TÉCNICO LISBOA

A Spoken Goal-Oriented Dialogue System for Service Robots

João Nuno Marcelino Barroca

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor(s): Prof. Luís Manuel Marques Custódio
Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. José Fernando Alves da Silva
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof[a]. Isabel Maria Martins Trancoso

December 2019

Dedicated to my parents, Alfredo and Lisete.

Acknowledgments

First, I would like to express my sincere gratitude to Professor Luís Custódio and Professor Rodrigo Ventura for the continuous availability, guidance and engagement throughout the development of this thesis.

Second, to the SocRob@Home team for all the technical and expert assistance in both practical and theoretical matters, including the opportunity to implement and test the developed system in a real service robot.

Furthermore, to Professor Pedro Lima and the SciRoc organization for the opportunity to compete in a robotics competition and test the system in a real world environment.

Last, my deepest gratitude goes to my parents and sister, for all the unconditional support and motivation since the start of my journey.

Resumo

Nos últimos anos, os investigadores têm tentado desenvolver sistemas automáticos de conversação capazes de interagir com os utilizadores de forma fácil e natural. Estes sistemas baseiam-se, essencialmente, na utilização de língua natural, ou seja, fala utilizada na comunicação entre seres humanos no dia-a-dia, permitindo assim uma interação mais eficiente entre o computador e o utilizador. Estes tipos de sistemas são extremamente utilizados em serviços de informação de rede, e em robôs domésticos, de serviço e sociais. Quando implementados no mundo real, uma grande variedade de fatores pode contribuir para uma degradação no seu desempenho. Alguns dos principais fatores são a elevada variedade no modo como os utilizadores falam e interagem com o sistema, e o elevado ruído encontrado em determinados ambientes. Assim, este tipo de sistemas de conversação deve ser capaz de lidar com a incerteza no reconhecimento de fala e compreensão do que o utilizador diz, explorando a natureza sequencial do diálogo para desambiguar tais incertezas. Nesta dissertação, desenvolvemos um sistema de diálogo capaz de lidar com essas incertezas. Em particular, projetamos três componentes essenciais que constituem este tipo de sistemas: um componente de compreensão de língua natural, um componente de seguimento do estado do diálogo e um componente de gestão do diálogo. Além disso, implementamos o sistema de diálogo desenvolvido num robô de serviço, testando o seu funcionamento em ambientes do mundo real.

Palavras-chave: Sistemas de Diálogo, Compreensão de Língua Natural, Seguimento do Estado do Diálogo, Gestão do Diálogo, Interação entre Humanos e Robôs

Abstract

In recent years, researchers have been trying to develop conversational systems capable of understanding and speaking in natural language, so that humans can interact with them easily and more naturally. Human-Computer dialogue systems, particularly goal-based dialogue systems, have been the most important component in conversational systems, since they allow users to speak naturally in order to accomplish tasks more efficiently. They are widely demanded in network information services, service, domestic and social robots, among others. When deployed to production, a spoken dialogue system may encounter a variety of difficulties, such as a large variation in the users of the system and high noise environments. Thus, the dialogue system must be able to use statistical frameworks to handle the uncertainty in both speech recognition and language understanding. In this thesis, we develop a spoken goal-oriented dialogue system capable of dealing with such uncertainties, by maintaining uncertainty about everything the user has said, and exploit the sequential nature of dialogue to disambiguate in the presence of errors. In particular, we design three main components: Natural Language Understanding (NLU), Dialogue State Tracking (DST) and Dialogue Management (DM). Furthermore, we implement the developed dialogue system in a service robot and test the complete system on a real world environment.

Keywords: Spoken Dialogue System, Goal-Oriented Dialogue System, Natural Language Understanding, Dialogue State Tracking, Dialogue Management, Human-Robot Interaction

Contents

List of Tables

List of Figures

Glossary

ASR	**A**utomatic **S**peech **R**ecognition
ATN	**A**ugmented **T**ransition **N**etworks
AUROC	**A**rea **U**nder the **R**eceiver **O**perating **C**haracteristics
BERT	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
CBOW	**C**ontinuous **B**ag-**O**f-**W**ords
CRF	**C**onditional **R**andom **F**ield
DARPA	**D**efense **A**dvanced **R**esearch **P**rojects **A**gency
DM	**D**ialogue **M**anagement
DSM	**D**ialogue **S**tate **M**achine
DSTC	**D**ialogue **S**tate **T**racking **C**hallenge
DST	**D**ialogue **S**tate **T**racking
ELMo	**E**mbeddings from **L**anguage **M**odels
ERL	**E**uropean **R**obotics **L**eague
FPR	**F**alse **P**ositive **R**ate
GPSR	**G**eneral **P**urpose **S**ervice **R**obots
GPT	**G**enerative **P**re-trained **T**ransformer
GRU	**G**ated **R**ecurrent **U**nits
GloVe	**G**lobal **V**ectors
HRI	**H**uman-**R**obot **I**nteraction
KL	**K**ullback-**L**eiber
LSTM	**L**ong-**S**hort **T**erm **M**emory
MDP	**M**arkov **D**ecision **P**rocess
MIT	**M**assachusetts **I**nstitute of **T**echnology
MLE	**M**aximum **L**ikelihood **E**stimation
MLM	**M**asked **L**anguage **M**odels
NER	**N**amed **E**ntity **R**ecognition
NLG	**N**atural **L**anguage **G**eneration
NLI	**N**atural **L**anguage **I**nference

NLP	**N**atural **L**anguage **P**rocessing
NLU	**N**atural **L**anguage **U**nderstanding
NSP	**N**ext **S**entence **P**rediction
PDF	**P**robability **D**ensity **F**unction
PMF	**P**robability **M**ass **F**unction
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
QA	**Q**uestion **A**nswering
RL	**R**einforcement **L**earning
RNN	**R**ecurrent **N**eural **N**etwork
ROC	**R**eceiver **O**perating **C**haracteristics
ROS	**R**obot **O**perating **S**ystem
SDS	**S**poken **D**ialogue **S**ystem
SGD	**S**tochastic **G**radient **D**escent
SLU	**S**poken **L**anguage **U**nderstanding
SVM	**S**upport **V**ector **M**achines
TPR	**T**rue **P**ositive **R**ate
TTS	**T**ext-**T**o-**S**peech
biLM	**B**idirectional **L**anguage **M**odel

Chapter 1

Introduction

1.1 Motivation

Robotics hold tremendous potential to benefit humans in all aspect of life. These benefits have been widely demonstrated in industrial environments, where automation and robotics have boosted the efficiency and productivity of industries, while reducing the costs in labor and production. Nowadays, due to technological advances, particularly in the field of Artificial Intelligence, autonomous robots are starting to be deployed and integrated into our daily environment. Autonomous robots are artificial agents whose capacities of perception, planning and action in the physical world allow them to autonomously achieve their goals. However, one of their main limitations is the inability to communicate with humans in a natural manner. *Language is the mark of humanity and sentience, and conversation or dialog is the most fundamental and specially privileged arena of language* [1]. Indeed, since we are born, language is one of our most important learnings. Essentially, it is the primary means through which humans have the ability to communicate and interact with one another. Therefore, the integration of robots into our daily lives has triggered the need to enhance Human-Robot Interaction (HRI) with speech and natural language.

In recent years, researchers have been trying to develop systems capable of understanding and speaking in natural language, so that humans can interact with them easily and more naturally. Those systems do not necessary have to be integrated in physical robots. In fact, conversational systems such as virtual personal assistants are increasingly becoming a part of daily life, with examples including Siri, Google Now, Alexa and Cortana [2]. Human-Computer dialogue systems, particularly spoken goal-oriented dialogue systems, have been the most important component in conversational systems, since they allow users to speak naturally in order to accomplish tasks more efficiently. They are widely demanded in network information services, service, domestic and social robots, among others.

When exposed to the public, a spoken dialogue system may encounter a variety of difficulties, such as a large variation in the users of the system (e.g. users usually have different ways of speaking), and high noise environments. Therefore, the dialogue system must be able to use statistical frameworks to handle the uncertainty in both speech recognition and language understanding. Statistical dialogue

systems are able to *maintain uncertainty about everything the user has said, and exploit the sequential nature of dialogue to disambiguate in the presence of errors* [3].

1.2 Objectives

This thesis aims at developing a spoken goal-oriented dialogue system capable of dealing with the uncertainty in speech recognition. A goal-oriented dialogue system can be defined as a computer system able to interact with humans on a task-oriented context. The main objectives of this work can be summarized as:

1. Develop a spoken goal-oriented dialogue system.
   (a) The developed dialogue system must be able to deal with the uncertainty in the speech recognition.
   (b) The dialogue system must be capable to use the sequential nature of the dialogue to disambiguate in the presence of errors.
2. Implement the dialogue system in a service robot and test the complete system.

1.3 Approach

Briefly, once a spoken dialogue system receives an audio signal, this signal is processed by a speech recognizer to give a probabilistic distribution over words [3]. Then, the dialogue system attempts to understand these words and how they affect the current state of the dialogue. Finally, it generates the system responses based on the dialogue state. Typically, this task is achieved by three different components: the Natural Language Understanding (NLU) extracts semantic information from the hypothesis recognized by the speech recognition; the Dialogue State Tracker (DST) takes this semantic information and computes the new state of the dialogue; the Dialogue Manager (DM) uses the dialogue state to generate the system responses.

Although a traditional spoken dialogue system is composed of more components, we consider that designing only the previous three components, together with an Automatic Speech Recognizer (ASR), provides a baseline framework for developing more complex dialogue systems.

In addition, we need a dialogue system capable of dealing with the uncertainty in speech recognition. Thus, the developed ASR must be able to recognize multiple hypothesis, and the DST must be capable of tracking multiple hypothesis and maintain beliefs over dialogue states.

Furthermore, the dialogue system must be able to use the sequential nature of dialogue to disambiguate in the presence of errors. Therefore, the developed DM must be capable of generating system responses for requesting more information from the users, or asking for clarifications.

Finally, we implement the dialogue system in a service robot and test the complete system on a real world environment.

1.4 Thesis Outline

To answer all challenges stated in this chapter, the structure adopted in this work follows the scheme presented in Figure 1.1.

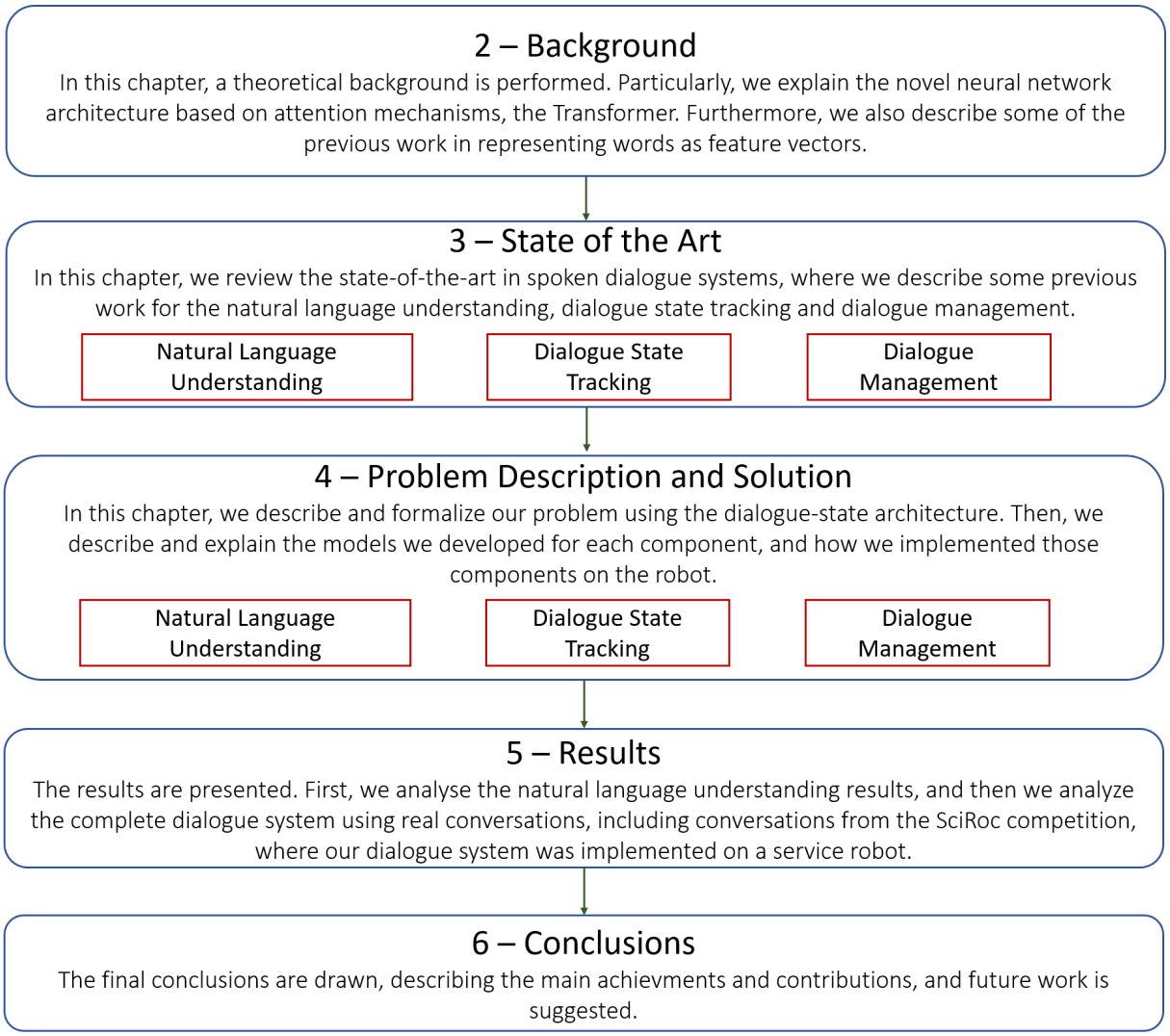


Figure 1.1: Thesis outline diagram.

Chapter 2

Background

This chapter gives an overview of some relevant background theory. It starts with a description about some probability and information theory concepts. In the following sections, there are described some machine learning principles, including artificial neural networks. Finally, it is explained how words can be represented, such that computers are able to represent and understand their semantics. If the reader is already familiar with the previous concepts, feel free to jump to the next chapter. However, we strongly recommend the reading of the following sections: Transformer (2.3.2) and Vector Representation of Words (2.4).

2.1 Probability and Information Theory

Probability theory is a mathematical framework for representing uncertain events [4]. Although there are many definitions regarding probability theory, all of them share the idea that, to be able to reason in the presence of uncertainty, we need to both represent uncertain statements and quantify their uncertainty, as well as use a set of axioms to derive new uncertain statements. For example, consider a robot in a non deterministic environment, with known location. Even though the initial location is exactly known, when the robot performs a certain action, it may lead to different outcomes (due to its non deterministic environment). How can the robot be able to reason and determine its location in such uncertainty? First, we need to represent the robot's location using a probability distribution over all the possible locations, commonly called belief. This allows us to not only represent the location of the robot, but also quantify the uncertainty of a specific location (the probabilities represent the degree of belief, with 1 indicating absolute certainty that the robot is in that position and 0 indicating absolute certainty that the robot is not in that position). Imagine now that the robot moves forward. We need to be able to update the probability distribution that represents the robot's location, taking into account the action performed. The previous example demonstrates a practical application of probability theory applied to a specific robotics problem.

However, *in fact, nearly all activities require some ability to reason in the presence of uncertainty* [4]. Goodfellow et al. justifies this affirmation by enumerating three possible sources of uncertainty: the inherent stochasticity of a system being modeled, the incomplete observability of a environment, and

the incomplete modeling of a system.

Despite steady progress over the last few decades in speech recognition technology, the process of converting conversational speech into words still incurs word error rates in the range $15\%\tilde{}30\%$ *in many real-world operating environments* [5]. Therefore, conversation systems, which interpret spoken commands, should be able to deal with the uncertainty of the input observations, and hence must require the ability to reason in the presence of such uncertainties.

2.1.1 Information Theory

Information theory is a branch of applied mathematics that revolves around quantifying how much information is present in a signal [4]. It was originally proposed to find fundamental limits on signal processing and communication operations [6].

Self-information

The basic intuition behind information theory is that the occurrence of an unlikely event represents more information than the occurrence of a likely event [4]. To formalize this intuition, we quantify the information in such a way that: likely events should have low information contents, and, in contrast, less likely events should have higher information content [4]. Therefore, a fundamental measure of information theory is *self-information*, which allows us to determine the information content of an event $x \in X$ [7]:

$$I(x) = -\log P(x) \tag{2.1}$$

where $\log$ is the natural logarithm, with base e. Therefore, the units of *self-information* is *nats*, which means that one *nat* is the amount of information gained by observing an event with probability $\frac{1}{e}$. By analyzing the Equation 2.1, we can easily verify that an event with high probability will hold a low information content, while an event with low probability will hold an higher information content.

Entropy

Another key quantity of information theory is *entropy*, which represents the expected amount of information in an event $x \in X$ drawn from a probability distribution $P(X)$ [7]:

$$H(x) = \mathbb{E}_{x\sim P}[I(x)] = -\mathbb{E}_{x\sim P}[\log P(x)] \tag{2.2}$$

In other words, the *entropy* quantifies the amount of uncertainty in an entire probability distribution.

Kullback-Leibler Divergence

If we extend the notion of entropy to two probability distributions, $P(x)$ and $Q(x)$, and measure the relative entropy of $P(x)$ with respect to $Q(x)$, we can calculate the *information gain* of one probability with respect to the other. This measure is known as *Kullback-Leibler divergence* (KL divergence), and

can be defined as [7]:

$$\begin{aligned} D_{KL}(P||Q) &= \mathbb{E}_{x\sim P}[\log P(x)] - \mathbb{E}_{x\sim P}[\log Q(x)] \\ &= \overbrace{\mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]}^{\text{linearity of expectation}} \\ &= \mathbb{E}_{x\sim P} \log \left[\frac{P(x)}{Q(x)}\right] \end{aligned} \tag{2.3}$$

The KL divergence between two distribution is always non-negative, or zero if the two distributions are equal.

Cross-Entropy

Another quantity that is closely related to the KL divergence is the *cross-entropy*, which can be defined as follows [7]:

$$H(P;Q) = -\mathbb{E}_{x\sim P}[\log Q(x)] \tag{2.4}$$

We can manipulate the previous equation, by adding and subtracting $\mathbb{E}_{x\sim P}[\log P(x)]$:

$$\begin{aligned} H(P;Q) &= -\mathbb{E}_{x\sim P}[\log P(x)] + \mathbb{E}_{x\sim P}[\log P(x)] - \mathbb{E}_{x\sim P}[\log Q(x)] \\ &= \underbrace{-\mathbb{E}_{x\sim P}[\log P(x)]}_{H(P)} + \underbrace{\mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]}_{D_{KL}(P||Q)} \end{aligned} \tag{2.5}$$

where the term $-\mathbb{E}_{x\sim P}[\log P(x)]$ represents the entropy of the probability distribution $P(x)$ and the term $\mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$ represents the KL divergence of $P(x)$ with respect to $Q(x)$. Thus, we can represent the *cross-entropy* as [7]:

$$H(P;Q) = H(P) + D_{KL}(P||Q) \tag{2.6}$$

2.2 Machine Learning

In this section we will introduce some fundamental general machine learning concepts. A machine learning algorithm is an algorithm that is able to learn from data [4]. Traditionally, when developing a computer program to perform a specific task, the developer would need to first acquire some knowledge about the task, and then develop all the logic and specific instructions for the program to be able to perform the task. In machine learning, the developer designs the program as a mathematical model capable of inferring the task knowledge from data.

2.2.1 Maximum Likelihood Estimation

The most common way to design a machine learning algorithm is to use the principle of *maximum likelihood estimation* (MLE). The MLE model can be defined as a function $p_{model}(\boldsymbol{x};\theta)$ that maps an input $\boldsymbol{x}$ to a probability using a set of parameters θ [7]. Thus, the objective of MLE is to learn the

parameters θ that best approximate the probability of our model $p_{model}(\boldsymbol{x};\theta)$ to the real data distribution, $p_{data}(\boldsymbol{x})$. *In other words, MLE seeks to maximize the likelihood of the data under the configuration of the model* [7], and can be defined as:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\, p_{model}(\boldsymbol{X};\theta) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} p_{model}(\boldsymbol{x_i};\theta) \tag{2.7}$$

Since usually many of the probabilities can take on values close to 0, and for a large number of samples n, the previous product of probabilities can lead to underflow, which results in a less precise estimation of the model. This can be solved by using log-probabilities, replacing the product operation with a sum [7]:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log p_{model}(\boldsymbol{x_i};\theta) \tag{2.8}$$

We can divide the previous equation by the total number of samples, n, to obtain an expectation with respect to the distribution of the data $p_{data}(\boldsymbol{x})$ [7]:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}} \frac{1}{n} \sum_{i=1}^{n} \log p_{model}(\boldsymbol{x_i};\theta) = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{x\sim p_{data}}[\log p_{model}(\boldsymbol{x_i};\theta)] \tag{2.9}$$

Rather than maximizing the likelihood of the data, MLE may also be seen as minimizing the dissimilarity between the data distribution, p_{data}, and the model distribution p_{model}, which is measured by the KL divergence [7]:

$$D_{KL}(p_{data}||p_{model}) = \mathbb{E}_{x\sim p_{data}}[\log p_{data}(x) - \log p_{model}(\boldsymbol{x_i};\theta)] \tag{2.10}$$

If we look at the previous equation, we note that the left term, $\log p_{data}(x)$, is only a function of the data, and hence, minimizing the KL divergence can only be achieved by minimizing the right term, $p_{model}(\boldsymbol{x_i};\theta)$, since it is the only term dependent on the model parameters. Minimizing a negative term is just the same as maximizing the term, so this objective is the same as the MLE objective:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmin}} \underbrace{-\mathbb{E}_{x\sim p_{data}}[\log p_{model}(\boldsymbol{x_i};\theta)]}_{H(p_{data};p_{model})} \tag{2.11}$$

Furthermore, this objective is also the same as minimizing the cross-entropy between the data distribution, p_{data}, and the model distribution, p_{model}:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmin}}\, H(p_{data};p_{model}) \tag{2.12}$$

Cross-entropy is a common loss in machine learning and the objective function that is most commonly used in neural networks.

2.2.2 Gradient Descent

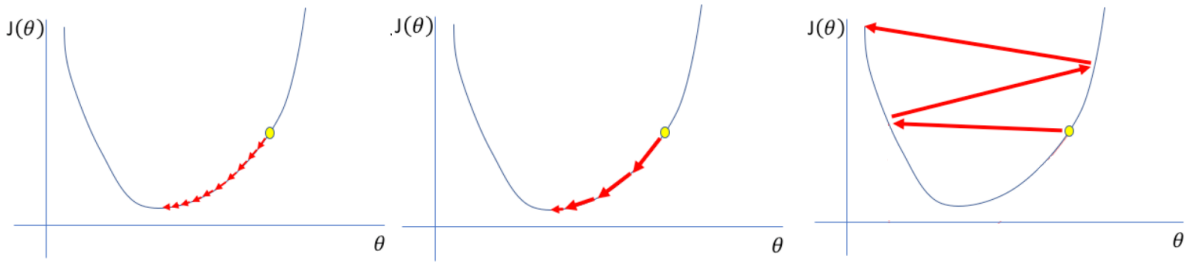
Gradient descent is an efficient iterative optimization method for minimizing an objective function $J(\theta)$. Essentially, gradient descent updates the parameters θ in the opposite direction of the gradient $\Delta_\theta J(\theta)$ of the function $J(\theta)$. These updates are done using Equation 2.13:

$$\theta = \theta - \eta \cdot \Delta_\theta J(\theta) \tag{2.13}$$

where η is the learning rate that determines the magnitude of the updates. Particularly, each parameter θ_i is updated by its partial derivative with respect to $J(\theta)$, $\frac{\partial}{\partial \theta_i} J(\theta)$, which is the i-th element of the gradient:

$$\theta_i = \theta_i - \eta \cdot \frac{\partial}{\partial \theta_i} J(\theta) \tag{2.14}$$

A good choice for the learning rate hyper-parameter represents one of the most important settings when training a model (see Figure 2.1), because it controls the convergence of the algorithm.



(a): Too low. **(b):** Just right. **(c):** Too high.

Figure 2.1: Effect of the learning rate on the gradient descent optimization. A small learning rate (left) requires many updates before reaching the minimum. The "optimal" learning rate (midle) swiftly reaches the minimum point. A learning rate too large (right) causes drastic updates that lead to divergent behaviours.

2.3 Artificial Neural Networks

Neural networks have become the tool choice of natural language processing in recent years [7]. In this section, we will give an overview of some fundamental concepts about neural networks. Neural networks can be seen as a composition of linear functions, interleaved with non-linear functions, that map an input vector of features $\boldsymbol{x}$ to an output vector $\boldsymbol{y}$. Traditionally, neural networks have more than one layer. The last layer of the network is usually called *output layer*, while non-output layers are referred to as *hidden layers*. The simplest neural network model is the *multilayer perceptron* (MLP), which consists of only one hidden layer and the output layer:

$$\begin{aligned} \boldsymbol{h} &= \sigma_1(\boldsymbol{W_1 x} + \boldsymbol{b_1}) \\ \boldsymbol{y} &= \text{softmax}(\boldsymbol{W_2 h} + \boldsymbol{b_2}) \end{aligned} \tag{2.15}$$

where $\boldsymbol{W}_1$, $\boldsymbol{b}_1$ and $\boldsymbol{W}_2$, $\boldsymbol{b}_2$ are the weights and biases of the first hidden layer and output layer, respectively, σ_1 is the activation function of the first hidden layer, $\boldsymbol{h}$ is the hidden state of the network, and

softmax$(\cdot)$ is the activation function of the output layer, defined by:

$$f(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\sum_{i=0}^{N} e^{z_i}} \tag{2.16}$$

Usually, most of the models use neural networks with more than one hidden layer. As stated before, a neural network can be seen as a sequence of linear and non-linear transformations. Since linear functions are not able to deal with non-linear problems, the expressiveness of neural networks mainly comes from its non-linear activation functions [7]. There are several types of activation functions. Traditionally, the output layer of a neural network uses either a *softmax* or a *sigmoid* activation functions. Furthermore, two more activation functions, *hyperbolic tangent* and *rectified linear unit*, are widely used in neural networks, particularly in the hidden layers. Nwankpa et al. performs a survey on the existing activations functions used in deep learning applications [8].

