# Human-robot speech interaction for service robots

Pedro Henrique Alves Martins
pedrohenriqueamartins@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2017

**Abstract**

To develop a system that is able to understand natural language instructions 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, being that the best results are achieved when the vectors are developed by the GloVe algorithm. For the intent detection and slot filling 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

## 1. Introduction

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 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.

The purpose of this work is to develop a program that will be able to understand the action requested by 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.

## 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. However, the focus of this work is only on the intent detection and slot filling, once there is only one domain for the robot.

LU4R, Language Understanding chain For Robots, similarly to this work is a Natural Language Understanding system that has as one of its purposes the recognition of the RoboCup GPSR instructions [3], [2].

The task of intent detection aims at classifying a speech utterance into one of various semantic classes. The first reported works in this area were developed for call routing, trying to direct the call to the right operator [9], [6]. With the advances in discriminative classifiers such as, Boosting [23], Support Vector Machines [11] and Minimum Classification Error [14], researches started using them in intent detection problems.

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, such as TINA [24] and Gemini [7]. Due to the difficulty of creating grammars, discriminative classifiers, such as Conditional Random Field and Support Vector Machines and generative classifiers, like Finite State Trandsucers, started being used for this task [22].

### 2.1. Neural Networks

After being one of the artificial intelligence biggest promises, in the 1990's, the interest in neural net-

works diminish significantly due to poor results. However, with the technological advances of this century 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 [26].

### Recurrent Neural Networks

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

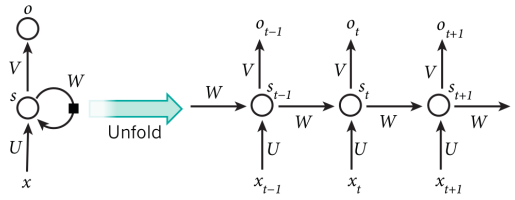



Figure 1: A recurrent neural network and the unfolding in time of the computation involved in its forward computation [15]

In the figure 1, 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:

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

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", once 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:

$$o_t = f_2(Vs_t) \tag{2}$$

where:

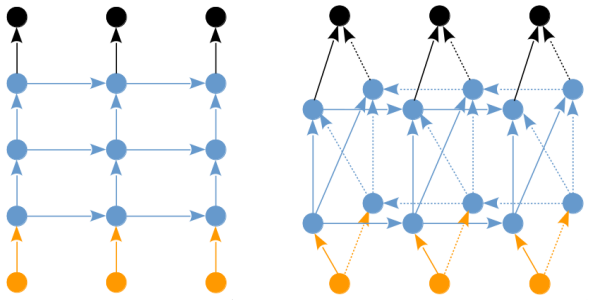
$f_2$ is an activation function, usually the $softmax$,

$V$ the weight used to obtain the output.

In this RNN every time step has an output. This network is used when doing sequence to sequence classification, 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, unrolled networks of DRNN and DBRNN can be seen.



(a) Deep Recurrent Neural Network

(b) Deep Bidirectional Recurrent Neural Network

Figure 2: Schemes of a deep RNN and a deep bidirectional RNN

The only difference between deep networks and the one presented before in figure 1, 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:

$$s^2{}_t = f(Us^1{}_t + Ws^2{}_{t-1}) \tag{3}$$

And, for the DBRNN is:

$$s^{2-}{}_t = f(U^- s^1{}_t + U^+ s^1{}_t + Ws^2{}_{t-1}) \tag{4}$$

$$s^{2+}{}_t = f(U^- s^1{}_t + U^+ s^1{}_t + Ws^2{}_{t+1}) \tag{5}$$

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

To confirm that RNNs outperform previously used algrithms, Kaisheng Yao et al. compared RNNs with FST, SVM and CRF in the ATIS slot filling task [26]. 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 [16]. 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 it was slightly worse, being their explanation the fact that in the entertainment domain the slots correspond to bigger expressions such as movie names [16].

