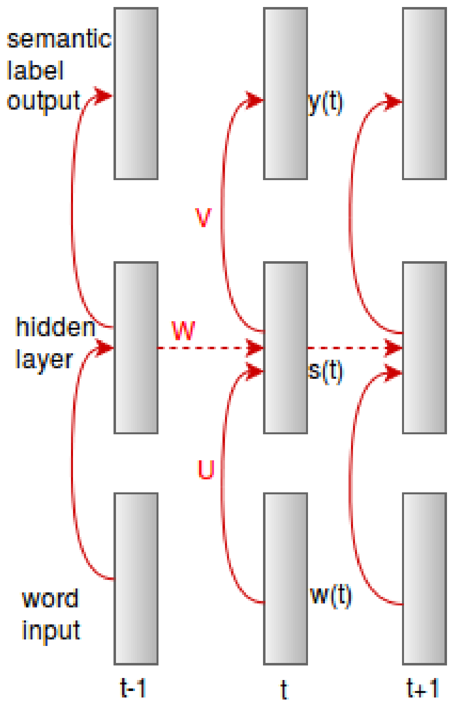

FIGURE 3.1: A Recurrent Neural Network for SF

3.1.3.2 CNN-CRF

This architecture is a combination of two difference architectures. The model which was proposed for the slot filling task is triangular Conditional Random Fields (TriCRF) [19]. Instead of having one simple input layer, this work places Convolutional Neural Network (CNN) layers at the input layer (see Figure 3.2). These CNN layers' job is to extract the feature which will then be the input of the TriCRF. Thus, it is supposed to be considered as a continuous space version of the TriCRF model. The CNN-CRF architecture achieved state-of-the-art results. Another interesting information about this work is that the CNN-CRF models two tasks, the slot filling and the intent determination, simultaneously. In addition to this, the bias issue [37] is handled by globally normalizing the neural network. That was the first model proposed to train both intent determination and slot filling jointly based on neural networks.

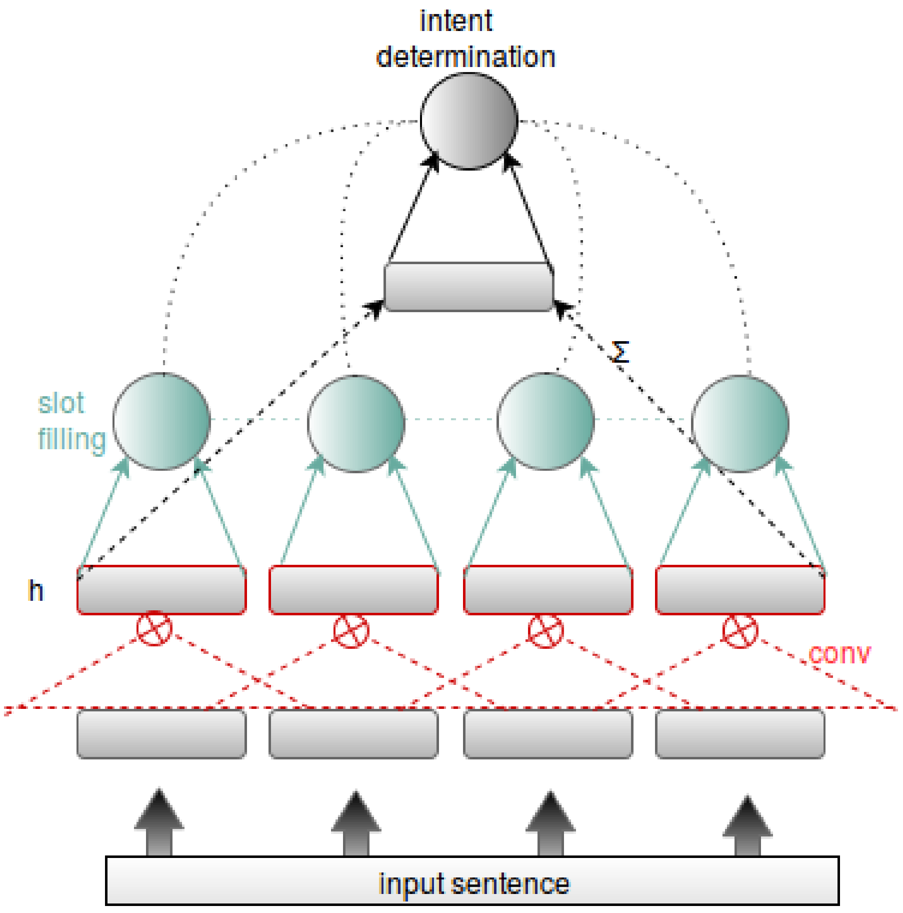

FIGURE 3.2: CNN-CRF for SF

3.1.3.3 Bi-directional RNN

In the simple recurrent neural network, the output of the hidden layer at time step $t-1$, $h(t-1)$ returns a feedback into the input of the hidden layer at time step t, $h(t)$. In other words, this neural network takes advantage of previous information. The exact same thing could be done by using the future instead of previous information. This means that the input of the hidden layer at time step t, $h(t)$, would get $h(t+1)$, not $h(t-1)$. However, an even better case would be the combination of these two ideas (see Figure 3.3). That is what the basic idea of the Bi-directional RNN [9] is. In this work, both bi-directional Elman RNN and bi-directional Jordan RNN were proposed and tested. Since the former results in better performance than the latter, here we provide some more details about the bi-directional Elman RNN using context window.

The first step is to define the forward hidden layer h(t):

$$\overrightarrow{\mathrm{h}}(t) = f\left(\overrightarrow{\mathrm{W}}_{in} in(t) + \overrightarrow{\mathrm{W}}_h \overrightarrow{\mathrm{h}}(t-1)\right)$$

where $\overrightarrow{\mathrm{W}}_{in}$, $\overrightarrow{\mathrm{W}}_h$ are the weights for the forward input and the hidden layers.

The backward hidden layer $h(t)$ is:

$$\overleftarrow{\mathrm{h}}(t) = f\left(\overleftarrow{\mathrm{W}}_{in} in(t) + \overleftarrow{\mathrm{W}}_h \overleftarrow{\mathrm{h}}(t+1)\right)$$

where $\overleftarrow{\mathrm{W}}_{in}$, $\overleftarrow{\mathrm{W}}_h$ are the weights for the backward input and the hidden layers.

Then, the bidirectional hidden layer $h(t)$ takes as input the forward and backward hidden layers:

$$h_{bi}(t) = f\left(W_{in} in(t) + \overrightarrow{\mathrm{W}}_h \overrightarrow{\mathrm{h}}(t-1) + \overleftarrow{\mathrm{W}}_h \overleftarrow{\mathrm{h}}(t+1)\right)$$

The bi-directional RNN cannot outperform the simple RNN both with context window.

3.1.3.4 LSTM

A long short-term memory (LSTM) neural network [16],[15] is another neural network which consists of an input layer, a hidden layer and an output layer. Like the simple recurrent neural network the output of the hidden layer returns as feedback to the hidden layer, which thus uses past information. However, the LSTM architecture has advanced properties compared to the simple RNN due to the fact that the hidden layer contains connected memory cells which modulate their input-output in a context-sensitive way. The memory cells are linearly activated and propagated between different time steps (see Figure 3.4).

An LSTM can be described by these functions:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_i)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

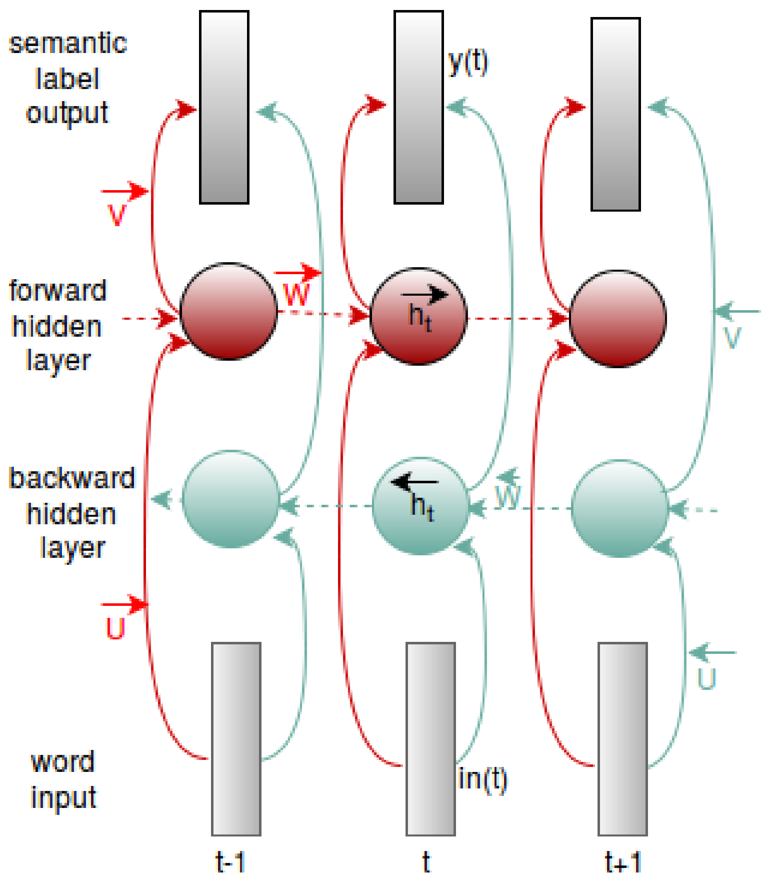

FIGURE 3.3: Bi-directional RNN for SF

$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o)$

$h_t = o_t \odot tanh(c_t)$

where σ is the logistic sigmoid function. i is the input gate, f is the forget gate and o is the output gate. c is the memory cell activation vectors h is the hidden vector

$\odot$ is the element-wise product of the vectors. The weight matrices from the cell to gate vectors are diagonal, whereas the weight matrices from input, hidden, and output vectors are not diagonal.

The LSTMs achieved state-of-the-art results on the ATIS database. In this work [15], they further extend the basic LSTM architecture to create deep LSTM. This deep LSTM consists of two layers of LSTMs. This extension improved the performance of the model and resulted in another state-of-the-art results on the ATIS database.

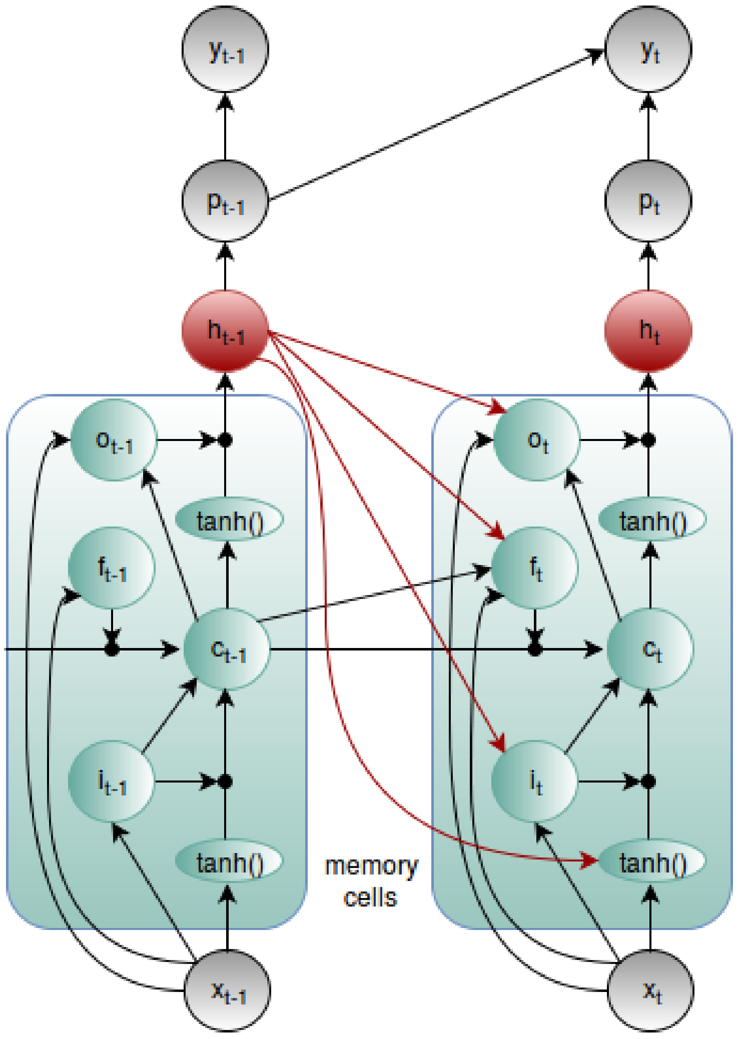

FIGURE 3.4: LSTM for SF

3.1.3.5 RNN-SOP

In [25], the Structured Output Prediction Recurrent Neural Network (RNN-SOP) was introduced (see Figure 3.5). The goal of this architecture is to capture and exploit label dependencies in model training. To do so, the model feeds the previous output label to the current sequence state, by using a sampling approach on true and predicted labels. This sampling approach makes the model more robust. The main idea is to makes the model learn to handle its own mistakes.

At time step t, the hidden layer of the neural network ht takes the word input x_t, previous hidden layer h_{t-1} and the output label y_{t-1} as input. During training, the true label at time step $t-1$ can be used as y_{t-1} whereas, during inference, the true label is unknown information. Thus, only the predicted label can be used. The problem that arises in this way is that the prediction power of the RNN does not generalize well during inference. An error made early in the input sequence can be propagated to the following predictions since they use wrong label information. To deal with such problems, the RNN-SOP applies a sampling approach in order to decide if true label or predicted label from the previous time step will be used.

Randomness to the output label is the key. With randomness the RNN model can learn to handle its own mistakes. The output label is either the predicted label with highest probability or a sampled label following label output probability distribution at $t - 1$. During the training process, the true label is chosen with probability P_i whereas the predicted label is chosen with probability $1 - P_i$. Different values for the P_i probability where examined in order to check the impact on the performance. Finally, performance gain over the baseline RNN system was achieved.