2.3.1 Training Neural Networks

As stated before, neural networks apply a sequence of linear and non-linear transformations along the hidden layers. Intuitively, when a neural network is trained on a specific dataset, it learns how to "transform" the input features so that, in the final hidden layer, those features are represented in such way that the output layer can easily use the new feature representation to perform the required task. In order to perform this *representation learning*, the neural network must be able to learn from the training data.

Back-propagation

Training a neural network consists in minimizing the loss function defined. This minimization can be achieved by using gradient descent. As described in Section 2.2.2, gradient descent techniques use the gradient of an objective function to update the parameters of a model. As neural networks consist of multiple layers, calculating the gradient of the loss function with regard to the parameters in non-trivial. To achieve this, we use a dynamic programming algorithm known as *back-propagation*. Back-propagation relies on the *chain rule* to compute the derivatives of the loss function with regard to the parameters. It starts by calculating the derivative of the loss function with regard to the output and each of the hidden layers, until it reaches the input layer. Then, after computing all the derivatives, the parameters of the network are updated.

Stochastic Gradient Descent

The typical gradient descent technique used in neural networks for updating the parameters is the Stochastic Gradient Descent (SGD). In SGS, the parameters are updated for each training sample.

$$\theta = \theta - \eta \cdot \Delta_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.17}$$

where $x^{(i)}$ and $y^{(i)}$ define the input feature vector and label, respectively, for the i-th training sample.

Adam

Adaptive Moment Estimation, or Adam, *is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments* [9]. Essentially, it is a modified stochastic gradient descent method that computes adaptive learning rates. The algorithm leverages the power of adaptive learning rates methods to find individual learning rates for each parameter. As the name suggests, Adam uses estimations of first and second order moments of the gradient to adapt the learning rate for each weight of the neural network.

2.3.2 The Transformer

Recurrent neural networks (RNNs) [10], in particular Long-Short Term Memory networks (LSTMs) [11] and bidirectional LSTMs (BiLSTMs) [12], have been the typical network architecture for processing language sequentially. Reading one word at a time, this forces RNNs to perform multiple steps to make decisions that depend on words far away from each other. In addition, this sequential nature of RNNs also makes it more difficult to fully take advantage of modern hardware.

In 2017, Google researchers proposed a novel network architecture called the Transformer, a model based solely on attention mechanisms, dispensing with recurrence and convolutions [13].

Like many SEQ2SEQ models, the transformer's baseline structure relies on an encoding and decoding components. The encoding component is a stack of 6 encoders, each one broken down into two sub-layers. *The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network* [13]. The decoder component is a stack of 6 decoders, which have the same two sub-layers as the encoders. In addition, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack [13]. A residual connection followed by a layer-normalization is employed around each of the sub-layers for both the encoders and decoders, which help training deep neural networks. The Transformer architecture can be seen of Figure 2.2.

Self-Attention

One of the core components of the Transformer is the self-attention, which enables the model to produce contextualized representations of the inputs. *An attention function can be described as mapping a query and a set of key-value pairs to an output* [13]. The first step in calculating self-attention is to create three vectors from each of the encoder's input vectors. Therefore, for each input x_i, we create the query, $\boldsymbol{q}_i$, the key, $\boldsymbol{k}_i$ and the value ,$\boldsymbol{v}_i$, vectors, by multiplying the input vector by three matrices, $\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v$, that we trained during the training process.

$$\begin{aligned} \boldsymbol{q}_i &= \boldsymbol{W}^q x_i \\ \boldsymbol{k}_i &= \boldsymbol{W}^k x_i \\ \boldsymbol{v}_i &= \boldsymbol{W}^v x_i \end{aligned} \tag{2.18}$$

The second step in calculating self-attention is to calculate a score that reflects how each input

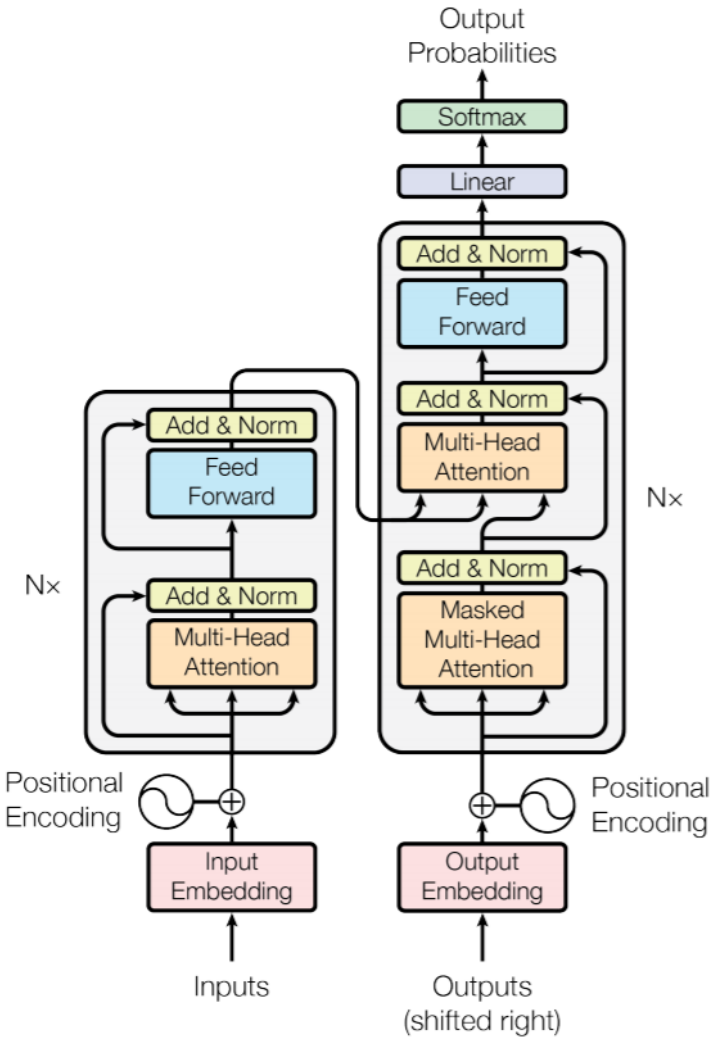


Figure 2.2: Transformer [13].

attends to the others. Thus, for each input x_i, we compute the score $s_{i,j}$ which is the dot product between its query vector $\boldsymbol{q}_i$ and the key vector $\boldsymbol{k}_j$ of all the other inputs, including itself.

$$s_{i,j} = \boldsymbol{q}_i \cdot \boldsymbol{k}_j \tag{2.19}$$

The third and forth steps are to divide the scores by the square root of the dimension of the key vectors, $\sqrt{d_k}$, which leads to more stable gradients, and then pass the result through a softmax operation to normalize the scores.

$$\overline{s}_{i,j} = softmax(\frac{s_{i,j}}{\sqrt{d_k}}) \tag{2.20}$$

Finally, we multiply each value vector by the softmax score and sum up the weighted value vectors to produce the output of the self-attention layer z_i, which corresponds to a contextualized representation of the input vector x_i.

$$z_i = \sum_{j=1}^{L} \overline{s}_{i,j} \boldsymbol{v}_j \tag{2.21}$$

where L is the input length.

Although the vector representation is easier for understanding the self-attention mechanism, the attention function is computed on a set of queries simultaneously (for all inputs), packed together into a matrix $\boldsymbol{Q}$. The keys and values are also packed together into matrices $\boldsymbol{K}$ and $\boldsymbol{V}$. Thus, the matrix of outputs is computed as [13]:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V} \tag{2.22}$$

Multi-Head Attention

In reality, instead of performing a single attention function, the Transformer uses a mechanism called *multi-head attention*, which *allows the model to jointly attend to information from different representation subspaces at different positions* [13]. Each attention head will compute an output value for the input token. Then, the multiple outputs from the different attention heads are concatenated and again projected, resulting in the final output value. Equation 2.23 describes the computations using multi-head attention [13].

$$\begin{aligned} \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)\boldsymbol{W}^O \\ \text{where head}_i &= \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \end{aligned} \tag{2.23}$$

where $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$ and $\boldsymbol{W}_i^V$ are the query, key and value matrices correspondent to the ith attention head, and $\boldsymbol{W}^O$ are the parameters of the final projection layer. Figure 2.3 shows the computational graphs of both the dot-product and multi-head attention.

The transformer uses multi-head attention in self-attention layers, both in the encoder and decoder components. The self-attention layers of the encoder component enable the model to produce contextualized representations of the inputs. In addition, the self-attention layers of the decoder component allow each position in the decoder to attend to what has already been decoded up to, and including, that position. Finally, the multi-head attention is also used in encoder-decoder attention layers, to allow the decoder to attend over all position in the input sequence. Intuitively, the Transformer uses the encoder component to extract contextualized representations of the input sequence. Then, it uses the decoder component to decode the input sequence, attending to both the contextualized representation of the input sequence, and also the contextualized representation of what has already been decoded.

Positional Embeddings

The transformer does not contain any recurrence or convolution. Therefore, we need a way for the model to make use of the order of the input sequence, and hence we must inject some information about the

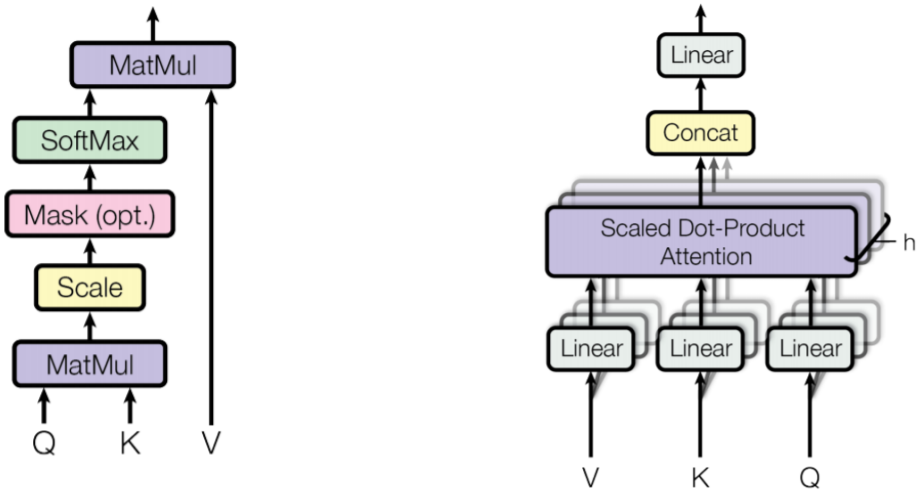


Figure 2.3: In the left, it is represented the scaled dot-product attention. In the right, the multi-head attention, which consists of several attention layers running in parallel [13].

relative or absolute position of the tokens in the sequence. The solution is to fuse positional embeddings with the input embeddings. Since the positional embeddings have the same dimension as the input embeddings, this fusion can be achieved by adding them. The positional embeddings make use of sinusoidal functions of different frequencies that allow the model to easily attend by relative positions. A main advantage of using sinusoidal signals (i.e. relative position) is the possibility to extrapolate to sequence lengths longer than the ones seen during training.

2.4 Vector Representation of Words

The majority of image and audio processing systems work with rich, dense, high-dimensional datasets encoded as vectors (e.g. individual raw pixels or power spectral density coefficients for image and audio data, respectively). However, in natural language processing, the datasets are traditionally composed of words (or characters) which are treated as unique and discrete symbols. This representation provides no useful information to the system regarding the relationship between words, and leads to data sparsity, since the words' vocabulary is usually very large, whatever the language is. Therefore, more data may be needed to successfully train statistical models. The solution is to use high-dimensional vectors as features to represent words.

A long tradition in computational linguistics has shown that textual information provides a good approximation to word meaning, since semantically similar words tend to have similar contextual distributions [14]. Distributional Semantics is a research area that develops and studies methods for quantifying and categorizing semantic similarities between linguistic items, based on their distributional properties in large samples of language data. Traditionally, these methods use distributional semantic models, which state that the meaning of a word can be inferred from its distribution in text. These models dynamically

build semantic representations in the form of high-dimensional vectors through a statistical analysis of the contexts in which words occur, and apply geometric techniques to these vectors to measure the similarity in meaning of the corresponding words [15].

Count-based vs. Predictive Methods

Methods based on distributional semantic models can be split into two categories: count-based and predictive methods (also called neural models or embeddings). The former compute the statistics of how often some word co-occurs with its neighbor words in a large text corpus, and then map these count-statistics down to a small, dense vector for each word. The predictive methods directly try to predict a word from its neighbor words using neural models, whose parameters are used as learned small, dense, embedding vectors.

Baroni et al. [15] has shown that predict-based models outperform the count-based models by testing multiple methods based on both approaches on a variety of benchmark tasks (semantic relatedness, synonym detection, concept categorization, selectional preferences and analogy), and using a corpus composed of 2.8 billion tokens constructed by concatenating ukWaC, the English Wikipedia, and the British National Corpus.

One of the first neural models was introduced by Bengio et al. [16], which consists of a feed-forward neural network, with a linear projection layer and a non-linear hidden layer, that predicts the next word given a sequence of previous words. Essentially, the idea of this approach is to associate to each word a distributed feature vector, then express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and, finally learn simultaneously the word feature vectors and the parameters of that probability function.

WOR2VEC

Following the previous work, Mikolov et al. [17] proposed a new model for learning distributed representations of words that try to minimize the computational complexity, named WORD2VEC. For this model, two different architectures were purposed, the Continuous Bag-of-Words (CBOW) and Skip-gram. Both architectures are similar to the previous described feedforward neural network, but with the non-linear hidden layer removed, since it is the major cause of the computational complexity [17]. In addition, the projection layer is shared for all words, i.e. the input word feature vectors are averaged in the projection layer, and therefore the order of words in the history does not influence this projection (hence the name bag-of-words). The two models are represented in Figures 2.4 and 2.5.

The major difference between them is that the CBOW predicts the current word based on the context, while the skip-gram predicts the context given the current word. Although simpler, they are able to compute very accurate high-dimensional word vectors, since they can be trained on a much larger dataset [17]. Following their previous work, Mikolov et al. presented some extensions and techniques that improve both the quality of the vectors and the training speed, such as the subsampling of frequent words, the hierarchical softmax, and negative sampling [18]. In this work, they also demonstrate that the

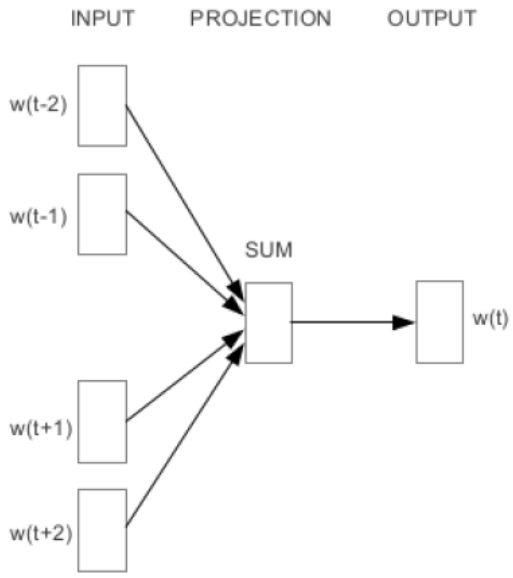


Figure 2.4: Continuous Bag-of-Words (CBOW) model. [17]

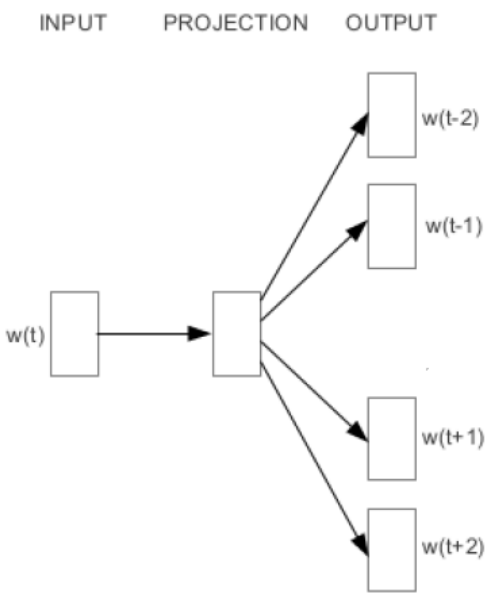


Figure 2.5: Continuous Skip-gram model. [17]

learned representations of words and phrases exhibit a linear structure that makes precise analogical reasoning possible using simple vector arithmetics. For example, we can use the learned vector representations to tell if words are similar, or opposites, or that a pair of words like "Stockholm" and "Sweden" have the same relationship between them as "Cairo" and "Egypt" have between them.

GloVe

Previous work have shown that predict-based methods outperform count-based methods across a range of tasks [15]. However, Pennington et al. argue that *both classes of methods are not dramatically different at a fundamental level, since they both exploit the underlying co-occurrence statistics of the corpus* [19]. Nevertheless, the efficiency with which the count-based methods capture global statistics can be advantageous, and therefore they developed a model that uses the major benefit of counting data while simultaneously capturing the meaningful linear structures prevalent in prediction-based models. In contrast with the previous WORD2VEC model, GloVe (Global Vectors) directly uses the words co-occurrence statistics to capture the global statistics of the corpus. Thus, since it is able to capture both global and local statistics of the corpus, it performs better than previous models on word analogy, word similarity and named entity recognition tasks [19].

ELMo

Although powerful, WORD2VEC and GloVe still hold some limitations. For each word, there is associated one and only one feature vector, no-matter what is the context in which this word appears. To solve this problem, Peters et al. presented the first *contextualized word-embeddings*, ELMo (Embeddings from Language Models) [20]. Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning a representation for each word. *Those representations are learned functions*

of the internal state of a deep bidirectional language model (biLM) pre-trained on a large text corpus[20].

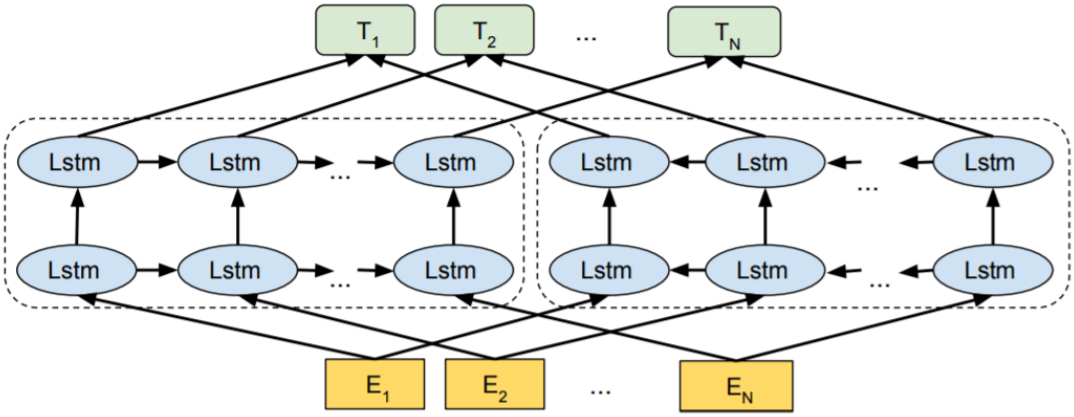


Figure 2.6: ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. [21]

ELMo uses a bidirectional LSTM (see Figure 2.6) trained to predict the next word in a sequence of words, a task called language modeling. The contextual representation of each word is the concatenation of the left-to-right and right-to-left representations obtained from each LSTM. Therefore, it extracts context-sensitive features from a left-to-right and a right-to-left language model. ELMo provided a significant step torwards pre-training in the context of NLP, since it can be trained on a massive text corpus, and then used as a component in other models that need to handle language.

OpenAI GPT

The release of the Transformer (see Section 2.3.2), and the results it achieved on tasks such as machine translation, triggered their use as replacement to LSTMs. In addition to dealing better with long-term dependencies, their architecture is better suited for taking advantage of modern hardware.

In 2018, Radford et al. [22] introduced the Generative Pre-Trained Transformer (GPT), which achieved strong natural language understanding through generative pre-training and discriminative fine-tuning. This model uses the Transformer decoders, slightly modified, and is trained on a language modeling task, using a very large unlabeled text corpus (an unsupervised pre-training procedure). Then, the model can be adapted to a supervised task, and fine-tuned on the labeled data. In contrast with ELMo, which is a feature-based approach (i.e. it is used as feature extractor) , the OpenAI GPT is a fine-tuning approach, since the model is pre-trained on a language modeling task and then the model parameters are fine-tuned on a specific discriminative task.

BERT

Although the OpenAI GPT can increase performance in discriminative tasks using a pre-trained model, it is only able to extract features based on left context. ELMo has already demonstrated that using both left and right context can yield better word representations. Thus, in 2019, Devlin et al. introduced a new language representation model called BERT (Bidirectional Encoder Representations from Transformers)

[21]. Both the OpenAI GPT and BERT architectures are shown is Figures 2.7 and 2.8, respectively.

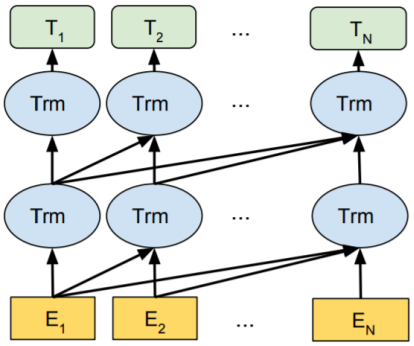


Figure 2.7: OpenAI GPT uses a left-to-right Transformer for learning token representations. [21]

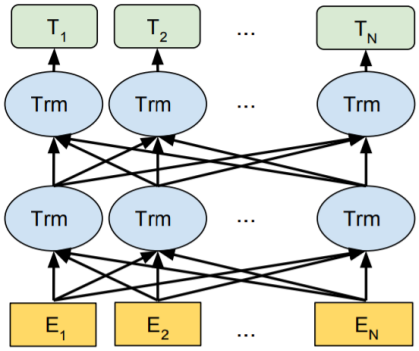


Figure 2.8: BERT uses a bidirectional Transformer for learning token representations. [21]

In contrast with the other language models, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. This is achieved by using the Transformer encoders, instead of using the decoders like OpenAI GPT. However, the Transformer encoders would allow each word to indirectly see itself in a multi-layered context. Therefore, they developed a new training objective, called the masked language model (MLM). The MLM randomly masks some of the input tokens, and the objective is to predict the masked token based on both left and right context [21]. In addition to the masked language model, they also introduced a next sentence prediction task, to train text-pair representations, which helps the model boosting performance in question answering (QA) and natural language inference (NLI) tasks [21]. In summary, BERT has advanced the state-of-the-art for eleven NLP tasks, and hence may represent one of the most important breakthroughs in the NLP field. Figure 2.9 shows a simple representation of the BERT-BASE model[1].

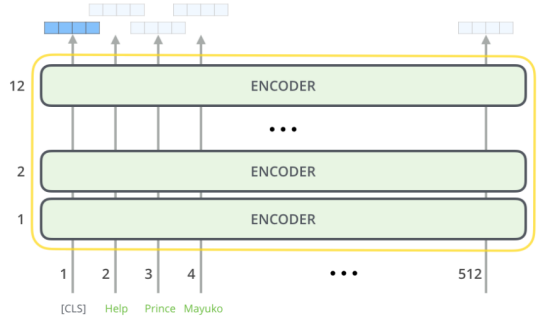


Figure 2.9: BERT-base model.

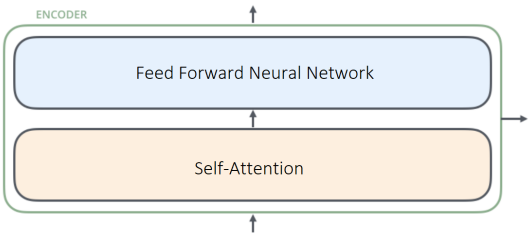


Figure 2.10: Encoder sub-layers.

The model architecture of BERT is a multi-layer bidirectional Transformer encoder [23]. Particularly, it consists of 12 encoder layers, each one containing two sub-layers: self-attention and fully connected feed-forward layers (see Figure 2.10). Actually, these encoder layers are very similar to those on the Transformer, presented in Section 2.3.2. The only difference is that, while the Transformer uses vectors with a size of 512 and 8 self-attention heads, BERT (BASE version) uses vectors with a size of 768 and the self-attention is composed of 12 heads. BERT takes the input representation of words as a

[1] Two versions of BERT were released, the BERT-BASE and BERT-LARGE.

concatenation of WordPiece embeddings [24] with 30,000 token vocabulary, positional embeddings with supported sequence length up to 512 tokens, and the segment embedding. In addition, a special token, representing a classification embedding, denoted [CLS] (for classification), is inserted as the first token in the input sentence sequence, which enables the model to provide a context-dependent representation of the full input sentence, $\boldsymbol{h}_{[\text{CLS}]}$.