### 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 [5]. In 1997, Hochreiter et al. proposed a novel RNN architecture, called Long Short-Term Memory (LSTM) [13].

### Long Short-Term Memory

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 [10]. 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 3, where the line junctions represent concatenations of vectors in a matrix, the $\odot$ represent the Hadamard product and the blue rectangles the activation functions.

The forget gate, $f_t$, decides the amount of information, from the previous cell, that should be taken into account. This is done by concatenating the output of the previous cell, $h_{t-1}$, and the input of the current cell, $x_t$.

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 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 $x_t$.

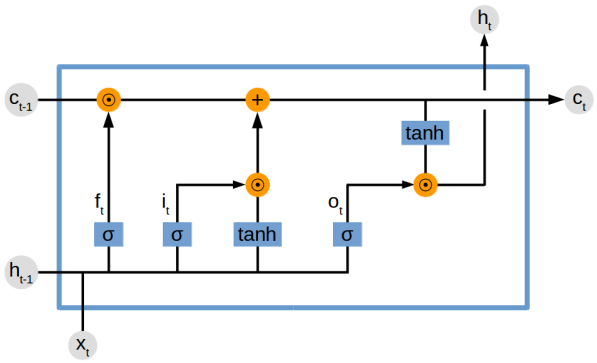



Figure 3: Scheme of LSTM memory cell

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 ouput gate, $o_t$ and the *tanh* of the cell state, $c_t$.

These steps, are represented in the following equations:

$$f_t = \sigma(W_f \cdot [h_{t-1},\ x_t] + b_t) \tag{6}$$

$$i_t = \sigma(W_i \cdot [h_{t-1},\ x_t] + b_i) \tag{7}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c \cdot [h_{t-1},\ x_t] + b_c) \tag{8}$$

$$o_t = \sigma(W_o \cdot [h_{t-1},\ x_t] + b_o) \tag{9}$$

$$h_t = o_t \odot tanh(c_t) \tag{10}$$

where:

$W$ 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 maintain 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 [21]. 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 [21], 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 [25]. 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 [12]. The bidirectional LSTM had the best performance, followed by the LSTM.

### 2.2. **Word Embeddings**

One way to represent the words in the input sentence is by using one-hot vectors, that contain one 1, that corresponds to the *id* of the word and multiple 0. 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. 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. 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 [26]. One of the first neural language models to produce word embeddings was developed by Yoshua Bengio at the beginning of the century [4]. Two other algorithms, Word2vec [17], [18] and GloVe [19] have produced better results, being also less computationally expensive.

## 3. Problem description

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, this intents can be divided in domains, being the domains determined before. Then depending on the domain, the intent is detected. However, as the set of actions for both competitions is small, 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 4.

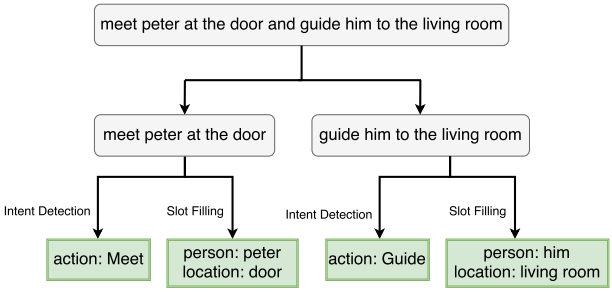



Figure 4: Problem example

### 3.1. **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 realize 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 and their explanation can be viwed in table 1

| Intents | Description |
|---|---|
| motion | moves to some place |
| meet | meets a person |
| grasp | grabs a object |
| place | places a object |
| take | takes object to some place or to someone |
| tell | tells something to someone |
| answer | waits for question |
| find | looks for an object |
| guide | guides a person to a location |
| follow | follows a person to a location |

Table 1: Set of intents and respective description 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, 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. The actions set and their description is presented in table 2.

| Intents | Description |
|---|---|
| motion | robot moves to some place |
| searching | looks for an object |
| taking | grabs a object |
| placing | places a object |
| bringing | takes object to a place or to someone |

Table 2: Set of intents and respective description 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".