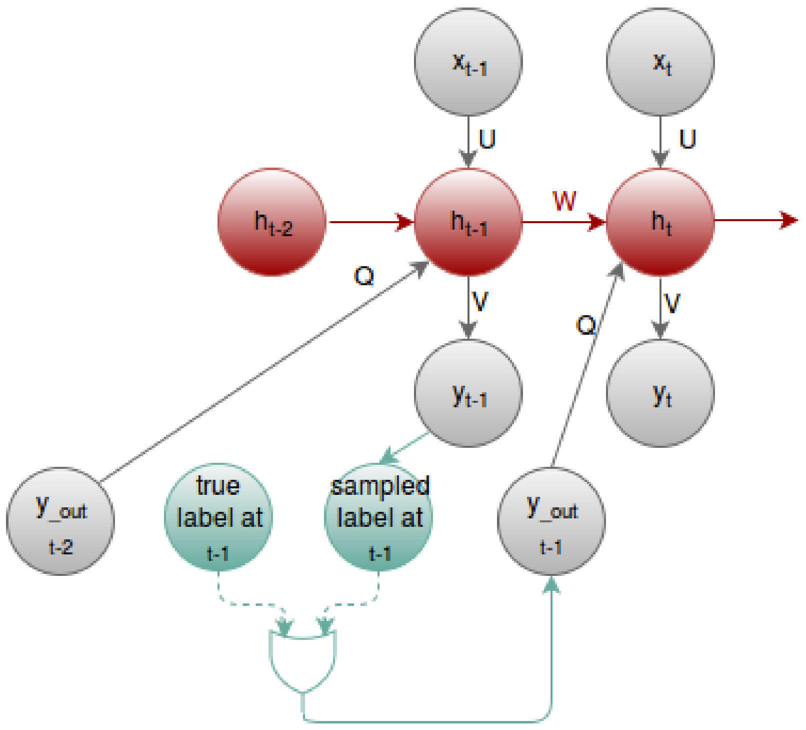

FIGURE 3.5: RNN-SOP for SF

3.1.3.6 Hybrid RNN

In [9] the Hybrid RNN is an RNN version which is a combination of the recurrences from the Jordan and the Elman RNN models (see Figure 3.6). The hidden layer activation and the output are:

$$h(t) = f\left(W_{in}in(t) + W_h h(t-1) + W_o P\{o(t-1)\} + b_h\right)$$

$$o(t) = softmax\{Wh(t) + b\}$$

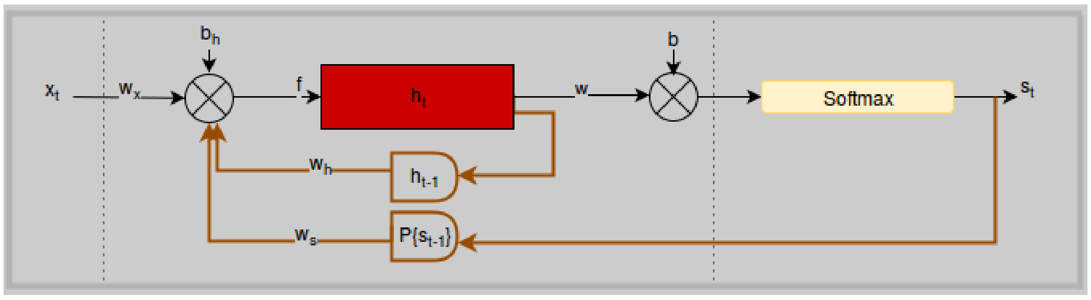

FIGURE 3.6: Hybrid RNN for SF

3.1.3.7 Deep LSTM

Here we present a recently proposed architecture, the Encoder-labeler LSTM and its extension Encoder-labeler Deep LSTM. In [15] the proposed encoder-labeler LSTM, the input words are represented into a vector using an LSTM encoder. Then the input vector enters a second LSTM layer. This second LSTM is responsible for the slot filling task (see Figure 3.7). In this way label dependencies and information of whole input sequence is possible to be captured. To sum up, this work managed to leverage sentence-level information and achieved new state-of-the- art in the slot filling task of the standard ATIS corpus. An Encoder-labeler Deep LSTM was also proposed providing some improvement to the performance of the model.

3.1.3.8 RNN-EM

The limited capacity of the memory of the simple RNNs has always been a concerning issue. The RNN-External Memory (RNN-EM) proposed by [17] is an alternative version of the RNN that uses an external memory in order to handle the limitation of memory capacity (see Figure 3.8). The RNN-EM model stores the past hidden layer activities both from the current input and from past inputs. The basic idea is that the input observation along with the content retrieved from the external memory is used for future predictions.

In detail, the RNN-RM consists of an input, a hidden and an output layer. There is no direct feedback of the hidden layer activity at the previous time step to the hidden layer. The feedback is taken as input to the hidden layer from the external memory. A weight vector is applied to retrieve the content from the external memory. The weights are proportional to the similarity of the current hidden layer activity with the content in the external memory. In practice this means that the content that is not relevant to the current hidden layer activity has small weights whereas the content that is relevant to the current hidden layer activity has high weights.

The philosophy of the RNN-EM architecture reminds of the long short-term memory neural networks. However, the RNN-EM gave higher performance than both the LSTM and the deep LSTM model and generally RNN-EM model achieved state-of-the-art results.

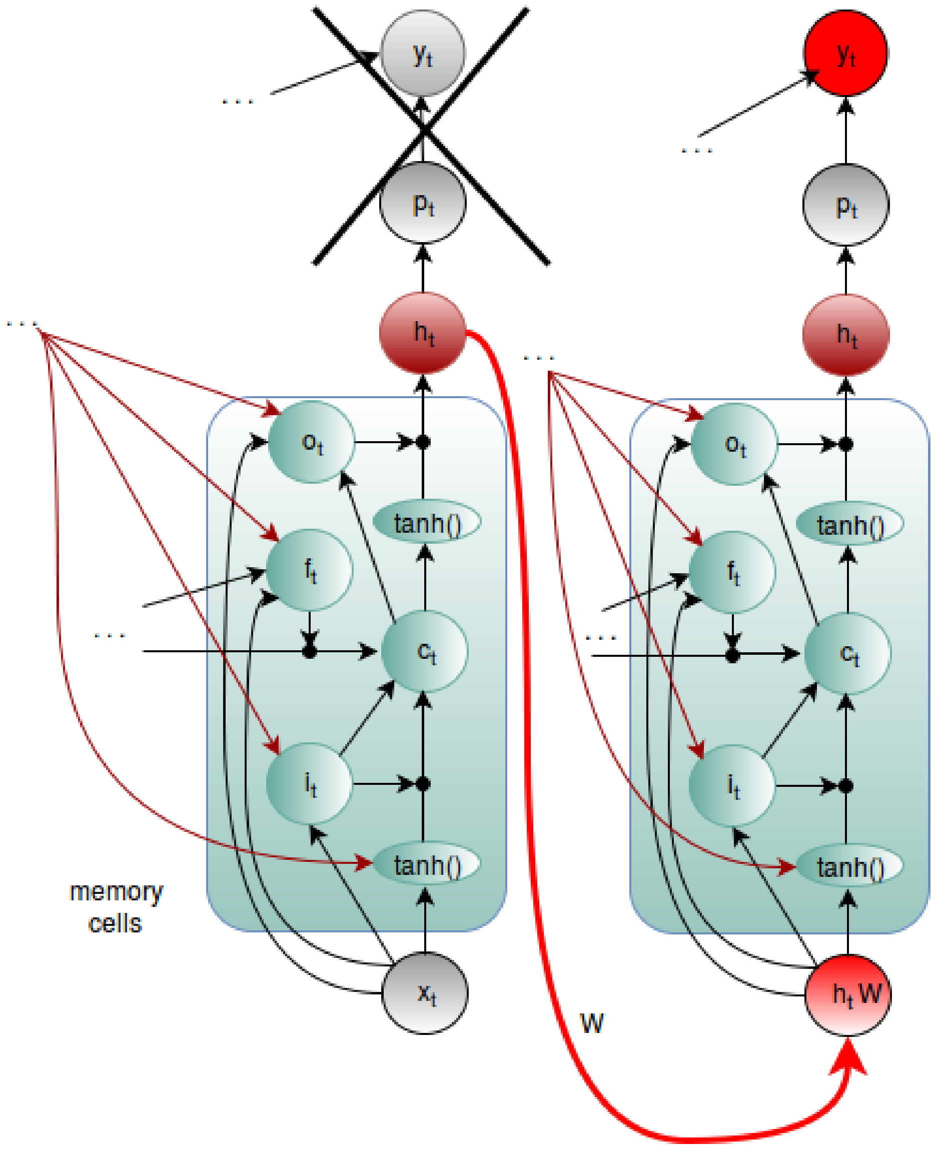

FIGURE 3.7: deep-LSTM for SF

3.1.3.9 Encoder-labeler LSTM

Here we present a recently proposed architecture, the Encoder-labeler LSTM and its extension Encoder-labeler Deep LSTM. In [18] the proposed encoder-labeler LSTM, the input words are represented into a vector using an LSTM encoder. Then the input vector enters a second LSTM layer. This second LSTM is responsible for the slot filling task. In this way label dependencies and information of whole input sequence is possible to be captured. To sum up, this work managed to leverage sentence-level information and achieved new state-of-the-art in the slot filling task of the standard ATIS corpus. An Encoder-labeler Deep LSTM was also proposed providing some improvement to the performance of the model.

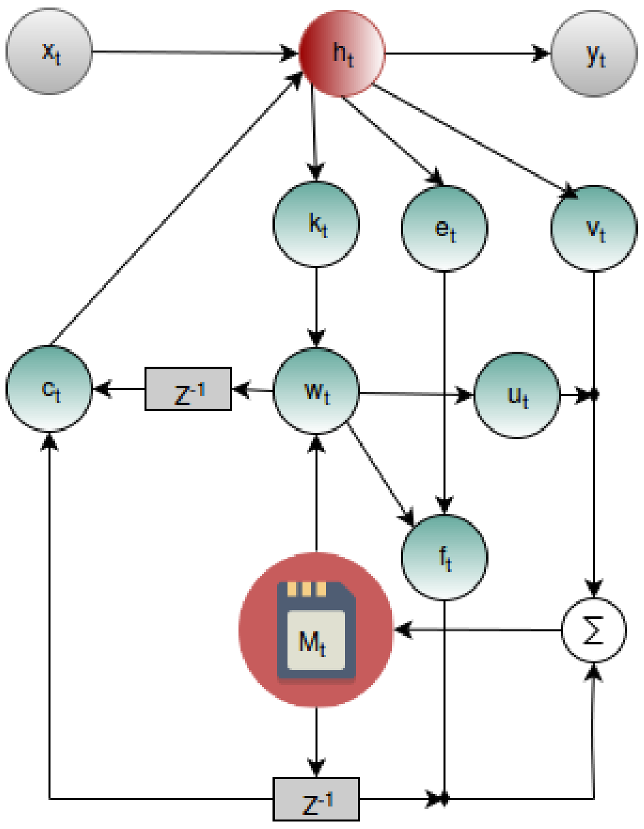

FIGURE 3.8: RNN-EM for SF

3.1.3.10 RSVM

In [21] Recurrent Support Vector Machines (RSVM) for slot tagging are proposed. RSVM combine a recurrent neural network (RNN) and the structured support vector machines (see Figure 3.9). The RNN is used for feature extraction. The support vector machine uses a sequence-level discriminative objective function. Furthermore, the model training was fast due to the fact that it skips the weight updating for non-support vector training samples. The model achieved new state-of-the-art results on the ATIS datasets.

3.1.3.11 Bi-directional RNN with ranking loss

In [22], a bidirectional recurrent neural network was proposed for the slot filling task. The bidirectional RNN was Elman-type. The feedback was information not only from the past but also from the future context, as in the case of the bidirectional RNN proposed by Mesnil in [9]. However, here ranking loss function is used to train the model. The use of the ranking loss function provides improvement on the performance over the cross entropy loss function.

3.1.3.12 Knowledge-guided Structural Attention Networks (K-SAN)

In [41], a more generalized form of the RNN is presented. This architecture incorporates non-flat network topologies in order to handle previously unseen test data, as well as finding the salient features to predict the semantic

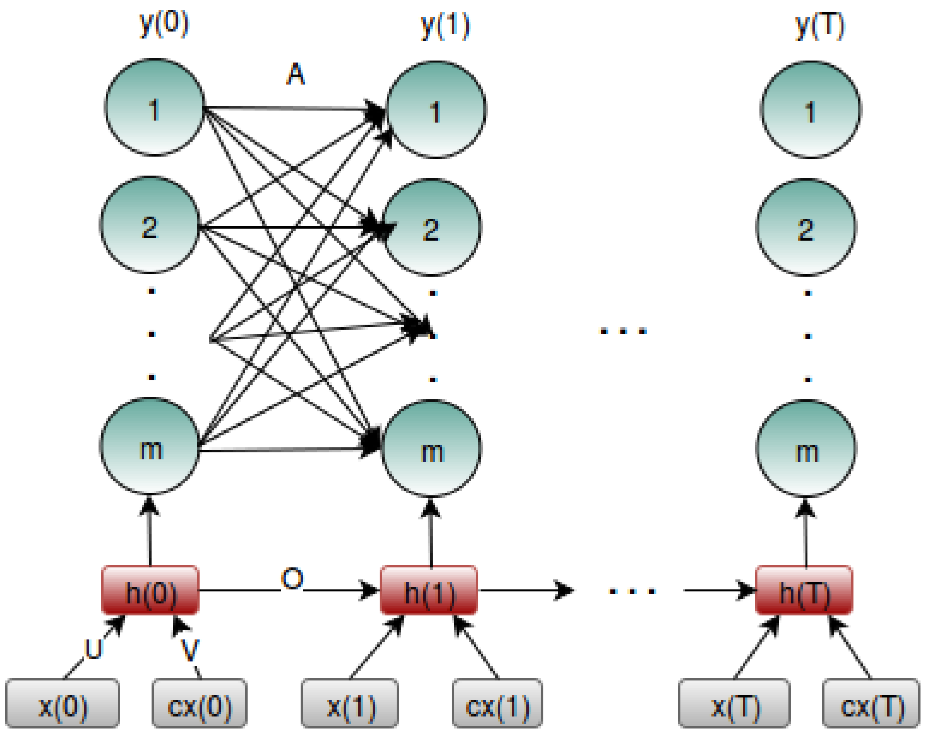

FIGURE 3.9: RSVM for SF

tags of the given sentences.

3.1.3.13 Attention-Based RNN

This RNN variant [40] jointly handles the intent detection and slot filling task. This work takes alignment into great account. This RNN encoder-decoder model with attention to the alignment manages to extract more information and finally to achieve a new state-of-the-art result.

Chapter 4

ClockWork Recurrent Neural Networks for Slot Filling

4.1 Novelty of the present work

Studying the state-of-the-art (Chapter 3), it is clear to see that almost all the neural network architectures proposed for the slot filling task are a variant of the simple recurrent neural network architecture (RNN). A simple RNN deals sufficiently with short-term dependencies of the input words. However, it lacks in the ability in handling long-term context. Consequently, the state-of-the-art proposals focus on providing a way to tackle this matter.

The issue that arises is that the more complex the dependencies a type of neural network tries to catch are, the more complex it becomes itself. All the recently proposed architectures require extra memory contrary to the simple Recurrent Neural Network, as well as a higher number of computations is necessary. This is mainly due to the fact that each model increases the number of the simple RNN parameters in order to improve the performance.

The difference of the present work is that our variant RNN simplifies the way of capturing long-term dependencies. More specifically, no extra memory of parameters are required. Instead of this, our neural network contains a reduced amount of parameters compared to the simple RNN. As a result the computational cost is lower. In addition to this, it is quite simple both to understand and implement. Even though our model is simple, it is capable of learning long-term information, apart from the short-term context that is captured anyway, even with the simple form of the simple RNN.

Our proposed neural network architectures are based on the clock-work RNN [20]. This neural network is a simple modification of the simple RNN. It was proposed and used for other tasks such as audio signal generation, TIMIT spoken word classification and on-line handwriting recognition. In all these three cases the Clock-Work RNN outperforms the simple RNN. Thus, it shows that it can be promising for other tasks as well. A similar time-scale architecture on LSTMs was only applied for modeling long texts [38].