Transfer Learning

Traditionally, there are two main paradigms for the adaptation of pre-trained models: *feature extraction* and *fine-tuning* [7]. In feature extraction, the pre-trained model's parameters are frozen and the features are used in the downstream model, similar to classic feature-based approaches. Alternatively, the pre-trained model's parameters can be fine-tuned on a new task [7]. Although feature extraction may be computationally cheaper (as features only need to be computed once), fine-tuning a pre-trained model allows us to adapt a general-purpose representation to many different tasks [7].

Chapter 3

State of the Art

This chapter describes the state of the art in dialogue systems. First, we give a brief overview of spoken dialogue systems theory, particularly their main types, strategies and architecture. In the following sections, we give a deeper description of some previous work related to Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), Dialogue State Tracking (DST) and Dialogue Management (DM).

3.1 Spoken Dialogue Systems

A spoken dialogue system can be defined as a computer system able to interact with humans on a turn-by-turn basic, and in which spoken natural language interface plays an important part in the communication [25]. Traditionally, they can be split into two classes: the goal-driven or non-goal-driven dialogue systems. The former represent the interaction between a human and a computer in a task-oriented context, while the latter, usually called chatbots, are related to chat communication.

First research in task-oriented dialogue systems began in the early 1990s, when MIT (Massachusetts Institute of Technology) developed an automatic flight booking system with the support of DARPA (Defense Advanced Research Projects Agency) [26]. Similar dialogue systems were developed in the following decade: HMIHY (How May I help You?), a spoken dialogue system based on call routing [27], JUPITER, a conversational interface that allows users to obtain worldwide weather information over the telephone using spoken dialogue [28], and a travel plan making system DARPA communicator [29].

Non-goal-driven dialogue systems usually respond to user utterances without any specific goal. ELIZA [30] might be the first chatbot, and one of the first programs capable of attempting the Turing test. Created in MIT Artificial Intelligence Laboratory, ELIZA simulated conversation by using a pattern matching and substitution methodology that gave users an illusion of understanding on the part of the program, but had no built in framework for contextualizing events [31].

Although there is a discrimination between goal-driven and non-goal-driven dialogue systems, its border is not strict. When we think of human interaction in any task-oriented scenario (e.g. flight booking), it is natural for any customer to chat with the human service staff while performing the required

task. Thus, a good dialogue system should be able to chat with users while helping them performing a certain task or achieving a specific goal [32]. Recently, some personnel assistant systems have been developed, such as Siri, Cortana, Google Now/Home, Alexa, among others. These systems are able to, not only perform certain requested tasks, but also maintain chat conversations with users.

Furthermore, the dialogue systems can also be implemented in physical systems, such as service and social robots. A service robot is a robot whose goal is to help humans, by giving them assistance in a wide range of tasks [33], while social robots are more concerned with social interactions and relationships with humans. Both type of robots require some form of human-robot interaction, and therefore make use of dialogue systems to enhance this interaction. One example of a social robot in which the human-robot interaction abilities are fundamental is *Gasparzinho* or MOnarCH, Multi-Robot Cognitive Systems Operating in Hospitals [34]. *Gasparzinho* interacts and plays with hospitalized children, acting as a robot companion. An identical robot, the *mbot*, is used by a robotics team from Instituto Superior Técnico, SocRob@Home, for scientific competitions. However, unlike *Gasparzinho*, *mbot* is a service robot that must perform a variety of domestic tasks.

One can also classify dialogue systems based on their dialogue strategy. A dialogue system can implement three different strategies: *system-initiative*, *user-initiative*, or *mixed-initiative*. In a system-initiative dialogue, the system has the full control of the dialogue flow by asking all the questions, and the user is only able to answer. On the other hand, in a user-initiative system the user makes all the requests and the system only answers. Finally, a mixed-initiative dialogue combines the advantages of the former two strategies. This combined strategy allows the user to take over control of the dialogue by asking questions or by giving more information than has been asked for, while, at the same time, enables the system to make proposals and help the user to reach his goal.

The majority of modern task-based dialogue systems make use of the frame-based architecture, first introduced in the GUS system for travel planing [35]. In a frame-based architecture, the system uses a domain ontology, a knowledge structure representing the kinds of information the system can extract from user utterances [1]. The ontology not only defines one or more frames, each a collection of slots, but also the possible values each slot can take. Briefly, a frame-based architecture tries to fill the required slots for a specific task during the conversation.

However, frame-based dialogue systems are not capable of asking questions, giving orders, or making informational statements [1]. For example, if the system cannot understand what a user said, it may need to ask clarification questions. The dialogue-state architecture, also called information-state architecture, like the GUS systems, is based on filling in the slots of frames. However, the dialogue-state architecture has a different way of deciding what to say next than the GUS systems. Simple frame-based systems often just continuously ask questions corresponding to unfilled slots, while the dialogue-state based systems use a dialogue policy to decide what to say.

The typical dialogue-state architecture consists of the following components, arranged in a pipeline: Automatic Speech Recognizer (ASR), Natural Language Understanding (NLU), Dialogue State Tracking (DST), Dialogue Management (DM), Natural Language Generation (NLG), and Text-to-Speech (TTS). First, the computer needs to convert speech into text and then extract task related information from user

utterances, which is done by the ASR and NLU, respectively. Second, the computer must be able to control the process of a dialogue. Initially, the DST estimates the state of the conversation, and then the DM generates actions based on the dialogue state. Finally, the computer needs to convert the system output actions to natural language sentences, and then to speech, which is the objective of NLG and TTS. This architecture is represented in Figure 3.1.

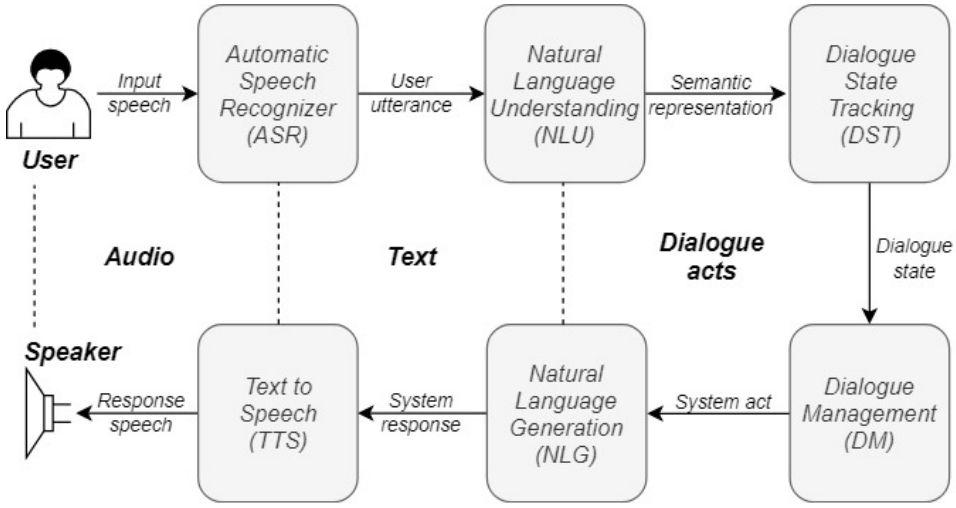


Figure 3.1: Dialogue-state architecture for dialogue systems.

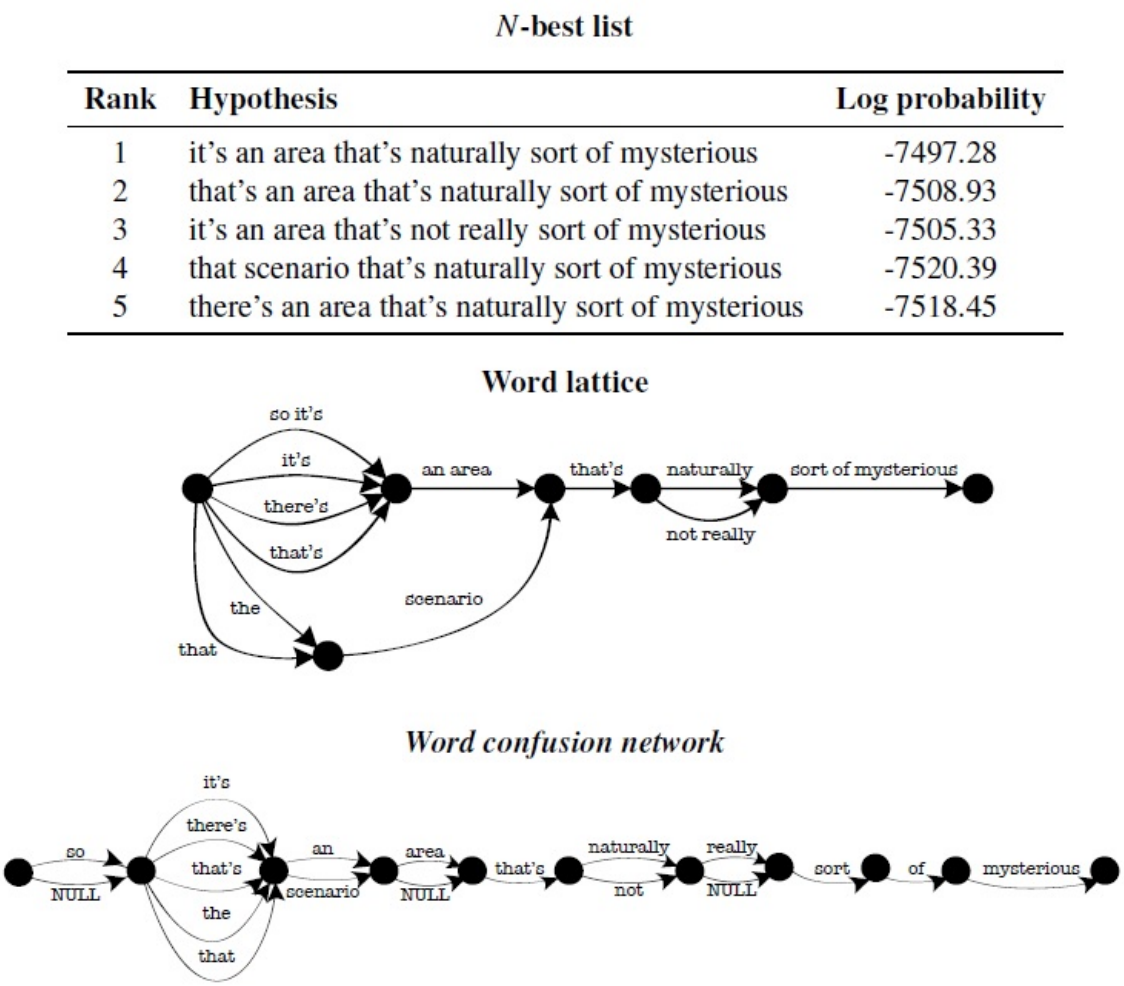
3.2 Automatic Speech Recognition

Automatic Speech Recognition (ASR) enables the recognition and translation of speech (audio) into text by computers and it is the first component in a dialogue system's pipeline. Therefore, ASR takes a key role in the full pipeline system because the errors accumulated in this process will be propagated to downstream tasks. Nevertheless, this thesis do not focus on the ASR module itself. It is rather concerned with understanding its output, since it will be used by the downstream systems such as Natural Language Understanding (NLU) or/and Dialogue State Tracking (DST).

Essentially, the ASR component assigns a posterior probability to the words of an utterance given its acoustics [3]. A typical form of the ASR output is a *N-best list* of hypothesis with corresponding probabilities or confidence scores. Although simple, this approach holds a clear limitation: it approximates the full probability distribution over all sentences with just the top N most probable. *Typically the variations in the top hypothesis involve mostly minor differences in articles and other short function words, with the result that many of the top hypothesis have the same meaning* [3]. This means that the words with low probability are likely to be omitted from the *N-best list* altogether, constraining the full ASR hypothesis.

An alternative approach is to use word lattices or word confusion networks, *which provide a more informative summary of the words' distribution, without pruning the lower scoring words as in the N-best list* [3]. Both word lattices or word confusion networks are directed graphs that encode possible word sequences, represented by paths from the start to the end node. Each edge of the graph gives not only information about word relationships, but also the probability weightings, which allows for the cal-

culation of the probability of a path. Although identical, word confusion networks are more restricted in their structure. In contrast with word lattices, each edge that connects two nodes can only hold a set of mutually exclusive word hypothesis. A graphic representation of both three structures can be seen in Figure 3.2.



***N*-best list**

Rank	Hypothesis	Log probability
1	it's an area that's naturally sort of mysterious	-7497.28
2	that's an area that's naturally sort of mysterious	-7508.93
3	it's an area that's not really sort of mysterious	-7505.33
4	that scenario that's naturally sort of mysterious	-7520.39
5	there's an area that's naturally sort of mysterious	-7518.45

Figure 3.2: Structures that summarize the posterior distribution over sentences found by ASR component ([1], as cited in [3]). The *N*-best list provides a list of *N* hypotheses ranked by probabilities. Word lattices and word confusion networks provide a directed graph which represents word relationships. However, word confusion networks are more restricted in their structure because each edge can only hold a set of mutually exclusive word hypothesis.

3.3 Natural Language Understanding

Natural Language Understanding (NLU), also called Spoken Language Understanding (SLU), allows the system to extract task related information from the user utterances. Traditionally, this information is extracted by three main NLU sub-tasks: *domain classification*, *intent detection* and *slot filling* [1]. Domain classification, as the name suggests, is responsible for determining the domain of the conversation. Although this task may be unnecessary for single-domain dialogue systems, the modern and standard approaches comprise multiple domains. The following task, intent detection or intent classification, is

aimed at determine what is the general task or goal that the user is trying to accomplish. Finally, we need to do slot filling, of which the aim is to extract semantic concepts, in particular the slots and fillers that the user intends the system to understand from their utterance [1].

One major difference among methods for NLU is whether they internally label sequences at the word level (a task usually referred as sequence labeling), or label the entire sequence (sentence classification) [3]. Intent detection is a classification problem that predicts the intent label y^i. Traditionally, slot filling is a sequence labeling task that tags the input word sequence $\boldsymbol{x} = (x_1, x_2, ..., x_T)$ with the slot label sequence $y = (y_1, y_2, ..., y_T)$ [23]. However, the slot filling task can also be decomposed into simple classification problems for each slot. For example, given a slot $s_i \in S$, we want to predict the slot label $y^i_{s_i}$ for all slots $S = (s_1, s_2, ..., s_S)$.

Initially, many natural language understanding systems used semantic template grammars to extract semantic concepts from the user utterances [3]. A particular example is TINA, a system for spoken language applications from MIT, developed in 1992, which integrates key ideas from context-free grammars, Augmented Transition Networks (ATN), and the unification concept [36]. However, grammar-based approaches tend to not generalize well and, in addition, they rely heavily on human expertise and effort to develop the grammar templates [33]. Thus, methods that comprise statistical approaches are best suited for the natural language understanding task.

Statistical methods for NLU are generally split into two categories: *generative* and *discriminative* models. Generative models learn a joint probability distribution $\boldsymbol{P(x, y)}$ between the input $\boldsymbol{x}$ and the labels $\boldsymbol{y}$. Next, they use Bayes' rule to calculate the conditional probability distribution $\boldsymbol{P(y|x)}$, which is then used to assign labels to input examples [3]. Discriminative models directly learn the conditional probabilities of the labels given a feature representation of the input utterance, $\boldsymbol{P(y|x)}$. In contrast with generative models, discriminative models do not make independence assumptions over the feature set and hence it is easy to include arbitrary potentially useful features [3]. For this reason, discriminative models can significantly outperform generative models in NLU tasks [37].

Common discriminative approaches for NLU have been using Support Vector Machines (SVMs) as discriminative classifiers [38], and Conditional Random Fields (CRF) to do sequential labeling on input sentences [39].

In recent years, there has been some work applying neural networks to NLU. Particularly, Recurrent Neural Networks (RNNs) and Long-Short Term Networks (LSTMs) have been the state-of-the-art for both intent classification and slot filing tasks [33, 40]. Prior work has also shown that joint modeling the intent classification and slot filing tasks can exploit and model the dependencies between them and improve the performance over independent models and, besides that, using models with attention mechanism can help with long-range dependencies [41]. Thus, attention-based joint learning methods were proposed and achieved state-of-the-art performance for intent classification and slot filing task [23].

However, recent breakthroughs in natural language processing have changed the state of the art in the NLP field. Usually, there is a lack of human-labeled data for the majority of NLP tasks, resulting in poor generalization capability [23]. To address this problem, a variety of unsupervised techniques were proposed for training general purpose language models using an enormous amount of unannotated

text (see Section 2.4). These pre-trained models can be fine-tuned on NLP tasks, yielding a significant improvement over training on task-specific annotated data and reducing the time needed for training a specific model.

Chen et al. [23] proposed a joint BERT-based model for both intent detection and slot filing, achieving significant improvements in both intent classification accuracy and slot filing F1, compared to the previous attention-based joint models. Besides that, they also demonstrate a large gain in the sentence-level semantic frame accuracy on the Snips dataset, which include multiple domains and has a large vocabulary, and therefore showing the strong generalization capability of using a general purpose language model.

3.4 Dialogue State Tracking

In a spoken dialogue system, dialogue state tracking refers to the task of correctly inferring the state of the conversation, such as the user's goal, given all of the dialogue history up to that turn [2]. This task is particularly difficult because of the common ASR and NLU errors (*the process of converting conversational speech into words still incurs word error rates in the range* $15\% - 30\%$ *in many real-world operating environments* [5]), which may cause the system to misunderstand the user. At the same time, the dialogue state tracking is the core of any dialogue system, because the dialogue policy relies on the estimated dialogue state (or distribution over dialogue states) to choose actions.

So far, three families of dialogue state tracking algorithms have been widely explored and implemented: hand-crafted rules, generative models, and discriminative models.

3.4.1 Hand-crafted Rules

Primarily, the research on state tracking used hand-crafted rules. In its earliest form, these approaches considered only the 1-best NLU result and tracked a single hypothesis for the dialogue state. Hence, these type of systems *reduce the dialogue state tracking problem to a single update rule* $F(s, \tilde{u}') = s'$ *that maps from an existing dialogue state* s *and the 1-best NLU result* $\tilde{u}'$ *to a new dialogue state* s' [2]. The MIT JUPITER weather information, a rule-based model, maintained a set of state variables which were updated using hand-designed rules in a dialogue control table [28]. The Information State Update approach, another rule-based model, used hand-written update rules to track a rich data structure called information state [42].

One shortcoming of tracking only a single hypothesis for dialogue state is the inability to make use of the full ASR/NLU N-best lists. Therefore, these models lack the benefits of tracking multiple dialogue states suggested by Pulman, who proposed modeling dialogue as a conversational game accounting for uncertainty using probabilistic frameworks [43]. Thus, more recent approaches to dialogue state tracking using hand-designed rules, compute scores for all dialogue states suggested by the whole ASR/NLU N-best lists, resulting in a posterior distribution over possible states [44–46].

Hand-crafted rule-based dialogue state trackers do not require any data to implement. Thus, there

is no data-driven model to import when implementing this type of systems in real-world applications, which is a benefit for bootstrapping. In addition, rules also provide developers an accessible method to incorporate knowledge of the dialogue domain [47].

However, a crucial limitation in these type of systems is that rules' formula parameters are not directly derived from real dialogue data, so they require careful tuning and optimization. *This limitation motivates the use of data-driven techniques, which automatically set parameters in order to maximize accuracy* [2].

3.4.2 Generative Models

Generative models try to learn how the data was generated, by specifying a joint probability distribution $P(x, y)$ over input features x and label sequences y. Then, they use the learned model to predict unseen data. *Generative models suggest that dialogue can be modeled as a Bayesian network that relates the dialogue state s to the system action a, the (true, unobserved) user action u, and the ASR/NLU result $\tilde{u}$* [2]. When the last system action and ASR/NLU results are observed, a distribution over possible states (commonly called *belief*) can be estimated by applying *Bayesian inference*. Equation 3.1[48] demonstrates one of the possible variants of belief state update.

$$b'(s') = \eta \sum_{u'} P(\tilde{u}'|u')P(u'|s', a) \sum_{s} P(s'|s, a)b(s) \tag{3.1}$$

where $b(s)$ is the previous probability distribution over states, $b'(s')$ is the (updated) probability distribution over states being estimated, $P(\tilde{u}'|u')$ is the probability of the user taking action $\tilde{u}'$ given the (true, unobserved) user action u', $P(u'|s', a)$ is the probability of the user taking action u' given the true dialogue state s' and system action a, $P(s'|s, a)$ is the probability of the dialogue changing to state s' given the previous dialog state s and systema action a, and η is a normalizing constant.

Early approaches to generative models for dialogue state tracking enumerated all possible dialogue states, and then used variants of Equation 3.1 to score them [49, 50]. However, it is trivial that enumerating all possible dialogue states in intractable, particularly given that these systems usually have to run in real time and the number of states can be considerable large.

Thus, two approximations have been done. First, by maintaining a beam of candidate dialogue states [51–54], the state-space of the dialogue state tracking problem is greatly reduced. Second, further factorizations of Equation 3.1 can be performed, assuming conditional independence between components of the dialogue state, [55, 56], and thereby reducing the complexity of the dialogue state tracking problem.

Although these approximations enable generative models to operate in real time, they impose other constraints. While the N-best approach enables to model all dependencies but with an incomplete distribution, the factoring approach can only handle a limited number of dependencies but can model the complete distribution [5].

In end-to-end evaluations, generative approaches have been shown to outperform hand-crafted rules [56]. However, in generative models all dependencies between features must be explicitly modeled,

which requires an impractical amount of data. Hence, generative models usually make independence assumptions which are invalid, or important features of dialogue history have to be ignored, which introduce a violation of the Markov assumption, resulting in poor estimates [2]. Together, these issues have triggered interest in discriminative models.

3.4.3 Discriminative Models

Discriminative models try to model the conditional probability $P(y|x)$ of the label sequences y given the observations x directly from the data. In contrast with generative models, *discriminative approaches for dialogue state tracking compute scores for dialogue states with discriminatively trained conditional models of the form $b'(s') = P(s'|f')$, where f' are features extracted from ASR, NLU, and dialogue history* [2]. Thus, these type of models can incorporate a large amount of arbitrary potentially useful features and can be optimized directly for prediction accuracy.

Williams [57] analyzed fundamental weaknesses in the formulation of statistical dialogue systems as generative models, suggesting that discriminative models could achieve better performance for belief state update. Indeed, as mentioned above, generative models learn the joint probability and can get wrong assumptions of the data distribution, in contrast with discriminative models which try to model directly the conditional probability.

The first presentation of a discriminative model for state tracking used a hand-written rule to enumerate a set of dialogue states to score [58]. By considering the top S_1 NLU hypothesis from the current turn, top S_2 hypothesis from the previous turn, the top S_3 hypothesis from the turn before that, and an additional state hypothesis $\bar{s}$ which account for the situation when none of the hypothesis is correct, it is possible to get a fixed number classes $k = S_1 + S_2 + S_3 + 1$. Standard multiclass logistic regression classification is then applied, in which one weight is estimated for every (class, feature) pair [58].

Subsequent work was done using some variations of the previous approach. Metallinou et al. [59] modified the logistic regression model to learn a single weight for each feature, which allows an arbitrary number of hypothesis to be scored, since the number of weights no longer increases with the number of hypothesis. Williams [60] applied a ranking algorithm which has the ability to construct conjunctions of features. Henderson et al. [61] applied a deep neural network as a classifier.

The majority of discriminative models encode the dialogue history in the features, to learn a simple classifier. However, there are other approaches which explicitly model dialogue as a sequential process. Conditional Random Fields (CRFs) can be used to determine the most likely dialogue state conditioned on the entire sequence [62]. Henderson et al. [63] used Recurrent Neural Networks (RNNs) that use the ASR/NLU results as inputs and output a distribution over dialogue states. This work is also notable for operating directly on ASR results only, mapping them to the dialogue state. *By using high-dimensional inputs (n-grams), with all the information, instead of inputs with a select few informative features, it avoids the need for an explicit semantic representation, and the possibility of information loss at the NLU stage* [63].

All the previous approaches require in-domain dialogue labeled data for training. However, when a

small amount of labeled data exists for the target domain, multi-domain learning can be applied [64]. If no labeled data exists, it is possible to use unsupervised learning adaptation from a base model for a related domain [65]. This technique tries to find points in the dialogue where a state component value is assigned a high score. Then it treats that predicted value as label, and adjust model parameters to try to predict that label earlier in the dialogue. This approach allows generic slot tracking model to be adapted to a specific slot for which labeled data does not exist.

3.4.4 Dialogue State Tracking Challenge

The literature provides numerous methods for dialogue state tracking. However, direct comparisons between those methods have not been possible because each author used different domains and system components. *Moreover, there has not been a standard task or methodology for evaluating dialogue state tracking, which limited the progress in this area* [2].

In 2013, Williams et al. [66] organized the first dialogue state tracking challenge (DSTC1) series, which introduced the first benchmark task and evaluation framework for dialogue state tracking, as well as labeled dialogue data.

The DSTC aimed at understanding which existing methods for dialogue state tracking have a better performance, encourage new work that advances the state-of-the-art, and examine which evaluation metrics are appropriate for dialogue state tracking.

Ever since, two more series of the dialogue state tracking challenge (DSTC2, DSCT3) were performed [67, 68], and, together, the three instances yield significant work on dialogue state tracking, such as new techniques and a standard set of evaluation metrics.