## 4. Implementation

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

In the first approach, schematized in figure 5, 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 correspondent

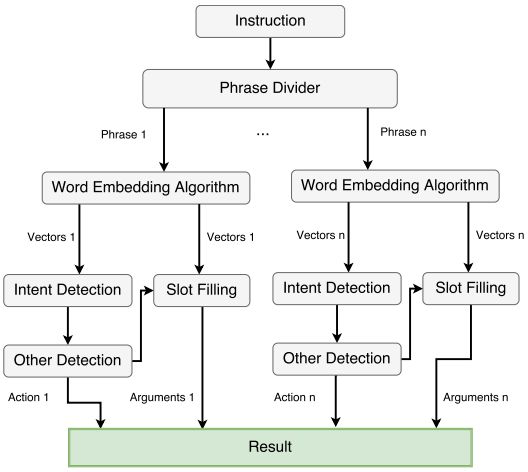



Figure 5: Scheme of the first approach

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.

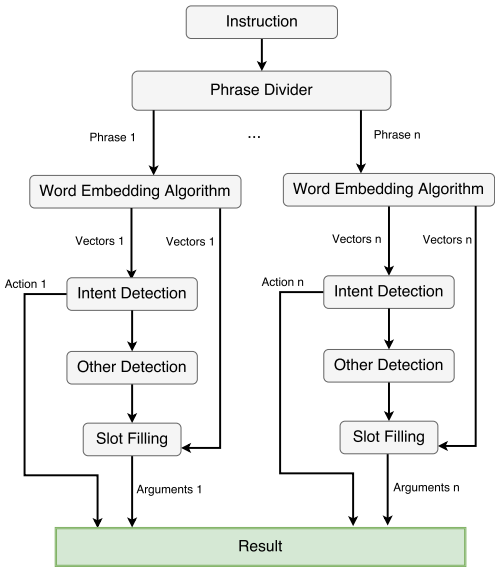



Figure 6: Scheme of the second approach

The second approach is represented in the scheme in the figure 6.

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.

### 4.1. **Dividing instructions in phrases**

The instructions given to a service robot often have more than one command. Because of that the instruction given has to be divided in phrases that will correspond to 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.

To divide the sentences, first the existent 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 that 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.

### 4.2. **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.

#### Word2vec

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 [18], 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.

#### 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.

By observing the tests made by Pennington et al. [19], 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 $\alpha$ was 3/4, as recommended.

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

### 4.3. **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.

In the first experiments, a softmax layer was used to convert this 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.

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

#### 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.

### 4.4. 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 [20]. 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.

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.

### 4.5. 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.

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 [8] 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.

## 5. Results

### 5.1. Metrics

The metric used to validate and test the models created is the accuracy, percentage of examples correct in relation to the total number of examples. 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.

### 5.2. 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 phrases 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 4, the annotations were modified.

### 5.3. Word vectors comparison

To compare the performance achieved using as input word embeddings created using the Word2vec and GloVe algorithms, and using one-hot vectors,

both GPSR and FBM3 test sets were used.

In the table 3, 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 |
|---|---|---|---|
| intent detection | 0.826 | 0.934 | 0.863 |
| slot filling | 0.817 | 0.895 | 0.871 |

Table 3: 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. [19], using a word analogy test, GloVe obtained a better performance.

Besides the word vectors comparison, this tests can confirm that with all the options 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. The architectures used are similar to the ones used for GPSR.

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

Table 4: 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 this 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.

### 5.4. **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 4, is described.

#### Approaches comparison

Firstly, a comparison of the two implementation methods was made. To do this, a bidirectional LSTM neural network with 1 layer of 500 cells was tested in both test sets.