The novelty of this work lies on the fact that a clock-work RNN has never been used for slot filling in spoken language understanding. In addition, we propose four novel models, the hybrid CW-RNN, the bi-directional CW-RNN, the deep CW-RNN and the Convolutional CW-RNN. While the present study is related to the simple RNN and hybrid RNN, it capitalizes the artificial network by partitioning the hidden layer. This was not considered in these earlier studies and may provide insights for future research. Our goal is to investigate the potentials of the Clockwork RNN variants for our slot filling task. For this purpose, we used a common dataset benchmark to conduct our experiments in order to compare our results with those of other neural network architectures. All proposed networks were modeled following the procedure described in Section 2.2.3.

4.2 Capturing short-term dependencies

In this work, we try to capture short-term dependencies, using a word-context window surrounding the word of interest. With each word mapped to an embedding vector, the word-context window is the ordered concatenation of word embedding vectors. For instance, if we want to use context window of size 3, then the word for tagging will be contexted by the previous and the following word features in the word feature vector. In detail, in the beginning we have an array of features [a,b,c]. The context window of size 3 for the word feature a is [-1,a,b], for feature b it is [a,b,c] and for feature c it is [b,c,-1]. So putting all these together as rows of a matrix, at the end we have a matrix of indexes where each line i corresponds to the context window surrounding the i-th word.

4.2.1 Word embeddings

To feed a neural network means to insert data as input. Undeniably this cannot be done with actual words in the case of the word tagging task. So, another representation of each word should be created, and finding a suitable one is not always an obvious thing to do. One way to handle this problem, is to extract some features from the data and then to represent each word as a vector and thus move to the continuous embedding space. However, there is no unique method to create these word embeddings. A frequently used technique is to train external data, like the Wikipedia, so that the embeddings cluster in case of similar word semantics. Of course, there is always a simpler way. One can randomly initialize the embedding vector, as long as this does not have negative effect on the performance of the system. For the ATIS data-set this simple approach seems to be effective. Consequently, for matters of simplicity our word embeddings were initialized randomly in our neural networks. For other data-sets the architecture could take advantage of better trained word representations, by initializing a word look-up table with vectors trained by back-propagation instead of randomly.

4.2.2 Context window

So far, we have described the procedure of extracting features from the words of a sentence produced either randomly or with the use of a look-up table. A vector containing these features is called Word Feature Vector. In machine learning two tasks are common to follow. The one would aim in tagging the sentence as a whole and the other would aim in tagging each element of the Word Feature Vector individually. The latter task is called Slot Filling and is the focus of this work. This word tagging task requires higher level dependencies among the words of the sentence tested. Consequently, the features contained in a Word Feature Vector need to be combined somehow and provide the next layer with more complex information. The extraction of higher level features can be achieved in different ways. Two of them are the window approach (context window), and the convolutional approach (section 4.1.3).

For the context window approach, a fixed window parses the Word Feature Vector in order to feed the next layer of the neural network. For instance, for window of size 3, the word we want to tag is accompanied by the previous and the following word in the window. This is the context of the word and the main reason why we do it is due to the fact that we suppose that each word in a sentence depends on its neighbors. The longer the window is, the more the neighbors that are taken into consideration for the decision of the network.

To sum up, a word enters a neural network in its context and thus the network is capable of capturing short-term dependencies. The only problem that appears is with the borders. In detail, the words both at the beginning and at the end of a sentence lack in context. For example, the first word of a sentence does not have a previous word. In such cases, we can handle this matter by using a technique called padding. We can pad the window whenever real context is not available. The elements that we add can be random or fixed values or features for the same sentence. For instance, it can be considered that the previous word of the first one, is the last word of the same sentence. Similarly, the following word of the last word of the sentence can be considered to be the first word of the sentence. In this work, the padding is done with the fixed index -1.

4.2.3 Convolutional layer

The other approach mentioned earlier uses a convolutional layer. Through convolution it is possible to combine the features of the Word Feature Vector into a global feature vector. For a specific word we want to tag, we first produce local features around it. Then, this local feature vector of the word is fed to the convolutional layer. This procedure is supposed to be done for each word in the sentence. The output of the convolutional layer should be fed to the standard hidden layers of the neural network and the size of the output of the network depends on the number of words in the sentence fed to the network.

However, applying convolution on a vector using a filter, the output of the convolutional layer would be of size $vector\_size + filter\_size - 1$. So, it is obvious that we need to turn the feature vector into its original size (vector_size) again.

To do so, we apply a Max Layer between the convolutional layer and the standard hidden layer of the neural network. For the max layer, max-pooling is frequently used. Max-pooling is a popular technique in image processing and it actually is a form of non-linear down-sampling. The main idea is to partition convolutional output and for each part it keeps the maximum value. In this way, not only the dimensionality is reduced, but also robustness is provided.

4.3 Capturing long-term dependencies

To capture long-term dependencies we apply a Clockwork Recurrent Neural Network. The Clockwork Recurrent Neural Network (CW-RNN) is a variant of the simple RNN architecture (Figure 4.2), proposed by Jan Koutnik, Klaus Greff, Faustino Gomez and Jurgen Schmidhuber [20]. The purpose of this architecture is to deal with the fact that RNN is not capable of capturing long-term dependencies although it works pretty well with short-term dependencies.

This issue is handled by CW-RNN by partitioning the RNN hidden layer into separated modules. These modules run at different clock speeds, timing their computation with different, discrete clock periods. Consequently, CW-RNN both trains and evaluates faster as not all modules need to be executed at every time step. Moreover, the model has a lower number of weights, because slower modules do not receive inputs from faster ones.

To sum up, rather than making the standard RNN architecture more complex, CW-RNN reduces the number of SRN parameters, improves the performance of the model and speeds up the network evaluation. The benefit of the CW-RNN is that its high-speed modules remember high-frequency information which is provided by the low speed modules and at the same time the low-clock-rate modules retain the long-term information obtained from the input.

4.3.1 CW-RNN architecture

A typical uni-directional RNN consists of input, hidden, recurrent and output layers. In the SF task, the input if size n_{in} $x_t \in \mathbb{R}^{n_{in}}$ at time t is a real-valued vector associated to each word (word embeddings). The output of the hidden layer of size n_h at a time step t is then defined as:

$$h_t = f\Bigg(Wx_t + Uh_{t-1} + b\Bigg) \tag{4.1}$$

where $W \in \mathbb{R}^{n_h \times n_{in}}$ and $U \in \mathbb{R}^{n_h \times n_h}$ are the weight matrices of the input and the hidden layer respectively, $b \in \mathbb{R}^{n_h}$ serves as a bias vector and $h_0 \in \mathbb{R}^{n_h}$ is the initial state. The function f at the hidden layer is the sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}}$$

The output layer can be described by the following equation

$$o_t = softmax\left(W_o h_t + b_o\right) \tag{4.2}$$

where $W_o \in \mathbb{R}^{n_{out} \times n_h}$ and $b_o \in \mathbb{R}^{n_{out}}$ define the weight matrix and bias vector of the output layer.

The CW-RNN is the same as the RNN above except that the hidden layer neurons are partitioned into m chunks (modules) of $k = n_h/m$ neurons each (Figure 4.3). For each module $i \in \{1, \cdots, m\}$, a different clock period $T_i \in \{T_1, ..., T_m\}$ is set which defines when each module will be activated and updated. In this paper we use a period $T_i = 2^{i-1}$ as in [20], although different clock period definitions can also be investigated.

We can, therefore, partition matrix $U = (U_{ij})$, $i, j \in \{1, \ldots, m\}$ into m^2 block sub-matrices, U_{ij}, of size $k \times k$, matrix $W = (W_i)$ into m block sub-matrices of size $k \times n_{in}$ (Figure 4.1), and vector $b = (b_i)$ into m sub-vectors of size $k \times 1$.

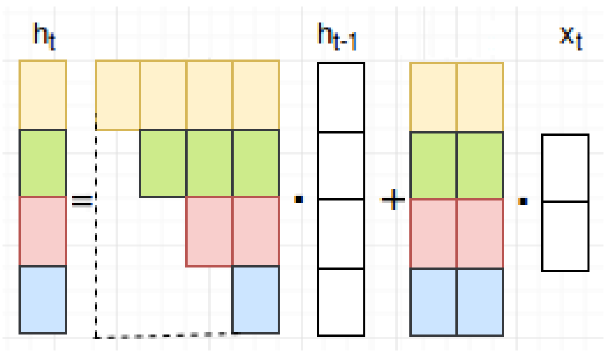

FIGURE 4.1: CW-RNN hidden layer matrix presentation for 4 modules

The connections among the neurons of the network depend on the time period. Specifically, although the neurons within each module are fully interconnected, the recurrent connections from module j to module i exist only if the period T_i is larger than the period T_j. In other words, the connections between the modules propagate the hidden state from slower modules to faster modules, which means the block U_{ij} is the zero matrix when $T_i > T_j$. The later inequality is used as in [20], although the opposite could do as well.

At each time step t, only the output of modules i that satisfy $(t \bmod T_i = 0)$ are active, where mod denotes modulo. The other modules retain their output values from the previous time-step. Therefore, the hidden state vector, which we have already named h_t, is partitioned into m sub-vectors of size k, $h_t = [h_t^{1\top}, \ldots, h_t^{i\top}, \ldots, h_t^{m\top}]^\top$, where h_t^i is the output of the i-th module at time-step t:

$$h_t^i = \begin{cases} f\left(\sum_{j=i}^{m} U_{ij} h_{t-1}^j + W_i x_t + b_i\right) & \text{if } t \bmod T_i = 0 \\ h_{t-1}^i & \text{otherwise} \end{cases}$$

It is clear that the low-clock-rate modules retain long-term information, while the faster modules contain the local information which is the context provided by the slower modules. In this example the module $g = 1$ is the fastest one and the module $g = 4$ is the slowest one.

The CW-RNN is a simplified RNN architecture, since using a smaller number of connections, decreases the number of parameters and the overall complexity of the network. Since a relatively small number of modules is activated at each time step, the computational complexity is largely decreased when training is performed in normal processing units.

4.3.2 Novelty of the present work: CW-RNN-based architectures

4.2.2.1 Bi-directional CW-RNN

The present work proposes bidirectional-CWRNN (biCW-RNN). It is based on the bidirectional RNN [9], described in Section 3.1.3.3. The bidirectional approach is able to exploit both future and past information in the input.

The hidden state vector h_t for the biCW-RNN is the addition of the forward $h_t^{(f)}$ as stated for the simple CWRNN and a backward $h_t^{(b)}$. The total hidden state is therefore $h_t = h_t^{(f)} + h_t^{(b)}$. (Figure 4.4)

4.2.2.2 Hybrid CW-RNN

In Section 2.2.2.3 we described the hybrid recurrent neural network. It is a simple combination of the Elman and the Jordan architecture. The difference is that both the past hidden activity and the past prediction is fed back to the hidden layer. In [9], the hybrid recurrent neural network slightly outperforms the simple recurrent neural network which is implemented in the paper. Inspired by [9], we created our Clock-Work Hybrid Recurrent Neural Network shown in figure 4.5.

The combined model is employed to learn more data patterns as both the past hidden activity and the past prediction o_{t-1} is fed back to the hidden layer. When CW-RNN is considered, several hidden layer modules are inactive at most time steps. The output layer of the network, which can also be seen as an average over past time-steps, can be set as feedback to the hidden layer. Specifically, we obtain the hidden layer dynamics using both the output posterior probabilities and the influence of past hidden states of the active models.

In this way, we introduce a hybrid CW-RNN architecture which combines the recurrences from the Jordan and the Elman models within the CW-RNN framework. The dynamics of our hybrid CW-RNN can be described from the following equation:

$$h_t^i = \begin{cases} f\left(\sum_{j=i}^{m} U_{ij} h_{t-1}^j + V_i o_{t-1} + W_i x_t + b_i\right) & \text{if } t \bmod T_i = 0 \\ h_{t-1}^i & \text{otherwise} \end{cases}$$

where o_t is given by 4.2 and the weight matrix, $V \in \mathbb{R}^{n_h \times n_{out}}$ of the connections from the output to the hidden layer, is partitioned into m blocks $V = [V_1^\top, \ldots, V_i^\top, \ldots, V_m^\top]^\top$ of size $k \times n_{out}$.

4.2.2.3 Deep CW-RNN

Deep neural networks have been extensively used in image processing. A popular type of deep neural networks is the deep convolutional neural network. The term deep refers to the fact that multiple hidden layers instead of one are used. The depth of a neural network is the number of such layers which are connected as the output of the one layer feeds the next one. Somehow, this procedure has shown great improvement in the models' performance.

Similarly, in spoken language understanding attempts have been made to explore deeper and more complex information from the input, through deep neural networks. One such example is [16]. In this paper a second LSTM is stack on top of the first LSTM layer. This variant provided significant improvement on the performance of the model.

Based on this idea, we developed a Deep Clock-Work RNN. Instead of two LSTM layers, our neural network contains two Clock-Work RNN layers. The output of the lower cw-rnn forms the input sequence for the upper cw-rnn. In detail, the input $in(t)$ of the upper cw-rnn takes h_t from the lower cw-rnn. A transformation matrix is applied on the $h(t)$ to transform it to $in(t)$. On top of the second cw-rnn layer a softmax layer is stack as in the regular cw-rnn case. In this work, we used depth 2 in our experiments (see Figure 4.7).

4.2.2.4 Convolutional CW-RNN

Previously, in Section 4.1 we discussed about capturing short-term dependencies. In Section 4.1.3 we briefly described how a Convolutional layer can be used for extraction of larger level features. The need to exploit long-distance dependencies among words far away in a sentence and out of a context word window, led us to create a convolutional clock-work recurrent neural network (CW-CRNN). To do so, we added a simple convolutional layer between the input and the hidden layer of our clock-work recurrent neural network introduced in Section 4.2.1.

A classical convolutional layer performs a convolution on an input $in()$ with a filter $f()$ and outputs:

$$conv(in, f) = in(m,n) \otimes f(m,n) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} in(u,v) f(m-u, n-v)$$

where the values of the filter $f() \in R^{flxfl}$ are considered to be parameters of the layer trained by back-propagation. The filter is small but extends through the full depth of the input. During the forward pass, we convolve each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. In this work, we consider the size of the filter as a hyper-parameter and we tested different sizes to check how the performance is affected.

Similarly to [42], we form the input matrix $X \in \mathbb{R}^{n_{in} \times N}$ by concatenating the N word embeddings x_t from an input sentence. Then a 2D filter defined through a filter matrix $F \in \mathbb{R}^{f_1 \times f_2}$ is used to perform wide convolution with the input matrix X. To address the problem of the missing words in the borders, we perform padding with past and future words. If we define $l_1 = \lfloor f_1/2 \rfloor$ and $l_2 = \lfloor f_2/2 \rfloor$, the convolution is described as follows:

$$(X * F)(i,t) = \sum_{j=-l_1}^{l_1} \sum_{\tau=-l_2}^{l_2} x_{t+\tau}(i+j) F(l_1+1-j, l_2+1-\tau)$$

where $i \in [-l_1+1, \ldots, n_{in}+l_1]\}$, $t \in [-l_2+1, N+l_2]$.