In particular, the DSTC series has triggered three key advances: *the transition from generative models to discriminative models* (the DSTC series have illustrated the weaknesses in generative models that degraded accuracy, such as the inability to handle a large amount of features); *the development of discriminative sequential models for dialogue state tracking* (unlike simple stationary classifiers, sequential models take as input a set of features at each turn, avoiding the need to encode the dialogue history in the features); *the development of models which use the ASR results as input directly* (by providing direct access to the raw input signal, they have the potential to provide further improvements in accuracy, which was demonstrated in the DSTC series).

3.5 Dialogue Management

Dialogue Management (DM) can be defined as controlling the flow of the conversation, deciding which action the system should take at each point of the dialogue. The dialogue management must, thereby, be able to choose the best action for a specific dialogue state. In addition, the dialogue management should also be responsible for applying a certain dialogue strategy (see Section 3.1).

Conventional dialogue systems typically maintain a single hypothesis for the dialogue state, effectively making the assumption that the state is known [3]. A variety of frameworks have been proposed to

choose the best action to perform, taking into account the single state. Approaches based on flow-charts (with nodes representing states and actions, and arcs representing user inputs) [69], and systems that use logical inference and planning [70].

Although the previous approaches introduce some advantages, such as bootstrapping, and an easy way of incorporating context knowledge, none of them suggest a way of learning which actions should be taken. Casting the problem as a Markov decision process (MDP) allows the learning of an action selection model. However, more recently approaches for dialogue systems allows to maintain a probability distribution over states instead of tracking a single state hypothesis, and thereby include details of the uncertainty in the user inputs. Thus, a more theoretically well-founded framework is to cast the problem as a partially observable Markov decision process (POMDP).

The POMDPs approach also assumes that dialogue evolves as a Markov process, i.e. *starting in some initial state s_0, each subsequent state is modeled by a transition probability $p(s_t|s_{t-1}, a_{t-1})$* [5]. However, *the state s_t is not directly observable reflecting the uncertainty in the interpretation of user utterances; instead, at each turn, the system regards the output of the natural language understanding (NLU) as a noisy observation o_t of the user input with probability $p(o_t|s_t)$* [5]. Hence, two probability functions need to be estimated - the transition and observation probability functions - which can be done by a suitable stochastic model, usually called dialog model. A stochastic model is a tool for estimating probability distributions of potential outcomes by allowing for random variation in one or more inputs over time.

Nowadays, the state-of-the-art approach is to use Reinforcement Learning (RL), in which the dialogue system is seen as a conversational agent, and the dialog system responses are interpreted as the agent actions. Thus, RL allows the agent to learn an optimal policy which can map the state of a dialogue conversation to a system response. However, usually this approach requires the use of a user simulator to interact with the conversational agent.

Chapter 4

Problem Description and Solution

In this chapter, we first give a brief problem description, where we formalize our problem based on the dialogue-state architecture. Then we give a brief description of the tasks the robot must be able to perform, particularly in the context of two robotics competitions. In the following sections, we will describe the models and algorithms used in the automatic speech recognizer (ASR), the natural language understanding (NLU), the dialogue state tracker (DST) and the dialogue manager (DM), including a description of the implemented dialogue state machine (DSM) that will control the dialogue flow. Finally, we explain the dialogue system implementation in the robot.

4.1 Problem Description

This thesis aims at developing a spoken goal-oriented dialogue system, which can be defined as a computer system able to interact with humans using speech and natural language, on a task oriented context. Our goal-oriented dialogue system is designed based on the dialogue-state architecture (recall Section 3.1). In a dialogue-state architecture, the system uses a domain ontology $\mathcal{D}$, which represents the kinds of information the system can extract from the users. The ontology defines one or more frames, each a collection of slots, s, and defines the values, v, that each slot can take. The set of slots specifies what the system needs to know, and the filler of each slot is constrained to values that the slot can take.

Furthermore, dialogue systems that make use of a dialogue-state architecture use a custom data structure, called DIALOGUE ACT, which represents an utterance in the context of conversational dialogue. A DIALOGUE ACT can be defined as a shallow representation of the semantics of either a user's utterance or the system's prompt [3]. Multiple formats have been proposed and are used for representing dialogue acts [71]. In this thesis, we adopt the formalization used in the Cambridge University Dialogue Systems group [3]. Thus, a DIALOGUE ACT consists of two components: a dialogue act type, D-TYPE, and a set of slot bindings, X. In their implementation, they define three types of slot bindings:

- *Bound slots*: slots bound to values, $s = v$,
- *Negated bound slots*: slots bound to values and negated, $s \neq v$,

- *Unbound slots*: slots not bound to any value, s.

However, we will only use two types of slot bindings: the *bound slots* and *unbound slots*. The *negated bound slots* are usually defined for systems that are able to deal with the denial of a certain value for a slot, which is not implemented in our dialogue system.

Tables 4.1 and 4.2 define the adopted dialogue act types and the correspondent slot bindings set for *user acts* and *system acts*, respectively.

Table 4.1: Description of dialogue act types and slot bindings for user actions.

Restrictions on X	Act Type	Description
X must be empty.	ACK	A back-channel such as "okay".
	AFFIRM	Replying affirmatively to the last system response.
	BYE	Saying good-bye.
	HELP	Asking for help
	NEGATE	Replying negatively to the last system response.
	REPEAT	Requesting the system to repeat the last response.
	RESTART	Requesting the system to restart.
	THANKYOU	Saying thank you.
X must contain only bound slots.	INFORM	The user is informing the system about the value of a specific slot. For example, INFORM(INTENT=TAKE, OBJECT=BOOK) might correspond to "take a book".

Table 4.2: Description of dialogue act types and slot bindings for system prompts.

Restrictions on X	Act Type	Description
X must be empty.	HELLO	Giving a welcome message to begin the dialogue.
	REPEAT	Asking the user to repeat himself.
	BYE	Giving a goodbye message to finish dialogue.
X must only contain bound slot(s).	CONFIRM	Confirming that the user goal has the slot-value pair(s) in the bound slots set.
X must only contain unbound slots.	REQUEST	Requesting for more information about a specific slot in the unbound slots set.

4.1.1 Domestic Domain

Our goal-oriented dialogue system will be implemented in a service robot, operating in a domestic environment. In a domestic environment, a robot must perform a variety of domestic/service tasks. One of the first problems the robot must solve is to discover which task it is being asked to perform and, hence, a slot of the utmost importance should be of the type INTENT. This slot holds the information about the user's intention. The second problem the robot must solve is to fill the slots of the task. Hence, other two important slots the system must fill are the SOURCE and DESTINATION. As the names suggest,

these slots hold information about the beginning and ending of a specific task, respectively. Finally, three more slots are defined: the OBJECT, the PERSON and the WHAT-TO-SAY slots, which refer to the object and the person involved in the task, and the latter refers to what the robot must say. A brief summary of the slots defined in the domestic domain ontology can be seen on Table 4.3.

Table 4.3: Slots defined in a domestic domain ontology.

Slot	Description	Example of values
INTENT	the type of task the user wants the robot to perform (intention of the user)	TAKE; FIND; ORDER; ANSWER; PLACE
SOURCE	the start of the task	BEDROOM; KITCHEN
DESTINATION	the end of the task	BATHROOM; KITCHEN
OBJECT	the object involved in the task	BOOK; COKE
PERSON	the person involved in the task	ME; PETER
WHAT-TO-SAY	what the robot must say	WEATHER;DATE

GPSR

In order to evaluate the performance of service robots, there are two international robotics competitions, RoboCup and European Robotics League (ERL), which have benchmark challenges to evaluate how well a robot can perform tasks in a domestic environment. One of these challenges is General Purpose Service Robots (GPSR), in which the robot is ordered to perform domestic tasks such as taking an object to a person or guiding a person to a certain destination. Table 4.4 gives a brief description of the GPSR tasks. As example, consider the user utterance *"take the book from the table"*, whose correspondent slots and fillers are: INTENT=TAKE, OBJECT=BOOK, SOURCE=TABLE.

Table 4.4: Tasks and respective description for the GPSR challenge.

Task (INTENT)	Description	Slots
Motion	Moves to some place	PERSON; SOURCE; DESTINATION
Meet	Meets a person	PERSON; DESTINATION
Grasp	Grabs an object	OBJECT; SOURCE
Place	Places an object	OBJECT; DESTINATION
Follow	Follows a person to a location	PERSON; DESTINATION
Tell	Tells something to someone	WHAT-TO-SAY; PERSON
Find	Looks for an object or person	OBJECT; PERSON
Guide	Guides a person to a location	PERSON; SOURCE; DESTINATION
Take	Takes an object from some place to another or gives it to someone	OBJECT; PERSON; SOURCE; DESTINATION

SciRoc

Recently, another project was created to support the ERL tournament in the context of smart cities. This project is called SciRoc, and the challenges focus on smart shopping and are organized in a series of episodes. In one of these episodes, the robot assists people in a coffee shop, and takes care of customers, by taking orders and bringing objects to and from customers' tables. Table 4.5 describes the task of the SciRoc coffee shop challenge. As example, consider the user utterance *"I would like a cappuccino, a sandwich and a bottle of water"*, whose correspondent slots and fillers are: INTENT=ORDER, OBJECT=CAPPUCCINO, OBJECT=SANDWICH, OBJECT=WATER.

Table 4.5: Task and respective description for the SciRoc coffee shop challenge.

Task (INTENT)	Description	Slots
Order	Order three items (objects) from the coffee shop menu	OBJECT

4.2 Approach

Recall the typical pipeline of a dialogue-state architecture (Figure 3.1). First the automatic speech recognition (ASR) receives the user's speech command as input and outputs an N-best list of recognized hypothesis, each one composed of the text utterance and its correspondent confidence score. Table 4.6 shows an example of a 5-best list of recognized hypothesis, in the context of the GPSR challenge.

Table 4.6: Example of a 5-best list of recognized hypothesis.

Hypothesis	Confidence
bring me a book from the tv table	0.994
bring me the book from the tv table	0.992
bring me a book from the table	0.976
bring me the book from the table	0.974
bring me that book from the tv table	0.971

Then, the natural language understanding (NLU) extracts the relevant information from each of the recognized hypothesis, and for each hypothesis outputs a *user act*.

Recalling the 5-best hypothesis of the example in table 4.6. The correspondent *user acts* are represented in table 4.7. Note that, for each *user act*, there are marginal confidence scores corresponding to the dialogue act type, D-TYPE, and for each of the slot-value pairs presented in X. In the following sections we will discuss how these confidence scores are computed.

After extracting the relevant information from the user's utterances, the following component of the dialogue system's pipeline, dialogue state tracker (DST), uses each of the *user acts* in the N-best list to compute a belief over dialogue states. This means that the dialogue state may have more than one value for a certain slot. Following the previous examples (tables 4.6 and 4.7), the resulting expected dialogue state is represented in table 4.8. The confidence scores associated with each slot-value pair

Table 4.7: Example of a 5-best list of *user acts*.

User acts
INFORM(INTENT=TAKE, OBJECT=BOOK, PERSON=ME, SOURCE=TV TABLE)
INFORM(INTENT=TAKE, OBJECT=BOOK, PERSON=ME, SOURCE=TV TABLE)
INFORM(INTENT=TAKE, OBJECT=BOOK, PERSON=ME, SOURCE=TABLE)
INFORM(INTENT=TAKE, OBJECT=BOOK, PERSON=ME, SOURCE=TABLE)
INFORM(INTENT=TAKE, OBJECT=BOOK, PERSON=ME, SOURCE=TV TABLE)

will also be discuss in the further sections.

Table 4.8: Dialogue state example.

Slot-value pairs	Confidence
INTENT=TAKE	1.0
OBJECT=BOOK	1.0
PERSON=ME	1.0
SOURCE=TV TABLE	0.7
SOURCE=TABLE	0.3

Finally, the dialogue management (DM) checks if the dialogue state holds all the required information to perform a certain task, and if it is confident enough about that information. If not, it generates a system act (system response), which is also defined as a DIALOGUE ACT, and the dialogue continues.

4.3 Automatic Speech Recognizer

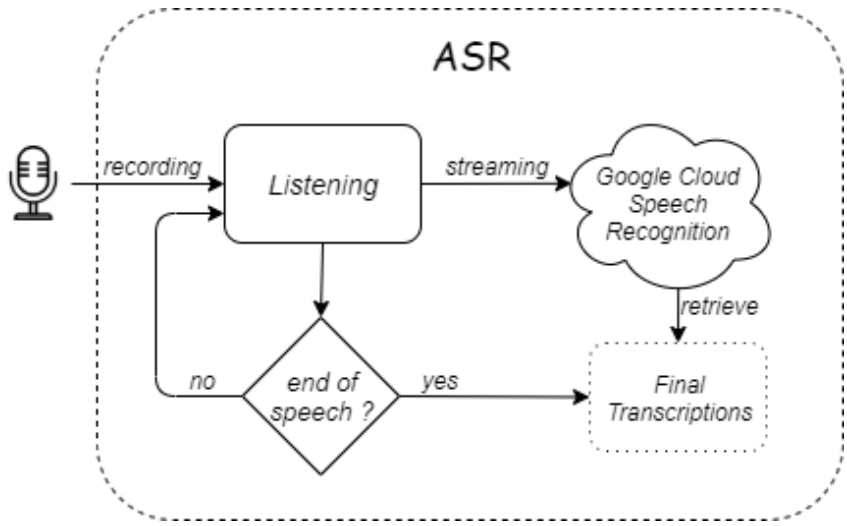


Figure 4.1: Architecture of the Automatic Speech Recognizer (ASR). First, the ASR starts recording audio and streaming it to the Google Cloud Speech Recognition, until it gets an end-of-speech signal. Then, it retrieves the final speech transcriptions.

First, we had to design an Automatic Speech Recognizer (ASR) capable of recognizing multiple hypothesis, and output a probability distribution over words. We achieved this by using the Google

Speech Recognition API to request an N-best list of transcriptions.

Figure 4.1 shows the architecture of the ASR component. Essentially, the ASR starts recording audio and streaming it to the Google Cloud Speech Recognition. When it gets an end-of-speech signal, which means the user has stopped talking, the ASR retrieves the final transcriptions as an N-best list of hypothesis, containing the text transcriptions and their correspondent confidence score.

4.4 Natural Language Understanding

Traditional natural language understanding (NLU) models perform three different sub-tasks: domain classification, intent detection and slot filing. However, when implemented in a dialogue system pipeline, the NLU must also be able to identify the dialogue act type, D-TYPE, which will help the dialogue system to extract some information about dialogue specific actions. For example, when the dialogue system asks for some type of confirmation, the NLU must be able to recognize if the user confirms or denies it. Thus, besides the intent detection and slot filing sub-tasks, our NLU model will also perform the dialogue act type classification of the user utterances. The domain classification sub-task will be ignored, since our dialogue system will be implemented in a single-domain domestic environment.

In Section 3.3, we have seen that the recent state-of-the-art models for the main NLP tasks are based on pre-trained general purpose language models. Because of the lacking of real-world labeled data for our task, we have decided to build a model based on BERT, and exploit its generalization capability.

Initially, we considered both the dialogue act type classification, intent detection, and slot filing tasks as multiple classification problems. Considering the input token sentence $\boldsymbol{x} = (x_1, ..., x_N)$, the model predicts a label for the dialogue act type y^d and for the intent y^i, but it also predicts a label for every slot in our domain ontology $y^{s_i}, \forall s_i \in \mathcal{D}$. This means that the domain ontology holds complete information about each slot, specifically all its possible values (labels), v. The model uses the pre-trained English uncased BERT-base model to output a context-dependent semantic representation of the input sentence, $\boldsymbol{h}_{[\text{CLS}]}$. Then, multiple classifier layers (linear layer + softmax) with a size of 768, all sharing the BERT output, are used to predict the dialogue act type, the intent, and the multiple slots. Figure 4.2 shows the architecture of the implemented model.

Based on the hidden state of the first special token [CLS], denoted $\boldsymbol{h}_{[\text{CLS}]}$, the dialogue act type, intent, and slots are predicted as:

$$y^d = \text{softmax}(\boldsymbol{W}^d \boldsymbol{h}_{[\text{CLS}]} + \boldsymbol{b}^d) \tag{4.1}$$

$$y^i = \text{softmax}(\boldsymbol{W}^i \boldsymbol{h}_{[\text{CLS}]} + \boldsymbol{b}^i) \tag{4.2}$$

$$y^{s_i} = \text{softmax}(\boldsymbol{W}^{s_i} \boldsymbol{h}_{[\text{CLS}]} + \boldsymbol{b}^{s_i}), \forall s_i \in \mathcal{D} \tag{4.3}$$

where $\boldsymbol{W^d}, \boldsymbol{b^d}$, $\boldsymbol{W^i}, \boldsymbol{b^i}$ and $\boldsymbol{W^{s_i}}, \boldsymbol{b^{s_i}}$ are the weights and biases associated with the classifier layer for

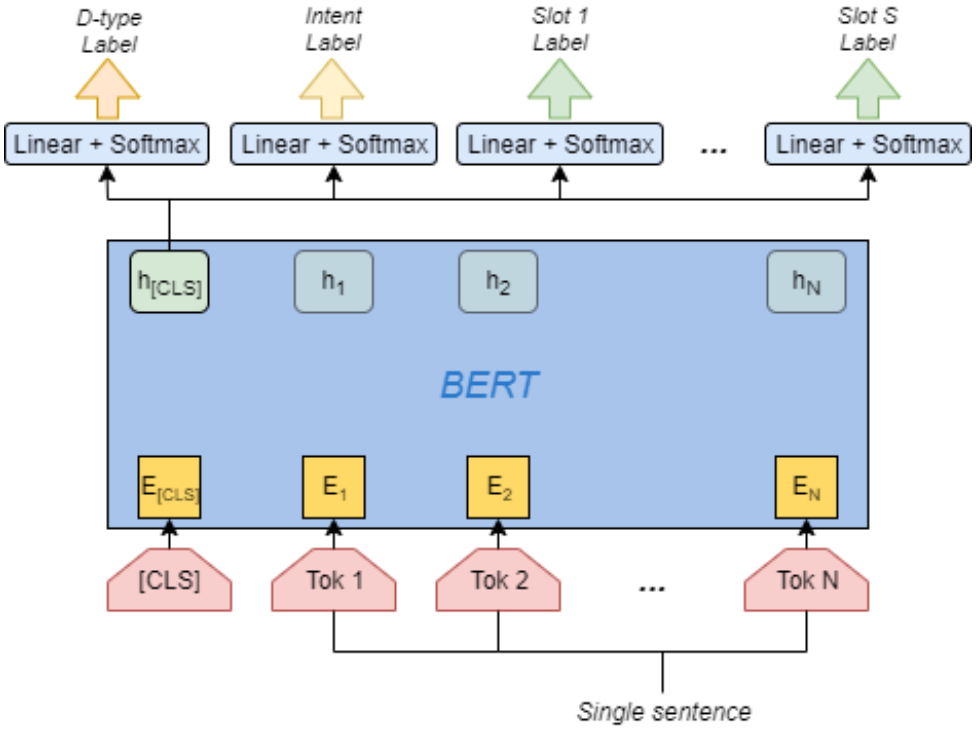


Figure 4.2: Joint multi-task classification model for natural language understanding. First, BERT outputs a context-dependent semantic representation of the input token sentence, $\boldsymbol{h}_{[\text{CLS}]}$. Then, multiple classifier layers use these features to predict the labels for each task, y^d, y^i, and $y^{s_i}, \forall s_i \in \mathcal{D}$. Adapted from [21].

the dialogue act type, the intent, and the multiple slots, respectively.

By jointly modeling the three tasks, we enable the model to exploit the relationships between them, but we also reduce the computational cost of using a single individual model for each slot (the number of models increase with the number of slots). The objective is formulated as:

$$P(y^d, y^i, y^{s_1}, ..., y^{s_S}|\boldsymbol{x}) = P(y^d|\boldsymbol{x})P(y^i|\boldsymbol{x})\prod_{i=1}^{S} P(y^{s_i}|\boldsymbol{x}) \tag{4.4}$$

where S is the total number of slots in our domain ontology. The learning objective is to maximize the conditional probability $P(y^d, y^i, y^{s_1}, ..., y^{s_S}|\boldsymbol{x})$, and, hence, the model is finetuned end-to-end via minimizing the cross-entropy loss (recall Equation 2.12).

One advantage of the previous approach is that, as we are considering the slot filing as multiple classification problems, the possible values for each slot are well defined in the knowledge base of the system. For example, consider the user says *"bring me my computer"*. The model will assign the slot OBJECT with the value COMPUTER. However, the user could also say *"bring me my laptop"* and the model will also assign the slot OBJECT with the value COMPUTER (instead of LAPTOP). This means that when the model understands an object, it is easy to identify it in the system's knowledge base, which represents a significant simplification for the following pipeline components of the dialogue system, as we will see in the next sections.

However, this also holds a clear limitation. Usually, the model will not generalize well to unseen values in the training data. For example, if the model has never seen the word laptop during training, it

will not know that the word is associated with the label COMPUTER. Hence, the model may not be able to correctly classify this slot. In addition, the model will not scale well, i.e. if we want to introduce a new value for a certain slot, we have to modify the number of labels in the classification task and retrain the model.

Therefore, we decided to change the implementation of the natural language understanding model. We continue to consider the dialogue act type classification and intent detection as classification problems, but the slot filing task is now considered as a sequence labeling task. Sequence labeling requires aligned data. Therefore we need to do aligned labeling in order to provide an alignment between the words in the input utterance and the target semantics. *BIO tags provide a method of aligning spans of a sequence with labels* [3]. Short for begin, inside, outside, this is a common tagging format for tagging tokens. The B-prefix before a tag indicates that the tag is the beginning of a sequence, and an I-prefix before the tag indicates that the tag is inside a sequence. An O-tag indicates that a token does not belong to any sequence. Table 4.9 represents an example of sequence labeling using BIO tags.

Table 4.9: Sequence labeling using BIO tagging. In this example, the slots are: PER(person)="me", OBJ(object)="book", and SRC(source)="living room table".

Tokens	Bring	me	a	book	from	the	living	room	table
Labels	O	B-PER	O	B-OBJ	O	O	B-SRC	I-SRC	I-SRC

First, we developed three independent models, instead of jointly modeling the tasks. We chose not to implement a joint model because the computational complexity of increasing the slots for slot-filing as a sequence labeling task is less significant compared to the implementation as a classification task (it increases the number of labels, but we do not need to have a single model for each slot). Both dialogue type classification, intent detection and slot filing models are composed of a pre-trained English uncased BERT-BASE model stacked with a classifier layer. However, in the dialogue type classification and intent detection we classify the whole sentence, while in slot filing we classify each word of the input sentence. Both type of models are represented in Figure 4.3.

Since both dialogue act type classification and intent detection are still being considered as classification tasks, they are predicted using the equations 4.1 and 4.2. However, note that the hidden state of the first special token [CLS], denoted $\boldsymbol{h}_{[\text{CLS}]}$, is now different for both dialogue act type and intent models, since they are now being independently modeled. Thus, each of the models will have an independent learning objective, which is to maximize the conditional probabilities $P(y^d|\boldsymbol{x})$ and $P(y^i|\boldsymbol{x})$, respectively. The models are finetuned end-to-end by minimizing the cross-entropy loss (recall Equation 2.12).

For slot filling, as a sequence labeling task, we need the final hidden states for the tokens representing each of the input words, denoted $\boldsymbol{h}_x = (\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_N)$. We then feed each of the hidden representation into a softmax layer, yielding a label for each token, $y_n^s, n \in 1...N$, where N is the total number of tokens.

$$y_n^s = \text{softmax}(\boldsymbol{W}^s \boldsymbol{h}_n + \boldsymbol{b}^s) \tag{4.5}$$

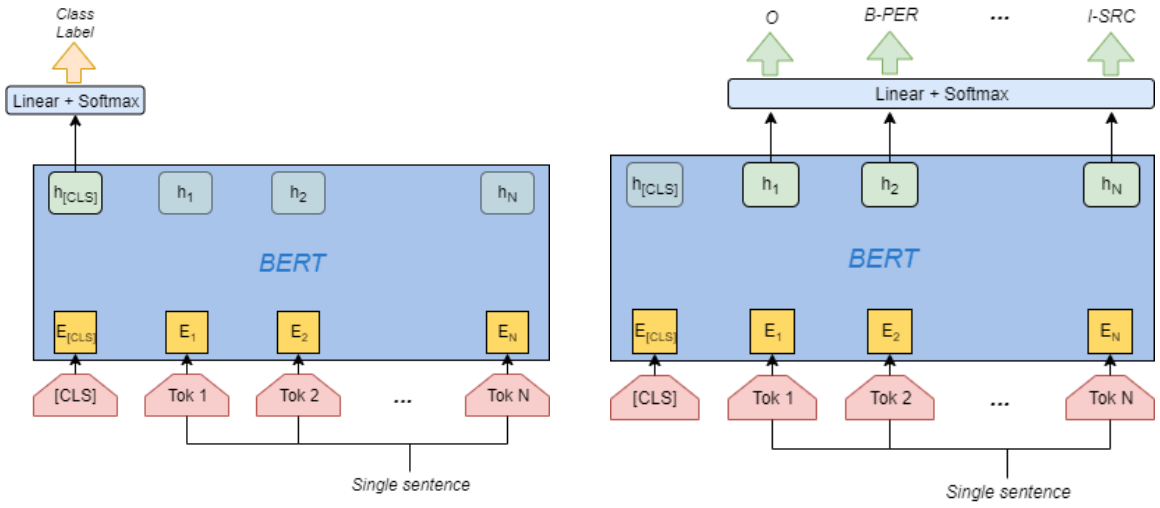


Figure 4.3: Classification and sequence labeling models used for dialogue type classification, intent detection and slot filing. Adapted from [21].

where $\boldsymbol{W}^s$ and $\boldsymbol{b}^s$ represent the weights and biases of the classifier layer of the slot filing model.

