TÉCNICO LISBOA

Human-robot speech interaction for service robots

Pedro Henrique Alves Martins

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Luís Manuel Marques Custódio
Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. José Fernando Alves da Silva
Supervisor: Prof. Luís Manuel Marques Custódio
Member of the Committee: Prof. André Filipe Torres Martins

November 2017

Dedicated to Cláudia, Ana Paula, António Amândio e Rui Manuel

Acknowledgments

I would first like to express my sincere gratitude to my thesis advisors, Prof. Luís Custódio and Prof. Rodrigo Ventura for the useful comments, remarks and engagement through the development of this master thesis. I would like to thank Prof. Pedro Miraldo and Tiago Dias for allowing, and helping me using a server to run the models used in this work.

I would also like to thank the SocRob team for all the knowledge that they transmitted me, specially Oscar Lima.

Resumo

Robôs de serviço têm o propósito de ajudar o ser humano realizando as mais variadas tarefas. Como o meio de comunicação mais comum entre humanos é a fala, é importante que os robôs a consigam perceber. Além disso, seria mais fácil integrar os robôs na sociedade se estes fossem capazes de compreender linguagem natural.

Neste trabalho, um sistema capaz de compreender linguagem natural é desenvolvido. Este sistema pode também ser usado em robôs de serviço, só sendo preciso alterar o dataset de acordo com o conjunto de acções que o robô realizará e treinar as redes neuronais e a $Support\ Vector\ Machine$.

Primeiramente, como as instruçoes que os robôs recebem podem conter mais do que um comando, um divisor de frases em orações é desenvolvido, usando o Parsey McParseface da Google [1]. De seguida, diferentes métodos são experimentados no que diz respeito aos vectores de palavras que irão entrar nas redes neuronais, sendo que os melhores resultados foram obtidos quando os vectores usados foram criados usando o algoritmo GloVe. Para a detecção da intenção, detectar a intenção pedida no comando, e detecção de argumentos, detectar os argumentos contidos no comando, como $pessoa$, $local$ e $objecto$, várias arquitecturas de $Recurrent\ Neural\ Networks$ e $Long\ Short\ Term\ Memory\ networks$ foram testadas, sendo que $LSTMs$ obtiveram melhores resultados. Como a acção requerida pelo humano pode não estar contida no grupo de acções que o robô realiza, uma $Support\ Vector\ Machine$ é usada para determinar se pertence ou não. Para implementar este sistema no robô um pacote de ROS foi criado usando uma máquina de estados $SMACH$.

Palavras-chave: Compreensão de Linguagem Natural, Robôs de Serviço, Redes Neuronais, Detecção da intenção, Detecção de argumentos, Vectores de palavras

Abstract

Service robots have the purpose of helping humans by performing a wide range of tasks. As the most used way of communication between humans is speech, it is important that robots are able to understand it. Moreover, it would be easier to integrate robots in society if they were able to understand natural language.

In this work, a system that is able to understand natural language instructions is developed. This system can be used in service robots, by just changing the datasets for the appropriate task and training the neural networks and Support Vector Machine.

Firstly, as the instructions given to robots can contain more than one command, a phrase divider is developed, in which the Parsey McParseface from Google is used. Following, some experimentation is done concerning the word vectors that will be the input of the neural networks used, being that the best results are achieved when the vectors are developed by the GloVe algorithm. For the intent detection, detects the action requested by the command, and the slot filling, determines the arguments contained in the command, such as $person$, $location$ and $object$, various architectures of Recurrent Neural Networks and Long Short Term Memory networks are tested, achieving LSTMs a superior accuracy. As the action requested by the human can be one that is not in the robot's domain, a Support Vector Machine is used to determine whether it is or not. To implement the system in a robot, a ROS package is created using a SMACH state machine.

Keywords: Natural Language Understanding, Service Robots, Neural Networks, Intent Detection, Slot Filling, Word Embeddings

Contents

List of Tables

List of Figures

Acronyms

ADAM Adaptive Moment Estimation.

ATIS Airline Travel Information System.

BLSTM Bidirectional Long Short Term Memory.

BRNN Bidirectional Recurrent Neural Network.

CRF Conditional Random Field.

DBLSTM Deep Bidirectional Long Short Term Memory.

DBRNN Deep Bidirectional Recurrent Neural Network.

DLSTM Deep Long Short Term Memory.

DRNN Deep Recurrent Neural Network.

ERL European Robotics League.

FBM3 Functional Benchmark 3.

GloVe Global Vectors.

GPSR General Purpose Service Robots.

LSTM Long Short Term Memory.

LU4R Language Understanding chain For Robots.

NLU Natural Language Understanding.

RNN Recurrent Neural Network.

ROS Robotics Operating System.

SGD Stochastic Gradient Descent.

SVM Support Vector Machine.

Chapter 1

Introduction

This chapter aims to briefly explain the relevance of human-robot speech interaction for service robots. A short overview of natural language understanding and the thesis objectives are exposed, as well as its outline.

1.1 Motivation

A service robot is a robot that has the purpose of helping humans, by assisting them on a wide range of tasks. There are two main subclasses of service robots, professional or industrial robots and domestic robots. Professional robots consist in robots that are used for commercial tasks that can go from welding and building cars to pick and place products. A domestic robot is primarily used for household chores, such as cleaning and storing objects, but can also be useful in therapy, assistance or entertainment.

Robotics holds tremendous potential to benefit humans in every aspect of life. These benefits have been increasingly availed in industrial environments, such as factories. Due to the technological evolution, robots are also starting to be integrated into the human environment, for everyday use. However, this integration can only be successful if the robots are able to interact with humans. Since language is the major and easiest way of communication between humans, Natural Language Understanding (NLU) will probably play a huge part in this. Furthermore, it can be helpful in the public acceptance, once it would be easier for people to trust robots that have the ability of understanding them.

One example of a robot in which interaction abilities are fundamental is Gasparzinho or MOnarCH, Multi-Robot Cognitive Systems Operating in Hospitals, [2]. It is a service robot that helps hospitalized children, by playing and interacting with them, improving the quality of the time that they spend in the hospital. This would not be possible if Gasparzinho could not interact with the children.

The introduction of service robots can also be helpful to people that require assistance, such as people with disabilities or the elderly. In Europe, there are about 70 million people with disabilities that require assistance [3], almost 10% of the population. With the advances in medicine and healthier lifestyles, a considerable increase in life expectancy has been observed. In the past 50 years it has augmented 10 years for people living in Europe [4]. There has also been observed notorious aging of

the population. In Europe the percentage of people with ages between 65 and 79 years, and with more than 80 years has increased by 1.1% and 1.3%, respectively [5]. As a consequence, more people will need help to continue having a normal life. Service robots could provide some of the needed assistance, helping them to be more independent.

Another aspect in which these robots could benefit humans is in saving time, by performing dull activities that are necessary. Nowadays, people are giving a greater importance to their careers, having less time for family and hobbies. With the insertion of domestic robots, the time spent in household chores could be spent with more pleasant activities.

Presently, most of the tasks performed by professional robots do not require natural language understanding. However, in the near future it is probable that robots will start having functions for which they need to interact with users and, consequently, will need to be able to understand natural language, such as serving at bars and restaurants or being cashiers at supermarkets.

The robotics team of Instituto Superior Técnico, SocRob, participates in two international robotics competitions, RoboCup and European Robotics League, ERL. In these competitions there are tasks in which the ability of understanding natural language is evaluated. In both competitions, one of the tests is General Purpose Service Robots, GPSR, in which the robots receive a voice instruction, that can contain more than one command. The robot has to understand these commands and perform the requested tasks. In ERL there is also a Speech Recognition Functionality Benchmark, FBM3, in which the robot receives audio files and has to understand them and write a file with the correspondent transcriptions, intents and arguments.

1.2 Natural Language Understanding

This work is focused in natural language understanding for service robots. NLU is an area of natural language processing in artificial intelligence that has been studied for more than 50 years. It encompasses a complex AI challenge, that consists in handling unstructured inputs governed by poorly defined and flexible rules, grammar, and convert them into a structured form that a machine can understand and act upon.

It has been a long-term dream to create machines with the ability to communicate with people by voice.

1.3 Objectives

The purpose of this thesis is to develop a program that will be able to understand the intent of an instruction, intent detection, and the meaning of the keywords in the utterance, slot filling, so that the robot knows what action it has to perform. An example of intent detection and slot filling is presented in figure 1.1.

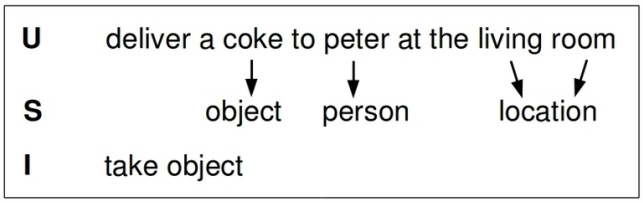


Figure 1.1: An example utterance (U) with respective slot labelling (S) and intent (I)

As can be seen in figure 1.1, in the utterance ”deliver a coke to peter at the living room”, the intent is ”take object”. With this information, the robot will know the action that needs to perform, take an object to someplace or deliver it to someone. It can also be observed that there are three relevant slots, an object, a person and a location. This completes the information that the robot will need to perform the action. The robot will know the object to deliver to which person and the location of the person. Also, if the robot does not have all the information needed to perform the desired action, it will be able to know what information is missing and ask for it.

As an instruction can contain more than one command, action of the robot, before applying the natural language understanding system, it has to be divided in phrases, being that each phrase corresponds to a command.

In addition to creating the system, it will also be implemented in the MOnarCH robot used by the SocRob team. This will be done by creating a ROS package. Furthermore, although being made for this robot, the package can be used in other robots and for other sets of actions and arguments, by just training the models with different datasets.

This thesis also aims to provide to the SocRob team a natural language understanding program that is able to understand the instructions asked in the GPSR task in the RoboCup and ERL. The system developed will also be used in FBM3 of ERL, despite not being its purpose once there are some differences related to the task of determining the keywords, such as, objects, locations, persons. In FBM3 the articles and adjectives that precede the keywords are also part of the slot.

Although the algorithms used in this thesis have already been used for other NLU tasks, as far as my knowledge goes, they have never been used together and applied in service robots. So, being these objectives fulfilled, not only a NLU system will be available for the SocRob team to use in competitions but it will also contribute to the NLU and robotics science fields.

In sum, the following tasks will be performed:

- Implementation of two word embedding algorithms, GloVe and Word2vec Skip-gram, in Python.
- Development and training of neural networks models for the intent detection and slot filling tasks, using the Tensorflow Python API.
- Implementation of a Support Vector Machine, using Scikit-Learn Python API, to identify if the instruction given to the robot is part of its set of actions.
- Development of an algorithm to divide sentences in phrases.

- Development of a generator of instructions for robots with automatic annotations, to train the neural networks.
- Development of a script to test the models created.
- Development of a ROS package to implement the system created in the robot.

1.4 Thesis Outline

In chapter 2, Background, some of previous attempts to solve this or similar problems are exposed. Some algorithms, such as Recurrent Neural Networks with and without Long Short Term Memory cells, and methods, like Word2Vec and GloVe, are also described.

In the chapter 3, the problem to solve will be presented and the pipeline of the system created in this thesis is thoroughly described.

The Results chapter, 4, reports the results achieved in several test sets, using different methods and algorithms. Possible reasons for these outcomes are also presented.

Finally, in chapter 5, the objectives achieved are reported and possible future work is described.

Chapter 2

Background

The natural language understanding process is, usually, divided in three steps: domain detection, intent detection and slot filling. For a personal assistant, the domain could be a theme like ”weather” or ”sports”, the intent could be an action, for example ”telling the result of a match” and the slot filling corresponds to the labelling of the keywords, such as the kind of sport or teams that played.

Typically the first two steps are treated as semantic utterance classification problems and the slot filling as a sequence classification problem. However, the focus of this work is only on the intent detection and slot filling, once there is only one domain for the robot.

In this chapter, firstly some previously used methods for intent detection and slot filling are described, starting in the most antique approaches and going through the methods used until the state of the art, neural networks. Comparisons between the results obtained with the different methods are also presented. A different approach, LU4R, also used for service robots is exposed.

The neural networks architectures most used in NLU, Recurrent Neural Networks and Long Short Term Memory, and the methods used to train them are presented.

Also, several word embedding techniques, methods that reduce the word vectors dimensionality while gaining meaning, are exposed.

Finally, an overview of these algorithms is made and a selection of the methods to use in this thesis implementation is made.

2.1 Intent detection

The task of intent detection aims at classifying a speech utterance into one of various semantic classes. In this case it consists in identifying the required action. To do this, the classifier needs to have significant freedom in utterance variations, since commands with the same intent can vary significantly [6], i.e. ”deliver the coke to peter at the living room” and ”robot can you bring me a bottle of water from the fridge” have the same intent but are very different.

The first reported works in this area were developed for call routing, trying to direct the call to the right operator [7], [8]. Approaches using discriminative classifiers such as, Boosting [9], Support Vector

Machines [10] and Minimum Classification Error [11], are also presented.