The filter values are considered to be parameters of the layer initialized randomly from a uniform(-1,1) distribution and estimated during model training with back-propagation. During the forward pass, we convolve each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. In this work, we consider the size of the filter as a hyper-parameter and we evaluated our system on different filter sizes, specifically $f_1 = n_{in}$ and $f_2 = \{3, 7, 9\}$.

Since the size of the features obtained from the convolutional layer is higher that the size of the input, we add a max-pooling layer after the convolutional layer. Max-pooling is a popular technique in image processing which is a form of non-linear down-sampling. The main idea is to partition convolutional output and to keep the maximum value for each part. In this way, we perform dimensionality reduction and we obtain robust features.

The result of the convolutional layer is the extraction of continuous space features. Such a continuous space feature extraction is able to capture more complex information than simple n-grams. Then, the extracted features are fed to the next layer for slot filling. In this work, we used one convolutional layer (see Figure 4.6). However, multiple convolutional layers can be stacked together to model wider range dependencies.

The size of the features is higher that the input and thus we add a max-pooling layer right after the convolutional layer. The max-pooling layer performs non-linear down-sampling. In detail, it partitions the input into a specific number of non-overlapping regions and for each region it outputs the maximum elements. It is obvious that by max-pooling we reduce the amount of parameters and the computations in the network and consequently we control over-fitting. Convolutional nn have been used in: [19][34][35][36].