The objective if formulated as:

$$P(y^s|\boldsymbol{x}) = \prod_{n=1}^{N} P(y_n^s|\boldsymbol{x}) \tag{4.6}$$

and the learning objective is to maximize the conditional probability $P(y^s|\boldsymbol{x})$ by minimizing the cross-entropy loss (recall Equation 2.12).

Figure 4.4 shows the NLU architecture. When the NLU receives the hypothesis from the speech recognition, it first performs some text pre-processing of the text transcriptions and then, using the NLU models described earlier, it predicts the dialogue act type, the intent and the slot-values. After that, it uses the grounding model to ground the predicted slot-values to the entities that are known to the system. The grounding model will be explained further, in section 4.4.1. Finally, after the predictions, the NLU computes a list of user acts (to each of the transcribed hypothesis corresponds a single user act).

Before proceeding to the grounding model description, we need to first explain how the user acts are formed. Consider an N-best list of transcribed hypothesis, $\boldsymbol{h}_n = (h_1, ..., h_N)$. Let h_i be the i-th hypothesis of the list, whose text transcription is u_i, and confidence score is c_i^u. First, we compute a probability distribution over the N hypothesis, $\boldsymbol{p}_n^u = (p_1^u, ..., p_N^u)$, using a softmax operation:

$$\boldsymbol{p}_n^u = \frac{\exp(\boldsymbol{c}_n^u)}{\sum_{i=1}^{N} \exp(c_i^u)} \tag{4.7}$$

yielding a probability p_i^u for each hypothesis h_i in the N-best list. As seen earlier, for each hypothesis' utterance transcription u_i, the NLU model will predict the dialogue act type, y^d, the intent, y^i, and a slot label for each input token n, y_n^s. Those predictions are performed using a softmax operation in the final layer of the model, and, hence, to each NLU prediction will also correspond a confidence score. Thus, there will be a dialogue act type confidence score c^d, an intent confidence score c^i, and a slot confidence score for each input token n, c_n^s.

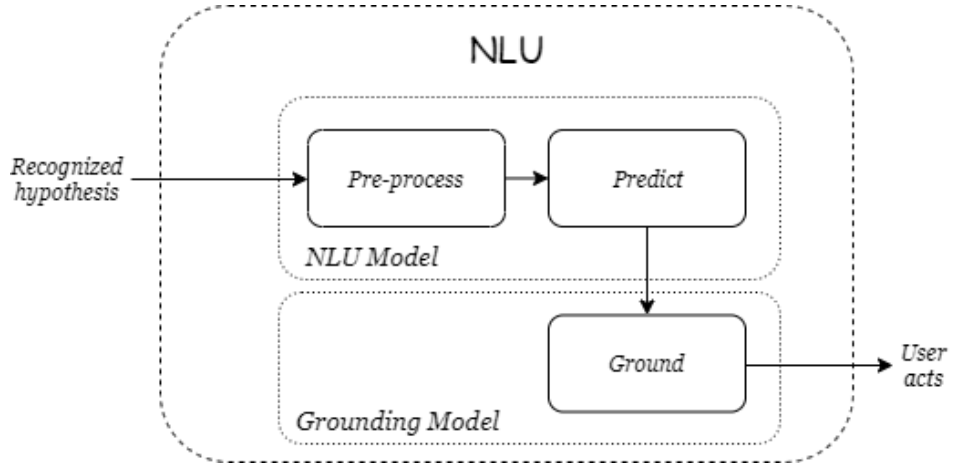


Figure 4.4: Architecture of the Natural Language Understanding (NLU). Each time the NLU receives a list of recognized hypothesis, it uses the NLU models to predict the dialogue act type, the intent and the slot-value pairs for each hypothesis. After that, it uses the grounding model to ground the slot-values to the entities known to the system. Finally it computes the user acts.

Each user act is composed of a dialogue act type, an intent and slot-value pairs. Let $\boldsymbol{d}_i^u$ be the user dialogue act corresponding to the i-th recognized hypothesis, h_i. The user dialogue act will be composed of a dialogue act type y_i^d, and an intent y_i^i, with confidence scores $\bar{c}_i^d = c_i^d \cdot p_i^u$ and $\bar{c}_i^i = c_i^i \cdot p_i^u$, respectively, i.e. the confidence scores of the NLU predictions will be normalized by the probability of the recognized hypothesis. Besides that, the user act can also have one or more slot-value pairs. After the slot labeling, each token n of the input sentence will be assign a label y_n^s. However, since what we need is a slot-value pair, each time a token n is labeled with a certain slot s, we will consider that the token n is the value of that slot. Let x_n be the n-th token of the i-th transcribed hypothesis h_i. If that token is assigned with a slot label s, the user dialogue act $\boldsymbol{d}_i^u$ will have the slot-value pair $(s, v = x_n)$, with confidence score $\bar{c}_i^{s,v} = c_n^s \cdot p_i^u$. Recall that the slot labeling task is performed using aligned labeling. Thus, there may be the case where a slot-value is composed of more than one token. In this situation, the slot-value pair will be composed of the concatenation of the l tokens that are labeled with slot s, and, hence, $(s, v = x_n \oplus ... \oplus x_{n+l})$, with confidence score $\bar{c}_i^{s,v} = \prod_{i=0}^{l-1} c_{n+i}^s \cdot p_i^u$.

4.4.1 Grounding

The sequence labeling model described in the previous section enables a better generalization for the slot filing task, since it is usually able to classify a word with the correct label, even without seeing it during training. However, we need to ground the words labeled by the model to the system's knowledge base, i.e. to the entities that are known to the system. Returning to the previous example, if the user says *"bring me my computer"*, the sequence labeling model will assign the label OBJECT to the word COMPUTER. On the other hand, if the user says *"bring me my laptop"*, the model will label the word LAPTOP as an OBJECT. However, both words share the same meaning. Besides that, will the system be able to relate any of those words with the real entity (computer)? In the case of the first approach for the NLU model, since we are dealing with the slot filing task as a classification problem, the labels are defined to correspond directly to the entities in the knowledge base. However, for the second approach

we need a way to relate the labeled words to the entities that the system knows.

A good approach is to define one or more names for each entity in the system's knowledge base, $\mathcal{KB}$. Thus, to each entity e, will correspond a set of words that correctly describes that entity, S^e. Then we can use those words, $\mathrm{w}_e \in \mathrm{S}^e$, to compare with the words labeled by the slot filing model. Although one could compare the labeled word with all the names of the entities and check if they match, it is usually intractable to define all possible names for all entities.

In Section 2.4 we have explained different techniques for representing words. Since some word embeddings' models represent words based on their semantic relationships, we can use the same idea to develop a model that is able to recognize if two words are semantically related. Thus, we developed a grounding model that is able to assign a confidence score between a certain word and an entity, which represents their semantical relatedness. Let w_k be the k-th word labeled by the slot filing model. First, we extract their embedding vector, $\boldsymbol{w}_k = \Phi(\mathrm{w}_k)$, using the grounding model, which makes use of a projection function $\Phi(.)$ of words into a multi-dimensional geometrical space [72]. Then, we compare the embedding vector of the k-th word, $\boldsymbol{w}_k$, with the embedding vector of all the possible names for all entities in the knowledge base, $\boldsymbol{w} = \Phi(\mathrm{w}), \forall \mathrm{w} \in S^e, \forall e \in \mathcal{KB}$. This comparison uses a GROUNDING FUNCTION that computes the confidence score as the cosine similarity between the word vectors (words with the same meaning will be closer in the multi-dimensional space of the embeddings). Given a word w_k, the GROUNDING FUNCTION of an entity e will be computed as:

$$g(\mathrm{w}_k, e) = \max_{\boldsymbol{w} \in \mathrm{S}^e} sim(\boldsymbol{w}, \boldsymbol{w}_k) = \max_{\boldsymbol{w} \in \mathrm{S}^e} \frac{\boldsymbol{w} \cdot \boldsymbol{w}_k}{\|\boldsymbol{w}\| \|\boldsymbol{w}_k\|} \tag{4.8}$$

Finally, we sort the confidence scores, $g(\mathrm{w}_k, e)$, and if the maximum score is bigger than a defined GROUNDING THRESHOLD, τ_G, we assign that entity to the slot filing label. If not, the word will be assign to no entity.

$$e = \underset{e \in \mathcal{KB}}{\operatorname{argmax}}\, g(\mathrm{w}_k, e), g(\mathrm{w}_k, e) \geq \tau_G \tag{4.9}$$

Since a lot of work has already been done in word embeddings training, we used pre-trained embeddings, particularly the GloVe embeddings trained on a large wikipedia corpus.

4.5 Dialogue State Tracker

In Section 3.4, we gave a description about the state of the art in dialogue state tracking. We chose to use a rule-based method, since we had no dialogue data for this particular domain to train a statistical model. In addition, we also wanted a dialogue state tracker capable of tracking multiple hypothesis and maintain a belief over dialogue states. In the dialogue state tracking challenge (Section 3.4.4), a particular rule-based method achieved significant results in the benchmark task. This generic dialogue state tracker maintains beliefs over user goals based on a few simple domain independent rules, using basic heuristic operations [44]. These rules are directly applied to observable system actions (i.e. the responses generated by the system) and partially observed user acts (i.e. the output of the NLU

component), without using any knowledge from external sources [44].

The heuristic rules are derived from some basic probabilistic mathematics. *Let $P(X)$ denote the probability of the occurrence of an event X, and, hence, the probability of X not occurring is $P(\neg X) = 1 - P(X)$. Accordingly, if X occurs two times, with independent probabilities $P_1(X)$ and $P_2(X)$, respectively, then the overall probability of its occurrence is $P(X) = 1 - P_1(\neg X)P_2(\neg X) = 1 - (1 - P_1(X))(1 - P_2(X))$* [44]. *The previous probability can be generalized to $P(X) = 1 - \prod_{i=1}^{k} P_i(\neg X) = 1 - \prod_{i=1}^{k}(1 - P_i(X))$, given a sequence of k independent events, with the probability of X occurring in the i-th event being $P_i(X)$. Recursively, we can compute this quantity as* [44]:

$$P^t(X) = 1 - (1 - P^{t-1}(X))(1 - P_t(X)) \tag{4.10}$$

where $P^t(X)$ denotes the value of $P(X)$ after X occurring t times, and we let $P^0(X) = 0$.

Now, consider A to be a binary random variable. Suppose we know the prior probability of A being true is $Pr(X)$. If there is a chance where with probability $P(B)$ we will observe an event B independent of A, and we assume that if B happens, we must set A to false, then the probability of A still being true will become $P(A = true) = Pr(A)P(\neg B) = Pr(A)(1 - P(B))$ [44].

Each turn, the dialogue system executes an action and receives an observation. The observation is an NLU N-best list of DIALOGUE ACTS, and can either be a DIALOGUE ACT without taking any SLOT-VALUE arguments (e.g. AFFIRM() or RESTART()) or an act presenting one or more SLOT-VALUE pairs (e.g. INFORM(OBJECT=BOOK) or INFORM(OBJECT=BOOK, SOURCE=LIVING ROOM TABLE)).

The main goal of our dialogue state tracker is to track the beliefs over individual slot-value pairs, starting with an initial belief b_0 with null confidence scores for all the slot-value hypothesis. As stated in section 4.2, the dialogue state tracker (DST) component will receive an N-best list of user acts (DIALOGUE ACTS). Each user act may contain one or more slot-value pairs, (s_i, v_j), with their correspondent marginal confidence scores, $c_{(s_i,v_j)}$. First, in each turn, the dialogue acts which have more than one slot-value pairs will be split into single slot-value dialogue acts. Then, the dialogue acts which have the same slot-value pairs will be merged across the N-best list by summing their confidence scores, yielding the marginal confidence scores for individual slot-value representations. Consider the following example:

INFORM(INTENT=TAKE, OBJECT=BOOK) $(c_{(intent,take)} = 0.7,\ c_{(object,book)} = 0.6)$
INFORM(INTENT=TAKE, OBJECT=COKE) $(c_{(intent,take)} = 0.3,\ c_{(object,coke)} = 0.1)$

After splitting-merging procedure, the user acts will become:

INFORM(INTENT=TAKE) $c_{(intent,take)} = 1.0$
INFORM(OBJECT = BOOK) $c_{(object,book)} = 0.6$
INFORM(OBJECT = COKE) $c_{(object,coke)} = 0.1$

Then, the dialogue state tracker will use the single slot-value dialogue acts to update the current dialogue state using a set of heuristic rules. *Let $P_t(u, s, v)$ denote the marginal confidence score for a user dialogue act $u(s = v)$ at turn t. Then, the belief $b_t(s, v)$ for the slot-value pair (s, v) is updated as* [44]:

- Rule 1: If $u = inform$, then $b_t(s, v) = 1 - (1 - b_{t-1}(s, v))(1 - P_t(u, s, v))$.

In addition, we also consider the effects of certain system actions on the belief updates as well. *Let $a(h)$ be one of the system actions performed in turn t, where h stands for a set of n slot-value arguments taken by a, i.e. $h = (s_1, v_1), ..., (s_n, v_n)$* [44].

- Rule 2: If a is an implicit or explicit confirmation action, CONFIRM, and an AFFIRM or NEGATE user act u is observed with confidence score $P_t(u)$:
  - Rule 2.1: If $u = affirm$, then $b_t(s_i, v_i) = 1 - (1 - b_{t-1}(s_i, v_i))(1 - P_t(u)), \forall (s_i, v_i) \in h$.
  - Rule 2.2: If $u = negate$, then $b_t(s_i, v_i) = b_{t-1}(s_i, v_i)(1 - P_t(u)), \forall (s_i, v_i) \in h$.

Finally, we add another rule to restart the belief of all slot-value hypothesis when a restart action is requested by the user.

- Rule 3: If a RESTART user act u is observed with confidence score $P_t(u) \geq \tau_R$, then $b_t(s_i, v_i) = 0, \forall (s_i, v_i) \in \mathcal{D}$, where $\mathcal{D}$ is the domain ontology and τ_R is a RESTART THRESHOLD.

Recalling the previous example, the dialogue state that corresponds to the first turn $t = 1$ will be:

INTENT=TAKE	$b_1(intent, take) = 1 - (1 - b_0(intent, take))(1 - c_{(intent,take)}) = 1.0$
OBJECT = BOOK	$b_1(object, book) = 1 - (1 - b_0(object, book))(1 - c_{(object,book)}) = 0.6$
OBJECT = COKE	$b_1(object, coke) = 1 - (1 - b_0(object, coke))(1 - c_{(object,coke)}) = 0.1$

Figure 4.5 shows the DST architecture. Essentially, each time the DST receives a list of user acts, it splits and merges the user acts, yielding user acts with single slot-value pairs, and then applies the previous described heuristic rules to update the dialogue state (i.e. the belief over individual slot-value pairs).

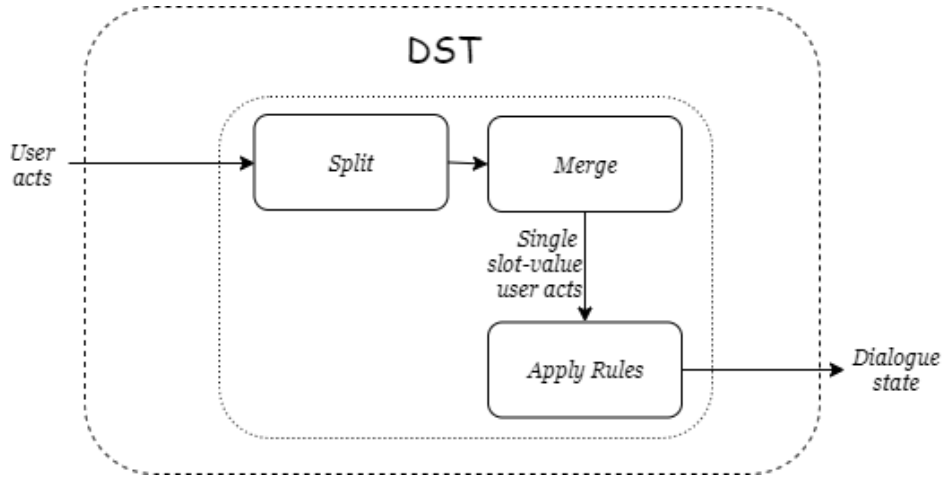


Figure 4.5: Architecture of the Dialogue State Tracking (ASR). When the DST receives a list of user acts, it first splits and merges the dialogue user acts, yielding single slot-value dialogue acts. Then, it uses these dialogue acts to compute the dialogue state by applying heuristic rules.

4.6 Dialogue Manager

The dialogue manager (DM) will control the flow of the conversation. Thus, it must decide which action the system should take at each turn, based on the current dialogue state. As stated before, we did not have any real world dialogue data for the dialogue system's domestic domain, neither a user simulator for training the dialogue manager using reinforcement learning. Thus, we decided to design the dialogue manager based on some hand-crafted rules. Briefly, the dialogue manager will compute a task state, t_t, and then use the hand-crafted rules to choose the next system's action. This task state is computed using a set of task templates, defined in a task ontology, $\mathcal{T}$.

Consider the dialogue state at turn t, $s_t = (b_t(s_1, v_1), ..., b_t(s_N, v_N))$, where N denotes the total number of slot-value pairs, and a system response dialogue act generated in the same turn, $d_t^a = a(h)$, where a denotes the dialogue act type and $h = (s_1, v_1), ..., (s_k, v_k)$ is a set of k slot-value arguments taken by a.

First, the dialogue manager will verify if the dialogue state contains a value for the INTENT slot. If not, this means that the system cannot know the intention of the user, i.e. the type of task the user wants the robot to perform. Thus, it must ask for that specific slot, generating the system response dialogue act REQUEST(INTENT).

- Rule 1: if $(intent, v) \notin s_t$, then $d_t^a = a(h) =$ REQUEST$(intent)$, where $a =$ REQUEST, and $h = (intent,$ NULL$)$.

On the other hand, when the dialogue state contains a value for the INTENT slot (e.g. INTENT=TAKE), but its confidence is lower than a SLOT THRESHOLD, τ_s, then the system will generate the dialogue act, CONFIRM(INTENT=TAKE).

- Rule 2: if $(s = intent, v)\ \exists\ s_t$, with $b_t(s = intent, v) \leq \tau_s$, then $d_t^a = a(h) =$ CONFIRM$(intent, v)$, where $a =$ CONFIRM, and $h = (s = intent, v)$.

Otherwise, if the dialogue state has a value for the INTENT slot, and is confidence enough about it, it will choose the task templates related to that specific INTENT. Each task template will have a set of required slots, s_{req}, which are the slots required to perform that specific task, and also a task confidence score c^t. In this case, the DM will start filling the task templates, using the slot-value pairs in the dialogue state. For each task template, the DM will search in the dialogue state for slot-value pairs whose slot is in the required slots of the task, i.e. $(s, v) \in s_t, s \in s_{req}$. If the confidence score of those slot-value pairs is bigger than the SLOT THRESHOLD, $b_t(s, v) > \tau_s$, their values will be added to the task template. In addition, for each slot-value pair added to the task template, the task confidence score will be updated as, $c_i^t = c_{i-1}^t \cdot b_t(s, v)$. The initial task confidence score will be the confidence score of the intent slot, $c_0^t = b_t(intent, v)$. In the end, at each turn t, we will have a set of task templates related to a specific task, defined by the INTENT slots in the dialogue state. Each of these task templates contains slot-value pairs extracted from the dialogue state, whose slots are required to perform that specific task, and a task confidence score. Finally, the DM will choose the task template with the higher confidence score, and compute the task state, t_t.

Thus, at each turn t, a task state composed of j slot-value pairs, $t_t = (s_1, v_1), ..., (s_j, v_j)$, with a certain task confidence c^t, will be computed. If the task confidence score is lower than a TASK THRESHOLD, τ_T, this means that the system is not entirely sure of all the information in the task state, and, hence, will generate the system response dialogue act CONFIRM$((s_1, v_1), ..., (s_j, v_j))$.

- Rule 3: for $t_t = (s_1, v_1), ..., (s_j, v_j)$, if $c^t \leq \tau_T$, then $d_t^a = a(h) =$ CONFIRM$((s_1, v_1), ..., (s_j, v_j))$, where $a =$ CONFIRM, and $h = ((s_1, v_1), ..., (s_j, v_j))$.

If the task confidence score is higher than the TASK THRESHOLD, then the DM will check if the task state (task template) has all the required slots full-filled, i.e. if all the required information is presented. If not, it must generate a system response dialogue act requesting one of the missing slots, REQUEST(s).

- Rule 4: for $t_t = (s_1, v_1), ..., (s_j, v_j)$, $c^t > \tau_T$, if $s_k \in s_{req}, s_k \notin t_t$ then $d_t^a = a(h) =$ REQUEST(s_k), where $a =$ REQUEST, and $h = (s_k,$ NULL$)$.

Consider the following dialogue state as an example:

INTENT=TAKE	$b_1(intent, take) = 1.0$
OBJECT = BOOK	$b_1(object, book) = 0.6$
OBJECT = COKE	$b_1(object, coke) = 0.1$

First, the DM will check for the INTENT slot. Since the slot is presented in the dialogue state, with a high confidence score, the system will retrieve the task templates related to the task TAKE. Consider one of those templates as TAKE(OBJECT, SOURCE, DESTINATION), where the required slots are the object, source and destination. Since there are two different slot-value pairs in the dialogue state (OBJECT=BOOK), (OBJECT=COKE), and in both of them, the slot is a required slot of the task template, two task templates will be computed. One of them is TAKE(OBJECT=BOOK, SOURCE, DESTINATION) and the other is TAKE(OBJECT=COKE, SOURCE, DESTINATION). However, the first task template has a higher confidence score, $b_1(intent, take) \cdot b_1(object, book) = 1.0 \times 0.6 = 0.6$, compared with the second template which has a confidence score of 0.1. Then, the task state is going to be TAKE(OBJECT=BOOK). Considering a task threshold $\tau_t = 0.8$, the task confidence will be lower than this threshold, and, hence, the dialogue system will generate a system response of the type CONFIRM(INTENT=TAKE, OBJECT=BOOK).

Now, consider that the user confirms the system response and, in the next turn, the dialogue state is updated to:

INTENT=TAKE	$b_1(intent, take) = 1.0$
OBJECT = BOOK	$b_1(object, book) = 1.0$
OBJECT = COKE	$b_1(object, coke) = 0.1$

The task state will still be TAKE(OBJECT=BOOK, SOURCE, DESTINATION). However, it will have a confidence score of 1.0, and since it is bigger than the TASK THRESHOLD, $\tau_T = 0.8$, the system is sure about the information. Nevertheless, the system does not have all the required information to perform the task and, hence, it will generate a system response of the type REQUEST(SOURCE) or REQUEST(DESTINATION).

Figure 4.6 shows the DM architecture. When the DM receives the dialogue state, it first computes the task state, and then apply the previously described rules and compute the system response.

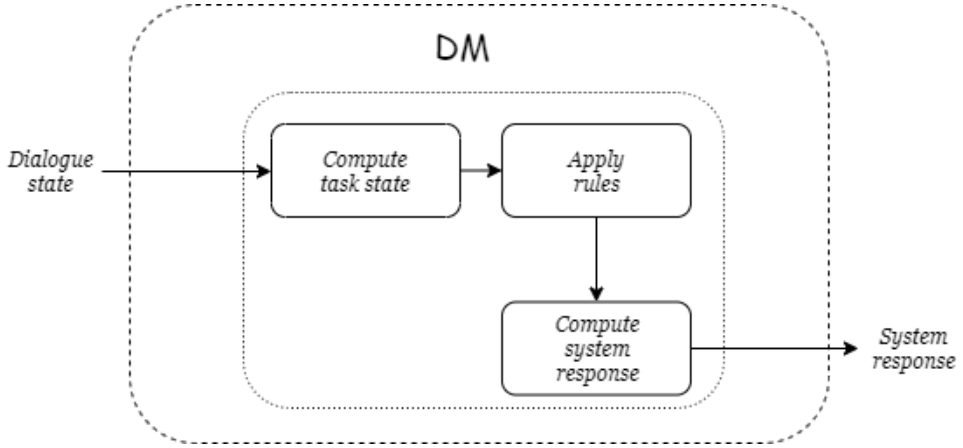


Figure 4.6: Architecture of the Dialogue Manager (DM). When the dialogue state is published, the DM will first compute the task state. Then, it applies the rules and computes the system response.

4.6.1 Dialogue State Machine

We have described how the dialogue manager (DM) will generate the system's response, based on the dialogue state. However, the DM is also responsible for controlling the flow of the conversation. Therefore, we developed a dialogue state machine (DSM), which is capable of controlling all the components of the dialogue system's pipeline. The DSM architecture can be seen in figure 4.7.