2.1.1 First approaches

In 1997, Gorin et al. [7] designed an automated system to perform call routing for AT&T call-centres. This system would ask ”How may I help you?” to the person calling encouraging people to speak naturally. Then, it would find salient phrase fragments, n-grams, which consist in expressions of n words, such as ”the area code for”, ”long distance call”. Conditional probabilities of each call type would be calculated for the phrase fragments found in the utterance, being the type chosen the one with the biggest probability. However, if this probability was smaller than a threshold the sentence would be classified as other. Finally, the system would ask the person calling to confirm. In case that the classification was confirmed, the call would be directed to the appropriate operator. Otherwise, the system would use the clarification that the caller gives to justify the classification incorrectness, to try to figure out what was the intent.

Some years later, Chu-Carrol et al. also developed a system for call routing, using a vector-state information retrieval model [8]. To select the proper destination of the call, firstly, salient n-gram terms would be extracted from the utterance provided by the caller, forming a bag of terms. A vector would then be computed doing the weighted sum of the term values and the cosine value between the vector obtained and each vector of the routing matrix, a matrix that contains the vectors of all possible destinations, was computed. Then, a confidence score for each destination was obtained by transforming the cosine using the sigmoid function of the destination. After that, the five best confidence scores would be compared with a threshold. If only one had higher confidence than the threshold, the call would be routed to that destination. Otherwise, if none had, the call would be routed to a human operator. It could also happen that there was more than one destination with a higher confidence level. In this case, the system would use a disambiguation algorithm, in which a query to clarify the intent of the caller would be performed. This query would be composed of a subset of salient n-grams selected by closeness to the difference vectors, relevancy and disambiguation power.

Both these methods were later outperformed by the use of discriminative classifiers.

2.1.2 Discriminative classifiers

Discriminative classifiers model the conditional probability of the classes, knowing the inputs, without making any assumptions, and use that probability to minimize a loss function while training. Other discriminative algorithms learn a direct map from the inputs to the respective classes [12], [13], [14].

With the advances in these classification techniques, researchers started using them in intent classification problems.

Boosting

Boosting is an iterative algorithm, that finds hypotheses given by several weak learners, which consist in learning algorithms whose accuracy is not very good but predicting better than by chance, and combine

them into a final result, doing the weighted sum. It is meant to be combined with any classifier, improving its performance.

Schapire et al. used the AdaBoost algorithm, a boosting algorithm previously developed by Freund et al. in 1997 [15] and improved by Schapire et al. [16], to perform multiclass text and speech categorization tasks [9].

In this case, decision trees were the weak classifier selected. Decision trees are sequential models that in each node compare a numeric or categorical attribute. After going through all the node comparisons, the leaf node decides which class the input belongs to [17].

This algorithm was tested on the same dataset used by Gorin et al. [7] in the AT&T call routing system and the accuracy performance achieved was similar. However, the false alarm rates improved significantly.

Support Vector Machines

Support Vector Machines (SVMs) are binary linear classifiers that are based on the Structural Risk Minimization principle, which says that an hypothesis that guarantee the minimum true risk should be found [10]. The idea behind SVMs is to find an hyperplane separating two classes, by discovering the best margin. This margin corresponds to the sum of the distance of the closest training points of the two classes being the chosen margin the biggest one.

The problem with the use of SVM for multiclass classification is that because of being a binary classifier, multiple SVMs have to be combined. In 2003 Haffner et al. suggested a method to do it [10]. The optimization techniques used were extracting the most information possible from the user within the smallest time possible, treating the classes independently and using exhaustive search when possible.

The results achieved were similar to the ones obtained by Schapire et al. [9] using boosting for the multiclass error. However, the threshold error improved.

Minimum Classification Error

Using the Minimum Classification Error (MCE) criterion, Kuo and Lee [11] improved Chu-Charrol et al. [8] system for call routing. They achieved a reduction of 10-30% on the relative error rate, by retraining the routing matrix using MCE. This improvement happened because, instead of only training the data tokens belonging to the true class, the training was also done for the competing classes.

Zitouni et al. [18] also experimented on improving the classification performance, with Boosting, MCE discriminative training and Automatic Relevance Feedback (ACR). ACR consists in improving user queries by asking them to specify which n-grams are more important. The conclusion obtained in these experiments was that the best improvement is obtained by the MCE training. They also concluded that combining MCE training with Boosting produced even better results.

Even though these algorithms achieved good performances for call routing purposes, the limited scope of n-grams and their sparseness compel the use of large amounts of training data to obtain good

generalization. Furthermore, the longer the n-grams are, the bigger the sparseness of the data is, making this a problem hard to solve [19].

2.2 Slot Filling

Slot filling corresponds to the process of automatically extracting semantic concepts, filling a set of arguments or slots. Most of the first approaches to solve this were based on syntactic and semantic grammars.

2.2.1 Using grammars

Semantically enhanced syntactic grammars

In 1992, Seneff developed a MIT's system for spoken language applications, called TINA [20]. This system was based on a context-free grammar augmented to enforce syntactic and semantic constraints, via unification, process of unifying syntactic concepts that usually have the same semantic content. By replacing low-level syntactic categories with semantic categories, domain dependent semantics were obtained. The mixing of semantic nodes and syntactic nodes, instead of using only syntactic nodes and separated semantic rules was advantageous because all the information needed could be extracted from the parse tree.

Dowding et al. also worked on a grammar-based system for spoken language understanding, Gemini [21]. Like TINA, Gemini also uses a unification grammar. However, domain-dependent semantics are separated from domain-dependent syntax, having a set of semantic rules associated to each node of the parse tree.

The biggest problem with these approaches is the fact that it often requires an exact matching of the input utterances to the grammar rules. This is not very common because people usually are not so concerned in being grammatically correct when they are speaking as they are when writing.

Semantic grammars

In 1991, Ward used a semantic grammar for natural language understanding, the Phoenix system [22]. In this system, the slots were filled by matching the input sentences, previously parsed, with the slot nets in the Recursive Transition Networks (RTN). RTN consist in a graph of nodes and edges in which the arcs can be labeled as terminal words or other networks. The final result corresponded to the parse that contained the most slots discovered.

This system achieved an impressive performance using the ATIS (Airline Travel Information System) dataset, one of the most popular datasets for slot filling [6].

Good results can be obtained using grammars. However, it is very difficult to create a grammar that generalizes well. The grammar used in the Phoenix system [22] had 3.2 thousand concepts and 13 thousand grammar rules.

2.2.2 Discriminative and Generative Classifiers

Both discriminative classifiers, such as Conditional Random Field (CRF) and SVMs and generative classifiers, like Finite State Tranducers (FST), are used in the slot filling task.

Generative classifiers learn a model of the joint probability of the inputs and the labels and make classifications using the Bayes rule, whereas discriminative classifiers use conditional probabilities directly [14].

Raymond et al. applied both generative and discriminative methods for the same tasks, the ATIS and MEDIA datasets and compared the results[23].

They used a SVM system developed by Kudo et al. that achieved an outstanding performance in chunking tasks [24]. As SVMs are binary classifiers they used pairwise classification. This consists in building classifiers for all pairs of classes and using as final result their weighted sum.

CRFs were also tested by Raymond et al.. CRF is a discriminative undirected probabilistic graphical model. Consists in a probabilistic framework for labelling and segmentating sequential data, using conditional probabilities. Given an observed sequence, the CRF undirected graph defines a single log-linear distribution over label sequences[25].

The generative model chosen for the comparison was the Stochastic Finite State Transducers (SFST). In this method, for each concept there is an Finite State Machine (FSM). The FSMs consist in transducers that receive an input of words and output the concept tag, being the result the best sentence segmentation computed over all possible hypothesis [23].

The results showed that the FST model is more robust to data sparseness and inconsistent training corpus. However, with consistent training data, CRFs had better results.

2.3 Language Understanding chain For Robots

LU4R is an adaptive spoken Language Understanding chain For Robots which has as one of its purposes granting robots the ability to understand the instruction given in RoboCup@home GPSR task.

To achieve this, firstly a morpho-syntactic analysis is performed over all the possible transcriptions, from speech to text, using Stanford CoreNLP Library [26]. To choose the best transcription available a SVM, developed by Basili et al. [27], that exploits a combination of linguistic kernels is applied [28]. The intent detection and slot filling, or argument labelling, are performed using a statistical semantic parser, based on the work of Croce et al. [29] and using a Markovian formulation of a structured SVM [30]. This algorithm consists in a combination of a local discriminative model, SVM, and a generative approach, Hidden Markov Model (HMM). The SVM estimates the individual observation probabilities of a sequence while the HMM chooses the most acceptable sequence. Also, grounded information, extracted from a semantic map, is provided to this step so the robot understands the instructions given not only based on the linguistic information but also on perceptual knowledge [31].

2.4 Neural Networks

After being one of the artificial intelligence biggest promises, in the 1990's, the interest in neural networks diminish significantly due to poor results.

However, with the technological advances of this century, such as the availability of large datasets and the increase in computational power available, deep networks can now be successfully trained. This led to big improvements in a variety of applications, in which NLU is included.

In recent years, language models using Recurrent Neural Networks (RNN) demonstrated outstanding performance in a variety of natural language processing tasks [32].

Before introducing RNNs, a short overview of how neural networks work and are trained will be exposed.

2.4.1 Loss function

Loss or cost function corresponds to a function that represents the accuracy of the model, being higher when the accuracy is smaller. This way, the objective of the train is finding the parameters, weights and bias, that minimize the function.

2.4.2 Activation functions

An activation function consists in the function that emulates the transformation that happens in the neurons. The input of this function, the z in equations 2.2, 2.3, 2.4 and 2.5, is the result of the expression 2.1.

$$z = wx + b \tag{2.1}$$

where:

w is a weight vector,

b is a bias,

x is the neuron inputs vector.

The weights of the neural networks are the values that do the association between the input of the neural network and the output. The biases correspond to values, one per layer that are summed to the neuron, basically to create an offset. Both the weights and the biases will be updated while training the network. The output of the neuron will be the result of the function. A scheme representing a neuron, its input, output and activation function can be seen in figure 2.1.

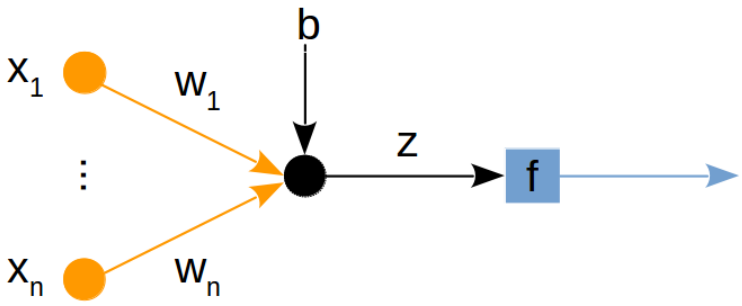


Figure 2.1: Scheme of a neuron, its input and output and activation function

Usually, the neurons of the output layer also have an activation function, so the desired result is obtained.

These functions are fundamental because by adding nonlinearity to the neural networks, allow them to learn nonlinear decision boundaries which are what make deep learning a great technique to solve such varied and complex problems.

There are several types of activation functions, being the most used ones described below.

Sigmoid

The sigmoid function is usually used for binary classification problems. The output of a sigmoid function varies between 0 and 1.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

Hyperbolic tangent

Hyperbolic tangent or tanh is similar to the sigmoid function, being also widely used. The output instead of varying between 0 and 1, varies betweeen -1 and 1. One important difference is the fact that the derivatives are steeper, making it converge to the minimum of the loss function quicker.

$$f(z) = \frac{2}{1 + e^{-2z}} - 1 \tag{2.3}$$

Rectified Linear Unit

The ReLU or Rectified Linear Unit is a function that outputs 0 if the input is negative and outputs the input if it is positive. It is very used as the activation function of hidden layers, once it helps reducing the vanish gradient likelihood. Also, according to Alex Krizhevsky et al. [33] using ReLU as activation function, the training of the neural network can be 6 times quicker.

$$f(z) = max(0,\ z) \tag{2.4}$$

Softmax

The softmax function is, commonly, used as the activation function of the output layer for multi-class classification problems. This happens because the results of this function can be considered a probability, once, as can be seen in equation 2.5, the output of the current neuron is divided by the sum of all the outputs. The result is, then, between 0 and 1, being the sum of the softmax results of the layer equal to 1.

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=0}^{k} e^{z_j}} \tag{2.5}$$

2.4.3 Training neural networks

Training a neural network consists in minimizing the loss function defined. This is, usually, achieved with the aid of a gradient descent optimizer.

A gradient descent optimizer does this minimization by updating the parameters θ, weights and bias, in the opposite direction of the gradient of the loss function[34]. This process is commonly described metaphorically as a particle trying to reach the valley of the surface created by the objective function.

Gradient descent algorithms have an hyper-parameter, the learning rate, η, that controls the size of the parameter updates. This hyper-parameter can be constant through the whole training or can be reduced when it starts to become more difficult to further minimize the loss function.

Several variants, exposed below, have been developed to increase the optimization performance. These gradients are, commonly, computed using the backpropagation algorithm.

Backpropagation

The backpropagation algorithm procedure consists in the application of the chain rule for derivatives, starting by computing the derivative of the output of the neural network and then the derivatives of the hidden layers until reaching the input layer. When the derivatives for all layers are calculated, the biases and weights error rates can be computed and the parameters can be updated.