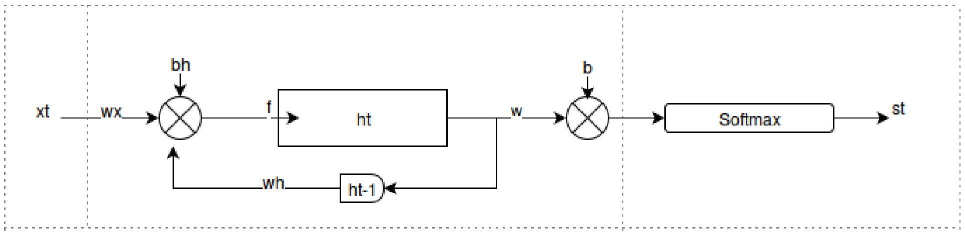

FIGURE 4.2: A Recurrent Neural Network

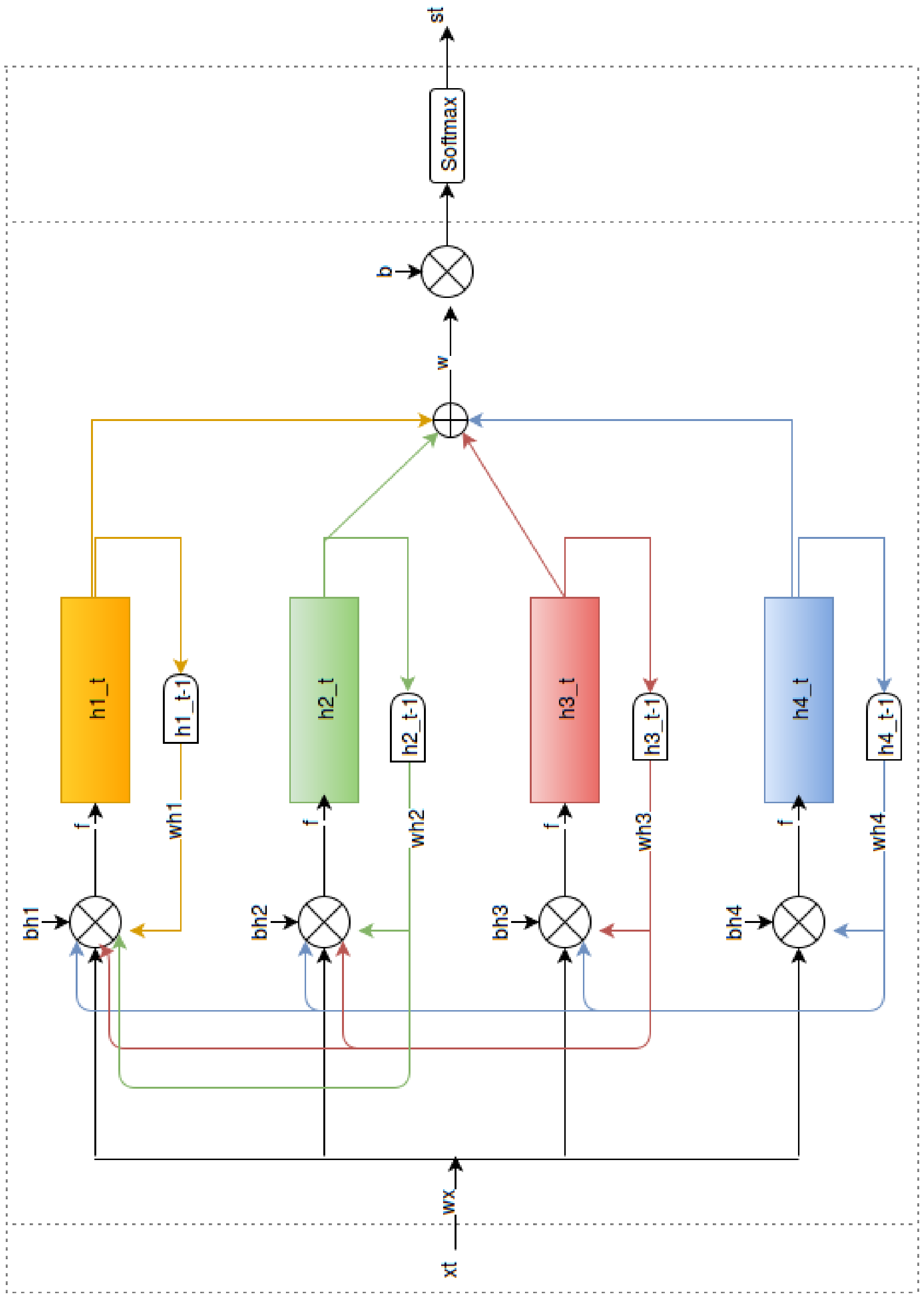

FIGURE 4.3: A ClockWork Recurrent Neural Network

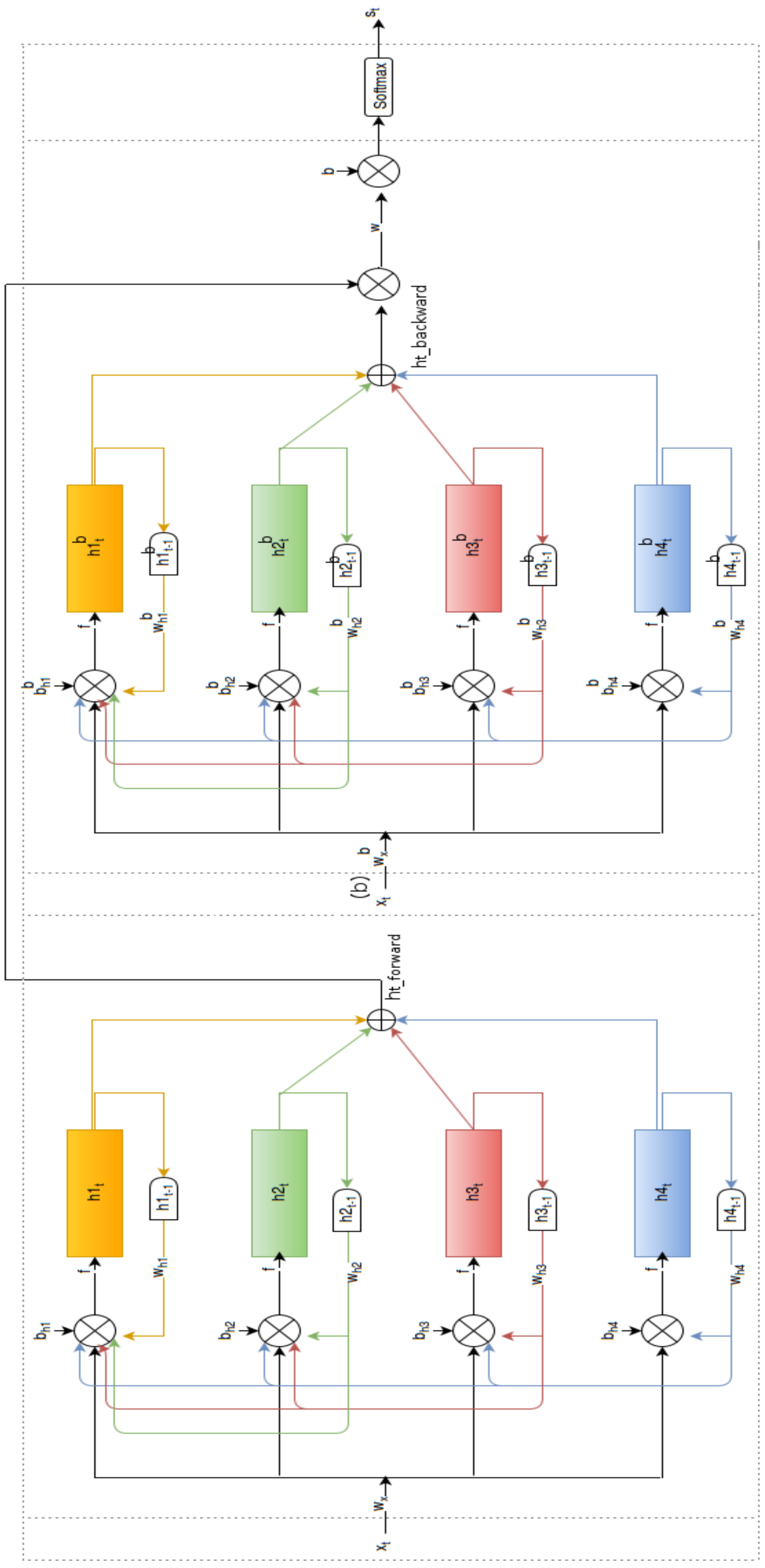

FIGURE 4.4: A bidirectional ClockWork Recurrent Neural Network

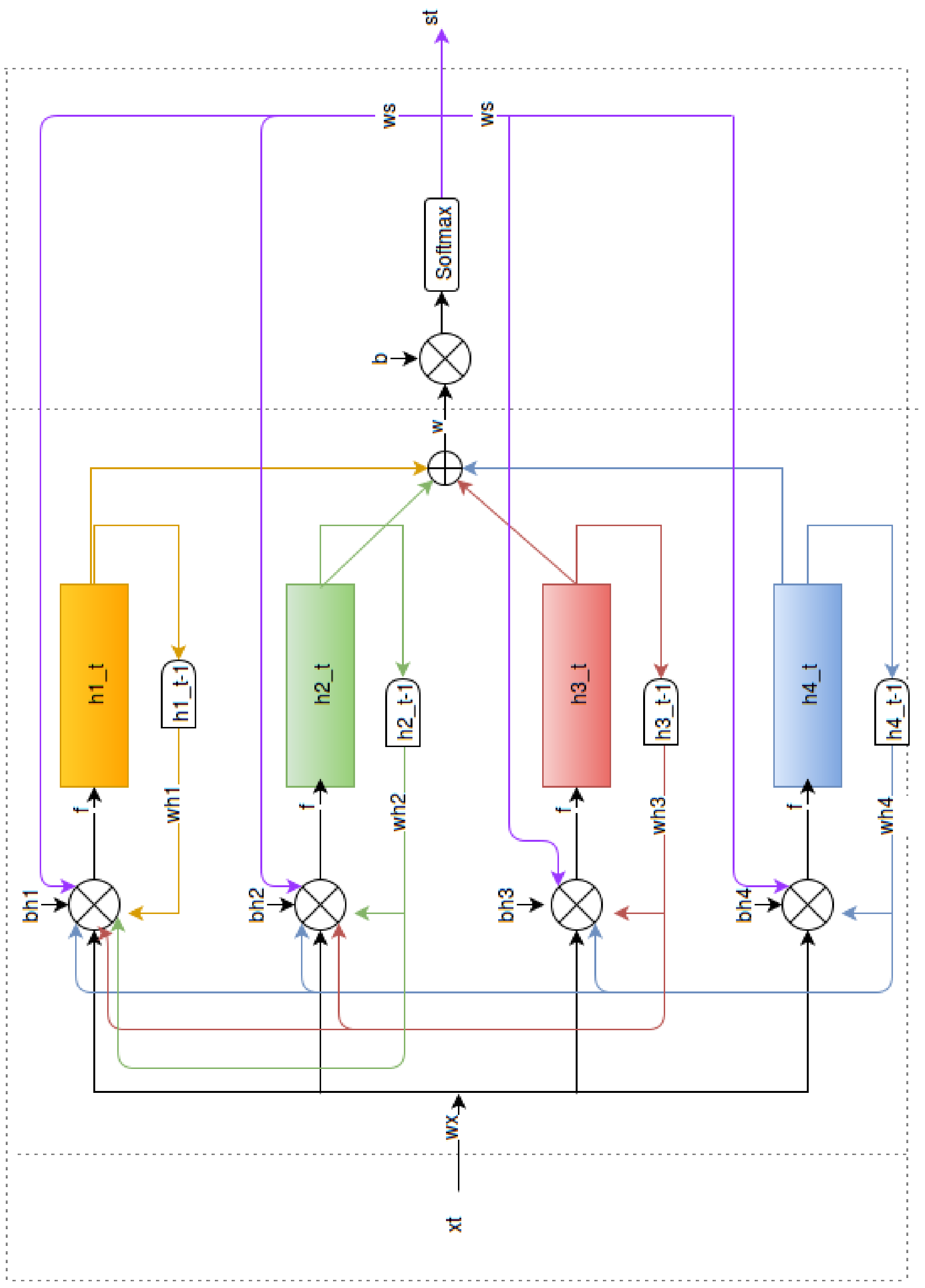

FIGURE 4.5: A hybrid ClockWork Recurrent Neural Network

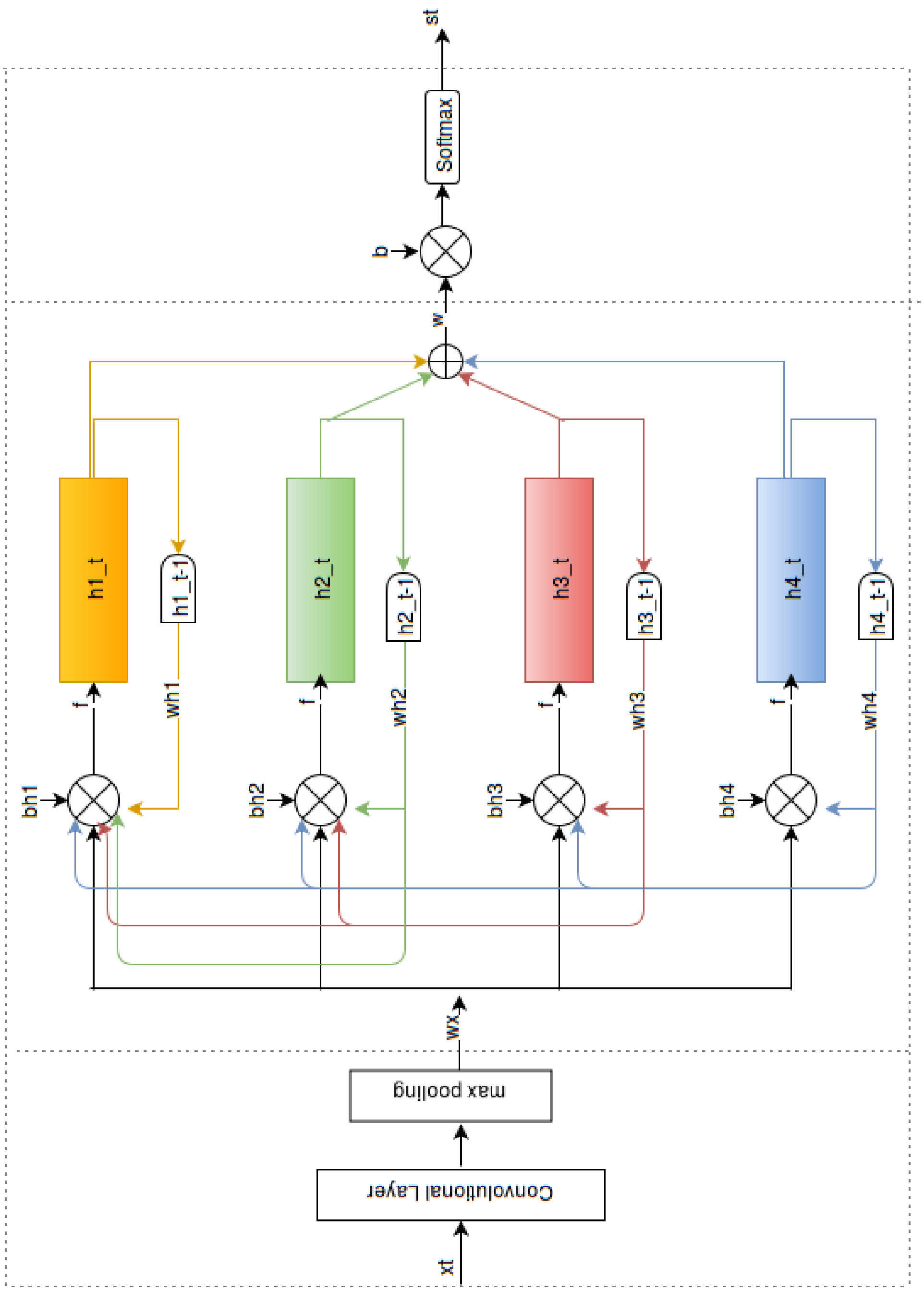

FIGURE 4.6: A convolutional ClockWork Recurrent Neural Network

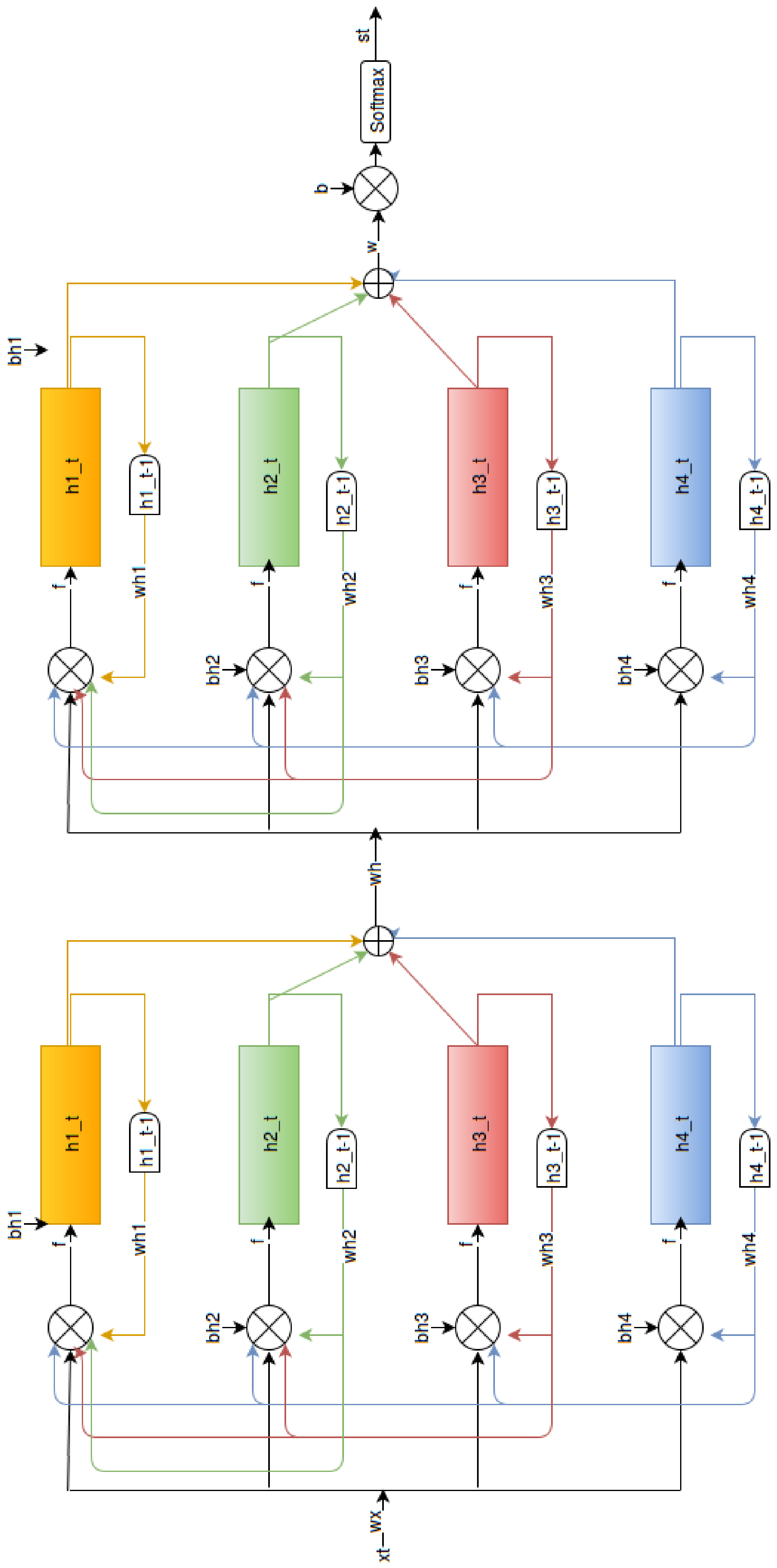

FIGURE 4.7: A deep ClockWork Recurrent Neural Network

Chapter 5

Data and Experiments

5.1 Corpus Analysis

In this work the corpus that we used is the popular and extensively used Airline Travel Information System (ATIS). ATIS [43] consists of audio recordings from speakers which make flight reservations or ask relevant information. For each of the recordings there is an annotated sentence as well as an annotated synchronous semantic interpretation (label) using the BIO (Begin - Inside - Outside) format. In this way, all words have a corresponding semantic label of either type B, I or O. The ATIS corpus was chosen for our experiments due to fact that it is the one that has been most used the by the SLU community particularly for the slot filling task e.g. [28,29,30,31].

In the original data, the training set includes 4978 sentences. These sentences were selected from Class A (context independent) training data in the ATIS-2 and ATIS-3 corpora. In this work, we used the database used in [14,9]. In this corpus, the 20 percent of the original training data was randomly sampled and used as the validation set. The rest 80 percent of the data was used as the model training set. As for the test data, they are from the ATIS-3 Nov93 and Dec94 datasets and they contain 893 sentences. The dataset provides us with the training set, the validation set, the test set and the dictionary (see Table 5.1 and Appendix[A]). For each sentence in each set we have this information: word id, name entity id and label id (see an example in Tables 5.2 and 5.3).

In this dataset there are 127 different types of slots (tags), including the common "null" label. The labels were created by [33] from the original annotations. In order to learn directly from data, we fully annotate automatically the train set to get the words/concept pairs. So, first we replace all class members by their corresponding label. Then we extract the sequence of concepts from the abstract semantic annotation. And finally, we build a regular expression to find concepts in words sequence. In this work, we preprocessed the data as in [14] for our experiments.

An example
In the example demonstrated in Table 5.2, train_lex is the vector with the word ids. Each word has a unique id. Train_ne is the name entity id. Here we notice that the words 'pittsburgh' and 'baltimore' have a different words id, but they have the same name entity id since they both are city names.

Furthermore, train_y is the vector which contains the label id of each word.

ATIS		database
vocsize	=	572
nclasses	=	127
nsentences:		
train	=	3983
validation	=	995
test	=	893

TABLE 5.1: Basic information on ATIS dataset

As we see, the words 'pittsburgh' and 'baltimore' may have the same name entity id, but they have different labels as the former is the departure city and the latter is the destination city.

EXAMPLE														
Sentence:	what	's	the	smallest	plane	that	flies	from	pittsburgh	to	baltimore	on	eight	sixteen
train_lex	554	4	481	444	381	480	192	208	379	502	69	358	169	441
train_ne	0	0	0	85	0	0	0	0	18	0	18	0	31	134
train_y	126	126	126	54	126	126	126	126	48	126	78	126	35	126

TABLE 5.2: A user's utterance example in ATIS

EXAMPLE	lex	ne	y	IOB
what	554	0	126	'O'
's	4	0	126	'O'
the	481	0	126	'O'
smallest	444	85	54	'B-mod'
plane	381	0	126	'O'
that	480	0	126	'O'
flies	192	0	126	'O'
from	208	0	126	'O'
pittsburgh	379	18	48	'B-fromloc.city_name'
to	502	0	126	'O'
baltimore	69	18	78	'B-toloc.city_name'
on	358	0	126	'O'
eight	169	31	35	'B-depart_time.time'
sixteen	441	134	126	'I-depart_time.time'

TABLE 5.3: Detailed utterance example in ATIS

5.2 Problems concerning Corpus

The biggest problem with this database is that the training set is too small. This fact results in finding many previously unseen sequences in the test set. Another significant category of errors is that we may have partially correct slot value annotations. For example, the word today can either match the tag Depart Date.Relative or Arrive Date.Relative. The third type of errors exists because of the disability of a model to capture long-term dependencies. This type of errors is our target in this work and it was been extensively analyzed in previous sections. Finally, other types of less errors are the annotation errors, which means that the slots were assigned the wrong category or tokenization issues, ill-formulated queries and ambiguous utterances. A more extensive study on the errors related to ATIS can be found in [31].

5.3 Data Processing

The task of SF is to predict the semantic label for each of the words in the ATIS sentences. To obtain proper input for our neural network architectures we need to represent each of the words in the sentences as d-dimensional feature vectors. Specifically, following the same data processing method as in [9,44,45] each word in the sentence is represented as an integer corresponding to the vocabulary id. Furthermore, for each of the word in the sentence we apply a context window which creates a vector representation by embedding the previous n_w and the next n_w words. In this way, short-term dependencies can be efficiently represented into the feature vector. Thus, the sentence is now represented by a matrix of integers. Finally, we associate this matrix of indexes to randomly selected embeddings which results to a continuous real-valued representation of the input sentence. Additional features could be incorporated into our model, however we use lexical features which are automatically extracted from word sequences to keep our model as simple as possible and make the model comparison more meaningful.

5.4 Evaluation Measures

The target of a classifier is to classify each test data in the correct class. So, to examine how successful our algorithm is, we calculate the accuracy of the classifier. As accuracy, we consider the number of the test data which are classified correctly over the total number of the test data. The formula which describes the accuracy of a classifier is:

$$Accuracy = \frac{\#Correctly \quad classified \quad test \quad data}{\#Total \quad test \quad data} \cdot 100(\%)$$

When we want to evaluate our classifier per class, there are some further measures we can use [47]. These measures are precision and recall. First, we define the terms tp, fp, tn, fn:

- tp (true positive): the number of the data which are classified correctly
- fp (false positive): the number of the data which are not classified correctly
- tn (true negative): the number of the data which should not be classified in a class and they weren't

- fn (false negative): the number of the data which should not be classified in a class but they were

Now we can define precision (Prec) and recall (Rec) as:

$$Prec = \frac{tp}{tp+fp} \quad \text{and} \quad Rec = \frac{tp}{tp+fn}$$

Precision can be considered as a measure of exactness or quality. More specifically, high precision means that an algorithm returned substantially more relevant results than irrelevant. On the other hand, recall is a measure of completeness or quantity. For instance, high recall means that an algorithm returned most of the relevant results [24].

Since we have more than two classes we need to find a way to combine the precision and recall we get from each class into one measure, since it is often useful to have a single measure. There are two ways to combine multiple performance measures into one quantity:

- Macro-averaging: compute performance for each class, then average.
- Micro-averaging: collect decisions for all classes, compute contingency table, evaluate.

So first the macro-averaged or micro-averaged precision and recall should be evaluated. Then, the combination of these two measures is the F-measure or balanced F-score (F1 score). The F1 score is the harmonic mean of precision and recall:

$$F1 = \frac{2 \cdot Prec \cdot Rec}{Prec + Rec}$$

Similarly to other works, in our experiments we evaluated our model performance using the macro F 1 score, the script of which was provided in the text chunking CoNLL shared task 2000 [46].

5.5 Baseline

Here we provide a brief description of the baseline models. In order to evaluate the results of our artificial neural network, we need to use the reported results of these relevant models and compare them with the results of this work in a following section. All the compared algorithms are applied on the same train and test dataset.

RNN

Yao in [14] trained a simple RNN. The model contained 200 hidden layer. The input was continuous space embedding of words. Two words of look-ahead was used and from the results it is obvious that it is a lot better than not using words of look-ahead, especially for networks of small size. We notice in the results that different numbers of hidden layers with over 100 layers perform almost the same. The highest performance of this simple model measured with the F1-score is 94.11%. For further improvement of the performance named entity features were also used. In this way, 3% improvement was achieved. Since, the features are highly related to the final output labels, this was expected. In other experiments, Yao used bag-of-words as input to the RNN and showed that the model is improved using bag-of-words input. The combination of the last two improvements, bag-of-words and named-entity features, resulted in the highest score which was 96.60%.

CNN-CRF

Xu and Sarikaya in [19] proposed a CNN-CRF. The CNN extracts features and the CRF performs the classification task. The input was 30 dimensional word vectors. The hidden layer of the CNN contained 110 dimensional hidden features from the context window size 5. In the experiments only lexical features automatically extracted from word sequences were used. Xu and Sarikaya jointly performed intent determination and slot filling. Their proposed CNN based TriCRF outperforms the simple TriCRF. The highest performance of this simple model measured with the F1-score is 94.35%.