The first state of the DSM is the DIALOGUE BEGIN. This state is responsible to begin the dialogue, by generating an HELLO() message. The next state is the START RECOGNITION, where the state machine will activate the recognition in the automatic speech recognition (ASR) node. Then, there will be a transition for the WAIT FOR RECOGNITION state, where the state machine will be waiting for a recognition from the ASR. In this state, there are two possible transitions. If there is a successful recognition, the state will transit to the state DIALOGUE MANAGEMENT. However, if there is an unsuccessful recognition (the ASR cannot recognize any hypothesis), or there is a timeout waiting for the recognition, the state will transit to RECOGNITION FAILED. This state will just compute a REPEAT() system response and continue to START RECOGNITION, unless there are N repeated failed recognitions, where the state will return failure and transition to the DIALOGUE FAILURE. The number of maximum failed recognition is a tune parameter of the dialogue system that can be changed. The DIALOGUE MANAGEMENT is the core state of the machine. After a successful recognition by the ASR component, the natural language understanding (NLU) and dialogue state tracker (DST) components will compute the user dialogue acts and dialogue state, respectively. Then, the dialogue manager (DM) will compute the system's response. The DIALOGUE MANAGEMENT state will be waiting for the computed system's response. Based on that response, it will decide if the dialogue will continue, and transit to the DIALOGUE CONTINUE state, or will finish and transit to the DIALOGUE END state. Besides that, the DIALOGUE MANAGEMENT can also return failure if there is a timeout waiting for the system's response, and transit to the DIALOGUE FAILURE state.

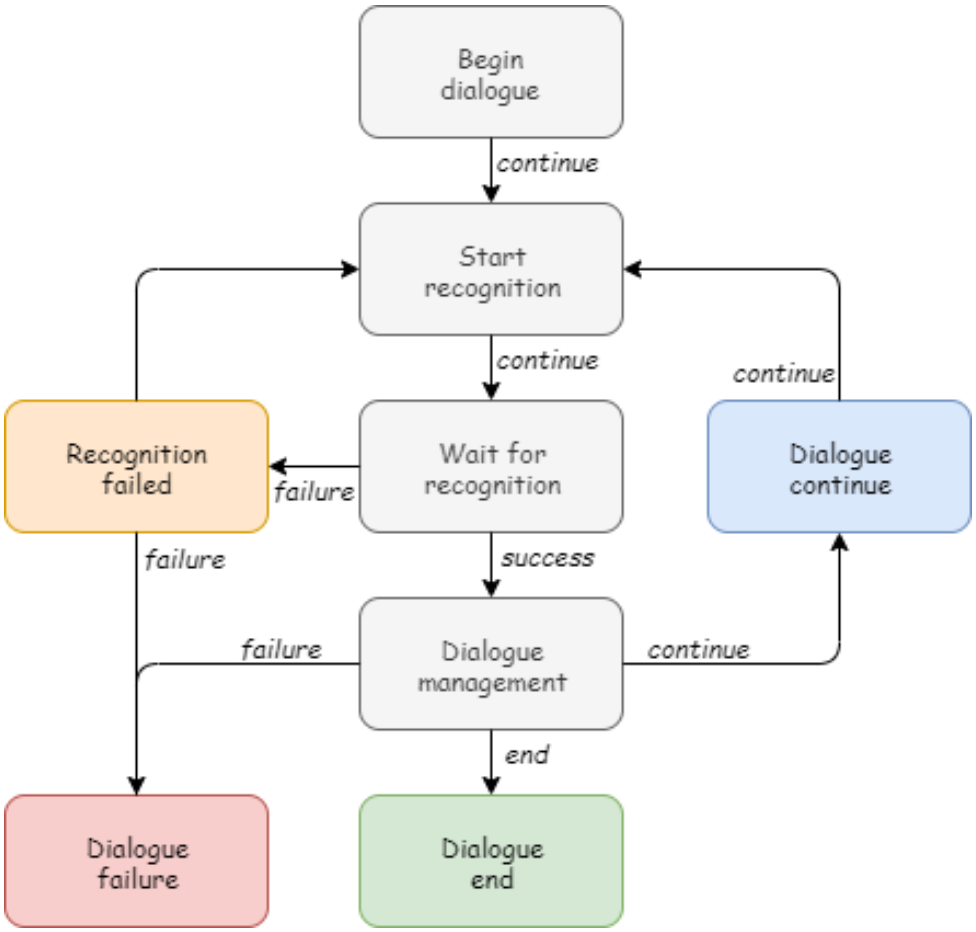


Figure 4.7: Architecture of the Dialogue State Machine.

The normal operation of the state machine is expected to be DIALOGUE BEGIN → START RECOGNITION → WAIT FOR RECOGNITION → DIALOGUE MANAGEMENT → DIALOGUE CONTINUE → ... until it has all the required information for a task, and then → DIALOGUE MANAGEMENT → DIALOGUE END.

4.7 Implementation

The developed dialogue system was implemented in the robot using ROS (Robot Operating System) [73]. ROS is a framework for writing robot software. It provides hardware abstraction, low-level device control (device drives), implementation of commonly used libraries and visualizers, message-passing between processes, and package management. ROS systems are made up of many independent programs running simultaneously and communicating with one another by passing messages. Therefore, ROS provides a loosely coupled structure that allows the creation of generic modules, which are organized as packages.

In our implementation, each component was designed as an individual ROS package, each of which contains the code for each individual component, together with their specific models. Usually, we can represent a ROS system using a graph, whose nodes represent the independent programs, and programs that communicate with one another are connected by edges (usually referred as topics). Figure

4.8 shows the ROS graph for our dialogue system implementation. The Automatic Speech Recognizer (ASR), Natural Language Understanding (NLU), Dialogue State Tracker (DST), and Dialogue Manager (DM) are independent programs (nodes), which run simultaneously and communicate with one another by sending messages. The Dialogue State Machine (DSM) is another program, which implements a SMACH state machine for controlling some of the nodes, particularly the ASR and DM, enabling the system to control the flow of the conversations.

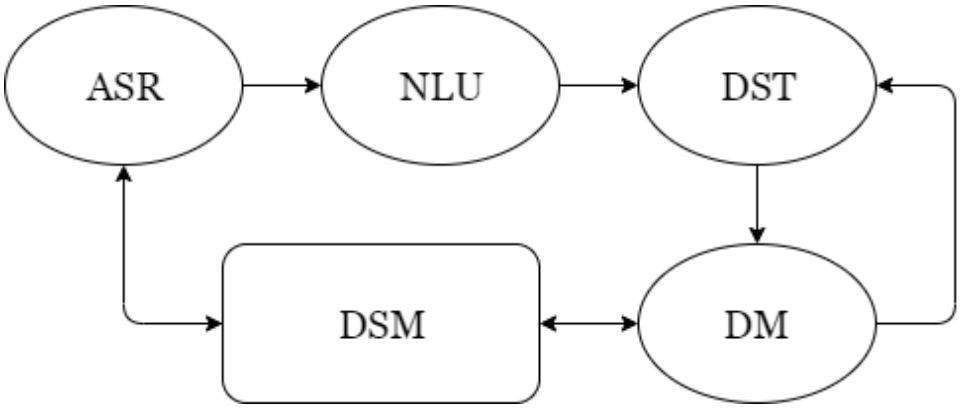


Figure 4.8: ROS graph of the dialogue system implementation. The ROS system is composed of five nodes: Automatic Speech Recognizer (ASR), Natural Language Understanding (NLU), Dialogue State Tracker (DST), Dialogue Manager (DM), and Dialogue State Machine (DSM), running simultaneously and communicating with one another.

NLU and Grounding models

Initially, we implemented the first version of the NLU models in `tensorflow` [74]. However, when we changed the approach for the NLU models, we decided to switch to `pytorch` [75], since we consider it easier for making modifications on pre-trained models, and customizing our own models. We implemented the grounding model using the `gensim` library [76]. Furthermore, all the developed code was done using `Python`.

Chapter 5

Results

In this chapter we discuss the quantitative results for the natural language understanding model. We will compare the performance of the model using two different transfer learning techniques: feature-extraction and fine-tuning, and then we will show the training, evaluation and validation results for the different models. Furthermore, we evaluate the complete dialogue system by analyzing real case conversations between the dialogue system and a user. In addition, we also demonstrate the main limitations of our implementation. Finally, we show some real conversations from the SciRoc2019 competition, where the dialogue system was implemented in a service robot.

5.1 Natural Language Understanding

The major problem when developing the dialogue system was the non-existence of any real data related to the tasks we want the robot to perform. The adopted solution was to design our own data generator to generate the training and test datasets. However, for validating the model, we used the available GPSR generator[1], which is the official generator used in the GPSR test of the RoboCup@Home competition. After generating the user utterances, they were manually labeled with intent labels and slot tags. Nevertheless, for dialogue act type classification, we were not able to generate any validation data, since the GPSR generator only generates text commands which correspond to a INFORM dialogue act type. The generated training, test, and validation sets contain 5000, 500, and 100 utterances, respectively. There are 11 slot labels, 10 dialogue act type labels, and 9 intent labels.

5.1.1 Metrics

The evaluation of the models was based on some standard classification metrics: precision, recall and F_1-score.

[1] https://github.com/kyordhel/GPSRCmdGen

Precision

In a classification task, the precision for a class is the number of true positives divided by the total number of elements predicted as belonging to that class:

$$\text{PRECISION} = \frac{tp}{tp + fp} \tag{5.1}$$

where tp denotes the true positives (i.e. the number of items correctly predicted as belonging to the class) and fp the false positives (i.e. the items that were incorrectly predicted as belonging to the class). Thus, precision reflects how precise the model is in identifying the true elements of a class, and, hence, a perfect precision model is a model that, when predicts an item as belonging to a certain class, those predictions are always right. However, the precision is not enough to evaluate the performance of a classification model. The model can have a perfect precision, but there may be a significant number of items that should be labeled as belonging to that class and are being classified as not belonging to the class (false negatives). Thus, we need to introduce the recall metric.

Recall

Recall indicates how many relevant items are selected, i.e. measures the proportion of the actual positives that are correctly classified as such. It is computed as the number of true positives divided by the total number of relevant elements (elements that in reality belong to the class):

$$\text{RECALL} = \frac{tp}{tp + fn} \tag{5.2}$$

where fn denotes the false negatives.

F1-Score

Together, the precision and recall metrics are capable of representing the performance of a classification model, since they can evaluate the proportion of items of a class that are correctly being predicted as belonging to that class. Usually, the precision and recall metrics can be represented together as the F_1 score, which computes their harmonic mean.

$$F_1 = 2 \cdot \frac{\text{PRECISION} \cdot \text{RECALL}}{\text{PRECISION} + \text{RECALL}} \tag{5.3}$$

Micro and Macro Averages

Since we are dealing with multi-class classification, we need to compute the average of the previous metrics. We can use both micro or macro-averages, which compute slightly different things. A macro-average computes the metric independently for each class and then take the average, i.e. this type of average treats all classes equally. The micro-average aggregates the contributions of all classes to compute the average metric.

Consider the following multi-class classification system with four classes and the following evaluation results:

- Class A: $1tp$ and $1fp$
- Class B: $10tp$ and $90fp$
- Class C: $1tp$ and $1fp$
- Class D: $1tp$ and $1fp$

First, we compute the precision for each class individually, yielding $\text{PRECISION}_A = \text{PRECISION}_C = \text{PRECISION}_D = 0.5$, whereas $\text{PRECISION}_B = 0.1$. If we compute a macro-average, then the resulting precision will be $\text{PRECISION} = \frac{0.5+0.1+0.5+0.5}{4} = 0.4$, whereas a micro-average will compute $\text{PRECISION} = \frac{1+10+1+1}{2+100+2+2} = 0.123$. These are quite different values for precision. Intuitively, in the macro-average, the higher precision of classes A, C and D is contributing to maintain the overall precision. While technically true, this value may be misleading since there is a high number of items that are not properly classified, which contribute only 1/4 towards the average, in spite of constituting 94.3% of the test data. The micro-average will adequately capture this class imbalance, and bring the overall precision down.

ROC curve

Traditionally, when we need to visualize the performance of a binary classification problem, we can use ROC (Receiver Operating Characteristics) curve. A ROC curve is a probability curve that represents the performance of a classification model at different classification thresholds. This curve plots two parameters: true positive rate (TPR), also called *recall*, and false positive rate (FPR). The false positive rate can be defined as:

$$fpr = \frac{fp}{fp + tn} \tag{5.4}$$

, where fp denotes the false negatives, and tn the true negatives.

We can extend this metric to a multi-class classification by using a one vs all methodology. Considering N number of classes, we plot N curves where each class is classified against all the others.

AUROC

The Area Under the Receiver Operating Characteristics (AUROC) measures the area underneath the ROC curve. Thus, AUROC provides an aggregate measure of performance across all classification possible thresholds, and hence represents a degree or measure of separability. Intuitively, it tells how much the model is capable of distinguishing between classes. A model with an excellent measure of separability will have an AUC near to the 1, while a model with no class separation at all will have an AUC near 0.5.

For a multi-class classification problem, since we will have N ROC curves (one per class), we can calculate the AUROC for each curve, and then compute the average.

5.1.2 Results

The NLU models uses the English uncased BERT-BASE model, which is composed of 12 layers, hidden states with a size of 768, and 12 self-attention heads. *BERT is pre-trained on BooksCorpus (800M words) and English Wikipedia (2500M words)* [23]. In Section 2.4, we have seen that we can use a pre-trained model both as feature-extractor or fine-tune it on a new task. We have decided to compare both approaches, and compare their performance.

Fine-tuning vs. Feature Extraction

Therefore, we trained both the intent detection (classification task) and slot filing (sequence labeling task) models using both approaches, and for each epoch we evaluated the models on the test set. We have decided not to analyze the dialogue act type classification model, since we already have a classification task represented in the intent detection analysis. Figure 5.1 shows the difference between the performance of the slot filing model, using the feature-extraction (EX) and fine-tuning (FT) approaches. The results for the intent detection model are presented in Chapter A (Figure A.1). The results were obtained by training the models several times using different combinations of hyper-parameters. It is clear that fine-tuning BERT during training leads to a significant increase in models performance, over the feature-extraction approach. Not only the models are able to achieve a greater F1-Score, but they also require a fewer number of training epochs.

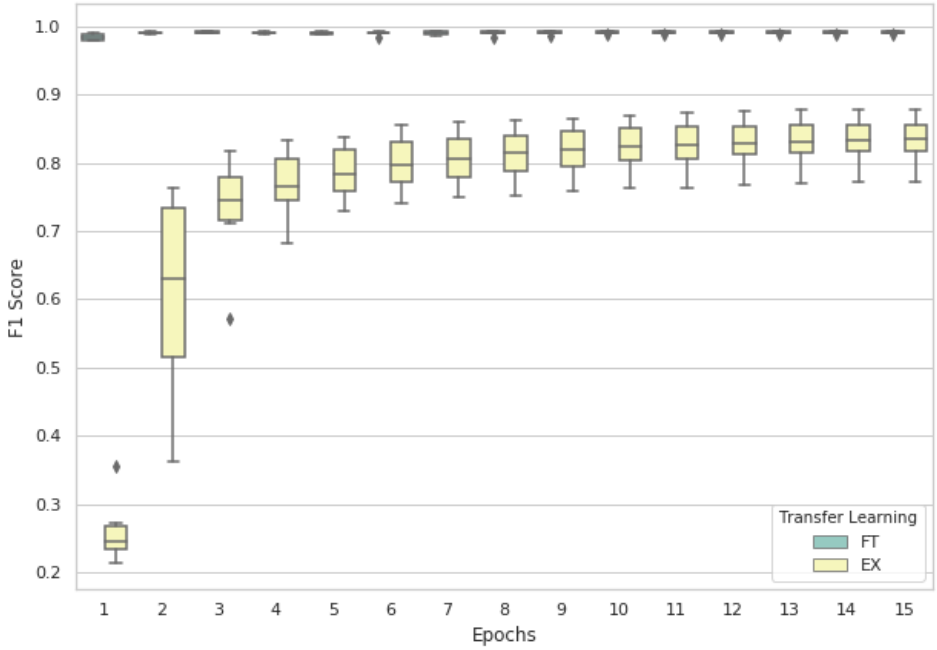


Figure 5.1: Fine-tuning (FT, green) vs feature extraction (EX, yellow) for slot filing.

BERT was pre-trained on both a mask language modeling (MLM), and a next sentence prediction (NSP) tasks, to learn context-representations for each input token and for a special token, [CLS], used for sentence classification. The fine-tuning approach can achieve a significantly better performance, because it can adapt the pre-trained representations to get better features for the new tasks.

After the previous analysis, we have decided to implement the fine-tuning technique for training the NLU models. For fine-tuning BERT, Devlin et al. [21] keeps most model hyper-parameters the same as in pre-training, with the exception of the batch size, learning rate and number of training epochs. Therefore, we will keep both the dropout probability and warm-up proportion at 0.1. In addition, we will use input sequences with a maximum sequence length of 15, and we will use Adam [9] as optimizer. The learning rate, batch size and number of training epochs are task-specific, but Devlin et al. [21] suggests the following range of possible values to work well across all tasks:

- Batch size: 16, 32.
- Learning rate: 5e-5, 3e-5, 2e-5.
- Number of epochs: 3, 4.

Hyper-parameter Search

Since fine-tuning is typically very fast, it is reasonable to run an exhaustive search over the above parameters. Thus, we train the models using combinations of the previous hyper-parameters, for a maximum number of epochs of 15, and for each epoch we save the correspondent model, and run the evaluation on the test set.

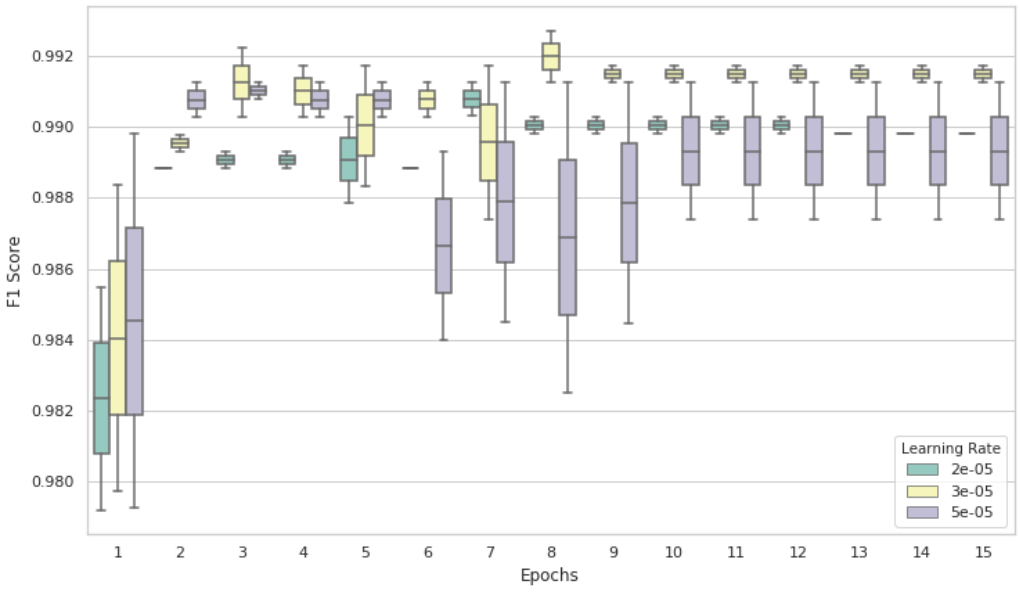


Figure 5.2: Performance results for the slot filing model using different learning rates (lr). (Green, lr $= 2e^{-5}$); (Yellow, lr $= 3e^{-5}$); (Purple, lr $= 5e^{-5}$)

Figure 5.2 shows the performance of the slot filing model for different learning rates. The results for the intent detection model are presented in Appendix A (Figure A.2). We verify that fine-tuning the model using a learning-rate of $3e^{-5}$ (yellow) yields a better performance for both the slot filing and intent detection models. For the batch size, Figure 5.3 shows that a batch size of 16 results in a better performance for the slot filing model. Again, the results for the intent detection model are presented in

Appendix A (Figure A.3). For the intent detection model, in contrast with slot filing, a batch size of 32 results in a better performance.

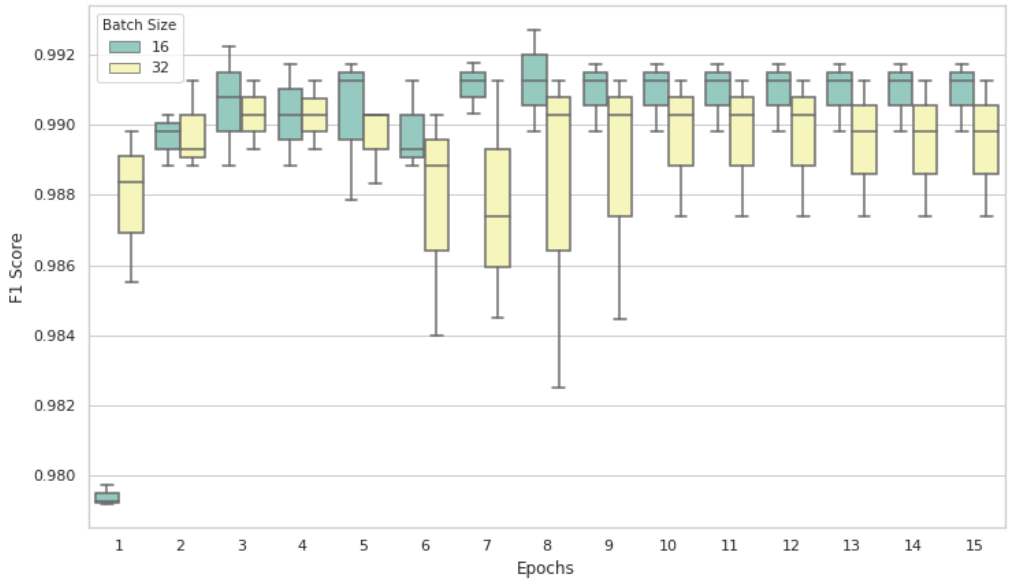


Figure 5.3: Performance results for the slot filing model using different batch sizes (bs). (Green, bs $= 16$); (Yellow, bs $= 32$)

After exhaustive searching over the previous parameters, we have decided to use the following hyper-parameters:

- Dropout rate: 0.1.
- Warm-up proportion: 0.1.
- Maximum sequence length: 15.
- Learning rate: 3e-5.
- Batch size: 16 (slot filing) and 32 (intent detection and dialogue act type classification)
- Number of epochs: 15, but saving a model for each epoch and running an evaluation over the test set on each epoch.

In addition, we also train the models multiple times, using a random parameters' initialization with different seeds. The training and validation results will be shown further.

Dialogue Act Type Classification

First, we trained the dialogue act type classification model. Figure 5.4 shows the evaluation performance of the model during training.

It is trivial to see that the model is able to predict the data really easy, with a perfect F_1 score after the first epoch. Intuitively, these results may be explained because of the BERT pre-training. BERT

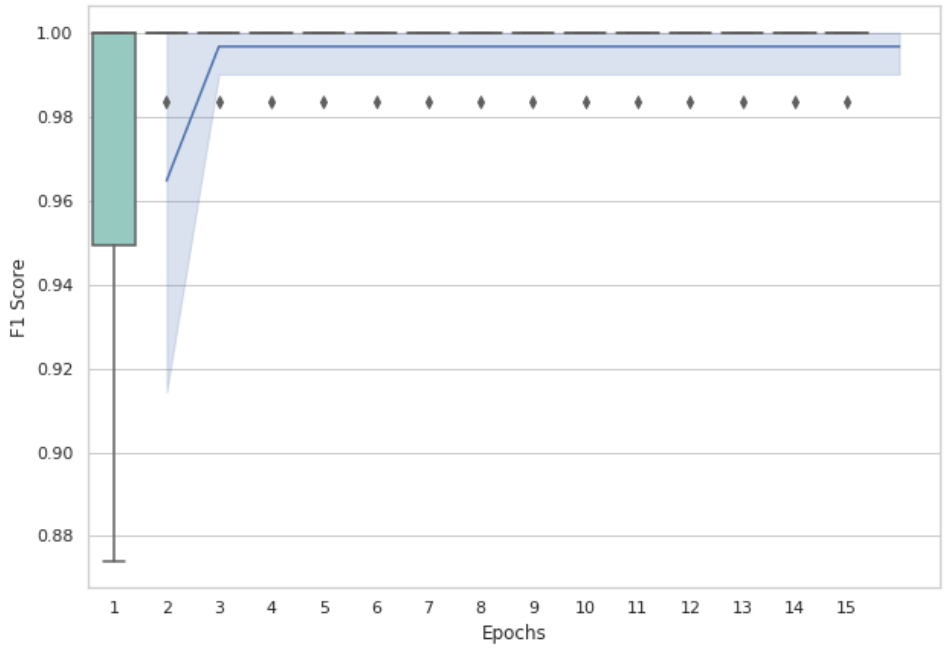


Figure 5.4: Dialogue act type classification model evaluation on test set during training

represents the input sentence as a context-dependent feature vector. Since BERT was already pre-trained on a large text corpus, it only needs some slightly adaptations on the pre-trained representations for the final classifier layer to be able to easily split the different classes.

As referred earlier, we were not able to generate a validation set for validating the dialogue act type classification model. However, several tests were performed to check the model's performance, where we tested different type of user commands and checked the model's predictions. Since usually the user commands reflecting confirmations, negations, repetitions, and greetings are fairly simple, the model was able to correctly identify the different dialogue act types of user commands.

Intent Detection

Next, we trained the intent detection model. Similarly to the dialogue act type classification model, this model was also able to achieve really good results after only one epoch (see Figure 5.5).