The error in the output layer, δ^L, is computed, using equation 2.6.

$$\delta^L = \nabla_\theta \, J \odot f'(z^L) \tag{2.6}$$

where:

L is the output layer,

l represents the layer,

J is the cost or loss function,

z is the weighted input, computed as in equation 2.1,

f is the activation function of the output layer,

∇_θ represents the gradient in relation to the neural network parameters,

$\odot$ represents the Hadamard product.

Then, for the other layers the error, δ^l, is calculated in terms of the error in the following layer, as expressed in equation 2.7.

$$\delta^l = ((w^{l+1})^T \ \delta^{l+1}) \odot f'(z^L) \tag{2.7}$$

Then, the rate of change of the biases and of the weights are given, respectively, by equations 2.8 and 2.9

$$\frac{\partial J}{\partial b^l_j} = \delta^l_j \tag{2.8}$$

$$\frac{\partial J}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{2.9}$$

where:

j and k represent the neurons,

a is the output of the neuron k of a previous layer and the input of the neuron j of the present layer,

b is a bias,

w is a weight.

After the error rates are computed, the weights and biases are updated by subtracting the error rates multiplied by the learning rate.

Batch descent

In Batch gradient descent or Vanilla gradient descent, the computation of the gradients of the loss function is made for all the examples in every update. The function used is presented in equation 2.10.

$$\theta = \theta - \eta \cdot \nabla_\theta \ J(\theta) \tag{2.10}$$

The parameters are then updated in the direction of the gradients.

Stochastic gradient descent

Stochastic gradient descent (SGD), computes the gradient of the loss function and performs parameter update for each training example, as can be seen in equation 2.11.

$$\theta = \theta - \eta \cdot \nabla_\theta \ J(\theta, \ x^{(i)}, \ y^{(i)}) \tag{2.11}$$

where:

$x^{(i)}$ is the training input,

$y^{(i)}$ is the training output.

This way the process is much faster than using batch descent [34].

Momentum gradient descent

One problem of SGD is the fact that when there are surface curves much steeper in one dimension than in another, it starts to oscillate and making very small progress.

In momentum gradient descent the parameters are updated in every example, like in SGD. However, a fraction of the update vector of the last time step is used to accelerate the process and diminish oscillations, capturing the momentum of the particle. The functions used to update parameters are presented in equations 2.12 and 2.13

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta \, J(\theta, \, x^{(i)}, \, y^{(i)}) \tag{2.12}$$

$$\theta = \theta - v_t \tag{2.13}$$

where:

v_t and v_{t-1} are, respectively, the update vector in the current step and in the previous step

γ is the momentum fraction

Adagrad

Adagrad is a modified stochastic gradient descent optimizer that adapts the learning rate to the parameters, increasing it for more sparse parameters and decreasing it for frequent parameters [35]. This method is represented in equation 2.14.

$$\theta_{t+1, \, i} = \theta_{t, \, i} - \frac{\eta}{\sqrt{G_{t, \, ii} + \epsilon}} \cdot \nabla_{\theta_t} \, J(\theta_{t, \, i}) \tag{2.14}$$

where:

$\sqrt{G_{t, \, ii} + \epsilon}$ corresponds to the part of the function responsible for the learning rate variation,

$G_{t, \, ii}$ is a diagonal matrix where each diagonal element $(i, \; i)$ is the sum of the squares of the gradients θ_i up to the time step t,

ϵ is a smoothing term to avoid division by zero.

The main benefits of this method are the better performance when dealing with sparse data and the fact that the learning rate does not need to be tuned manually. Its main disadvantage is that the accumulation of squared gradients in the learning rate denominator causes the learning rate to shrink, making it impossible to gain new knowledge.

Adam

Adaptive Moment Estimation or Adam is a modified stochastic gradient descent method, that, like Adagrad, computes adaptive learning rates [36]. The difference is that in Adam, the adaptive learning rates are obtained by computing estimates of first and second moments, mean and uncentered variance, of the gradients. The moments are computed using the equations 2.15 and 2.16.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_{\theta_t} J(\theta_t) \tag{2.15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_{\theta_t} J(\theta_t))^2 \tag{2.16}$$

being:

m_t and v_t the estimates of the first and second moments respectively,

β_1 and β_2 correspond to decay rates.

As m_t and v_t are initialized as zero vectors, they are biased towards zero, especially when the decay rates are small. This is annulled by computing bias-corrected estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.17}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.18}$$

Then, the parameters are updated as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.19}$$

The Adam optimizer solves the monotonically decreasing learning rate problem observed in Adagrad. It also stores momentum changes for each parameter, which increases the performance in convex problems.

2.4.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are very useful when dealing with sequential problems since, instead of assuming that the inputs are independent of each other, like traditional neural networks, RNNs use the sequential information of the input.

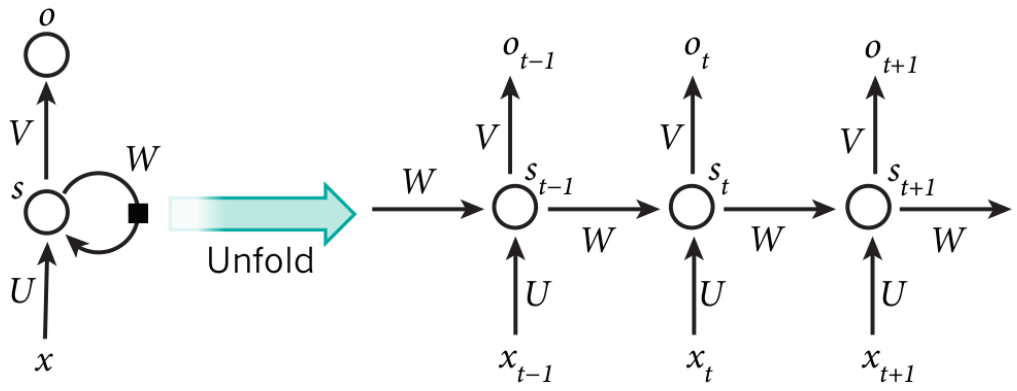


Figure 2.2: A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Adapted from [37].

In the figure 2.2, a diagram of a RNN can be seen. When unfolded, the network for all time steps is exposed. The time is represented by the indices, being $t-1$ the input before the present i.e. if the input x is a sentence, x_{t-1} would represent the word before x_t and x_{t+1} would be the word after. The same happens for the hidden states s and for the output o.

The hidden state of the current step is calculated by:

$$s_t = f(Ux_t + Ws_{t-1}) \tag{2.20}$$

being:

f an activation function, usually the $tanh$, or $ReLU$,

U the weight that influences the input passed to the hidden state,

W the weight given to the previous step.

The hidden state is sometimes called as the ”memory”, since is there that the value of the last step is taken into account.

The output of the network at the present step is obtained by:

$$o_t = f_2(Vs_t) \tag{2.21}$$

where:

f_2 is an activation function, usually the $softmax$,

V the weight used to obtain the output.

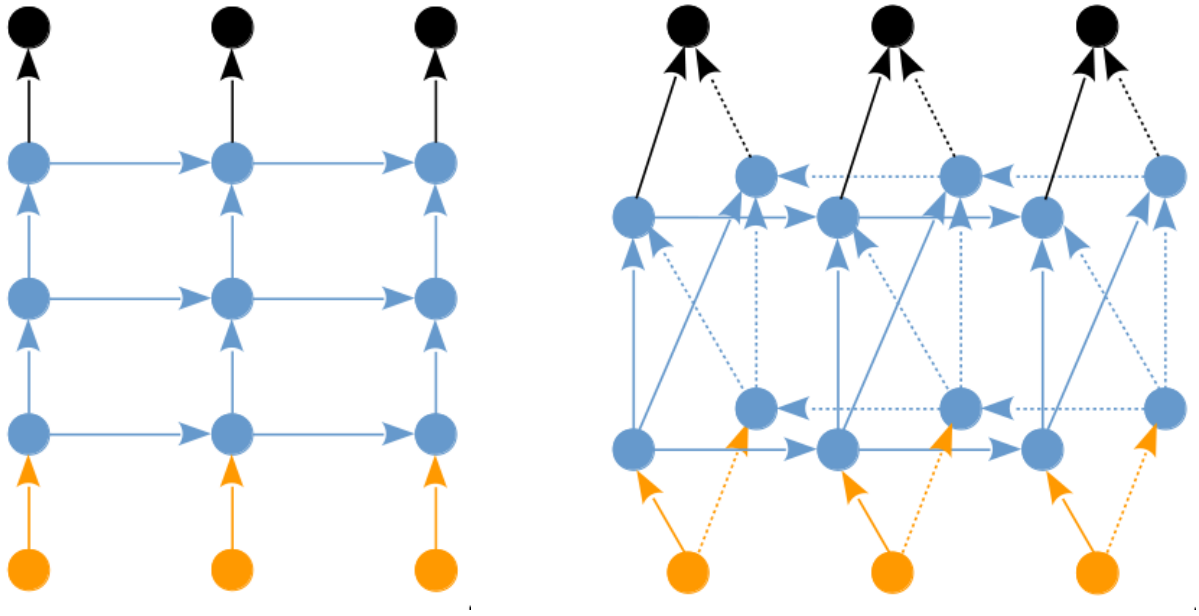
The parameters U , V and W are shared through all steps, reducing the number of parameters needed to be learned.

In this RNN every time step has an output. This network is used when doing sequence tagging, such as slot filling. However, in classifications like intent determination, only the output of the final step of the network is considered.

Variances of Recurrent Neural Networks

Due to the fact that in some problems the output not only depends on the previous time steps but also on the future ones, a type of RNNs that take that into account was developed, Bidirectional Recurrent Neural Networks (BRNN). A BRNN consists in two RNNs stacked, with one being influenced by the information of the previous steps and the other by the future ones. The output corresponds to the sum of the outputs of the two RNNs.

Both RNNs and BRNNs can have multiple layers of hidden states, being called Deep RNN or Deep BRNN. In the schemes of figure 2.3, unrolled networks of DRNN and DBRNN can be seen.



(a) Deep Recurrent Neural Network

(b) Deep Bidirectional Recurrent Neural Network

Figure 2.3: Schemes of a deep RNN and a deep bidirectional RNN

The only difference between deep networks and the one presented before in figure 2.2, is that because they have more than one layer, the input of the second layer is the output of the first layer and so on. For the DRNN, the calculation of the step t in the second layer is as follows:

$$s^2{}_t = f(Us^1{}_t + Ws^2{}_{t-1}) \tag{2.22}$$

And, for the DBRNN is:

$$s^{2-}{}_t = f(U^- s^1{}_t + U^+ s^1{}_t + Ws^2{}_{t-1}) \tag{2.23}$$

$$s^{2+}{}_t = f(U^- s^1{}_t + U^+ s^1{}_t + W s^2{}_{t+1}) \tag{2.24}$$

representing the $-$ the RNN that is influenced by the past steps and the $+$ the RNN influenced by the future.

The use of more layers gives the network a higher learning capability. However, the dataset to train it has to be bigger. The number of layers is, usually, considered an hyper-parameter that depends on the complexity of the problem and so to find the more appropriate number, various should be tested.

To confirm that RNNs outperform previously used algorithms, Kaisheng Yao et al. compared RNNs with FST, SVM and CRF in the ATIS slot filling task [32]. The results showed that RNNs outperform all the previous methods used for this task.

Grégoire Mesnil et al. tested DRNNs and DBRNNs on a slot filling task [38]. To test the performance and compare to CRFs, they used the ATIS dataset, a movies dataset and an entertainment dataset. For the movies dataset and ATIS, the performance of RNNs was better than CRFs. However, for the entertainment dataset the performance was slightly worse, being their explanation the fact that in the entertainment domain the slots correspond to bigger expressions such as movie names [38].

The problem of long-term dependencies

The idea behind the development of RNNs is the ability to use memorized information in the next steps. In fact when the distance between the present and the relevant information is small RNNs learn well. Nonetheless, when learning long-term dependencies, which happens when information from several steps before is relevant, which is the case of this work, the performance obtained is not so good [39].

Multiple algorithms have been proposed by researchers to handle this problem and improve the efficiency of RNNs. Some of these approaches had good results, but were too complex and computationally costly [40].

In 1997, Hochreiter et al. proposed a novel RNN architecture, called Long Short-Term Memory (LSTM) [40].

2.4.5 Long Short-Term Memory

Since LSTMs emerged as an effective model for sequential data problems, being capable of capturing long-term temporal dependencies, they have been used to advance the state of the art of several difficult problems, including NLU [41]. LSTMs have been widely used for NLU tasks with very good results [19], [42], [43].

A LSTM consists in memory cells, one per time step, that maintain its state over time, and gates which regulate the information flow into and out of the cells [41]. All of the gates have the sigmoid as activation function, so their output is a value between 0 and 1, that represents the amount of information that passes. As for the cell state and the cell output, the activation functions used are the hyperbolic tangent to overcome vanish gradient descent.

A scheme of a LSTM cell can be seen in figure 2.4, where the line junctions represent concatenations of vectors in a matrix, the $\odot$ represent the Hadamard product and the blue rectangles the activation functions.

