Natural Language Generation for Open Domain Human-Robot Interaction

António Luís Vilarinho dos Santos Lopes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors:	Doctor David Manuel Martins de Matos Doctor Ricardo Daniel Santos Faro Marques Ribeiro

Examination Committee

Chairperson:	
Supervisor:	Doctor David Manuel Martins de Matos
Member of the Committee:	Doctor Paulo Quaresma

May 2017

Acknowledgements

I want express my gratitude to my advisor, Doctor David Matos, for being an exceptional professor, advisor, and friend, who is tireless and always brings out the best in his students. I have been working with him since I was an undergraduate student and I hope we can still continue to collaborate for many years to come. I also want to thank him for all his support and fruitful discussions, not only during this thesis, but also during these years. I also want to thank my co-advisor, Doctor Ricardo Ribeiro, for all the fruitful discussions and support.

I also want to address my thanks to all my friends who have been with me for the past years, João Tornada, José Rosa, Rodrigo Granja, Frederico Sabino, Francisco Duarte, José Arvela, and many others, and the new friends, who are now great friends, and with whom I had innumerable fruitful discussions, Miguel Ramos. I also want to express my gratitude to Miguel Ramos for helping me with the development of the ongoing deep learning framework, which was used in my experiments.

Finally, but not least, I want to express my deepest gratitude to Patrícia for being supportive and comprehensive through all the past years, as well as being one of the most influential and important person in my life.

Lisboa, May 12, 2017
António Luís Vilarinho dos Santos Lopes

For my family and friends…

Resumo

A comunicação verbal desempenha um papel muito importante na interação homem-homem. Assim, possui também o potencial de desempenhar um papel importante na interação homem-máquina, especialmente se não for restringida a um único domínio.

Neste trabalho, abordamos um dos principais componentes dos sistemas de diálogo, geração da língua natural, para estudar como é afectado em comunicação de domínio aberto. Usamos métodos estatísticos, nomeadamente modelos de tópicos e aprendizagem profunda, e abordamos a arquitetura tradicional de geração optimizando o planeamento de frases e realização de superfície, como tarefas diferentes.

Utilizamos as legendas de documentários para modelar aspectos específicos de domínio e um conjunto de dados com um vocabulário grande para atender às preocupações linguísticas independentes do domínio. Usamos a Alocação Latente de Dirichlet para descrever a as relações finas do corpus específico do domínio e para o vocabulário grande usamos uma representação de "word embeddings" (providencia relações geométricas com semântica).

As tarefas de geração de língua natural são modeladas como problemas de aprendizagem profunda. Especificamente, o planeador de frases é implementado com redes neuronais feed-forwad e convolucionais. O micro-planeamento e a realização de superfície são implementados com redes neuronais recorrentes, que têm em conta os aspectos sequencias da linguagem.

Avaliamos o nosso método de construção de corpus através da detecção de diferentes segmentos de cenas e como esses parâmetros afetam a construção dos modelos de tópicos. Avaliamos o planeador de frases usando a similaridade de coseno e a realização de superfície com análise subjetiva. Os resultados sugerem que o planeador de frases aprender a mapear um espaço genérico independente do domínio para um espaço específico de domínio. A qualidade dos resultados da realização da superfície deve ser considerada preliminar.

Abstract

Verbal communication plays a very important role in human-human interaction. It has the potential to also play a very important role in human-machine interaction, especially if it is not restricted to a single domain.

In this work, we approach one of the key components of dialogue systems, natural language generation, to study how this component is affected by open domain communication. We rely on statistical methods, namely topic models and deep learning, and approach the traditional generation architecture by optimizing the sentence planning and surface realization, as different tasks.

We use documentaries' subtitles to model domain-specific aspects and a large vocabulary dataset to account for domain-independent linguistic concerns. Latent Dirichlet Allocation is used for describing the fine-grained relationships in the domain-specific corpus, while word embeddings (providing geometric semantic relations) are used to represent the large vocabulary.

Natural language generation tasks are modelled as deep learning problems. Specifically, sentence planning is implemented with feedforwad and convolutional neural networks. Microplanning and surface realization are implemented with recurrent neural networks, to account for sequential aspects of language.

We evaluate our corpus construction method by analysing different time boundaries in the scene detection algorithm and how those parameters affects the topic models. We evaluate sentence planning using cosine similarity and surface realization with subjective analysis. Our results suggest that the sentence planner can learn a mapping from the generic domain-independent space into the domain-specific space. The quality of surface realization results must be considered preliminary.