In contrast with the dialogue act type classification, for intent detection we were able to validate the model on a validation dataset. As referred before, this validation data was generated using the official GPSR command generator, and then manually labeled with the different intent types. The validation results are shown on Table 5.1 and Figure 5.6. Table 5.1 shows that the model is performing really well, with an overall F_1 score of 97.1%. However, it is also clear that the intent labels TAKE and GUIDE are the cases where the model is performing worse, with an F_1 score of 94.1% and 89.7%, respectively. A deeper analysis of the validation dataset indicates that the problem could be related to the fact that both TAKE and GUIDE commands can be represented with the verb take, e.g. *"take the book from the table"* and *"take John to the living room"*. However, by analyzing the model predictions on the validation set, we verified that the model was capable of distinguish between both types of intents, taking into account

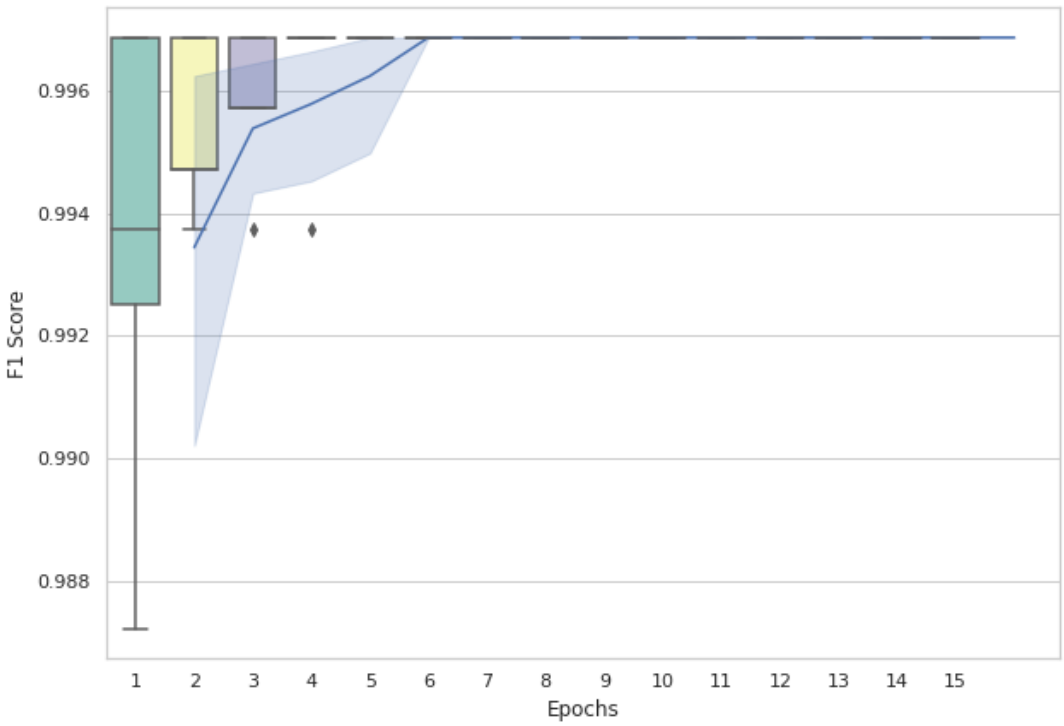


Figure 5.5: Intent detection model evaluation on test set during training.

that a TAKE intent command is usually highly correlated with an OBJECT, while a GUIDE intent command is correlated with a PERSON. Intuitively, this means that the self-attention mechanism from BERT is able to model these correlations. Nevertheless, since the validation set is generated using a more complex generator than the training and test sets, it contains more general and unusual commands, such as *"take the pointing right person to the kitchen cabinet"* or *"escort the sitting person to the bathroom"*. We noted that, in both this specific commands, and others similar to them, the model was not capable of labeling them as a GUIDE intent, because it was not interpreting *"the pointing right person"* as a word sequence describing a PERSON.

Figure 5.6 shows the ROC curve. For every class, the AUROC is 1.0 (or real close to 1.0), which means that the model has an excellent measure of separability.

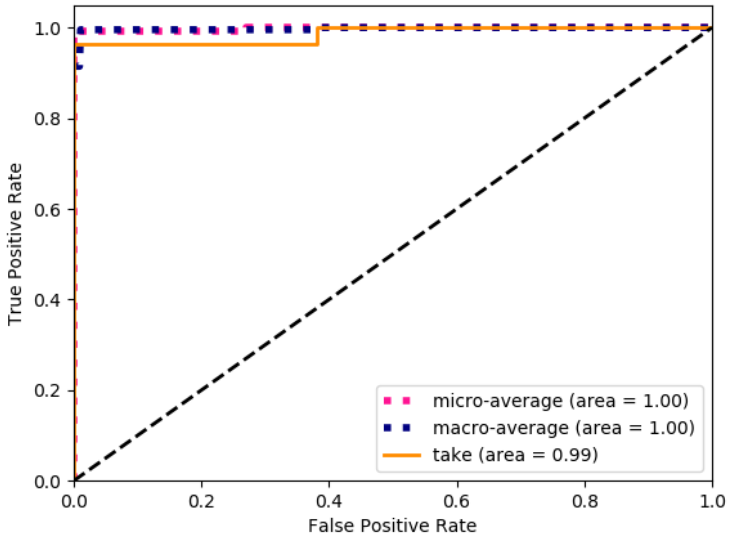


Figure 5.6: ROC curve for the intent detection model performance on validation data using a one vs all methodology. Classes with an AUROC of 1.0 are omitted for visualization.

Table 5.1: Validation results for the intent detection.

	Precision	Recall	F1
TELL	1.0000	1.0000	1.0000
FIND	1.0000	1.0000	1.0000
GO	1.0000	1.0000	1.0000
TAKE	0.9412	0.9412	0.9412
GUIDE	0.9286	0.8667	0.8966
FOLLOW	1.0000	1.0000	1.0000
ANSWER	1.0000	1.0000	1.0000
PLACE	1.0000	1.0000	1.0000
Micro Average	0.9770	0.9659	0.9714
Macro Average	**0.9765**	**0.9659**	**0.9710**

Slot Filing

Finally, we trained the last model, the slot filing model, whose evaluation results are shown on Figure 5.7. After the first/second epochs, the model is already performing really well. We saved the model correspondent to the 10th epoch (purple), since it is when the F_1 score stabilizes.

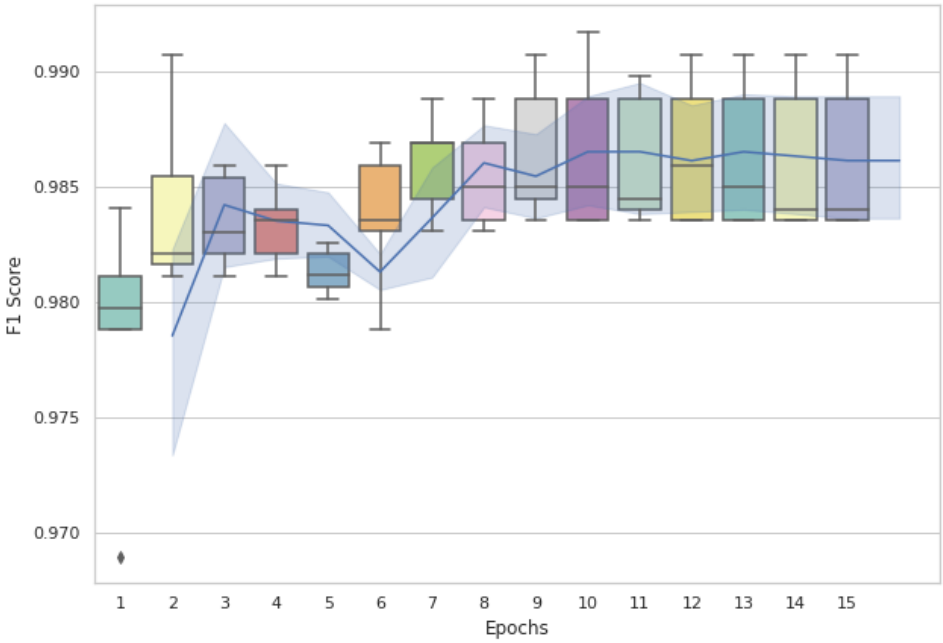


Figure 5.7: Slot filing model evaluation on test set during training.

Next, we validated the model, yielding the results represented on Table 5.2 and Figure 5.8. Table 5.2 shows that the model is having some difficulty labeling the words that correspond to the slot WHAT-TO-TELL. Usually, in the user commands that ask the robot to tell something, the sequence of words

that indicate what the robot must say have an high length. For example, consider the command "please tell me the day of the month". To the slot WHAT-TO-TELL, must correspond the value "the day of the month", which has a length of 5. However, the model predicts this by labeling each word individually, using BIO tagging, i.e. the word sequence ("the", "day", "of", "the", "month") must be predicted with labels (B-WHAT_TO_TELL, I-WHAT_TO_TELL, I-WHAT_TO_TELL, I-WHAT_TO_TELL, I-WHAT_TO_TELL). Sometimes, the model is predicting the label B-WHAT_TO_TELL instead of I-WHAT_TO_TELL, which is causing the performance degradation, since the model considers them as different labels. In addition, we can also verify that the slot PERSON has a lower recall compared with the model overall performance. This is due to the same problem described in the intent detection model, with user utterances similar to *"take the pointing right person to the kitchen cabinet"*, where the model is not predicting the tokens with the label PERSON, and hence the recall drops. Figure 5.8 shows that the slot filing model also has an excellent measure of separability.

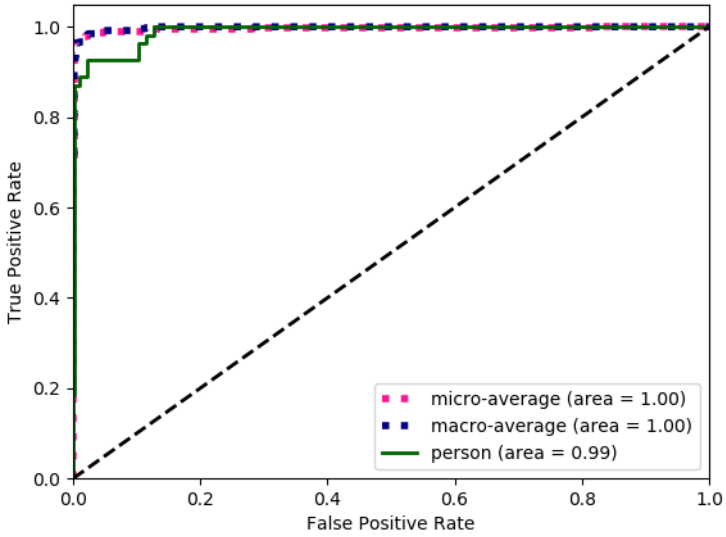


Figure 5.8: ROC curve for the slot filing model performance on validation data using a one vs all methodology. Classes with an AUROC of 1.0 are omitted for visualization.

Table 5.2: Slot filing model results on validation data.

	Precision	Recall	F1
PERSON	0.9792	0.8704	0.9216
OBJECT	0.9444	1.0000	0.9714
SOURCE	1.0000	0.9355	0.9667
DESTINATION	0.9577	1.0000	0.9784
WHAT-TO-TELL	0.9000	0.9000	0.9000
Micro Average	0.9639	0.9492	0.9565
Macro Average	**0.9650**	**0.9492**	**0.9558**

Overview

When we designed our own data generator for both dialogue act type classification, intent detection and slot filing, we did not analyze the grammars that the official GPSR command generator (which was used for the validation set) was using, since we wanted the validation set to be composed of different and more complex user commands. Thus, the grammars that compose our generator are far more simple. Table 5.3 shows a summary of the training, test and validation results for both the three models. When evaluated on the test set, the models achieve really good results, which was already expected since they were evaluated on a dataset generated with the same generator of the training set. However, the validation set allowed us to check if the model was not over-fitting to the training data. The results on the validation set are still very good, showing that the model is performing and generalizing well. However, the validation data cannot be seen as a representation of the real world data distribution.

Thus, the validation of the NLU models on a real world dataset would represent a better indicator for the performance and generalization capability of the model.

Table 5.3: NLU model results.

Task	Precision			Recall			F1		
	Train	Test	**Val**	Train	Test	**Val**	Train	Test	**Val**
Dialogue act type classification	1.0000	1.0000	**-**	1.0000	1.0000	**-**	1.0000	1.0000	**-**
Intent detection	1.0000	1.0000	**0.9765**	1.0000	1.0000	**0.9659**	1.0000	1.0000	**0.9710**
Slot filing	0.9984	0.9941	**0.9650**	0.9990	0.9894	**0.9492**	0.9987	0.9916	**0.9558**

5.2 Dialogue System

After training and validating the natural language understanding model, and since the other developed components are rule-based, we decided to analyze the complete dialogue system, rather than analyzing each component individually. To analyze the entire dialogue system, we will show some real conversations between a user and the system. Note that our developed dialogue system does not contain a Natural Language Generation (NLG) component (recall the typical dialogue-state architecture described in Section 3.1). Thus, we hard-coded some text responses, regarding the system dialogue acts generated by the Dialogue Manager (DM).

This first example of conversation highlights the advantage of using a dialogue system that tracks multiple recognition hypothesis. In this first example, the user wants the robot to bring him an apple from the kitchen cabinet. Table 5.4 represents the conversation between the user and the dialogue system.

Table 5.4: First conversation example.

SYSTEM	Hello. How can I help you?
USER	Bring me an apple.
SYSTEM	From where?
USER	From the kitchen cabinet.
SYSTEM	Do you want me to deliver you an apple from the kitchen cabinet?
USER	Yes.
SYSTEM	Ok, I will do it.

We can verify that the dialogue system was capable of easily handling the conversation, requesting more information about the task, and asking for a confirmation. However, let us take a deeper look on the internal states of the system. Figure 5.9 shows the output of the dialogue system components for the first turn of the conversation. Note that the confidence scores are rounded to the first decimal place. Looking at the ASR output, we can see that the 1-best recognized hypothesis is *"bring me an airport"*, which is not what the user said. However, since our dialogue system is looking for the first 5 best hypothesis, it is still able to extract the correct information from the other hypothesis, particularly from the second one, which corresponds to the correct speech command transcription.

The NLU output holds the information extracted from each of the recognized hypothesis. Although they contain some user acts with the incorrect value for slot OBJECT, such as $airport$ and $air\ pump$, the grounding model will parse those values and check if they represent entities that exist in the system's knowledge base. Since not, the DST did not consider them as relevant and only tracked the correct OBJECT value, $apple$, in the dialogue state.

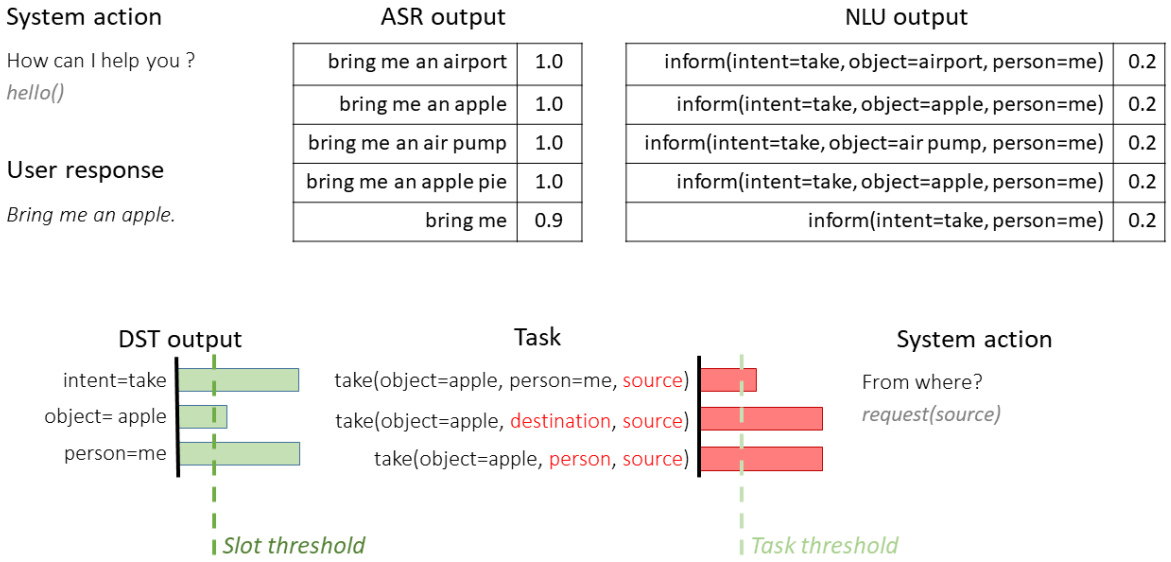


Figure 5.9: First dialogue turn of the first conversation example. The dialogue system begins the conversation by generating an HELLO() message. Then, the user asks the robot to bring him an apple. The dialogue system is able to successfully recognize all the information presented in the command, and generates the system response REQUEST(SOURCE), since it needs the SOURCE slot to complete the task.

The previous example highlights the robustness of our spoken dialogue system against noise in the speech recognition. In contrast with conventional dialogue systems, which consider only the 1-best ASR/NLU result and track a single hypothesis for the dialogue state, our approach is still able to understand the user speech command, even with ASR hypothesis that do not correspond to that command. Furthermore, the previous conversation also highlights the advantage of using a grounding model. Although the NLU predicts the values $airport$, $air\ pump$ and $apple$ as objects, the knowledge base of the system does not contain any entities that can be described by the names $airport$ and $air\ pump$. Therefore, the dialogue system will filter the $airport$ and $air\ pump$ values, and will only consider $apple$ as an object in the dialogue state.

Now, consider the following example, in which the user asks the robot to do same task. Table 5.5 represents the conversation between the user and the dialogue system.

Although the conversation seems similar to the previous example, the dialogue system is not capable of handling the conversation after the second turn, and therefore cannot perform the task. Again, let us take a deeper look on the internal states of the system. Figure 5.10 shows the output of the dialogue system components for the second turn of this second conversation.

By looking at the ASR output, we verify that the hypothesis are similar to each other, and correspond

Table 5.5: Second conversation example.

SYSTEM	Hello. How can I help you?
USER	Bring me an apple.
SYSTEM	Where is the apple?
USER	It is on the kitchen cabinet.
SYSTEM	Where is the apple?
USER	On the kitchen cabinet.
SYSTEM	Where is the apple?
...	...

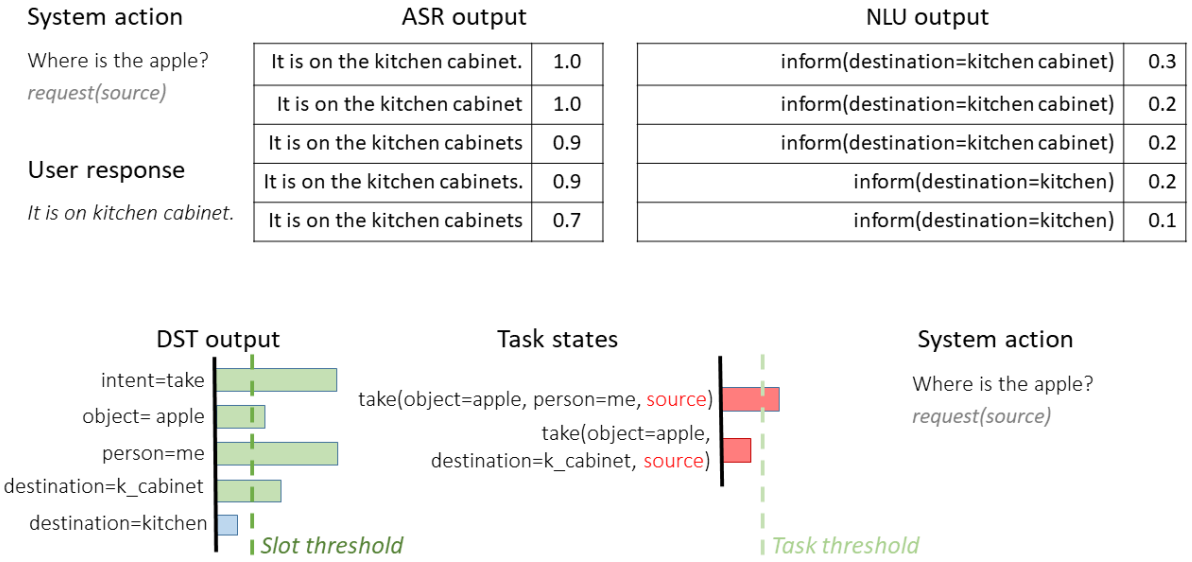


Figure 5.10: Second turn of the second conversation example. After requesting the source, REQUEST(SOURCE), the dialogue system receives a user speech utterance indicating the kitchen cabinet as the location of the apple. Although, due to the NLU bad performance, the dialogue system cannot fill the SOURCE slot, and thus will request it again.

to the real user utterance. However, when looking at the NLU output, we see that the NLU is assigning the values $kitchen\ cabinet$ and $kitchen$ to the DESTINATION slot, while it should be assigning those values to the SOURCE slot. Thus, since in the following turns the user continues to use similar utterances, the dialogue system will enter an "infinite" loop because it does not recognize the SOURCE slot and will continue to request it.

The dialogue state tracking (DST) component used in our dialogue system is based on simple and generic heuristic rules, which rely entirely on the believability of the NLU hypothesis. Therefore, although our dialogue system may be more robust to speech recognition errors, it is still limited by the NLU performance. This NLU limitation is, mainly, due to the fact that the NLU models are only using the user utterances as input features. Intuitively, we could counteract such limitation by using the previous system responses as input features, together with the user utterances.

5.3 SciRoc Competition

The previous examples of conversations highlight some advantages of our dialogue system implementation over more simple approaches. However, they were produced in a simulation environment by the dialogue system designers. Thus, we needed a way to test the complete system in a real world scenario. The SciRoc 2019 competition, held in Milton Keynes, provided this opportunity. The dialogue system was implemented on the robot used for this competition. However, some minor modifications had to be made, since the task was slightly different from the typical GPSR tasks, which took approximately two days to be done. In addition to test the complete system, this competition also enabled us to check if our NLU models are, indeed, generalizing well.

In the SciRoc task, the robot must be able to serve customers in a coffee shop, and therefore recognize their orders. Each table, with one or more customers, can order three different items. Thus, for each table, the robot must annotate the orders, composed of three different objects. The slot filing model was already capable of recognizing objects, hence it was not necessary to modify the NLU component. However, we needed to change the domain ontology to incorporate the items in the coffee shop menu, so that the grounding model could recognize them. In addition, we needed to change the task ontology to include a task ORDER template, so that the dialogue management could use this task template to compute the task state and generate the system responses. Finally, since our intent detection model has not been trained to recognize ORDER commands, we did not use it. On the other hand, when computing the user dialogue acts, we force the NLU model to always classify the INTENT with the label ORDER, with a confidence score of 1.0.

Next, we will show an example of a real conversation[2] from the SciRoc competition, in which the robot was able to successfully take an order from a customer in a coffee shop environment. Table 5.6 shows the conversation between the user and the dialogue system.

Table 5.6: SciRoc first conversation example.

SYSTEM	Hello. Today in the menu we have (...). What would you like to order?
USER	I would like one cappuccino, one water, one crisps.
SYSTEM	Your order is water, cappuccino and tyrell crisps?
USER	Correct.
SYSTEM	Your order is water, cappuccino and tyrell crisps?
USER	Yes.
SYSTEM	Good choice. I will get it right away.

We verify that the dialogue system is able to successfully take the order. However, in the second turn of the conversation, although the user has explicitly confirmed the order, the system generated the confirmation again. Similarly to the previous examples, let us take a look on the internal states of the dialogue system. Figure 5.11 shows the output of the dialogue system components for the second turn of the conversation.

By looking at the ASR output, we verify the existence of an hypothesis with a complete opposite

[2] A video of the full conversation is available at https://youtu.be/FGTSXI_By7s

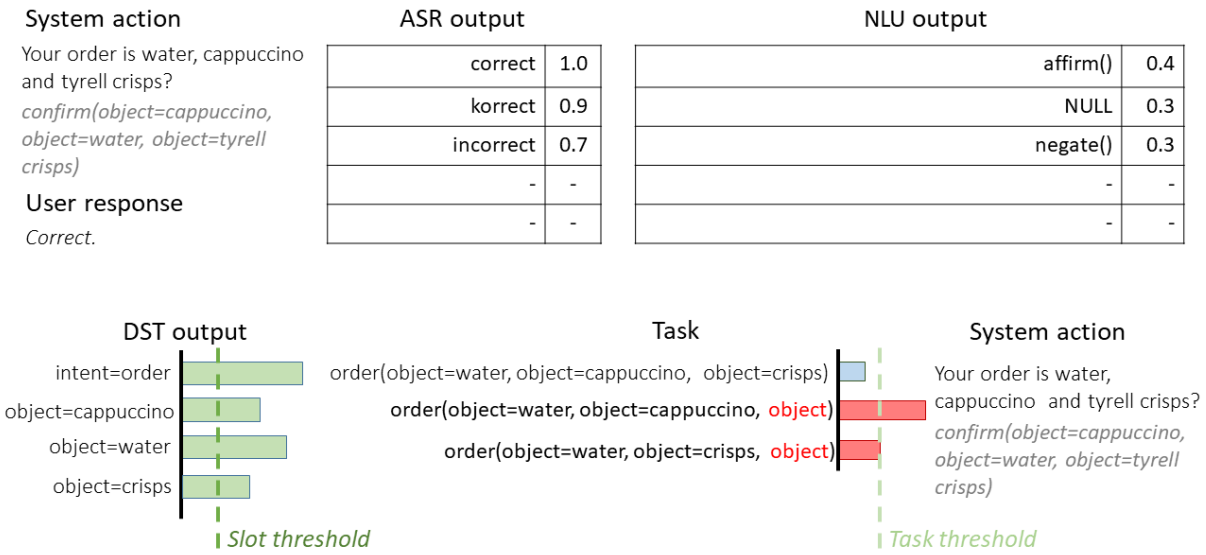


Figure 5.11: Second dialogue turn of the SciRoc trial example. After generating a confirmation action, the system receives an affirmation user utterance. However, some noise in the speech recognition makes it recognize an hypothesis representing a negation. The negative effect of this hypothesis counteracts the positive effect of the affirmation. Therefore, the system remains unsure about the information and generates the confirmation action again.

effect of the real speech utterance. This negative effect will propagate to the downstream components of the dialogue system, which prevents the dialogue system to increase its confidence on the task information (i.e. the slots that are presented on the confirmation action). Therefore, the system generates a confirmation action again.