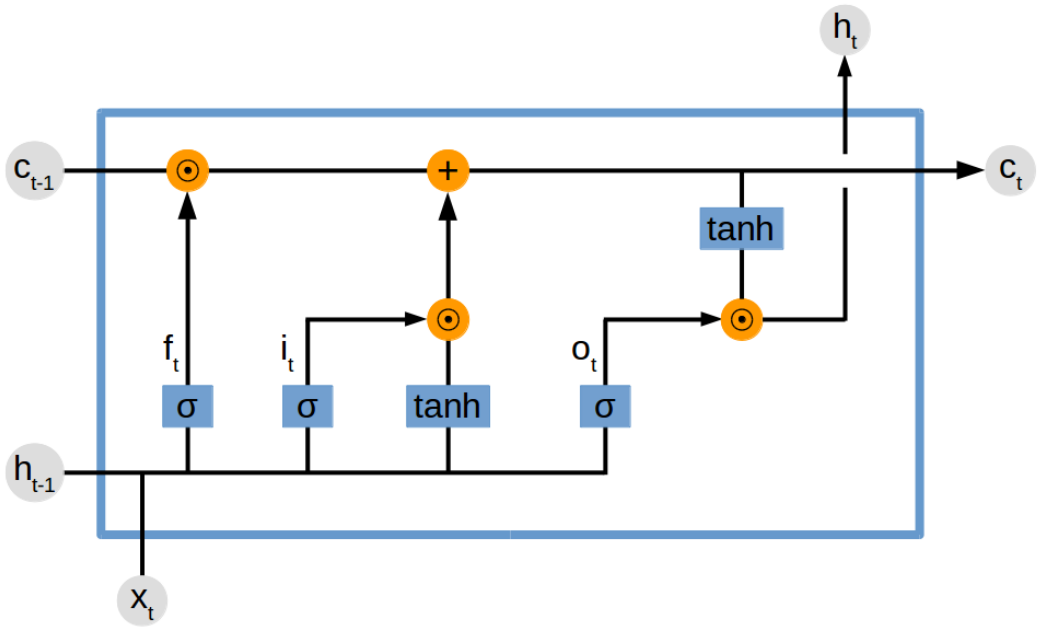


Figure 2.4: Scheme of LSTM memory cell

The forget gate, f_t, decides the amount of information, from the previous cell, that should be taken into account. This is done by performing the sigmoid over the concatenation of the output of the previous cell, h_{t-1}, and the input of the current cell, x_t multiplied by the weight vector and summing the bias.

The input gate i_t selects the amount of the input that should be used in the current cell. This gate works in the same way as the forget gate.

The cell state, c_t, is then computed by doing the Hadamard product of the result of the forget gate, f_t and the state of the previous cell, c_{t-1}. This product is then summed with the hadamard product between the input gate result, i_t and the $tanh$ of the concatenation of the output of the previous cell, h_{t-1} and the input of the current one x_t.

Finally, the output gate decides what information should be outputed by the cell doing the sigmoid of the concatenation of the previous cell output, h_{t-1} and the current cell input, x_t. The output of the cell, h_t is then the hardamard product between the result of the output gate, o_t and the $tanh$ of the cell state, c_t.

These steps, are represented in the following equations:

$$f_t = \sigma(W_f \cdot [h_{t-1},\ x_t] + b_t) \tag{2.25}$$

$$i_t = \sigma(W_i \cdot [h_{t-1},\ x_t] + b_i) \tag{2.26}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c \cdot [h_{t-1},\ x_t] + b_c) \tag{2.27}$$

$$o_t = \sigma(W_o \cdot [h_{t-1},\ x_t] + b_o) \tag{2.28}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.29}$$

where:

x_t is the input at time step t

f_t is the forget gate at time step t,

i_t is the input gate at time step t,

c_t is the cell state at time step t,

o_t is the output gate at time step t,

h_t is the output of the cell at time step t,

W is the weight matrix to be learned,

b corresponds to the bias matrix.

Like RNNs, LSTMs can also have multiple layers and be bidirectional. The cell maintains the same architecture, being the only change the way the inputs of the cells are connected with each others, as explained before for RNNs.

Ravuri and Stolcke applied RNNs and LSTMs to the intent determination problem [19]. For the ATIS dataset both RNNs and LSTMs had better performances than a Maximum Entropy language model also tested. However, the RNN performed slightly worse than a boosting model. For the Conversational Browser (CR) dataset RNNs and LSTMs outperformed other models, but in contrast to the ATIS dataset, LSTMs performed worse than RNNs. This contrast is due to the fact that in ATIS, sentences are bigger [19], being the peak length 11 words. In the CR dataset most sentences have less than 5 words.

As for the slot filling task, Kaisheng Yao et al. compared the performance of CRFs, RNNs, LSTMs and Deep LSTMs, using the ATIS dataset [42]. Deep LSTMs outperformed the other algorithms followed by LSTMs and RNNs.

Hakkani-Tür et al. also used the ATIS dataset slot filling task to compare RNNs, LSTMs and bidirectional LSTMs [43]. The bidirectional LSTM had the best performance, followed by the LSTM.

2.5 Word embeddings

One way to represent the words in the input sentence is by using one-hot vectors, that contain one 1 and multiple 0. Each word in the vocabulary has an id that corresponds to the index of the 1 in the vector. A

small vocabulary of the most used words in English can have from 30000 to 50000 words, which implies that the vectors would have a very big number of dimensions.

Another approach to represent the words is by using word embeddings, feature vectors. This correspond to vectors that express the features of the words, trying to capture their meaning. Usually the number of dimensions of these vectors ranges from 50 to 600 dimensions. In the vector space of a vocabulary, represented with word embeddings, the similar words tend to be close together. Because of this, when adjusting the model to increase the likelihood of a word in a context, it also increases the likelihood of similar words in similar contexts [32]. That is, if a model was trained with the sentence "guide me to the bedroom" and a person says "guide me to the bathroom" to the robot, despite not being trained with the second sentence the model will detect "bathroom" as a destination anyway, as a result of being similar to "bedroom".

One of the first neural language models to produce word embeddings was developed by Yoshua Bengio at the beginning of the century [44].

2.5.1 Bengio's approach

The language model proposed by Bengio consists in a feed-forward neural network, that predicts the next word in a sequence. In this neural network, there is a layer with the feature vectors, the embedding layer. This layer is followed by a hidden-layer with the $hyperbolic\ tangent$ as activation function and finally a $softmax$ output layer that guarantees positive probabilities summing to 1. The output corresponds to the conditional probabilities of a word being the next, knowing the previous ones, words inside the input window [44]. The size of input window, which goes through the corpora, is an hyper-parameter that is previously selected.

The training is achieved by finding the θ that maximizes the log-likelihood, equation 2.30. θ corresponds to the feature vectors and all the parameters, weights and bias, of the network.

$$L = \frac{1}{T} \sum_{t} \log f(w_t, w_{t-1}, ..., w_{t-n+1}; \theta) + R(\theta) \tag{2.30}$$

where:

T is the number of words in the corpora,

n is the number of words, input window,

w represents the words,

f is the output of the network,

θ represents the parameters that will be learned by the neural network, such as, weights and biases,

R is a regularization term.

All the weights, biases and feature vectors are updated doing backpropagation. When the train is complete, the log-likelihood function was maximized, the word embeddings, vectors in the embedding layer, are ready to be used in other models as inputs.

2.5.2 Word2vec

Tomas Mikolov suggested a new model for learning word embeddings, named Word2vec. For this model, two different architectures were proposed, Continuous Bag-of-Words (CBOW) and Skip-gram [45], [46]. The goal was to develop a model with the ability to learn word embeddings on huge datasets, which was not possible with previous models due to the high computational cost.

Continuous Bag-of-Words model

In this architecture, instead of just using previous words for the predictions, the following words are also taken into account. As can be seen in the scheme of figure 2.5, a feed-forward neural networks is used, but in contrast to Bengio's approach [44], there is no hidden layer.

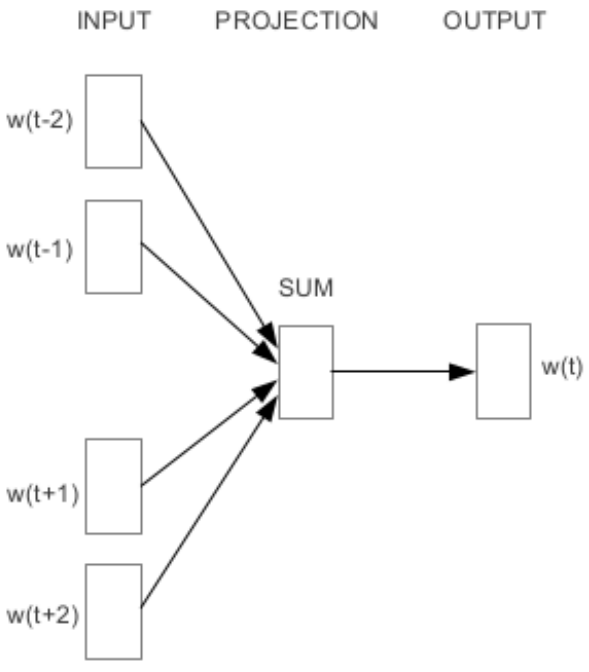


Figure 2.5: Scheme of the Continuous Bag-of-Words model. Adapted from [45]

As the name suggests, the order of the words inside the window input do not influence the prediction being their vectors averaged in the projection layer. In this model, these input vectors are not the feature vectors, they correspond to one-hot vectors. The projection layer has as activation function the $softmax$, being the output obtained the conditional probability of the output word, knowing the previous and following words. When training, the objective function that is maximized is given by equation 2.31

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n}) \tag{2.31}$$

where:

T is the number of words in the corpus,

θ represents the parameters of the neural network, weights and biases,

w represents the words,

t represents the time steps.

Skip-gram model

The Skip-gram model, instead of predicting the current word based on the context, tries to predict the words before and after the current word.

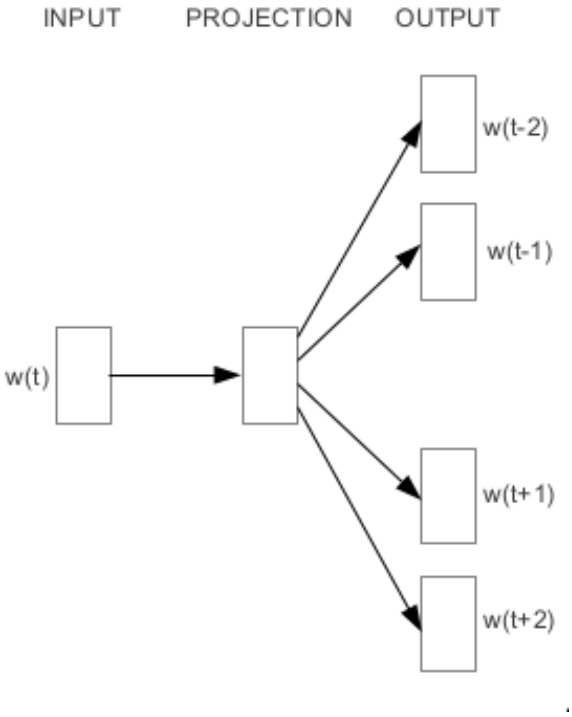


Figure 2.6: Scheme of the Skip-gram model. Adapted from [45]

As can be seen in the scheme presented in figure 2.6, the network used for this model consists, like the CBOW network, in an input layer and a projection layer with $softmax$ as the activation function. The output layer has a variable number of nodes, depending on the window size chosen. Each node corresponds to a conditional probability of a word being in that position, knowing the center word.

The objective function maximized in Skip-gram is:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j}|w_t) \tag{2.32}$$

where:

T is the number of words in the corpus,

θ represents the parameters of the neural network, weights and biases,

w represents the words,

t represents the time steps.

j is the distance of the word to the center, being positive to the right and negative to the left,

c is the number of context words used to the right or to the left.

The softmax function used in the Skip-gram formulation is:

$$p(w_o|w_I) = \frac{exp({v'_{w_o}}^T v_{w_I})}{\sum_{w=1}^{W} exp({v'_w}^T v_{w_I})} \tag{2.33}$$

being:

w_o the outside word or context word,

w_I the inside word or center word,

v'_w the input vector representation of a word w,

v_w the output vector representation of a word w,

W the number of words in the vocabulary,

T is the matrix transposition symbol.

The cost of computing this function is proportional to the size of the vocabulary which is usually large, $10^5 - 10^7$ terms. This makes this formulation impratical [46]. For this reason, Mikolov et al. developed extensions of the original model, such as the use of hierarchical softmax, negative sampling and subsampling of frequent words.

Hierarchical Softmax

One of the extensions proposed by Mikolov et al. was the use of hierarchical softmax instead of full softmax, since its computational cost is lower.

The hierarchical softmax uses a binary tree representation of the output layer with the words as the leaves. In this case, a Huffman tree based on word frequencies is used.

In this algorithm, each node of the tree has the relative probabilities of its child nodes. Using the full softmax function, to obtain the probability of a word it is required to normalize it over the probabilities of all words in the vocabulary. In contrast, using the binary tree, the probability is calculated by following the path to the leaf node of the respective word. As a binary tree has a depth of $\log_2(W)$, instead of evaluating all the words in the vocabulary, only $\log_2(W)$ words have to be evaluated.