The results of the comparison for GPSR and FBM3 can be seen in 5 and 6, respectively.

| | Approach 1 | Approach 2 |
|---|---|---|
| Accuracy | 0.895 | 0.874 |

Table 5: Comparison of the performance for the different approaches implemented, for GPSR

| | Approach 1 | Approach 2 |
|---|---|---|
| Accuracy | 0.728 | 0.637 |

Table 6: Comparison of the performance for the different approaches implemented, for FBM3

As can be seen in the tables, the accuracy was slightly higher using approach 1. This result was expected for GPSR and 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. However, once 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. This result can be explained by the fact that 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.

Besides the better result, other reason to use approach 1 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. This way, from now on the approach 1 is used.

### Intent detection

Firstly, a comparison between the use of RNNs and LSMSs was made. The architecture used is similar, just replacing RNN cells by LSTM cells. The results for the GPSR and FBM3 are showed in tables 7 and 8, repectively.

| | RNN<br>1 layer of 500 cells | LSTM<br>1 layer of 500 cells |
|---|---|---|
| Accuracy | 0.869 | 0.931 |

Table 7: Comparison of the performance in the intent detection task when using RNNs and LSTMs, for GPSR

| | RNN<br>1 layer of 500 cells | LSTM<br>1 layer of 500 cells |
|---|---|---|
| Accuracy | 0.894 | 0.978 |

Table 8: Comparison of the performance in the intent detection task when using RNNs and LSTMs, for FBM3

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, once some of the commands present in the datasets are quite extensive, reaching 15 words.

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

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

Table 9: Comparison of the performance in the intent detection task when varying the neural network architecture, for GPSR

Observing table 9, 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.

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

Table 10: 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.

Comparing the performance in GPSR and in FBM3, the intent detection results were higher in the FBM3. This can be a consequence of the FBM3 task having a smaller set of intents.

### 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.

| | RNN<br>1 layer of 500 cells | LSTM<br>1 layer of 500 cells |
|---|---|---|
| Accuracy | 0.836 | 0.873 |

Table 11: Comparison of the performance in the slot filling task when using RNNs and LSTMs, for GPSR

Observing tables 11 and 12, it can be observed that the accuracy is better when using a single layer

| | RNN<br>1 layer of 500 cells | LSTM<br>1 layer of 500 cells |
|---|---|---|
| Accuracy | 0.584 | 0.637 |

Table 12: Comparison of the performance in the slot filling task when using RNNs and LSTMs, for FBM3

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. 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.

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

Table 13: 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 13, 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, once it is a sequence to sequence 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.

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

Table 14: 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 14, 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. Comparing the performance in the two tests, is can be observed that the accuracy is quite smaller for FBM3. This is due to the superior arguments complexity, i.e. in the GPSR test the arguments were composed by less words.

With this 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.

### 5.5. **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.

### 5.6. **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 4. 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.

The result in this competition was better than expected, once, as explained before in the chapter 4, 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. Although, the result would have probably been better if the model used had been the final one.

## 6. Conclusions

The purpose of this work was to develop a natural language understanding system for all types of service robots. This objective was successfully achieved, as a NLU system that obtains good accuracies, showed in chapter 5, was built. Furthermore, the SocRob team was in need of a NLU system that is now available for the team to use in competitions.

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.


### Acknowledgements

I would first like to express my sincere gratitude to my thesis advisors, Prof. Lus Custdio 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.


### References


[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 Inc*, 2016.

[2] E. Bastianelli, D. Croce, A. Vanzo, R. Basili, and D. Nardi. A discriminative approach to grounded spoken language understanding in interactive robotics. *IJCAI*, 2016.

[3] E. Bastianelli, D. Croce, A. Vanzo, R. Basili, and D. Nardi. Perceptually informed spoken language understanding for service robotics. *IJCAI*, 2016.

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

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

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

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

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

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

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

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

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

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

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

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

[25] 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.

[26] 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.