Bi-directional RNN

Mesnil in [9] tried to benefit from both future and past information. To do so, he tested a bidirectional recurrent neural network. The bidirectional model combines the past and future information through its hidden state. The input with a context window is the input of the model in order to encode the future and past information together. Mesnil conducted 200 experiments with random sampling [8] of the hyper-parameters. For the context window the sizes tested where 3, 5, ... , 17. For the word embeddings the dimensions checked where 50 and 100 where the embedding matrix and the weight matrices were randomly initialized from the uniform of range [-1,1]. The regularization method was early-stopping over 100 epochs. From the results we see that the context window does not provide improvements to the bidirectional architecture. The highest performance of the bidirectional RNN measured with the F1-score is 94.73%.

LSTM and Deep LSTM

Yao in [15] proposed the LSTM and the Deep LSTM models. In the LSTM the input word in a context window of size 3 is fed to the model. For the simple LSTM 300 hidden layers and minibatches of size 30 were used. In the case of the deep LSTM Yao used 200 hidden layers for the lower LSTM and 300 hidden layers for the upper LSTM and the minibatch size was 10. Measured with the F1-score, the highest performance for the LSTM was 94.85% whereas for the Deep LSTM 95.08%.

RNN-SOP

In [25] Liu and Lane proposed the Structured Output Prediction Recurrent Neural Network (RNN-SOP). For the input of the network only word features are used. Both dropout and L2 regularization were used to prevent overfitting but the improvement was limited for ATIS slot filling task. Liu and Lane used context window of size 4 in order to incorporate future context features. In the experiments the P_i was set to values from 0.20 to 1.0. Measured with the F1-score, the highest performance for the RNN-SOP achieved was 94.89% with P_i = 0.8.

Hybrid RNN

Mesnil in [9] experimented with the Hybrid RNN. The hybrid model outperformed the simple RNN. n the experiments, the highest performance of the bidirectional RNN measured with the F1-score is 95.06%. Mesnil conducted 200 experiments with random sampling [8] of the hyper-parameters. For the context window the sizes tested where 3, 5, ... , 17. For the word embeddings the dimensions checked where 50 and 100 where the embedding matrix and the weight matrices were randomly initialized from the uniform of range [-1,1]. The regularization method was early-stopping over 100 epochs. For further improvement of the performance, he concatenated the Named Entity (NE) information feature as a one-hot vector feeding both to the context window input and the softmax layer. In the case of ATIS dataset, he used the gazetteers of flight related entities, e.g. airline as named entities. The results showed gains for all the methods tested.

RNN-EM

In [17] Peng and Yao proposed the RNN-EM. In the experiments the input has a context window of size 3. During the training procedure AdaDelta was applied for the gradients update. The model runs for 50 epochs. The highest performance for the RNN-EM was achieved for 100 hidden layers and 8 memory slots each of dimension 40. The RNN-EM performed LSTM by almost 0.38% and the highest performance measured with the F1-score was 95.22%.

Bidirectional RNN with ranking loss (R-biRNN)

In [22], a Bidirectional RNN with ranking loss was proposed. The model is trained with stochastic gradient descent in 25 epochs. The activation function that was used is the sigmoid function. The regularization technique was L2 regularization with weight 1e-7. The mini batch size was 1 and 100 hidden layers were used. Techniques like AdaGrad [26] and AdaDelta [27] were also used. Wherever the learning schedule does not need cross-validation, the model was fed with all the training data. The highest F1-score achieved by the simple form of the model R-biRNN is 95.47%. For further improvement of the performance they trained 5 models with different initializations and combined them. In case of a tie, one of the most frequent classes is randomly picked. In this way the F1-score is 95.56%.

Recurrent Support Vector Machines For Slot Tagging In Spoken Language Understanding

In [21] Recurrent Support Vector Machines (RSVMs) were used for the slot filling task. For the input only lexical features are used. During training with 20 epochs, the learning rate changes by AdaGrad [32]. Furthermore, 300 hidden layers were used as well as the two surrounding words of each word are used as bag of words. In the experiments, 10 models are trained. Each model has the same parameters but different random initialization. The highest F1-score achieved is 95.5%.

Encoder-labeler LSTM(W) and Encoder-labeler Deep LSTM(W)

In [18], Kurata,Xiang, Zhou, Yu proposed the Encoder-labeler LSTM(W). They used word embeddings of dimension 30, with context window of size 0, 1, 2. The size of hidden units in the LSTM was 100, 200, 300. The model runs for 100 epochs during the training process. The highest F1-score achieved was 95.4%. Using both a labeler LSTM(W) and a labeler LSTM(W+L), they got improvement by explicitly modeling label dependencies. In [18], an encoder-labeler deep LSTM(W) is also proposed. In both models, the hyper-parameter were randomly searched as proposed in [8]. For the encoder-labeler deep LSTM(W), they used word embeddings of dimension 30, 50, 75, with context window of size 0, 1, 2. The size of hidden units in the LSTM was 100, 150, 200, 250, 300. The highest F1-score achieved is 95.66%.

Attention-Based Recurrent Neural Networks

In this category, we have two attention-based encoder-decoder neural network models which have recently shown promising results. K-SAN [41] learns the representation for the whole sentence by paying different attention on the substructures and leverages prior knowledge as guidance to incorporate non-flat topologies. It has better generalization and robustness and its performance reaches 95%. The second and most important model in this category is an attention-based RNN model for joint intent detection and slot filling [40]. A bidirectional RNN reads the input sentence forward and backward, and at each time step, the concatenated forward and backward hidden states is used to predict the slot label. This attention-based bidirectional RNN model jointly deals with intent detection and slot filling. This joint model appears to simplify the dialog system, as only one model needs to be trained and deployed. Finally, this model achieved state-of-the-art performance, 95.78% for slot filling, on the benchmark ATIS task.

5.6 Data Processing

Reading directly from the database we need to transport our data in a proper way so that they can be the input of our neural network (see Figure 5.1).

- A sentence is a sequence of words. Each word being listed in a vocabulary table can be associated to a specific word id.

model	**F1(%)**
RNN	94.11
CNN-CRF	94.35
Bi-directional RNN	94.73
LSTM	94.85
RNN-SOP	94.89
K-SAN	95.00
Hybrid RNN	95.06
Deep LSTM	95.08
RNN-EM	95.22
Encoder-labeler LSTM(W)	95.40
CNN TriCRF	95.42
R-biRNN	95.47
5×R-biRNN	95.56
Bi-directional CNN	95.61
Encoder-labeler Deep LSTM(W)	95.66
Attention-Based RNN	95.78

TABLE 5.4: Baseline F1 scores on ATIS

- So, a sentence can easily be represented by a vector of integers.
- Next, we want to capture short-term dependencies between the words, so each word is accompanied by the words among which it is located. To do so, we apply a context window, which means that each word is not a single number any more, but a vector of numbers including the previous and the following words in the sentence. Thus, the sentence is now represented by a matrix of integer numbers.
- Finally, we associate this matrix of indexes to randomly selected embeddings which gives a matrix of real numbers.

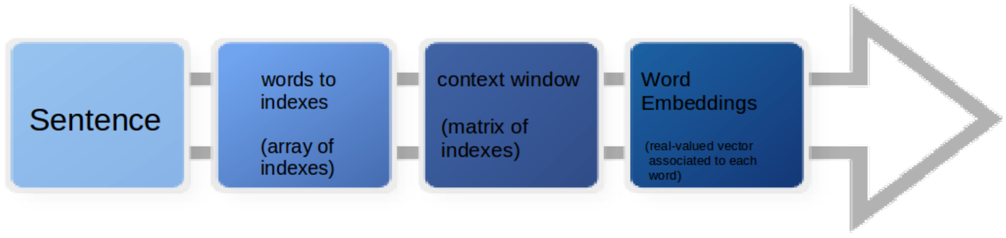

FIGURE 5.1: Data processing

Now that we have associated every single word to a real-valued vector, we have a continuous representation of the input to feed the neural network. Additional features could be incorporated into our model, however we use lexical features which are automatically extracted from word sequences to keep our model as simple as possible and make the model comparison more meaningful.

Sentence	<from montreal to las vegas>
IOB	['O', 'B-fromloc.city_name', 'O', 'B-toloc.city_name', 'I-toloc.city_name']
words to indexes	[208 319 502 260 539]

TABLE 5.5: Example of a sentence in ATIS

context window
-1, 208, 319
208, 319, 502
319, 502, 260
502, 260, 539
260, 539, -1

TABLE 5.6: context window example

Word Embeddings
-0.06971462, 0.07759826, 0.05064092, ..., 0.06876717, -0.05846993, -0.10999857
-0.06971462, 0.07759826, 0.05064092, ..., 0.10192625, 0.08745284, 0.04919793
-0.06971462, 0.07759826, 0.05064092, ..., -0.01003582, 0.10694719, 0.1247109
0.10038264, -0.10563177, -0.18760249, ..., -0.01003582, 0.10694719, 0.1247109
0.17441745, 0.16767969, -0.069546 , ..., -0.01003582, 0.10694719, 0.1247109

TABLE 5.7: Word Embeddings example

5.7 Experiments and Results

In this section all the experiments and the results are shown. All the experiments were conducted in order to maximize the models F1 score and to do so, we use the high-level programming language Python. To implement all the artificial neural networks we used the Theano toolkit [7]. For all of our neural networks we use random sampling of the hyper-parameters [8] (see Table 5.8).

parameter	**value**
Context window size	$\{3, 5, 7, 9\}$
Number of hidden units	200
Dimension of word embedding	$\{50, 100\}$
Learning rate	$rand(0.1 - 1)$
Seed	$rand(1 - 1000)$
early-stopping epochs	100
depth	$\{1, 2\}$
embedding matrix initialization	$unif(-1, 1)$
weight matrices initialization	$unif(-1, 1)$

TABLE 5.8: Model's parameters

5.8 Comparison of this work to the state-of-the-art

By construction, CW-RNN has less parameters than an RNN or an LSTM with the same number of neurons. For instance, assuming that a CW-RNN has its hidden layer partitioned into m modules and each module consists of k neurons (figure 5.2), this makes a total of $n_h = k \times m$ neurons. Since each neuron receives input only from the neurons having the same or larger clock period, the number of parameters for the recurrent matrix U of CW-RNN is:

$$parameters(U) = \sum_{j=1}^{m} jk^2 = \frac{n_h^2 + n_h k}{2} \tag{5.1}$$

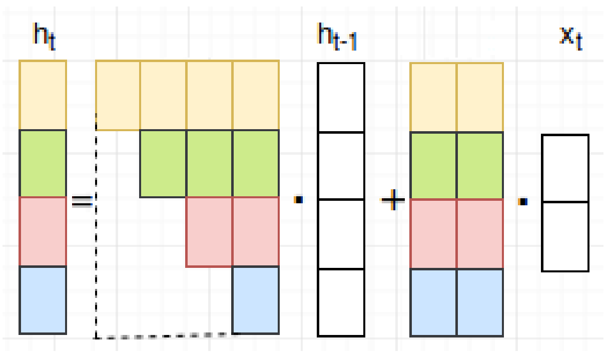

FIGURE 5.2: CW-RNN modules

For a direct comparison, the recurrent matrix of a similar sized RNN has n_h^2 parameters which means that the RNN has $\frac{2m}{m+1}$ more parameters than the CW-RNN. For instance, for a $m+1$ typical experiment with $m = 4$, the number of RNN parameters are 1.6 times more than the corresponding CW-RNN. Furthermore, although both the CW-RNN and the RNN have the same number of input connections, which are represented by matrix W this is not the case when LSTMs are considered. The blocks of an LSTM are described by 7 interconnection matrices, which account for $7 \times n_h^2$ parameters, and by 4 input matrices.

The CW-RNN also reduces the vanishing gradient problem which is prominent to other RNN-based architectures. Since a module i with period T_i is updated every T_i time steps and the back-propagation in time is also performed with steps of length T_i, the error signal for slow modules remains strong enough during longer time intervals. For example, given a sequence $\{(x_i, y_i)|i = 1, \ldots, N\}$ of data sample pairs, the parameter estimation formula for the slowest module m of the CW-RNN are identical to the formula corresponding to an RNN having the same size as the module m and is estimated from the following sequence of data samples:

$$\{(x_1, \bar{y}_t), (x_{T_m+1}, \bar{y}_{T_m+1}), (x_{2T_m+1}, \bar{y}_{2T_m+1}), \ldots\}$$

where x_t is now sampled at $t \in \{1, T_m + 1, 2T_m + 1, 3T_m + 1, \ldots\}$, instead of $t \in \{1, 2, \ldots\}$ in the Vanilla RNN. The mean value $\bar{y}_{iT_{m+1}}$ of y_t

for $iT_m + 1 \leq t \leq (i+1)T_m$,$i \in \{0, 1, \ldots\}$, is defined as:

$$\bar{y}_{iT_{m+1}} = \frac{1}{T_m} \sum_{t=iT_m+1}^{(i+1)T_m} y_t \tag{5.2}$$

Table 5.11 demonstrates the results of all the models implemented in this work and in table 5.3 we compare our results to the baseline results. Although we did not achieve state-of-the-art results, we managed to outperform some more complex models.

In Figure 5.3 we provide some statistics for a single experiment, where we can clearly observe that there is not parameter to guarantee generally high performance. The purpose of these diagrams is to show that we cannot predict the behavior of the model when changing the hyper-parameters and thus, for different data, the parameters of the model must be searched.

model	**F1**(%)
Convolutional CW-RNN	95.44
Bi-directional CW-RNN	95.31
Hybrid CW-RNN	95.25
CW-RNN	95.23
dropConnect CW-RNN	95.21
Deep CW-RNN	94.65

TABLE 5.9: Results of this work

model	**F1(%)**
RNN	94.11
CNN-CRF	94.35
Bi-directional RNN	94.73
LSTM	94.85
RNN-SOP	94.89
K-SAN	95.00
Hybrid RNN	95.06
Deep LSTM	95.08
RNN-EM	95.22
CW-RNN	**95.23**
Hybrid CW-RNN	**95.25**
Bi-directional CW-RNN	**95.31**
Encoder-labeler LSTM(W)	95.40
CNN TriCRF	95.42
Convolutional CW-RNN	**95.44**
R-biRNN	95.47
RSVM	95.50
5×R-biRNN	95.56
Bi-directional CNN	95.61
Encoder-labeler Deep LSTM(W)	95.66
Attention-Based RNN	95.78

TABLE 5.10: Best F1 scores on ATIS

Architecture	**CW-RNN**	**Vanilla RNN**
Elman-type	**95.23**	94.98
Hybrid	**95.25**	95.06
Bi-directional	**95.31**	94.73

TABLE 5.11: F1-scores of CW-models in this work compared to their simple versions for the same number of hidden units