Negative Sampling

Without extensions, in the Skip-gram model, using softmax, the weights for all the words in the vocabulary are adjusted with every training example. Using negative sampling, a previously stipulated number of words, besides the correct word, are chosen as negative samples. This way only the weights of these words and of the correct word are modified and, consequently, the computational cost is much lower.

With negative sampling, the $\log p(w_{t+j}|w_t)$ in the objective function of the Skip-gram, equation 2.30, is replaced by equation 2.34, being w_o one of the context words represented by w_{t+j}.

$$\log \sigma({v'_{w_o}}^T v_{w_I}) + \sum_{i=1}^{k} E_{w i \sim P_n(w)}[\log \sigma(-{v'_{w_i}}^T v_{w_I})] \tag{2.34}$$

where:

E is the expected value,

P_n is the noise distribution,

k is the number of wrong words sampled.

The noise distribution corresponds to a free parameter. In Word2vec, Mikolov et al. defined it as:

$$P_n(w) = \frac{U(w)}{Z}^{3/4} \tag{2.35}$$

being:

$U(w)$ the unigram distribution of a word, frequency of the word,

Z the sum of the frequencies of all words .

Using this function as noise distribution, the more frequent the word is in the corpus, the more times it is chosen as negative sample.

For the number of words to be used as negative samples, k, the researcher advise the range of 5 to 20 for small training datasets. Using big datasets this number could be reduced to 2 to 5 words [46].

Subsampling of frequent words

In a large corpus, frequent words, such as ”a”, ”the” and ”of”, can appear millions of times. These words, usually, add less information to the word embeddings than rare words, i.e. the co-occurrence of ”take” and ”the” is not very beneficial, once almost every word co-occurs with ”the”, in contrast the co-occurrence of ”take” and ”bottle” is beneficial.

Mikolov et al. used a simple approach to subsample the frequent words. It consisted in calculating the following probability:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.36}$$

where:

t is a threshold, typically around 10^{-5},

$f(w_i)$ is the word frequency.

Then, the frequency of these words is diminished by randomly removing the words with frequencies bigger than the threshold t, the ones that have a positive probability, from the context windows.

A test set of Semantic-Syntactic Word Relationships questions, word analogies, was used by Mikolov et al. to compare the two Word2vec models with some neural network and RNN models [45]. The conclusion obtained was that the Skip-gram model produces the best word embeddings, being only defeated in the syntactic accuracy test by a neural network language model. The CBOW model performed poorly in the semantic accuracy but did better than most of the language models in the syntactic one. It was also observed that training the network for 3 epochs instead of just 1, using the same data, did not produce better results.

Using the same dataset, the feature vectors resulting from the Skip-gram model with the extensions described before were compared [46]. The vectors created using negative sampling, with $k = 5$ or $k = 15$ performed better than the ones created using hierarchical softmax. Furthermore, when $k = 15$, though the computational cost is slightly higher, the word embeddings had better results. It was also concluded that the subsampling of frequent words improved the vectors.

2.5.3 GloVe

GloVe is an abbreviation for Global Vectors. This algorithm was developed by Pennington et al. in 2014 [47]. In opposition to the previous explained models, in GloVe the feature vectors are achieved by directly using the word co-occurrence statistics.

Unlike Word2vec, the GloVe algorithm instead of going through the whole corpus in an online fashion, when trying to predict the center or the context words, uses a previously generated co-occurrence matrix. To generate this matrix, firstly a window size is selected, usually of 4 to 10 words. The corpus is then searched and the matrix filled, corresponding the elements of the matrix to the number of times the word represented by the column number appears in the window of the center word, that corresponds to the line number. Moreover, as the meaning of a word is closer related to the nearby words, instead of counting the co-occurrences of the words inside the context window equally, the distance factor is taken into account, being the value added to the matrix element $\frac{1}{d}$, where d represents the distance between the context and the center words.

Two weight matrices are created, one for the feature vectors when the words are in the center of the window and other for when in the context.

A least squares loss function, equation 2.37, is used to train the word embeddings.

$$J = \sum_{i,j=1}^{V} f(X_{ij})({w_i}^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{2.37}$$

where:

V is the vocabulary size,

i represents the center word and j the context word,

X_{ij} is the element of the matrix X that corresponds to the number of times the word j in in the context window of word i,

w_i and $\tilde{w}_j$ are the center and context word vectors, respectively,

b_i and $\tilde{b}_j$ correspond to bias values for the main and context words.

This loss function includes a weighting function, equation 2.38, preventing that frequent co-occurrences are overweighted, similarly to the subsampling of frequent words in the Word2vec model [46].

$$f(X_{ij}) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{2.38}$$

The value usually used for x_{max}, maximum of the co-occurrences elements of the matrix, is 100, number with which the performance is superior. Also only with empirical justifications, the value recommended for α is $\frac{3}{4}$.

After training, the word embeddings are obtained by summing the two weight matrices.

The same word-analogy test used by Mikolov et al. [45] was used to compare GloVe with the Skip-gram and CBOW models. GloVe outperformed both Word2vec algorithms in the semantic and syntactic questions sets.

The performance obtained in the ATIS slot filling task using one-hot vectors as input versus the use of word embeddings was compared by Kaisheng Yao et al. [32]. The results obtained were superior when feature vectors were the RNN input.

The use of word features was also tested by Hakkani-Tür et al. for slot filling and intent classification tasks [43]. However significantly improvements were not observed, hence, only results using one-hot vectors were reported.

2.6 Methods overview

As seen throughout this chapter, the algorithms that have been achieving better results for both the intent detection and the slot filling tasks are neural networks, more specifically Recurrent Neural Networks and Long Short Term Memory networks. Though in some comparisons, exposed before, researchers obtained better results with LSTMs, both RNNs and LSTMs will be tested in this work.

For the neural network inputs, as there is no consensus of whether it is better to use one-hot encoding or word embeddings, both are implemented and compared. The word embedding algorithms that will be used are the word2vec and GloVe, since the computing expense of the Bengio's approach is very high.

Chapter 3

Problem description and solution

As descibed in the previous chapter, the algorithms that will be used in the intent detection and slot filling tasks correspond to neural networks, more specifically RNNs and LSTMs. For the input of the neural networks, two word embeddings algorithms will be used, Word2vec and GloVe and one-hot encoding will also be experimented.

In this chapter, a detailed description of this thesis problem is made and the different approaches used to try to solve this problem are described, as well as their steps. The competitions in which this system will be used by the SocRob team, and their characteristics are also exposed.

Finally, the implementation of the system in the robot is described.

3.1 Problem description

To command a robot by speech there are two options: previous selected instructions or natural language. The disadvantages of using previous selected instructions is the fact that the user has to know the exact commands the robot can recognize. In contrast, when using natural language the user can say the command in whatever form he wants. The disadvantage of natural language is the difficulty of teaching the robot to understand it. In this work, a system that is able to understand natural language is going to be developed and will be applied in the competitions in which the SocRob team participates, described in 3.1.1.

For the robot to perform an action, it has to know which is the required action so it can run the program or group of programs that make the robot perform the action. Also, for each action there are parameters that have to be filled, such as locations, objects and so on.

These two requirements can be achieved by doing intent detection, being the intent detected the action to perform, and slot filling, in which the slots represent arguments to fulfill the action parameters. Also, if the number of actions the robot is able to perform is large, these intents can be divided in domains, being the domains determined before. Then depending on the domain, the intent is detected. However, as can be seen in 3.1.1, the set of actions for both competitions is small, so it is not worthy to have more than one domain.

Before applying the intent detection and slot filling, as the instruction can contain various commands, the instruction sentence is divided in phrases, being each a different command.

A scheme of an example instruction and its solution is showed in figure 3.1.

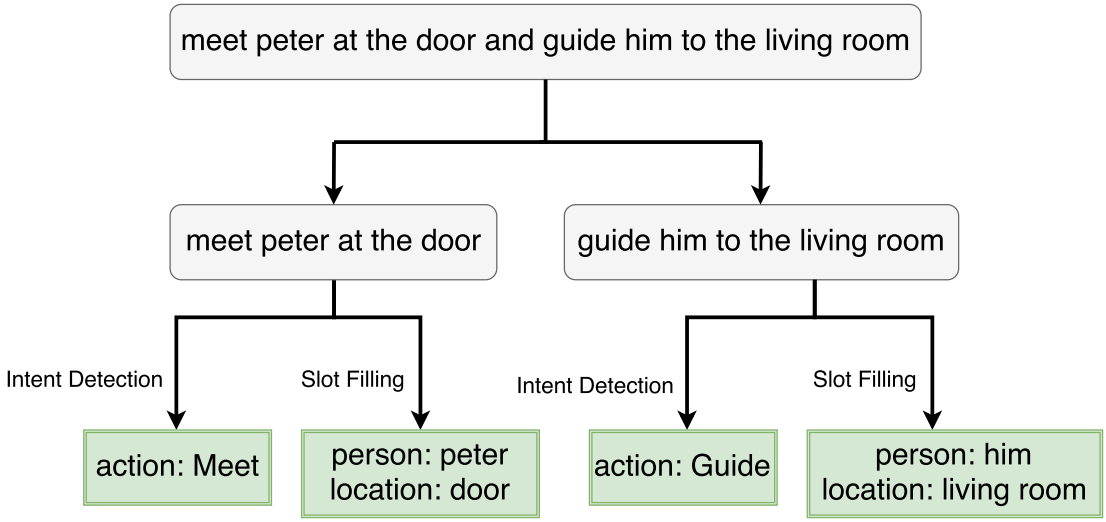


Figure 3.1: Problem example

3.1.1 Competitions

General Purpose Service Robots

GPSR is a task in which a voice instruction is given to the robot that the robot is supposed to understand and perform it. These instructions can contain more than one action the robot has to perform.

Observing instructions created by the GPSR generator available publicly, a set of actions for the intent detection and arguments for the slot filling task was created. The set of intents, their descriptions and the arguments that can be part of a instruction that as that intent, can be viewed in the table 3.1.

Intents	Description	Set of arguments
motion	moves to some place	source; destination; person; object
meet	meets a person	person; destination; object
grasp	grabs a object	object; destination; person
place	places a object	object; destination; person
take	takes object from one place to other or gives to someone	object; source; destination; person
tell	tells something to someone	what to tell; person; destination; object
answer	asks a person what they want to know and waits for question	person; destination; object
find	looks for an object or person	object; destination; person
guide	guides a person to a location	person; source; destination
follow	follows a person to a location	person; source; destination

Table 3.1: Set of intents and respective description and set of arguments for GPSR

The set of arguments for GPSR contains ”source” and ”destination”, which are the location where the action starts and ends, respectively, ”person” that corresponds to a person involved in the action, ”what to tell” which consists in what the robot has to say in the action ”tell” and ”object”. The location arguments can also correspond to location of objects or furniture.

To create the training sets, firstly sentences were created using the competition's generator. However, as these instructions were not annotated, the dataset had to be small. Otherwise, it would be very time consuming. Because of this a command generator that automatically annotates the sentences was created, allowing to create big datasets. For testing a dataset created using the RoboCup generator was used.

Speech recognition functionality benchmark

FBM3 or Speech Recognition Functionality Benchmark is slightly different from the approach used for the GPSR, relatively to the slot filling, i.e. for the GPSR the arguments in the sentence ”search the

kitchen for the coke" would be "coke" that is a object and "kitchen" a location, whereas for FBM3 the object would be "for the coke" and "the kitchen" would be the location.

The dataset was created the same way, but differing in the set of actions and arguments, presented in table 3.2 .

Intents	Description	Set of arguments
motion	robot moves to some place	goal; path
searching	looks for an object	theme; ground
taking	grabs a object	theme; source
placing	places a object	theme; goal
bringing	takes object from one place to other or gives to someone	theme; goal; source; beneficiary

Table 3.2: Set of intents and respective description and set of arguments for FBM3

The set of arguments for the Speech Recognition Functionality Benchmark contains "theme", which consists in the object used in the action, "goal" which corresponds to destination, "path" which is the path robot has to take in the action "motion", "ground" that consists in the location the robot has to look for something in the "searching" action and "beneficiary" which corresponds to the person that the robot has to give an object in the action "bringing".

3.2 Approaches

Two different approaches were implemented to understand the instructions given to the robot in natural language.