This conversation example from the SciRoc competition demonstrates another limitation of the dialogue system. In the DST implementation, we made the rules for our belief tracker as generic as possible, to ensure the generality of the component. However, in practice, it is reasonable to use more detailed rules to address additional phenomena in a particular system [44]. Thus, one way to overcome the previous limitation is to introduce one or more rules that are able to disambiguate when a list of user acts contains both AFFIRM() and NEGATE() hypothesis.

Table 5.7 contains another example from the SciRoc competition, in which the user ordered one item on each turn. The dialogue system was able to always recognize the item asked each turn. However, we were not able to get any video records of this example.

In summary, during the trials, the dialogue system was able to take the orders from the customers. This reflects the good generalization capability from the NLU models, particularly the slot filing model, which is able to label multiple objects in the same utterance, without even seeing this type of sentences during training (the training data is composed of sentences that contain at most one object). Besides that, it is also capable of labeling objects that were not part of the training data. However, during some of the trials, the internet connection was significantly slow. Since our dialogue system needs to connect to the Google cloud to use the speech recognition, most of the times this connection would

Table 5.7: SciRoc second conversation example.

SYSTEM	Hello. Today in the menu we have (...). What would you like to order?
USER	I want a wrap.
SYSTEM	Besides the wrap, what do you want more?
USER	An espresso.
SYSTEM	Besides the espresso and the wrap, what do you want more?
USER	A smoothie.
SYSTEM	Your order is espresso, smoothie and wrap?
USER	Yes.
SYSTEM	Good choice. I will get it right away.

take a really long time, activating the timeout of the WAIT FOR RECOGNITION state in the dialogue state machine (recall Section 4.6.1). Therefore, in some of the trials, the dialogue system was not able to operate normally because it was consistently getting timeouts in the speech recognition, leading to dialogue failures. However, the robot was still able to complete the task because, when the dialogue state machine returns a DIALOGUE FAILURE state, the robot launches a menu in the screen in which the customers can select their orders.

Chapter 6

Conclusions

In this chapter we describe the main achievements and contributions of our work. Furthermore, we also discuss a few limitations of our system. Finally, we refer some modifications and future work that can improve or complement our work.

6.1 Achievements

The first main contribution of this work is the development of an NLU component, which takes advantage of using a state-of-the-art pre-trained language model, BERT (Bidirectional Encoder Representations from Transformers). BERT is pre-trained on a large corpus, and then fine-tuned on our specific tasks. Therefore, our NLU models can be trained faster and using a smaller number of training samples. In addition, these models also yield a better performance and generalization capability, since they are making use of transfer learning techniques, which are shown to allow our models to generalize better [7]. Another contribution, still related to the NLU component, is the development of a grounding model. This model enables the NLU to assign the labels predicted by the slot filing model to entities that are known to the system. This way, the NLU can interpret the instructions taking into account grounded information [33].

Another main contribution of this work is the implementation of a dialogue state tracking (DST) component. The DST is based on a simple and generic rule-based belief tracker, which maintains beliefs over marginal representations of user goals [44]. Although simple, this is a core component of the dialogue system, since it enables the system to track multiple hypothesis and maintain beliefs over dialogue states. At the same time, it is also a crucial component, because the dialogue policy relies on the estimated dialogue state to choose actions.

The final major contribution is the design of a dialogue management (DM) module. The DM uses hand-crafted rules to map dialogue states to system actions. Thus, this component enables the dialogue system to generate responses to request more information from the users or ask for clarifications. In addition, this module is also composed of a dialogue state machine (DSM), which controls the flow of the conversation.

The main purpose of this work was to develop a spoken goal-oriented dialogue system capable of dealing with the uncertainty in the speech recognition. By implementing a speech recognizer capable of recognizing multiple hypothesis, and designing a DST capable of tracking multiple hypothesis for the dialogue state, we are able to deal with such uncertainties, which makes our system robust to errors in speech recognition. In addition, the DM enables the dialogue system to exploit the sequential natural of dialogue to disambiguate in the presence of errors, by asking for clarifications and requesting more information from the users.

As final remark, we were able to test the dialogue system in a real world environment. We implemented the system in a robot that participated in the SciRoc 2019 competition, held in Milton Keynes. The robot had to serve customers in a coffee shop environment, and hence one of its main tasks was to take orders from customers using speech interaction. In general, our dialogue system handled most of the conversations and was able to successfully take the orders.

6.2 Current Limitations and Future Work

In general, the developed spoken dialogue system performs well in simple environments, where most of the entities are known and can be defined in the knowledge base of the system. Nevertheless, it is still restricted by the NLU performance, since it relies entirely on the semantic information extracted by the NLU for computing the dialogue state. Although performing and generalizing well, the NLU still holds some limitations, particularly because the models were trained using generated datasets and not real world data. Besides that, we are only using the user utterances as input features for the models. Including some system actions from previous dialogue turns in the features may help overcoming some of those limitations. Another modification that could improve the NLU performance is to jointly model the dialogue act type classification, intent detection and slot filing tasks. By jointly modeling the three tasks, we enable the model to exploit the relationships between them. Finally, we can also explore the use of a structured predictor in the slot filing model. Since this model is performing sequence labeling, the use of a structured predictor such as a conditional random field (CRF) can help modeling the relationships between the predictions. However, there are some previous works, particularly Chen et al. [23], which explored the use of structured predictors for intent detection and slot filing, using BERT. They have shown that there is no improvement in performance by combining BERT with CRF. Intuitively, *this is probably due to the self-attention mechanism in Transformer, which may have successfully modeled the label structures* [23].

Another limitation of our dialogue system is the inability to disambiguate between concurrent hypothesis, i.e. when the system receives a list of user acts with hypothesis that are usually mutually exclusive (e.g. AFFIRM() and NEGATE()), it must be able to decide which of the hypothesis is more probable and discard the other one. This can be achieved by introduced more detailed rules in the DST.

In our system, both the dialogue state tracking (DST) and dialogue management (DM) are simple rule-based components, designed for an initial dialogue system installation. Using our dialogue system, we are now able to collect real world conversations. This dialogue dataset can then be used to train

statistical discriminative models for DST. Furthermore, the dialogue dataset can be used to design a user simulator, so we can train a DM using a reinforcement learning approach.

In summary, the following topics are recommended for future work:

- Build a dataset with real world user commands for this particular domain.
- Explore the jointly modeling of the dialogue act type classification, intent detection and slot filing tasks.
- Explore structured prediction, by combining BERT with a conditional random field (CRF).
- Include the dialogue system responses as input features for the NLU model, jointly with the user utterances.
- Include the influence of phonetic similarity, jointly with the semantic similarity, in the grounding model [72].
- Modify the DST to include more detailed rules to address additional phenomena.
- Build a dataset with real world conversations for this particular domain, using the developed dialogue system.
- Use a dataset of real world conversations to train a statistical discriminative model for dialogue state tracking (DST).
- Design a user simulator to train a dialogue management (DM) model using reinforcement learning.

Bibliography

[1] D. Jurafsky and J. H. Martin. *Speech and Language Processing- An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 3 edition, 2008.

[2] J. Williams, A. Raux, and M. Henderson. The dialog state tracking challenge series: A review. *Proc SIGdial Conf on Discourse and Dialogue*, 7(3):4–33, 2016.

[3] M. Henderson. *Discriminative Methods for Statistical Spoken Dialogue Systems*. PhD thesis, University of Cambridge, 2015.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] S. Young, M. Gasic, B. Thomson, and J. D. William. POMDP- based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, PP(99):1–20, 2013.

[6] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3): 378–423, 1948.

[7] S. Ruder. *Neural Transfer Learning for Natural Language Processing*. PhD thesis, National University of Ireland, Galway, 2019.

[8] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

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

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

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

[12] J. N. Graves, A. and A. Mohamed. Hybrid speech recognition with deep bidirectional lstm. *Automatic Speech Recognition and Understanding (ASRU)*, 2013.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[14] G. Miller and W. Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991.

[15] M. Baroni, G. Dinu, and G. Kruszewski. *Don't count, predict!* A systematic comparison of context-counting vs. context-predicting semantic vectors. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 1:238–247, June 2014.

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

[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*, 2013a.

[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*, 2013b.

[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] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *Proceedings of NAACL*, 2018.

[21] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[22] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding with unsupervised learning. *Technical report, OpenAI*, 2018.

[23] Q. Chen, Z. Zhuo, and W. Wang. Bert for joint intent classification and slot filling. 02 2019.

[24] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

[25] S. Arora, K. Batra, and S. Singh. Dialogue system: A brief review. *CoRR*, abs/1306.4134:2–5, 2013.

[26] S. Seneff, L. Hirschman, and V. W. Zue. Interactive problem solving and dialogue in the ATIS domain. *Proceedings of the Workshop on Speech and Natural Language, Pacific Grove, California*, pages 354–359, 1993.

[27] A. L. Gorin, G. Riccardi, and J. H. Wright. How may I help you? *Speech Communication*, 23(1): 113–127, 1997.

[28] V. Zue, S. Seneff, J. R. Glass, J. Polifroni, C. Pao, T. J. Hazen, and L. Hetherington. JUPITER: A telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1):85–96, 2000.

[29] M. Walker, J. Aberdeen, J. Boland, E. Bratt, J. Garafolo, L. Hirschman, A. Le, S. Lee, S. S. Narayanan, K. Papineni, B. Pellom, J. Polifroni, A. Potamianos, P. Prabhu, A. I. Rudnicky, G. Sanders, S. Seneff, D. Stallard, and S. Whittaker. DARPA communicator dialog travel planning systems : The June 2000 data collection. *Proceedings of 2001 European Conference on Speech Communication and Technology*, 2001.

[30] J. Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, 1966.

[31] P. Norvig. Paradigms of artificial intelligence programming. *Morgan Kaufmann Publishers, New York*, pages 151–154, 1992.

[32] X. Wang and C. Yuan. Recent advances on human-computer dialogue. *CAAI Transactions on Intelligence Technology*, 1(4):303–312, 2016.

[33] P. Martins. Human-robot speech interaction for service robots. Master's thesis, Instituto Superior Técnico, Lisboa, 2017.

[34] 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.

[35] D. G. Bobrow, R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd. GUS, a frame-driven dialog system. *Artificial Intelligence*, 8:155–173, 1977.

[36] S. Seneff. TINA: A natural language system for spoken language applications. 1992.

[37] Y. Wang and A. Acero. Discriminative models for spoken language understanding. *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2006.

[38] F. Mairesse, M. Gašic, F. Jurcicek, S. Keizer, B. Thomson, K. Yu, and S. Young. Spoken language understanding from unaligned data using discriminative classification models. *International Conference on Acoustics, Speech, and Signal Processing. IEEE*, 2009.

[39] D. Zhou and Y. He. Learning conditional random fields from unaligned data for natural language understanding. *Advances in Information Retrieval, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg*, (11):283–288, 2011.

[40] Y. Z. D. Y. G. Z. Kaisheng Yao, Baolin Peng and Y. Shi. Spoken language understanding using long short-term memory neural networks. *In 2014 IEEE Spoken Language Technology Workshop, SLT 2014, South Lake Tahoe, NV, USA*, pages 189–194, 2014.

[41] B. Liu and I. Lane. Attention-based recurrent neural network models for joint intent detection and slot filling. *In Interspeech, San Francisco, CA, USA*, pages 685–689, 2016.

[42] S. Larsson and D. Traum. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 5(3/4):323–340, 2000.

[43] S. Pulman. Conversational Games, Belief Revision and Bayesian Networks. *CLIN VII: 7th Computational Linguistics in the Netherlands meeting*, 1996.

[44] Z. Wang and O. Lemon. A simple and generic belief tracking mechanism for the dialog state tracking challenge: On the believability of observed information. *Proc SIGdial Conf on Discourse and Dialog, Metz, France*, 2013.

[45] K. Sun, L. Chen, S. Zhu, and K. Yu. A generalized rule based tracker for dialogue state tracking. *Proc IEEE Workshop on Spoken Language Technologies (SLT), South Lake Tahoe, Nevada, USA*, 2014a.

[46] K. Sun, L. Chen, S. Zhu, and K. Yu. The SJTU system for dialog state tracking challenge 2. *Proc SIGdial Conf on Discourse and Dialogue, Philadelphia, USA*, 2014b.

[47] M. Henderson. Machine learning for dialog state tracking: A review. *Proceedings of The First International Workshop on Machine Learning in Spoken Language Processing*, 2015.

[48] J. D. Williams. Exploiting the ASR N-best by tracking multiple dialog state hypotheses. *Proc INTERSPEECH Conf, Brisbane, Australia*, 2008.

[49] D. Heckerman and O. Lemon. Inferring informational goals from free-text queries: a bayesian approach. *Proc 14th Conf on Uncertainty in Artificial Intelligence (UAI)*, pages 230–238, 1998.

[50] H. Meng, C. Wai, and R. Pieraccini. The use of belief networks for mixed-initiative dialog modelling. *IEEE Transactions of Speech and Audio Processing*, 11(6):757–773, 2003.

[51] S. Young, J. Schatzmann, K. Weilhammer, and H. Ye. The hidden information state approach to dialog management. *Proc Int Conf on Acoustics, Speech and Signal Processing (ICASSP), Honolulu, Hawaii*, pages 149–152, 2007.

[52] J. Henderson and O. Lemon. Mixture model POMDPs for efficient handling of uncertainty in dialogue management. *Proc Association for Computational Linguistics Human Language Technologies (ACL-HLT), Columbus, Ohio, USA*, pages 73–76, 2008.

[53] J. D. Williams. Incremental partition recombination for efficient tracking of multiple diaogue states. *Proc Intl Conf on Acoustics, Speech and Signal Processing (ICASSP), Dallas, Texas, USA*, pages 5382–5385, 2010.

[54] M. Gašić and S. Young. Effective handling of dialogue state in hidden information state POMDP dialogue manager. *ACM Transactions on Speech and Language Processing*, 7(3), 2011.

[55] J. D. Williams. Using particle filters to track dialog state. *Proc. IEEE Workshop on Automatic Speech Recognition Understanding (ASRU), Kyoto, Japan*, pages 502–507, 2007.

[56] B. Thomson and S. Young. A POMDP framework for spoken dialog systems. *Compututer, Speech and Language*, 24(4):562–588, 2010.

[57] J. D. Williams. A critical analysis of two statistical spoken dialog systems in public use. *Proc IEEE Workshop in Spoken Language Technology (SLT)*, 2012.

[58] D. Bohus and A. Rudnicky. A 'K hypothesis + other' belief updating model. *Proc AAAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems, Boston, USA*, 2006.

[59] A. Metallinou, D. Bohus, and J. D. Williams. Discriminative state tracking for spoken dialog systems. *Proc Association for Computational Linguistics (ACL), Sofia, Bulgaria*, 2013.

[60] J. D. Williams. Web-style ranking and SLU combination for dialog state tracking. *Proc SIGdial Conf on Discourse and Dialogue, Philadelphia, USA*, 2014.

[61] M. Henderson, B. Thomson, and S. Young. Deep neural network approach for the dialog state tracking challenge. *Proc SIGdial Conf on Discourse and Dialogue, Metz, France*, 2013.

[62] S. Lee and M. Eskenazi. Recipe for building robust spoken dialog state trackers: Dialog state tracking challenge description. *Proc SIGdial Conf on Discourse and Dialogue, Metz, France*, 2013.

[63] M. Henderson, B. Thomson, and S. Young. Word-based dialog state tracking with recurrent neural networks. *Proc SIGdial Conf on Discourse and Dialogue, Philadelphia, USA*, 2014d.

[64] J. D. Williams. Multi-domain learning and generalization in dialog state tracking. *Proc SIGdial Conf on Discourse and Dialogue, Metz, France*, 2013.

[65] M. Henderson, B. Thomson, and S. Young. Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. *Proc IEEE Workshop on Spoken Language Technologies (SLT), South Lake, Tahoe, Nevada, USA*, 2014e.

[66] J. D. Williams, A. Raux, D. Ramachandran, and A. Black. The dialog state tracking challenge. *Proc SIGdial Conf on Discourse and Dialogue, Metz, France*, 2013.

[67] M. Henderson, B. Thomson, and S. Young. The second dialog state tracking challenge. *Proc SIGdial Conf on Discourse and Dialogue, Philadelphia, USA*, 2014b.

[68] M. Henderson, B. Thomson, and S. Young. The third dialog state tracking challenge. *Proc IEEE Workshop on Spoken Language Technologies (SLT), South Lake Tahoe, Nevada, USA*, 2014a.

[69] S. Lee and M. Eskenazi. "Voicexml". *Communications of the ACM*, 43(9):53, 2000.

[70] D. R. Traum and S. Larsson. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 6(3):323–340, 2000.

[71] D. R. Traum. 20 questions on dialogue act taxonomies. *Journal of semantics*, 17(117):7–30, 2000.

[72] E. Bastianelli, D. Croce, R. Basili, and D. Nardi. Using semantic models for Robust natural language human robot interaction. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9336 LNCS:1393–1397, 2015.

[73] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL `https://www.ros.org`.

[74] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[75] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[76] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

Appendix A

Intent Detection Results

This appendix contains the results for the analysis of the intent detection model. Figure A.1 shows the performance of the intent model using two different transfer learning paradigms: feature-extraction and fine-tuning. Figures A.2 and A.3 show the performance of the model using different hyper-parameter configurations.

A.1 Fine-tuning vs. Feature-Extraction

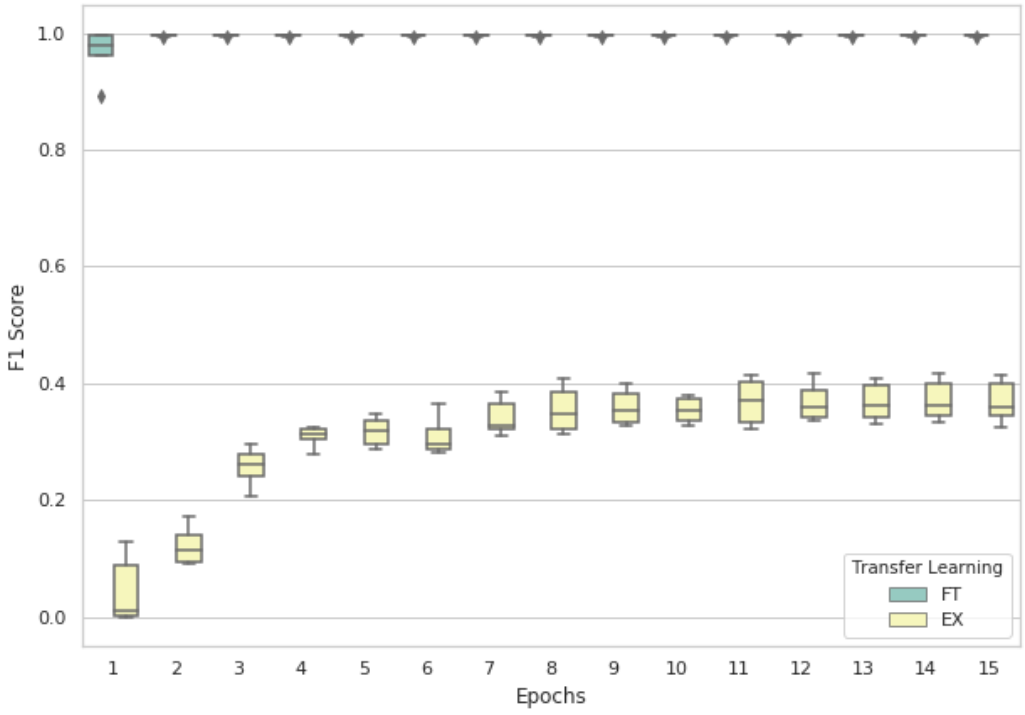


Figure A.1: Fine-tuning (FT, green) vs feature extraction (EX, yellow) for intent detection.

A.2 Hyper-parameters Comparison

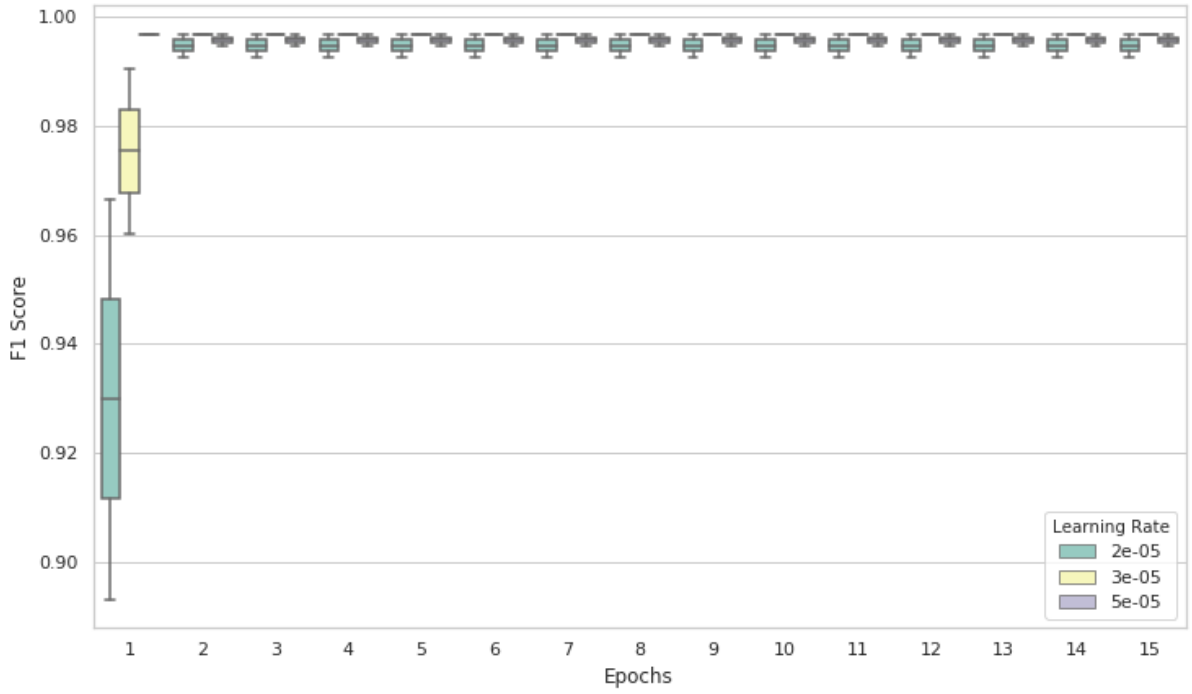


Figure A.2: Performance results for the intent detection model using different learning rates (lr). (Green, lr $= 2e^{-5}$); (Yellow, lr $= 3e^{-5}$); (Purple, lr $= 5e^{-5}$)

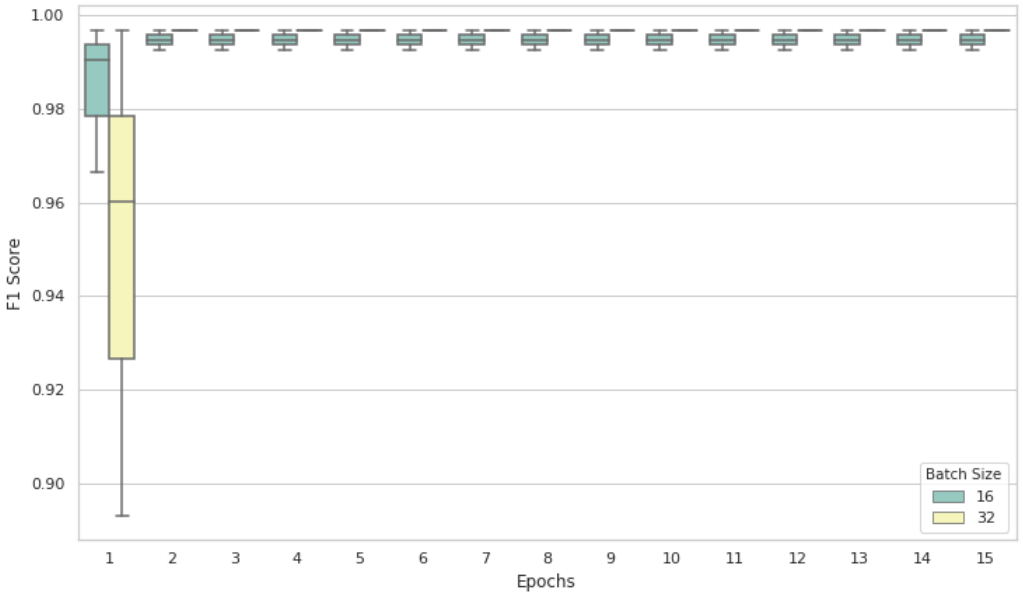


Figure A.3: Performance results for the intent detection model using different batch sizes (bs). (Green, bs $= 16$); (Yellow, bs $= 32$)