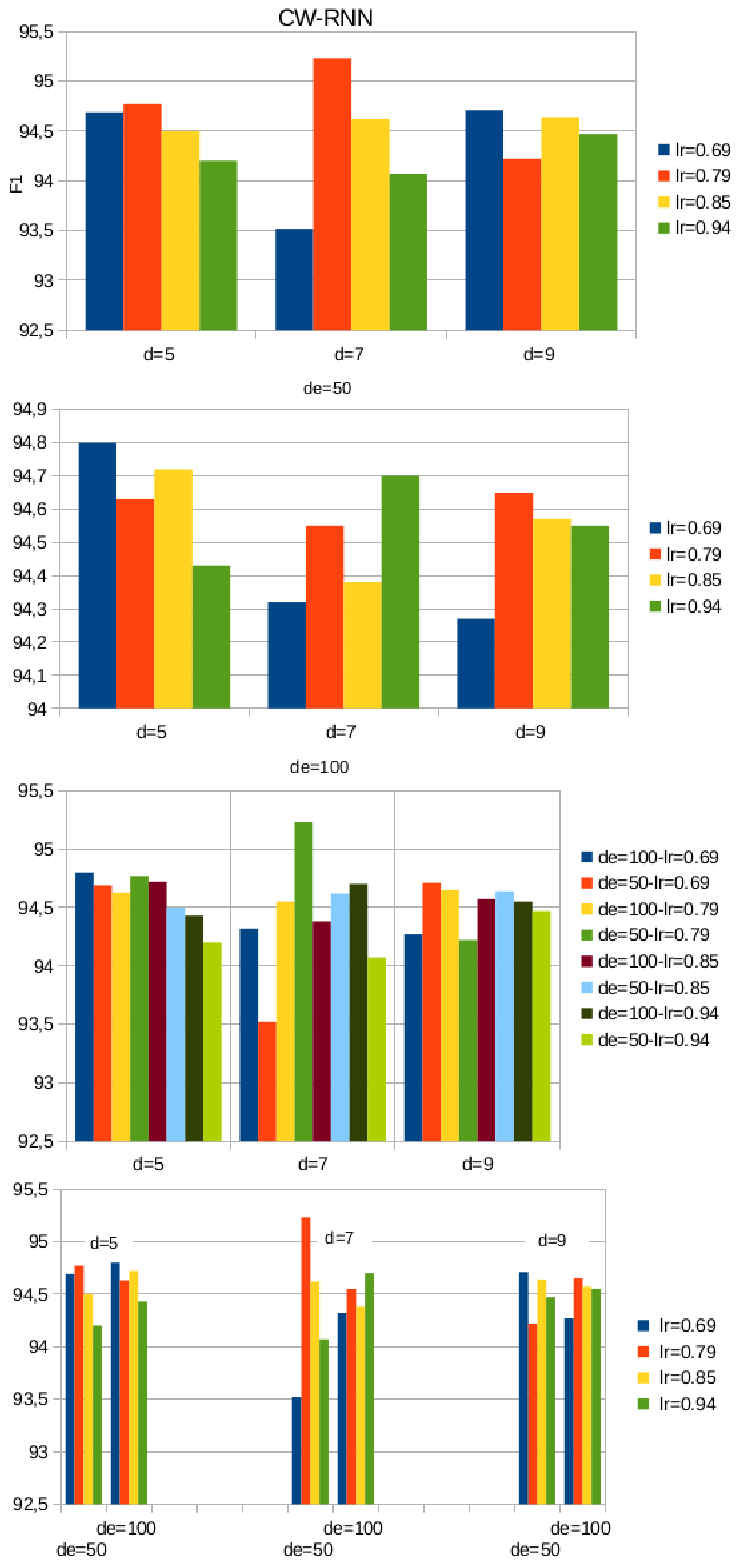

FIGURE 5.3: an example of the experiments

Chapter 6

Concluding remarks and future work

6.1 Concluding remarks

In this work, we presented a modified Recurrent Neural Network to perform a language understanding task and specifically the slot filling task in order to automatically extract significant information from utterances. Based on the RNN proposed in [9] we managed to boost the performance of this neural network without actually changing its architecture. More specifically, we created a simple RNN, then we broke it in 4 parts and we forced each part to run in an individual pace.

With this simple modification the ClockWork-RNN outperformed the simple RNN by 0.25%. Moreover, the hybrid version of our ClockWork-RNN outperformed the simple hybrid RNN by 0.19% and the bidirectional version of the ClockWork-RNN outperformed the simple bidirectional RNN by 0.58%. The Convolutional CW-RNN, which is a novel architecture proposed in the present work, led to even higher performance than these models and gave comparative to the state-of-the-art results.

The interesting part of the present work is that the slot filling task is dealt with a neural network architecture which manages to overcome the difficulties of the simple RNN, while at the same time, it has some superior features. More specifically, the Clockwork RNN is able to expand its memory exploiting its own structure, without using further external memory. By succeeding that, a CW-RNN can capture long-term dependencies and it reduces the vanishing gradient problem. As for the superior features, the CW-RNN architecture has less number of parameters than in the case of the simple RNN (almost the half parameters). Thus, the CW-RNN needs fewer operations per time step, so it is faster than RNN. Actually, the overall CW-RNN speed-up over RRN is m/4.

The importance of the proposed model lies on the fact that it provides an artificial neural network which could substitute the simple RNN wherever the latter is used in more complex architectures. It would be very interesting to find out if the baseline models could be boosted, by such a simple modification.

6.2 Future work

As future work, we have plenty possible extensions of the work. As previously mentioned, it would be very intriguing to test the performance of all the baseline models by substituting the simple RNN with the ClockWork-RNN in order to enhance these architectures. Another area of interest would be to check the adaptability of the proposed neural network when testing other datasets.

Furthermore, we are interesting in exploring new types of artificial Neural Networks to deal with noise owing to either recognition errors or speaker's error such as hesitations or side-speech.

Finally, our biggest interest is to search the possibilities of building a neural network for dialog state tracking, where the basic idea is to take into account the dialog history.

Appendix A

Appendix

ATIS Vocabulary and Classes

WORDS

0: "'d", 1: "'ll", 2: "'m", 3: "'re", 4: "'s", 5: "'t", 6: '72s', 7: '<UNK>', 8: 'DIGIT', 9: 'DIGITDIGIT', 10: 'DIGITDIGITDIGIT', 11: 'DIGITDIGITDIGITDIGIT', 12: 'DIGITDIGITDIGITDIGITDIGITDIGIT',

13: 'a', 14: 'abbreviation', 15: 'abbreviations', 16: 'about', 17: 'ac', 18: 'actually', 19: 'after', 20: 'afternoon', 21: 'again', 22: 'air', 23: 'aircraft', 24: 'airfare', 25: 'airline', 26: 'airlines', 27: 'airplane', 28: 'airplanes', 29: 'airport', 30: 'airports', 31: 'alaska', 32: 'all', 33: 'along', 34: 'also', 35: 'am', 36: 'america', 37: 'american', 38: 'amount', 39: 'an', 40: 'and', 41: 'angeles', 42: 'another', 43: 'any', 44: 'anywhere', 45: 'ap', 46: 'ap57', 47: 'ap80', 48: 'approximately', 49: 'april', 50: 'are', 51: 'area', 52: 'arizona', 53: 'around', 54: 'arrange', 55: 'arrangements', 56: 'arrival', 57: 'arrivals', 58: 'arrive', 59: 'arrives', 60: 'arriving', 61: 'as', 62: 'at', 63: 'atl', 64: 'atlanta', 65: 'august', 66: 'available',

67: 'b', 68: 'back', 69: 'baltimore', 70: 'be', 71: 'beach', 72: 'before', 73: 'between', 74: 'boeing', 75: 'book', 76: 'booking', 77: 'boston', 78: 'both', 79: 'bound', 80: 'breakfast', 81: 'burbank', 82: 'business', 83: 'but', 84: 'buy', 85: 'bwi', 86: 'by',

87: 'c', 88: 'california', 89: 'can', 90: 'canada', 91: 'canadian', 92: 'capacity', 93: 'car', 94: 'carolina', 95: 'carries', 96: 'cars', 97: 'charlotte', 98: 'cheap', 99: 'cheapest', 100: 'chicago', 101: 'choices', 102: 'cincinnati', 103: 'cities', 104: 'city', 105: 'class', 106: 'classes', 107: 'cleveland', 108: 'close', 109: 'co', 110: 'coach', 111: 'code', 112: 'codes', 113: 'colorado', 114: 'columbus', 115: 'coming', 116: 'connect', 117: 'connecting', 118: 'connection', 119: 'connections', 120: 'continental', 121: 'cost', 122: 'costs', 123: 'could', 124: 'county', 125: 'cp',

126: 'd', 127: 'daily', 128: 'dallas', 129: 'database', 130: 'day', 131: 'days', 132: 'dc', 133: 'dc10', 134: 'december', 135: 'define', 136: 'delta', 137: 'denver', 138: 'depart', 139: 'departing', 140: 'departs', 141: 'departure', 142: 'departures', 143: 'describe', 144: 'destination', 145: 'detroit', 146: 'dfw', 147: 'diego', 148: 'difference', 149: 'different', 150: 'dinner', 151: 'dinnertime', 152: 'direct', 153: 'display', 154: 'distance', 155: 'dl', 156: 'do', 157:

′does′, 158: ′dollars′, 159: ′downtown′, 160: ′dulles′, 161: ′during′,

162: ′ea′, 163: ′each′, 164: ′earlier′, 165: ′earliest′, 166: ′early′, 167: ′eastern′, 168: ′economy′, 169: ′eight′, 170: ′eighteenth′, 171: ′eighth′, 172: ′either′, 173: ′eleventh′, 174: ′evening′, 175: ′ewr′, 176: ′expensive′, 177: ′explain′, 178: ′express′,

179: ′f′, 180: ′f28′, 181: ′far′, 182: ′fare′, 183: ′fares′, 184: ′february′, 185: ′ff′, 186: ′field′, 187: ′fifteenth′, 188: ′fifth′, 189: ′find′, 190: ′first′, 191: ′fit′, 192: ′flies′, 193: ′flight′, 194: ′flights′, 195: ′florida′, 196: ′fly′, 197: ′flying′, 198: ′fn′, 199: ′following′, 200: ′for′, 201: ′fort′, 202: ′four′, 203: ′fourteenth′, 204: ′fourth′, 205: ′francisco′, 206: ′friday′, 207: ′friends′, 208: ′from′,

209: ′general′, 210: ′georgia′, 211: ′get′, 212: ′give′, 213: ′go′, 214: ′goes′, 215: ′going′, 216: ′great′, 217: ′ground′, 218: ′guardia′,

219: ′h′, 220: ′has′, 221: ′have′, 222: ′heading′, 223: ′hello′, 224: ′help′, 225: ′here′, 226: ′hi′, 227: ′highest′, 228: ′hours′, 229: ′houston′, 230: ′how′, 231: ′hp′,

232: ′i′, 233: ′if′, 234: ′in′, 235: ′include′, 236: ′indianapolis′, 237: ′information′, 238: ′interested′, 239: ′international′, 240: ′into′, 241: ′is′, 242: ′it′, 243: ′itinerary′,

244: ′january′, 245: ′jersey′, 246: ′jfk′, 247: ′jose′, 248: ′july′, 249: ′june′, 250: ′just′,

251: ′kansas′, 252: ′kind′, 253: ′kinds′, 254: ′know′,

255: ′la′, 256: ′lake′, 257: ′land′, 258: ′landing′, 259: ′landings′, 260: ′las′, 261: ′last′, 262: ′lastest′, 263: ′late′, 264: ′later′, 265: ′latest′, 266: ′layover′, 267: ′least′, 268: ′leave′, 269: ′leaves′, 270: ′leaving′, 271: ′less′, 272: ′let′, 273: ′like′, 274: ′limo′, 275: ′limousine′, 276: ′list′, 277: ′listing′, 278: ′live′, 279: ′lives′, 280: ′located′, 281: ′logan′, 282: ′long′, 283: ′look′, 284: ′looking′, 285: ′los′, 286: ′louis′, 287: ′love′, 288: ′lowest′, 289: ′lufthansa′, 290: ′lunch′,

291: ′m′, 292: ′m80′, 293: ′make′, 294: ′makes′, 295: ′making′, 296: ′many′, 297: ′march′, 298: ′maximum′, 299: ′may′, 300: ′mco′, 301: ′me′, 302: ′meal′, 303: ′meals′, 304: ′mean′, 305: ′meaning′, 306: ′memphis′, 307: ′miami′, 308: ′michigan′, 309: ′midnight′, 310: ′midway′, 311: ′midwest′, 312: ′milwaukee′, 313: ′minneapolis′, 314: ′minnesota′, 315: ′missouri′, 316: ′mitchell′, 317: ′monday′, 318: ′month′, 319: ′montreal′, 320: ′more′, 321: ′morning′, 322: ′mornings′, 323: ′most′, 324: ′much′, 325: ′my′,

326: ′name′, 327: ′names′, 328: ′nashville′, 329: ′nationair′, 330: ′near′, 331: ′need′, 332: ′new′, 333: ′newark′, 334: ′next′, 335: ′night′, 336: ′nineteenth′, 337: ′ninth′, 338: ′no′, 339: ′nonstop′, 340: ′nonstops′, 341: ′noon′, 342: ′noontime′, 343: ′north′, 344: ′northwest′, 345: ′november′, 346: ′now′, 347: ′number′, 348: ′numbers′, 349: ′nw′, 350: "o′clock", 351: ′oakland′, 352: ′october′, 353: ′of′, 354: ′offer′, 355: ′offers′, 356: ′ohio′, 357: ′okay′, 358: ′on′, 359: ′one′, 360: ′only′, 361: ′ontario′, 362: ′options′, 363: ′or′, 364: ′ord′, 365: ′originate′, 366: ′originating′, 367: ′orlando′, 368: ′other′, 369: ′out′, 370: ′over′,

371: ′passengers′, 372: ′paul′, 373: ′pennsylvania′, 374: ′people′, 375: ′petersburg′, 376: ′philadelphia′, 377: ′philly′, 378: ′phoenix′, 379: ′pittsburgh′, 380: ′plan′, 381: ′plane′, 382: ′planes′, 383: ′please′, 384: ′pm′, 385: ′possible′, 386: ′price′, 387: ′prices′, 388: ′provide′, 389: ′provided′,

390: ′q′, 391: ′qo′, 392: ′quebec′, 393: ′qw′, 394: ′qx′,

395: ′rate′, 396: ′rates′, 397: ′reaching′, 398: ′rent′, 399: ′rental′, 400: ′rentals′, 401: ′repeat′, 402: ′requesting′, 403: ′reservation′, 404: ′reservations′, 405: ′restriction′, 406: ′restrictions′, 407: ′return′, 408: ′returning′, 409: ′right′, 410: ′round′,

411: ′s′, 412: ′sa′, 413: ′salt′, 414: ′same′, 415: ′san′, 416: ′saturday′, 417: ′saturdays′, 418: ′say′, 419: ′schedule′, 420: ′schedules′, 421: ′seating′, 422: ′seats′, 423: ′seattle′, 424: ′second′, 425: ′see′, 426: ′september′, 427: ′serve′, 428: ′served′, 429: ′serves′, 430: ′service′, 431: ′serviced′, 432: ′services′, 433: ′serving′, 434: ′seventeenth′, 435: ′seventh′, 436: ′sfo′, 437: ′shortest′, 438: ′should′, 439: ′show′, 440: ′six′, 441: ′sixteen′, 442: ′sixteenth′, 443: ′sixth′, 444: ′smallest′, 445: ′snack′, 446: ′so′, 447: ′some′, 448: ′sometime′, 449: ′soon′, 450: ′sorry′, 451: ′southwest′, 452: ′st.′, 453: ′stand′, 454: ′stands′, 455: ′stapleton′, 456: ′starting′, 457: ′still′, 458: ′stop′, 459: ′stopover′, 460: ′stopovers′, 461: ′stopping′, 462: ′stops′, 463: ′sunday′, 464: ′sundays′, 465: ′sure′,