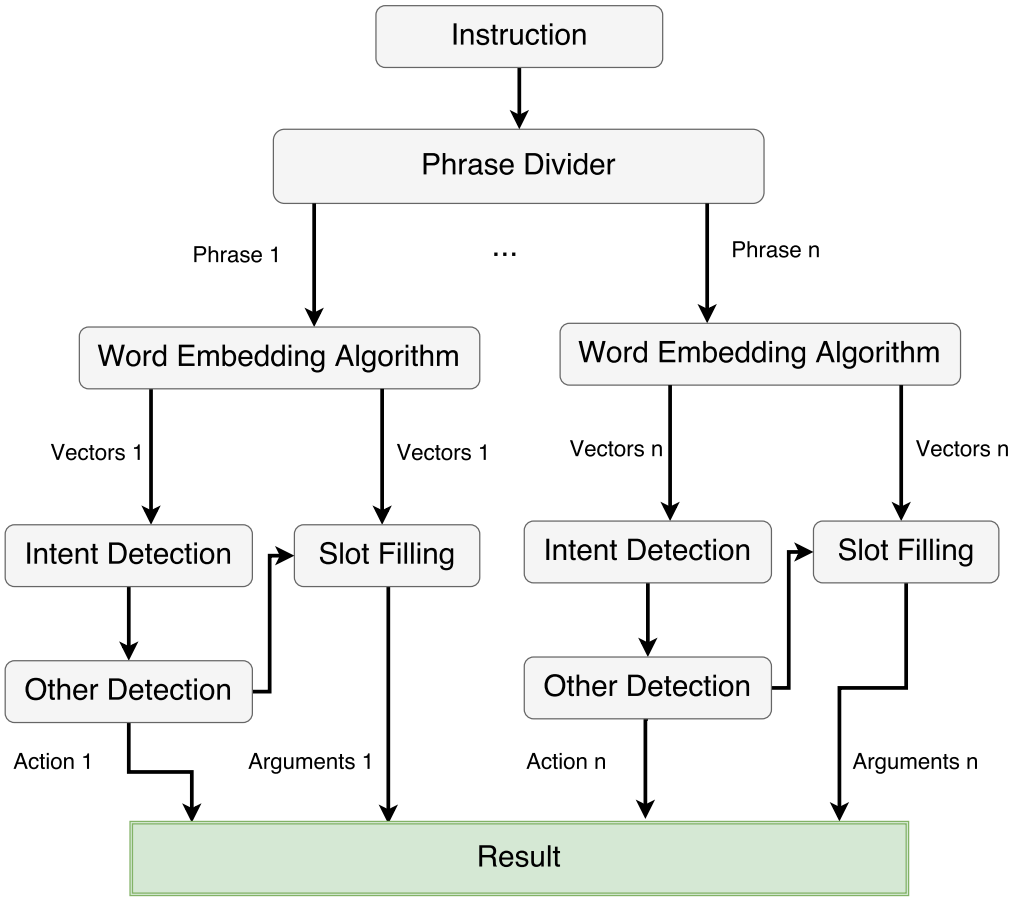


Figure 3.2: Scheme of the first approach

In the first approach, schematized in figure 3.2, after the instruction is received by the robot it is recognized by the speech recognition system. The resulting *string* is divided in phrases, being each resulting phrase a command to the robot. The words in the commands are then converted to feature vectors. The sequence of feature vectors corresponding to the commands are the input to the intent detection and slot filling models. Finally, the intent detection result is passed to an algorithm that determines if the intent is one of the previous defined actions or other. If it is other the result of the slot filling classification is erased and the result corresponds only to ”Other”. Otherwise the result is the intent detected and the slots identified.

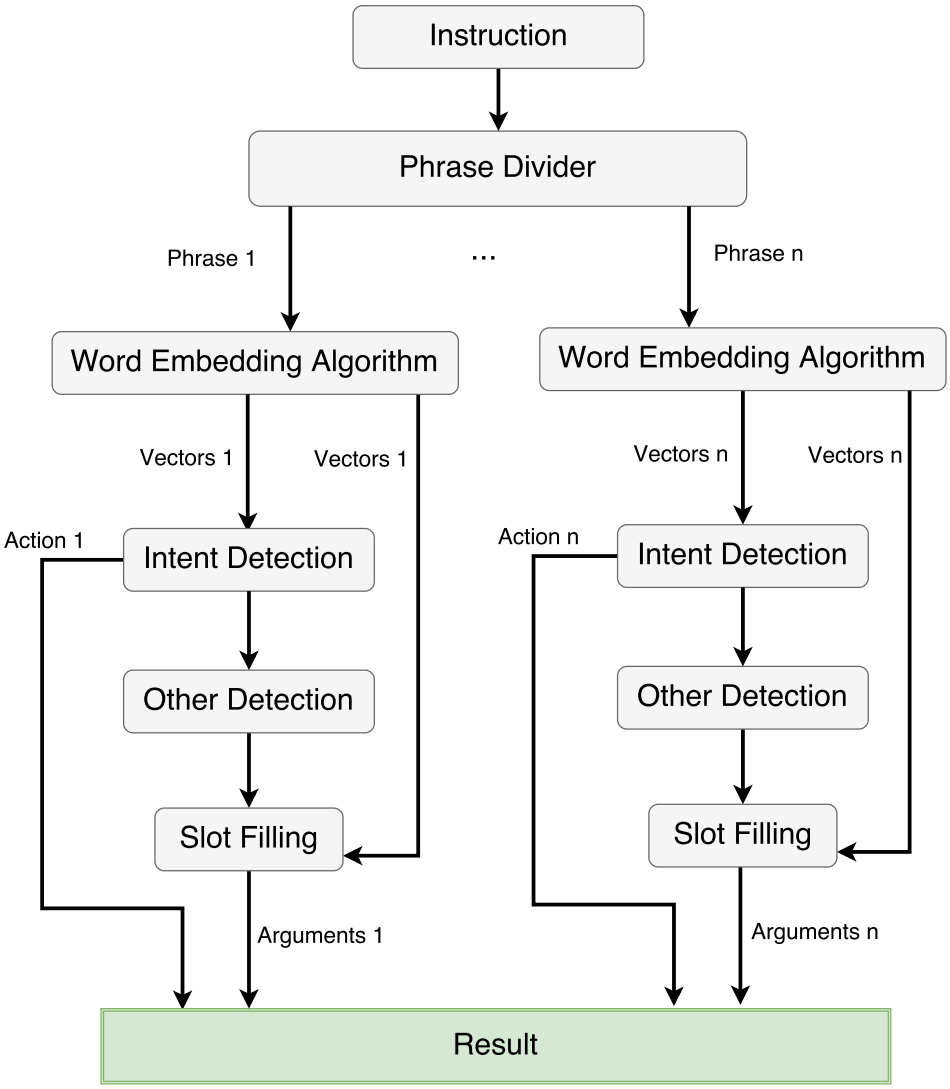


Figure 3.3: Scheme of the second approach

The second approach is represented in the scheme in the figure 3.3.

In this algorithm, instead of having the same slot filling model for all the intents that can be detected, there is a different model for each one. The slot filling model used is chosen according to the result of the intent detection system, after going through the system that determines if the action is 'Other' or one of the defined. The result obtained is similar to the one in the first method.

3.3 Dividing instructions in phrases

The instructions given to a service robot often have more than one command, for example, a person can ask the robot to find an object and deliver it or to go to the door and guide the person knocking to a location.

Because of that the instruction given has to be divided in phrases that will be the commands the robot needs to recognize.

To achieve this an open source syntactic parser from Google was used, Parsey McParseface, which

is part of Syntaxnet [1]. Given a sentence as input, Syntaxnet attributes a part-of-speech (POS) tag to each word, describing its syntactic function, such as noun, verb, adverb, determinant, adjective. It also determines the relationships between the words and represents it in a parse tree.

The model consists of a simple feed-forward neural network that operates on a task-specific transition system and uses global normalization with a conditional random field (CRF) objective. Though being a simple model, it achieves state of the art performance in part-of-speech tagging, syntactic dependency parsing and sentence compression [1].

To divide the sentences, first the existing verbs are found. The auxiliary verbs are differentiated from the principal verbs by the word they are related to. The principal verbs have a relationship with the verb of the previous phrase and the auxiliary verbs are related to a principal verb which is the next or previous word in the sentence.

By assembling the words that have relationships with the principal verbs, the words that are related to those words and so on, the phrases are constituted. As sometimes terms are related to others in previous phrases, the word order is taken into account when dividing the sentence, i.e. if a word is related to one in the previous command it still belongs to the phrase it is in. This is achieved by looking at the position of the principal verbs and of conjunctions when existent.

Finally the conjunctions connecting the commands are removed, once they do not add meaning to the sentence. An example of this process for the instruction ”I forgot my phone, find it in the living room and bring it to me.” can be found in the following table and figure.

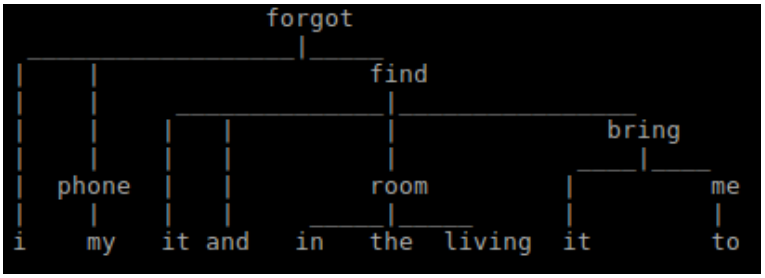


Figure 3.4: Parse tree obtained using Parsey McParseface

Word number	word	POS tag	related to word
1	i	pronoun	2
2	forgot	verb	-
3	my	pronoun	4
4	phone	noun	2
5	find	verb	2
6	it	pronoun	5
7	in	preposition	10
8	the	determinant	10
9	living	noun	10
10	room	noun	5
11	and	conjunction	5
12	bring	verb	5
13	it	pronoun	12
14	to	preposition	15
15	me	pronoun	12

Table 3.3: Result obtained using Parsey McParseface

In the table 3.3, the POS tags and the relationships between the words can be seen. The parse tree resultant is represented in figure 3.4. With this information, the instruction is divided in three phrases, ”i forgot my phone”, ”find it in the living room” and ”bring it to me”.

3.4 Word embeddings

Although, according to some literature, models using word embeddings have better performance than using one-hot encoding, some cases where the difference is not significant have been reported. Because of this, in this thesis both one-hot vectors and word feature vectors will be tested.

There is also not much agreement on which model has the best performance, Word2vec skip-gram model or GloVe. As a consequence, the two methods will be analyzed.

3.4.1 Word2vec

As described before in section 2.5.2, several extensions to the original skip-gram model were developed. In this thesis, the extensions used are negative sampling, being the number of words sampled $k = 15$, as advised [46], and subsampling of frequent words with a threshold of 10^{-5}, as advised in the original

Word2vec article.

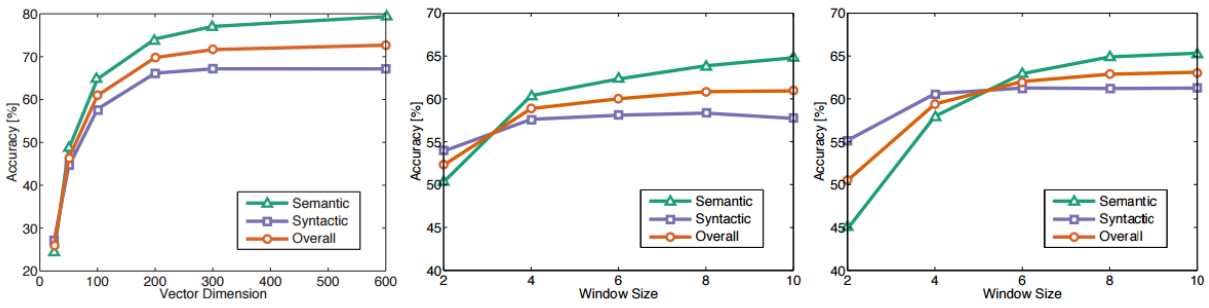
The corpus used to train the model was the Wikipedia of 2014, which contains 1.6 billion tokens. Before being used the corpus was tokenized and lowercased using the NLTK Python package. The punctuation and the numbers were also removed, once, frequently, speech recognition systems do not recognize punctuation and recognize numbers in full.

A vocabulary constituted by the 50000 most frequent English words was used. The window size chosen was of 5 words, the center one, two before and two after. To train the neural network the Adam optimizer was used to minimize a softmax loss function. The learning rate chosen was fixed and of 0.01.

3.4.2 GloVe

To create feature vectors using the GloVe method, the same corpora and vocabulary was used. The tokenization applied was also similar to the one in Word2vec.

In the following figure, graphics provided by Pennington et al. can be seen [47]. These graphics shows how the accuracy on the analogy task, previously discussed in section 2.5.3, varies with the word vector size, the window size and the type of window used.



(a) Symmetric context (b) Symmetric context (c) Asymmetric context

Figure 3.5: Accuracy on the analogy task as function of vector size and window size/type. Adapted from [47]

By looking at these graphics, a vector dimension of 300 was chosen, because in this value the increase of the performance declines significantly. As for the context, a symmetric window with a size of 10 words was used.

The maximum number of the matrix elements, x_{max}, used in the weighting function was 100 and the value of α was $\frac{3}{4}$, as recommended.

The model was trained using the Adaptive Gradient optimizer, AdaGrad, since it has achieved very good results for problems with sparse data and was also used in the original GloVe paper.

3.5 Intent detection

In the intent classification task, all the words in the instruction can be useful to determine the action the robot as to perform. To have the ability of capturing the meaning and connections between all the words

in the phrase, the neural network architecture chosen was LSTMs. Several types were tested, such as one layer LSTM, deep LSTMs, bidirectional LSTMs and deep bidirectional LSTMs.

The input provided to the neural networks corresponds to a word vector, computed by one-hot encoding or a word embedding algorithm, for each word in the sentence. The output is a group of confidence values, one for each of the previously selected actions, intent classes. The bigger value is then selected as the action determined. However, the intent could be one that is not represented in the set of classes. Hence, it is necessary to classify if the intent corresponds to the action with higher confidence value or to other action that does not belong to the available classes. This process is explained in the section 3.5.1.

In the first experiments, a softmax layer was used to convert these confidence values into probabilities. However, when predicting whether the intent of the instruction was in the set of classes, the performance decreased using the probabilities.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for\ j = 1, ..., K \tag{3.1}$$

As can be seen in the softmax function, equation 3.1, when the value z_j, that corresponds to one of the confidence values, is substantially bigger than the others, the resultant probability will be, approximately, 1. The values would then lose their magnitude and just be expressed in relation to the rest. Because of this, the output results were left unaltered.

To train the LSTMs, the Adam optimizer with varying learning rates was used to minimize a softmax cross entropy loss function.

3.5.1 Action Other determination

To predict if the action that the robot has been instructed to do is the one resultant from the neural network previously explained or if it is one that was not part of the classes, a SVM is used. A SVM was the algorithm chosen because this corresponds to a simple binary classification problem.

To train the SVM a dataset, containing a subset of the data used to train the neural network for the intent detection task and some commands for which the output should be $Other$, is used. The input of the SVM corresponds to the maximum value that is outputed by the intent detection neural network. Although not all types of sentences with an intent that is not present in the actions set are represented in the dataset, the value that they will output in the intent detection neural network is similar. This way, the SVM works similarly to a threshold that is automatically set, identifying the intent as $Other$ if the confidence value is not reached.

3.6 Slot filling

The algorithm selected to perform this task is also a LSTM neural network. In both implementation methods, LSTMs, deep LSTMs and bidirectional LSTMs were experimented.

The input corresponds to word vectors, like in the intent determination task. The output corresponds to a tag for each word that follows the popular IOB format [48]. The first word of a slot is tagged with a B of begin, the other words inside the slot are represented with the I of inside and the words that do not belong to a slot are tagged with the O of outside. An example of this output can be found in table 3.4.

Instruction	deliver	a	coke	to	peter	at	the	living	room
Slot tags	O	O	B-object	O	B-person	O	O	B-destination	I-destination

Table 3.4: Example of the slot filling output

In implementation method 1, as the possible classes are the same for all intents, if for some reason a utterance term has a tag that does not make sense for that action, is not part of the set of arguments correspondent to the intent detected, it is replaced by O. As an example, for the action "robot go to the kitchen", if "kitchen" is identified as the argument $what\ to\ tell$, it would be changed to O.

The training of the neural network was performed similarly to the one for the intent determination task.

For both the intent determination and slot filling tasks, as the input sentences belonging to the dataset did not have all the same length, the shorter ones had to be padded. To do this, firstly the maximum length was found. Then, for all sentences with a smaller length, zero vectors were added.

3.7 Implementation in the robot

To implement the system in the robot, a ROS package was created. A SMACH state machine was used with the first state being $Wait\_for\_instruction$, in which the robot waits for a voice instruction. Then, when an instruction is received, it passes to the state $Speech\_Recognition$, that corresponds to the speech to text transformation. After having a string of the instruction received, the robot would enter in the final state, $Natural\_Language\_Understanding$, in which the system developed is put in practice and the resultant intents and arguments are obtained. A sheme of this state machine can be seen in figure 3.6.

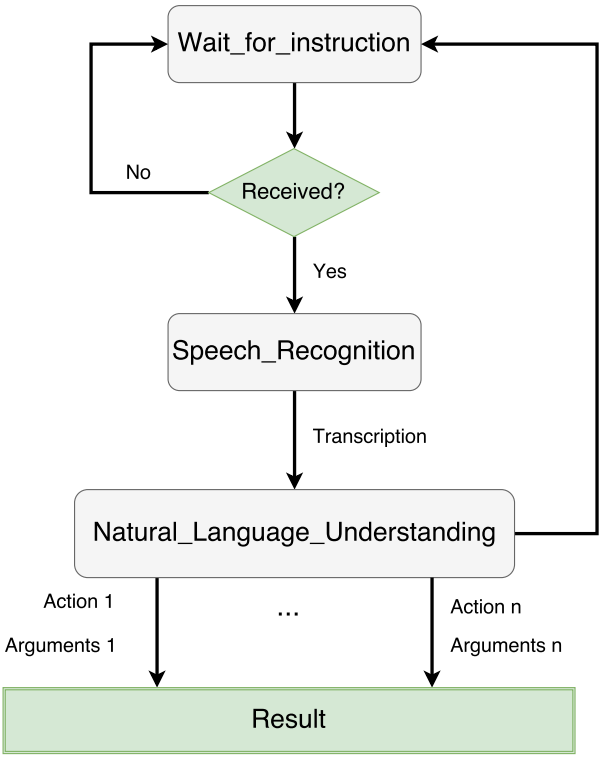


Figure 3.6: Scheme of state machine

Since the previously speech recognition system used by the SocRob team was grammar based, and, consequently, not appropriate for language understanding, the Google Cloud speech API [49] to text service was used. This speech recognition system is able to recognize natural language with a very good performance. Can also directly interpret a audio file without reproducing it, which is required in FBM3.

Chapter 4

Results

In this chapter, firstly different approaches to create word vectors are compared. Then, already using the word vectors that performed better, the two different implementation models are compared. To try to improve the model, variations of their constituents, such as the neural network architecture and its parameters, are tested. Also, possible justifications for the differences in the accuracies are presented.

The performance obtained in the ERL 2017 is also described.

4.1 Metrics

The metric used to validate and test the models created is the accuracy, being the result a value between 0 and 1 that represents the amount of correct results obtained out of the number of results, where 1 corresponds to obtaining all the correct ones. This was the metric used because in this case we are only interested in knowing if the instruction was understood by the robot or not. Also, as the training set is balanced there is no need to use a metric that deals better with unbalanced data.

4.2 Test Datasets

The various implementations were tested with two test sets, one related to the GPSR task and one with the FBM3. Both datasets are already converted to text to prevent the influence of speech recognition errors.

The test dataset for the GPSR was obtained using the generator provided by RoboCup and was annotated by hand. 100 instructions were generated randomly, being that some are composed by more than one phrase which represent different commands to the robot.

For the FBM3 task, a dataset of the RoCKIn@Home in 2014, competition that originated ERL, was used. This dataset contains 180 instructions, also with multiple commands. The annotations were already part of the set provided. However, since the slot filling output is different than the one used in this thesis, as explained in chapter 3, the annotations were modified.

4.3 Word vectors comparison

To compare the performance achieved using as input word embeddings created using the Word2vec and GloVe algorithms, described in chapter 2, and using one-hot vectors, both GPSR and FBM3 test sets were used.

An analogy or word similarity test could also have been performed. However, as the purpose is to compare the accuracy in these specific tasks and not the overall performance of the different methods, these were the tests performed.

In the table 4.1, the accuracy achieved for both the intent detection and slot filling tasks, using the GPSR dataset, is presented, for all the word input vectors methods tested. For the intent detection, the models used consisted in bidirectional LSTM with just one layer of 500 cells. The same neural network architecture was used for the slot filling task.

	one-hot vectors	GloVe	Word2vec skip-gram
intent detection	0.826	0.934	0.863
slot filling	0.817	0.895	0.871

Table 4.1: Comparison of the performance when varying the type of word vectors, for GPSR

As expected the accuracy using one-hot vectors was the worst. This happened because, as explained before, the one-hot vectors do not capture the meaning of the words. This can be a problem when a word does not appear in the training set but appears in the test set.

Regarding the word embedding methods tested, GloVe and Word2vec skip-gram, the accuracy for both the intent detection and the slot filling tasks is slightly higher using the feature vectors created by the GloVe algorithm. This was also an expected result, considering that in the comparison made by Pennington et al. [47], using a word analogy test, GloVe obtained a better performance.

Besides the word vectors comparison, this tests can confirm that for all the options tested good performances are obtained using this implementation.

The accuracy achieved in the FBM3 test set, for the intent detection and slot filling tasks, for all the word input vectors methods tested, are showed in table 4.2. The architectures used consist in bidirectional LSTM with two layers of 250 cells.

	one-hot vectors	GloVe	Word2vec skip-gram
intent detection	0.934	0.978	0.985
slot filling	0.653	0.728	0.675

Table 4.2: Comparison of the performance when varying the type of word vectors, for FBM3

Concerning the word vectors comparison, the results obtained in the FBM3 test were similar to the ones for GPSR. However, the intent detection accuracy was slightly higher using the Word2vec vectors.

By analyzing these results, it is clear that the performance in both tasks improves when vectors produced using the GloVe method are used. Consequently, in the following tests, the neural network inputs are GloVe word embeddings.

4.4 Model improvements

Having a working baseline model, several attempts were made to try to improve it. Some different neural network architectures, with varying sizes were tested. Also, the comparison between using implementation method 1 and 2, described in chapter 3, is described.

4.4.1 Approaches comparison

Firstly, a comparison of the two implementation methods was made. As explained before in chapter 3, the difference in the two approaches is related to how the slot filling task is performed. In approach 1, the slot filling task is not influenced by the intent detection. This means that the model used to select the arguments of the command is the same for all the possible intents, the two processes run in parallel. On the other hand, in approach 2 the slot filling model depends on the previous detected intent. There is then a different model, with a different set of arguments for each of the actions.

To do this comparison, a bidirectional LSTM neural network with 1 layer of 500 cells was used.

The results of the comparison for GPSR can be seen in 4.3.

	Approach 1	Approach 2
Accuracy	0.895	0.874

Table 4.3: Comparison of the performance for the different approaches implemented, for GPSR

As can be seen in table 4.3, the accuracy was slightly higher using approach 1. This can be explained by the fact that the number of arguments that constitute the set, 6, is not big, and some of them, such as $destination$ and $person$, can be present in a command of almost any action. Also, as, in approach 2, the selection of arguments depends on the intent detection task, when errors occur while determining the action, the wrong model is used, increasing the chance of arguments being badly selected.

For GPSR the method chosen was 1, firstly because the accuracy on the comparison made is better. Other reason to use it is the fact that the computational cost when training is higher for approach 2, once a model has to be trained for each of the intents while in approach 1 only one model has to be trained. Also, when the system is being executed method 1 is faster because the slot filling and intent detection tasks run in parallel.

The results of the comparison for FBM3 can be seen in 4.4.

	Approach 1	Approach 2
Accuracy	0.728	0.637

Table 4.4: Comparison of the performance for the different approaches implemented, for FBM3

Since the arguments present in FBM3 dataset are more complex than in GPSR dataset, it was expected that approach 2 would achieve a better result than method 1. However, as showed in table 4.4, the accuracy obtained using approach 1 is better. Similarly as for GPSR, the reasons that can explain this result are the fact that some of the arguments are common for almost all the possible intents, such as $theme$ and $destination$. The propagation of errors that result from the intent detection task can also prejudice the method's 2 result.

This way, from now on the approach used is 1.

4.4.2 Intent detection

Firstly, a comparison between the use of RNNs and LSTMs was made. The architecture used is similar, just replacing RNN cells by LSTM cells. The results for the GPSR are showed in table 4.5.

	RNN	LSTM
	1 layer of 500 cells	1 layer of 500 cells
Accuracy	0.869	0.931

Table 4.5: Comparison of the performance in the intent detection task when using RNNs and LSTMs, for GPSR

As expected, the accuracy achieved when using LSTMs was higher. This can be caused by the existence of long-term dependencies that could not be memorized by the RNNs.

The results for the FBM3 are showed in table 4.6.

	RNN	LSTM
	1 layer of 500 cells	1 layer of 500 cells
Accuracy	0.894	0.978

Table 4.6: Comparison of the performance in the intent detection task when using RNNs and LSTMs, for FBM3

As for GPSR, in FBM3 the result was better when LSTMs were used. The difference observed is

quite expressive, because some of the commands present in the dataset are quite extensive, reaching 15 words.

Due to these results, the following architectures that were tested consisted of different LSTMs neural network architectures and with different parameters.

The results obtained in the GPSR intent detection tests are presented in table 4.7.

	LSTM	DLSTM	BLSTM	DBLSTM	DBLSTM
	1 layer of 500 cells	2 layers of 500 cells	1 layers of 500 cells	2 layers of 250 cells	2 layers of 500 cells
Accuracy	0.931	0.900	0.934	0.923	0.908

Table 4.7: Comparison of the performance in the intent detection task when varying the neural network architecture, for GPSR

Observing table 4.7, it can be seen that there are no big differences in the accuracies obtained using the different types of LSTMs. Moreover, there is a small decrease in accuracy when the neural networks possess more than one layer. A reason that can provoke this is the architecture of the neural network being too complex for the problem. It can also be observed that the accuracy does not improve significantly when using bidirectional LSTMs. This result was expected, since the output of the intent detection task consists in the final output of the sequence.

Although the best result was obtained when using a neural network containing only 1 layer of 500 BLSTMS, the architecture chosen for the final model consists in a layer of 500 LSTM cells, once the difference between the two accuracies is not substantial and the computational cost is lower for the simple LSTM, both when training and when executing the NLU system.

The results obtained in the FBM3 intent detection tests are presented in table 4.8.