466: ′tacoma′, 467: ′take′, 468: ′takeoff′, 469: ′takeoffs′, 470: ′taking′, 471: ′tampa′, 472: ′taxi′, 473: ′tell′, 474: ′ten′, 475: ′tennessee′, 476: ′tenth′, 477: ′texas′, 478: ′than′, 479: ′thank′, 480: ′that′, 481: ′the′, 482: ′their′, 483: ′then′, 484: ′there′, 485: ′these′, 486: ′they′, 487: ′third′, 488: ′thirteenth′, 489: ′thirtieth′, 490: ′thirty′, 491: ′this′, 492: ′those′, 493: ′three′, 494: ′thrift′, 495: ′through′, 496: ′thursday′, 497: ′thursdays′, 498: ′ticket′, 499: ′tickets′, 500: ′time′, 501: ′times′, 502: ′to′, 503: ′today′, 504: ′tomorrow′, 505: ′too′, 506: ′toronto′, 507: ′total′, 508: ′tower′, 509: ′train′, 510: ′transcontinental′, 511: ′transport′, 512: ′transportation′, 513: ′travel′, 514: ′traveling′, 515: ′trip′, 516: ′trips′, 517: ′trying′, 518: ′tuesday′, 519: ′tuesdays′, 520: ′turboprop′, 521: ′twa′, 522: ′twelfth′, 523: ′twentieth′, 524: ′twenty′, 525: ′two′, 526: ′type′, 527: ′types′,

528: ′ua′, 529: ′under′, 530: ′united′, 531: ′up′, 532: ′us′, 533: ′use′, 534: ′used′, 535: ′uses′, 536: ′using′, 537: ′utah′,

538: ′various′, 539: ′vegas′, 540: ′very′, 541: ′via′,

542: ′want′, 543: ′washington′, 544: ′way′, 545: ′we′, 546: ′wednesday′, 547: ′wednesdays′, 548: ′week′, 549: ′weekday′, 550: ′weekdays′, 551: ′well′, 552: ′west′, 553: ′westchester′, 554: ′what′, 555: ′when′, 556: ′where′, 557: ′which′, 558: ′who′, 559: ′will′, 560: ′wish′, 561: ′with′, 562: ′within′, 563:

'without', 564: 'worth', 565: 'would',

566: 'y', 567: 'yes', 568: 'yn', 569: 'york', 570: 'you', 571: 'your'

Classes

In this section we provide the reader with a table containing all the possible slot tags or labels as it is used to calling them. These labels, shown in Table A.1, are really descriptive and one can understand easily what kind of information it would include. The tag which is not so obvious is the 'O'. The same is for the first letter of each tag which is B or I.

These three letters B, I, O represent the words Beginning, Inside and Outside respectively. By using this type of tagging we actually represent each sentence with the use of the IOB format [https://en.wikipedia.org/wiki /Inside_Outside_Beginning]. The Inside Outside Beginning (IOB) representation is a popular way of tagging in computational linguistics. The letter B is the prefix before a labels which indicates the beginning of a chunk. For instance, considering that the departure city is 'New York' the word 'New' would have the label 'B-fromloc.city_name', whereas the word 'York' would have the label 'I-fromloc.city_name'. B is also used when a tag is followed by a tag of the same type without O tokens between them.

Similarly, the letter I is the prefix of a label indicates that the label is inside a chunk. Finally, the letter O indicates that a token does not belong to chunk, which in practice means that the specific word is of no interest for the specific task.

ATIS Classes	
0: 'B-aircraft_code'	1:'B-airline_code'
2: 'B-airline_name'	3: 'B-airport_code'
4: 'B-airport_name'	5: 'B-arrive_date.date_relative'
6: 'B-arrive_date.day_name'	7:'B-arrive_date.day_number'
8: 'B-arrive_date.month_name'	9: 'B-arrive_date.today_relative'
10: 'B-arrive_time.end_time'	11:'B-arrive_time.period_mod'
12: 'B-arrive_time.period_of_day'	13: 'B-arrive_time.start_time'
14: 'B-arrive_time.time'	15: 'B-arrive_time.time_relative'
16: 'B-booking_class'	17: 'B-city_name'
18: 'B-class_type'	19: 'B-compartment'
20: 'B-connect'	21: 'B-cost_relative'
22: 'B-day_name'	23: 'B-day_number'
24: 'B-days_code'	25: 'B-depart_date.date_relative'
26: 'B-depart_date.day_name'	27: 'B-depart_date.day_number'
28: 'B-depart_date.month_name'	29: 'B-depart_date.today_relative'
30: 'B-depart_date.year'	31: 'B-depart_time.end_time'
32: 'B-depart_time.period_mod'	33: 'B-depart_time.period_of_day'
34: 'B-depart_time.start_time'	35: 'B-depart_time.time'
36: 'B-depart_time.time_relative'	37: 'B-economy'
38: 'B-fare_amount'	39: 'B-fare_basis_code'
40: 'B-flight'	41: 'B-flight_days'
42: 'B-flight_mod'	43: 'B-flight_number'
44: 'B-flight_stop'	45: 'B-flight_time'
46: 'B-fromloc.airport_code'	47: 'B-fromloc.airport_name'
48: 'B-fromloc.city_name'	49: 'B-fromloc.state_code'
50: 'B-fromloc.state_name'	51: 'B-meal'
52: 'B-meal_code'	53: 'B-meal_description'
54: 'B-mod'	55: 'B-month_name'
56: 'B-or'	57: 'B-period_of_day'
58: 'B-restriction_code'	59: 'B-return_date.date_relative'
60: 'B-return_date.day_name'	61: 'B-return_date.day_number'
62: 'B-return_date.month_name'	63: 'B-return_date.today_relative'
64: 'B-return_time.period_mod'	65: 'B-return_time.period_of_day'
66: 'B-round_trip'	67: 'B-state_code'
68: 'B-state_name'	69: 'B-stoploc.airport_code'

TABLE A.1: ATIS dataset classes(1)

ATIS Classes	
70: 'B-stoploc.airport_name'	71: 'B-stoploc.city_name'
72: 'B-stoploc.state_code'	73: 'B-time'
74: 'B-time_relative'	75: 'B-today_relative'
76: 'B-toloc.airport_code'	77: 'B-toloc.airport_name'
78: 'B-toloc.city_name'	79: 'B-toloc.country_name'
80: 'B-toloc.state_code'	81: 'B-toloc.state_name'
82: 'B-transport_type'	83: 'I-airline_name'
84: 'I-airport_name'	85: 'I-arrive_date.day_number'
86: 'I-arrive_time.end_time'	87: 'I-arrive_time.period_of_day'
88: 'I-arrive_time.start_time'	89: 'I-arrive_time.time'
90: 'I-arrive_time.time_relative'	91: 'I-city_name'
92: 'I-class_type'	93: 'I-cost_relative'
94: 'I-depart_date.day_number'	95: 'I-depart_date.today_relative'
96: 'I-depart_time.end_time'	97: 'I-depart_time.period_of_day'
98: 'I-depart_time.start_time'	99: 'I-depart_time.time'
100: 'I-depart_time.time_relative'	101: 'I-economy'
102: 'I-fare_amount'	103: 'I-fare_basis_code'
104: 'I-flight_mod'	105: 'I-flight_number'
106: 'I-flight_stop'	107: 'I-flight_time'
108: 'I-fromloc.airport_name'	109: 'I-fromloc.city_name'
110: 'I-fromloc.state_name'	111: 'I-meal_code'
112: 'I-meal_description'	113: 'I-restriction_code'
114: 'I-return_date.date_relative'	115: 'I-return_date.day_number'
116: 'I-return_date.today_relative'	117: 'I-round_trip'
118: 'I-state_name'	119: 'I-stoploc.city_name'
120: 'I-time'	121: 'I-today_relative'
122: 'I-toloc.airport_name'	123: 'I-toloc.city_name'
124: 'I-toloc.state_name'	125: 'I-transport_type'
126: 'O'	

TABLE A.2: ATIS dataset classes(2)

Bibliography

[1] Susan J. Boyce and Allen L. Gorin, "User Interface Issues for Natural Spoken Dialog Systems", Proceeding of the *international symposium on spoken dialog*, 1996

[2] G.Tur and R.DeMori, "Spoken Language Understanding: Systems for Extracting Semantic Information from Speech.", NewYork, NY, USA: Wiley, 2011.

[3] Dan Jurafsky and James H. Martin, "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.", Prentice Hall, 2009

[4] J. Elman "Finding structure in time" in Cognitive Science, 14 (2), 1990.

[5] M. Jordan "Serial order: a parallel distributed processing approach" in Tech. Rep no 8604, San Diego, University of California, Institute of Computer Science, 1997.

[6] MacLeod Christopher, "An Introduction to Practical Neural Networks and Genetic Algorithms For Engineers and Scientists", Robert Gordon University, 2010

[7] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio, "Theano: A CPU and GPU Math Expression Compiler." Proc. Python for Scientific Computing Conference (SciPy) 2010.

[8] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization", in Journal of Machine Learning Research 13 (2012) 281-305, 2012.

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

[10] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding.", in *Interspeech*, 2007.

[11] J. Lafferty, A. McCallum and F. Pereira, "Conditional randomfields: Probabilistic models for segmenting and labeling sequence data." in *ICML*, 2001

[12] T. Kudo and Y. Matsumoto, "Chunking with support vector machine.", in *ACL* 2001.

[13] R. Sarikaya, G. E. Hinton, and B. Ramabhadran, "Deep belief nets for natural language call-routing." in *ICASSP*, 2011.

[14] K. Yao, G. Zweig, M-Y. Hwang, Y. Shi and D. Yu, "Recurrent neural networks for language understanding." in *Interspeech*, 2013.

[15] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi, "Spoken language understanding using long short-term memory neural networks.", in *IEEE SLT*, 2014.

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

[17] B. Peng, K. Yao. "Recurrent Neural Networks with External Memory for Language Understanding", in *NLPCC*, ser. Lecture Notes in Computer Science, vol. 9362. Springer, 2015, pp. 25–35

[18] G. Kurata, B. Xiang, B. Zhou, and M. Yu, "Leveraging sentence-level information with encoder lstm for natural language understanding", *CoRR*, vol. abs/1601.01530, 2016.

[19] P. Xu and R. Sarikaya, "Convolutional neural nteworks based triangular CRF for joint intent detection and slot filling", in *ASRU*, 2013.

[20] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. "A clockwork rnn." *ICML*, 2014.

[21] Yangyang Shi and Kaisheng Yao and Hu Chen and Dong Yu Yi-Cheng Pan and Mei-Yuh Hwang, "Recurrent Support Vector Machines For Slot Tagging In Spoken Language Understanding", in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Microsoft*, 2016.

[22] Ngoc Thang Vu, Pankaj Gupta, Heike Adel, Hinrich Schutze, "Bidirectional recurrent neural network with ranking loss for spoken language understanding", in 2016 *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016*, Shanghai, China, March 20-25, 2016, 2016, pp. 6060–6064.

[23] G. Mesnil, X. He, L. Deng and Y. Bengio, "Investigation of Recurrent-Neural-Network architectures and learning methods for spoken language understanding" in *Interspeech*, 2013.

[24] Precision and recall. URL http://en.wikipedia.org/wiki/Precision _and_recall.

[25] Bing Liu and Ian Lane, "Recurrent neural network structured output prediction for spoken language understanding." In *Proc. NIPS Workshop on Machine Learning for Spoken Language Understanding and Interactions*, 2015.

[26] J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization", Journal of Machine Learning Research, vol. 12, 2010.

[27] M.D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method." *CoRR*, 2012.

[28] G. Tur and R. De Mori, "Spoken Language Understanding: Systems for Extracting Semantic Information from Speech" in *Eds. John Wiley and Sons*, 2011

[29] Y. Wang, L. Deng and A. Acero, "Semantic Frame Based Spoken Language Understanding", Chapter 3, *Eds. John Wiley and Sons*, 2011.

[30] G. Tur and D. Hakkani-Tur and L. Heck and S. Parthasarathy, "Sentence simplification for spoken language understanding", in *ICASSP*. pp. 5628-5631, 2011.

[31] G. Tur, D. Hakkani-Tür, and L. Heck, "What is Left to be Understood in ATIS", in *IEEE SLT*, 2010.

[32] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of *Machine Learning Research*, 2011.

[33] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding", in *INTERSPEECH*, 2007.

[34] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa, "Natural language processing (almost) from scratch" in *Journal of Machine Learning Research*, vol 12, 2011.

[35] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *CIKM*, 2014

[36] P. Xu, R Sarikaya, "Joint Intent Detection and Slot Filling with Convolutional Neural Networks", *IEEE Automatic Speech Recognition and Understanding Workshop*, 2013.

[37] J. Lafferty, A. McCallum and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data" in *ICML*, 2001

[38] P. Liu, X. Qiu, X. Chen, S. Wu, X. Huang, "Multi-Timescale Long Short-Term Memory Neural Network for Modelling Sentences and Documents", *Conference on EMNLP*, 2015, L. Marquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, Eds.The Association for Computational Linguistics, pp. 2326-2335, 2015

[39] D. Hakkani-Tur, G.Tur, A.Celikyilmaz, Y.N.Chen, J.Gao, L.Deng, and Y.Wang, "Multi-domain joint semantic frame parsing using bi-directional rnn-lstm", in Proceedings of *The 17th Annual Meeting of the International Speech Communication Association (INTERSPEECH)*. San Francisco, *CA:ISCA*, 2016.

[40] B.Liu and I.Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling", *CoRR*, vol. abs/1609.01454, 2016. [Online]. Available: http://arxiv.org/abs/1609.01454

[41] Y.Chen, D.Z.Hakkani-Tur, G.Tur, A.Celikyilmaz, J.Gao, and L. Deng, "Knowledge as a teacher: Knowledge-guided structural attention networks", *CoRR*, vol. abs/1609.03286, 2016. [Online]. Available: http://arxiv.org/abs/1609.03286

[42] N. T. Vu, "Sequential convolutional neural networks for slot filling in spoken language understanding", *CoRR*, vol. abs/1606.07783, 2016

[43] P. J. Price, "Evaluation of spoken language systems: The ATIS domain", in Proc. of the *Speech and Natural Language Workshop*, 1990, pp. 91–95.

[44] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model", in Advances in Neural Information Processing Systems 13, Papers from *Neural Information Processing Systems (NIPS) 2000*, Denver, CO, USA, 2000, pp. 932–938.

[45] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding", in *INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association*, Antwerp, Belgium, August 27-31, 2007, pp. 1605–1608.

[46] Tjong Kim Sang and Buchholz, Introduction to the conll-2000 shared task: Chunking. In Proceedings of the *2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, 127–132. Association for Computational Linguistics, 2000.

[47] Dan Jurafsky, NLP course, *Stanford University*, *https://web.stanford.edu/ jurafsky/NLPCourseraSlides.html*