	LSTM	DLSTM	BLSTM	DBLSTM	DBLSTM
	1 layer of 500 cells	2 layers of 500 cells	1 layers of 500 cells	2 layers of 250 cells	2 layers of 500 cells
Accuracy	0.978	0.969	0.978	0.917	0.903

Table 4.8: Comparison of the performance in the intent detection task when varying the neural network architecture, for FBM3

Similarly as with GPSR dataset, using the FBM3 dataset it can be seen that the accuracy decreases when using multiple layers. The use of bidirectional LSTMs also decreased the performance.

The best results were obtained when a single layer of 500 LSTM or BLSTM cells was used. Thus, also due to the BLSTM higher computational cost, the architecture chosen also consisted in a single layer of 500 LSTM cells.

Furthermore, comparing the performance in GPSR and in FBM3, the intent detection results were higher in the FBM3. This is a consequence of it having a smaller set of intents.

4.4.3 Slot filling

Similarly as for the intent detection task, the first comparison made when searching for the best neural network architecture to use was between RNNs and LSTMs.

The results of this comparison, for GPSR, are showed on table 4.9

	RNN	LSTM
	1 layer of 500 cells	1 layer of 500 cells
Accuracy	0.836	0.873

Table 4.9: Comparison of the performance in the slot filling task when using RNNs and LSTMs, for GPSR

It can be observed, in table 4.9 that the accuracy is better when using a single layer of 500 LSTM cells than when a single layer of 500 RNN cells is used. This happens for the same reason as in intent detection, there are long-term dependencies that the RNNs are not able to capture.

In table 4.10, the comparison for the FBM3 dataset is showed.

	RNN	LSTM
	1 layer of 500 cells	1 layer of 500 cells
Accuracy	0.584	0.637

Table 4.10: Comparison of the performance in the slot filling task when using RNNs and LSTMs, for FBM3

The same happens using FBM3 dataset, as can be seen in table 4.10.

Because of this, for both tasks the LSTMs are elected as the model to be used in the final model.

Following, different LSTM neural network architectures, with some changes in the parameter, were tested, using the GPSR and the FBM3 test sets.

The results obtained using the GPSR dataset are presented in table 4.11.

	LSTM	DLSTM	BLSTM	DBLSTM	DBLSTM
	1 layer of 500 cells	2 layers of 500 cells	1 layers of 500 cells	2 layers of 250 cells	2 layers of 500 cells
Accuracy	0.873	0.922	0.895	0.914	0.947

Table 4.11: Comparison of the performance in the slot filling task when varying the neural network architecture, for GPSR

Differently than what happens in the intent detection task, as can be seen in table 4.11, the accuracy improves when more complex models, with more hidden layers, are used. This happens because the

slot filling task corresponds to a more complex task, since it is a sequence tagging problem instead of a single classification. Also, the results obtained are better using bidirectional LSTMs, due to the fact that a classification of a word can depend on the following words instead of only the previous words.

The accuracies obtained using the FBM3 dataset are presented in table 4.12.

	LSTM	DLSTM	BLSTM	DBLSTM	DBLSTM
	1 layer of 500 cells	2 layers of 500 cells	1 layers of 500 cells	2 layers of 250 cells	2 layers of 500 cells
Accuracy	0.637	0.648	0.687	0.728	0.762

Table 4.12: Comparison of the performance in the slot filling task when varying the neural network architecture, for FBM3

Using the FBM3 dataset it was also determined, based on table 4.12, that the results improve with the use of more than one layer and bidirectional LSTMs. The reasons that justify these results are the same as for the GPSR testset.

With these results in mind, the neural network chosen for the slot filling task, for both GPSR and FBM3 is a DBLSTM composed by 2 layers of 500 cells each.

By observing this results, it can also be seen that the performance in the FBM3 test was quite smaller. This is due to the superior arguments complexity, i.e. in the GPSR test the arguments were composed by less words.

4.5 Overview

After comparing the word embedding options and various neural network architectures, the final models for GPSR and FBM3 were chosen. In the final models, firstly, the word vectors are created using the GloVe algorithm. Then, for the intent detection a neural network containing one layer of 500 cells is utilized. The slot filling task is executed by a neural network with 2 layers of 500 BLSTMs. As the implementation approach 1 achieved better results, these two tasks can be performed in parallel, diminishing the time of execution.

4.6 ERL 2017

For the 2017 ERL, a model, using this thesis work, was created to compete in the FBM3 task. As the competition happened before the comparisons explained above were made, the model used was not the final one, chosen based on the results presented above.

The system used was based on the second implementation approach, described in 3. Furthermore, for the intent detection, a deep bidirectional LSTM with 2 layers of 250 cells was used, being the input word embeddings created using the Word2vec algorithm. For the slot filling, the architecture used in

each of the models, one per intent, was also deep bidirectional LSTM with 2 layers of 250 cells. The same word embeddings were used as input.

As far as my knowledge goes, only other team, used an automatic natural language understanding system, LU4R, previously described in chapter 2. As the list of verbs, names and objects were small and available to the teams, it was a possible to use different approaches, such as hard coding.

In ERL, the evaluation is performed by computing the accuracy over the complete result, i.e. a result is correct only if all the intents and all the slots of the correspondent instruction are correct. In case of draw other variables, such as, the accuracy of the intent detection and of the speech recognition are used.

The result in this competition was better than expected, once, as explained before in the chapter 3, the slot filling output in this task is different than the one used in this thesis. SocRob was placed in the third place, achieving a better performance than the team using LU4R.

However, the result would have probably been better if the model used had been the final one.

Chapter 5

Conclusions

In this chapter an overview of the fulfillment of this thesis objectives, previously declared in chapter 1, is made. Modifications and future work that could improve or complement this work are also described.

5.1 Achievements

The purpose of this work was to develop a natural language understanding system that can be used by service robots. This objective was successfully achieved, as a NLU system that obtains good accuracies, showed in chapter 4, was built. Furthermore, the SocRob team was in need of a NLU system that is now available for the team to use in competitions. In fact, it was already used in a competition with a quite good performance. However, the system used was not the final one, since the comparisons between the steps of the system's pipeline had not been done yet. It is then believed that using the final model created in this work the results obtained in the following competitions will be better.

Considering the final model, after dividing the sentences in phrases, commands, the words are converted in word vectors using the GloVe algorithm, since a better accuracy was achieved when using it rather than Word2vec and one-hot encoding. These vectors consist then on the input of a neural network with a single layer of 500 LSTM cells which is able to detect the intent of the command. The maximum output value, that corresponds to the correct intent, is the input of a SVM that will detect if this intent is correct or if the required command does not belong to the intents set. In parallel, other neural network, this one with 2 layers of 500 bidirectional Long Short Term Memory cells, that performs the slot filling task, does sequence to sequence classification and outputs whether the words belong to an argument and the nature of that argument. Finally, the system outputs the intent and the arguments obtained and the robot is able to perform the action requested.

Two of the system's steps, the phrase divider and the other action detector, were not compared in chapter 4. However, since their performance influences the accuracy of the tests realized for the other steps, they also suggest that these were successful implemented and are obtaining good results.

In conclusion, although the principal objectives were achieved, a NLU system that can be used in service robots, such as MOnarCH, was developed, a lot of work can be done to complement it, as

described in the following section.

5.2 Future Work

Even though the proposed objectives were achieved, further work, that is not of the scope of this thesis or that was not possible to do due to time constraints, could improve or complement this work. Therefore, the following topics are suggested as future work:

- Build a dataset with commands made by humans instead of using an automatic generator. The increased variances obtained would ease the neural network training and improve its ability to generalize well.
- Use a bigger dataset to create the word embeddings. The corpora of wikipedia used is constituted by 1.6 billion tokens. There are other corpus, such as Common Crawl and Twitter corpora, that can reach 840 billion tokens. This was not a possibility in this work due to the incredibly high computational cost.
- Further experiment different neural network architectures, such as deeper Long Short Term Memories, specially for the slot filling task since an accuracy increase was noticed when layers were added. This was not a possibility because of time constraints and higher computational cost.
- Modify the algorithm so the robot can interpret the instructions taking into account grounded information provided by a semantic map, like was done in LU4R [31].
- Develop a dialogue manager so the robot can answer to the human command's to ask a clarification or further information and be able to understand it. This could be done using deep reinforcement learning.
- Develop a ROS node to interact between the Natural Language Understanding system developed and the robot's action servers.

Bibliography

[1] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins. Globally normalized transition-based neural networks. *Google ACL*, 2016.

[2] J. Messias, R. Ventura, P. Lima, J. Sequeira, P. Alvito, C. Marques, and P. Carriço. A robotic platform for edutainment activities in a pediatric hospital. *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2014.

[3] Eurostat. Disability statistics - need for assistance. Internet: http://ec.europa.eu/eurostat/statistics-explained/index.php/Disability_statistics_-_need_for_assistance, November 2015.

[4] Eurostat. Mortality and life expectancy statistics. Internet: http://ec.europa.eu/eurostat/statistics-explained/index.php/Mortality_and_life_expectancy_statisticsLife_expectancy_is_increasing, June 2016.

[5] Eurostat. Population by age group. Internet: http://ec.europa.eu/eurostat/web/products-datasets/-/tps00010, May 2017.

[6] G. Tur and R. D. Mori. *Spoken Language Understanding Systems for Extracting Semantic Information from Speech*. John Wiley Sons, Ltd, 2011.

[7] A. Gorin, G. Ricciardi, and J. Wright. How may i help you? *Speech Communication 23*, 1997.

[8] J. Chu-Carrol and B. Carpenter. Vector-based natural language call routing. *Association for Computational Linguistics*, 1999.

[9] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 2000.

[10] P. Hafiner, G. Tur, and J. H. Wright. Optimizing svms for complex call classification. *ICASSP*, 2003.

[11] H.-K. J. Kuo and C.-H. Lee. Discriminative training of natural language call routers. *IEEE Transaction on Speech and Audio Processing, VOL. 11, NO. 1*, 2003.

[12] G. Bouchard and B. Triggs. The trade-off between generative and discriminative classifiers. *COMPSTAT Symposium*, 2004.

[13] R. Nallapati. Discriminative models for information retrieval. *SIGIR*, 2004.

[14] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in Neural Information Processing Systems*, 2002.

[15] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 1997.

[16] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. 1998.

[17] S. B. Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 2011.

[18] I. Zitouni, H.-K. J. Kuo, and C.-H. Lee. Boosting and combination of classifiers for natural language call routing systems. *Speech Communication*, 2003.

[19] S. Ravuri and A. Stolcke. Recurrent neural network and lstm models for lexical utterance classification. 2015.

[20] S. Seneff. Tina: A natural language system for spoken language applications. 1992.

[21] J. Dowding, J. M. Gawron, and D. A. et al. Gemini: A natural language system for spoken-language understanding. 1993.

[22] W. Ward. Understanding spontaneous speech: The phoenix system. *Proceedings of the IEEE International Conference on Acoustics*, 1991.

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

[24] T. Kudo and Y. Matsumoto. Chunking with support vector machines. 2001.

[25] H. M. Wallach. Conditional random fields: An introduction. 2004.

[26] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. *Association for Computational Linguistics (ACL) System Demonstrations*, 2014.

[27] R. Basili, E. Bastianelli, G. Castellucci, D. Nardi, and V. Perera. Kernel-based discriminative reranking for spoken command understanding in hri. *Advances in Artificial Intelligence*, 2013.

[28] E. Bastianelli, D. Croce, A. Vanzo, R. Basili, and D. Nardi. Perceptually informed spoken language understanding for service robotics. *Proceedings of the IJCAI Workshop on Autonomous Mobile Service Robots*, 2016.

[29] D. Croce, G. Castellucci, and E. Bastianelli. Structured learning for semantic role labeling. *Intelligenza Artificiale*, 2012.

[30] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. *Proceedings of the International Conference on Machine Learning*, 2003.

[31] E. Bastianelli, D. Croce, A. Vanzo, R. Basili, and D. Nardi. A discriminative approach to grounded spoken language understanding in interactive robotics. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI*, 2016.

[32] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu. Using recurrent neural networks for slot filling in spoken language understading. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 2015.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2012.

[34] S. Ruder. An overview of gradient descent optimization algorithms. 2016.

[35] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[36] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[37] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 2015.

[38] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tür, X. He, L. Heck, G. Tur, D. Yu, and G. Zweig. Using recurrent neural networks for slot filling in spoken language understading. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 2015.

[39] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.

[40] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[41] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. 2015.

[42] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi. Spoken language understanding using long short-term memory neural networks. *Proceedings of the IEEE SLT Workshop*, 2014.

[43] D. Hakkani-Tür, G. Tur, A. Celikyilmaz, Y.-N. Chen, J. Gao, L. Deng, and Y.-Y. Wang. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. *Interspeech*, 2016.

[44] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning*, 2003.

[45] T. Mikolov, G. Corrado, K. Chen, and J. Dean. Efficient estimation of word representations in vector space. *Proceedings of the International Conference on Learning Representations*, 2013.

[46] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 2013.

[47] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representations. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

[48] L. A. Ramshaw and M. P. Marcus. Text chunking using transformation-based learning. *Third Workshop on Very Large Corpora*, 1995.

[49] Google. Google cloud speech api. Internet: https://cloud.google.com/speech/.