Palavras Chave
Keywords

Palavras Chave

Geração Automática de Língua Natural

Domínio Aberto

Aprendizagem Profunda

Redes Neuronais Recorrentes

Modelos de Tópicos

Keywords

Natural Language Generation

Open Domain

Deep Learning

Recurrent Neural Networks

Topic Models

Index

List of Tables

List of Figures

1 Introduction

Verbal communication, one of the modalities of human-human communication, is a very important characteristic of human interaction, as it allows humans to communicate with each other in a very rich (possibly ambiguous) way. Thus, human interaction is shaped to a great extent by verbal communication: this type of communication allows humans to express needs, knowledge, beliefs, among others, with each other. Thus, having a very important role in human-human interaction, verbal communication has, arguably, a very important role in human-machine interaction, as it allows humans and machines to communicate with each other in a natural way, resembling human-human interaction. Arguably, dialogue is an important characteristic of human-human interaction and human-machine interaction, as it provides a natural interaction between the interlocutors and listeners. Therefore, the ability to communicate in any domain provides a natural way for human and machines to communicate and collaborate to achieve a task.

Intrinsically, dialogue requires the interlocutor to be able to communicate about a certain subject within a specific context. Thus, Dialogue Systems (DSs) have been, usually, developed to fit a given domain (Litman and Silliman 2004; Ferguson and Allen 1999; Ferguson, Allen, and Miller 1996), i.e., DSs are, usually, developed to perform a well-defined task on a well-defined domain, which implies that these systems are not developed for addressing open domain communication. Therefore, traditional approaches to DS, and its components, focus on accomplish a specific task, covering only the vocabulary, and consequently the semantic interpretation, of the domain in question and not generalizing for new domains, in fact, usually, these systems require hand-crafted rules for the specific domain. However, to consider an open domain communication the system must be able to generalize to new domains and talk about new concepts.

Therefore, we consider that for a system to be truly open domain, it must be able to communicate about any domain, learn more about a given domain, and learn and adapt to new domains (this does not imply necessarily that the system is aware of the domain *a priori*). A true open domain system has not yet been achieved and as such we will limit our work to studying methods for communicating about any domain. In addition, every system is, ultimately, designed to achieve a goal and to accomplish a certain task. However, in contrast with the traditional DS, where the task is limited to the domain, an open domain DS can be applied to any task which requires verbal communication. Thus, an open domain DS does not have a specific task, rather, the task is the implicit goal of communication and interaction.

Addressing the open domain problem is a concern of the scientific community for the past years, as having domain-independent methods that can address an open domain environment would allow systems to interact with humans regarding any domain. With the development of deep learning (and other statistical) techniques, the ability to develop increasingly larger models that deal with larger vocabularies and semantics will be possible, as more data will be available. Thus, an open domain system must be able to learn new concepts and its interpretation in different contexts, modelling the fine-grained relations of words in different contexts, i.e., concepts are intrinsically connected to one or more topics and its interpretation comes from the context, where the topic is inserted. A practical example is how humans learn, we learn about different domains but domains can have new concepts, for instance in the context of mineralogy, a new previously unseen mineral being discovered, and is very important the ability to generalize and learn new concepts. The practical applications for open domain DS are numerous: from being deployed in a call center and allowing a natural and flexible way to communicate, to space probes having the ability to describe what they encounter in different expeditions.

1.1 Goal

A DS is usually decomposed into different modules, in a sequential or parallel (Ferguson and Allen 1999) architecture. The generic architecture of a DS is depicted on Figure 1.1, while a well known architecture, TRIPS, is depicted on Figure 1.2, which uses a parallel approach to the DS agent. As we address the problem of how to communicate, our main focus is the natural language generator, namely addressing the generation problem in an open domain context, i.e., we approach the generation problem in a domain-independent way. The generation module is the bottleneck in a DS for communicating in an open domain way, as it is responsible for deciding how should the system's goal be mapped into a utterance that best represents the goal, i.e., the Natural Language Generation (NLG) component is responsible for determining "how to say" the intention of the system.

Therefore, the NLG component of the DS is our main focus, as it is responsible for mapping the system's intentions and goals into natural language. Furthermore, we study statistical approaches to generation by approaching both sentence planning and surface realization in a statistical manner. In addition, we will not consider any other component from the DS architecture, the DS is only regarded as relevant background for the generation problem application.

Finally, to consider open domain and communicate about any domain, domain adaptation and learning is crucial, as it allows the system to learn and adapt to new, previously unseen, domains or transfer the learning method. This way, domain-independent methods for the generation problem are our main focus.

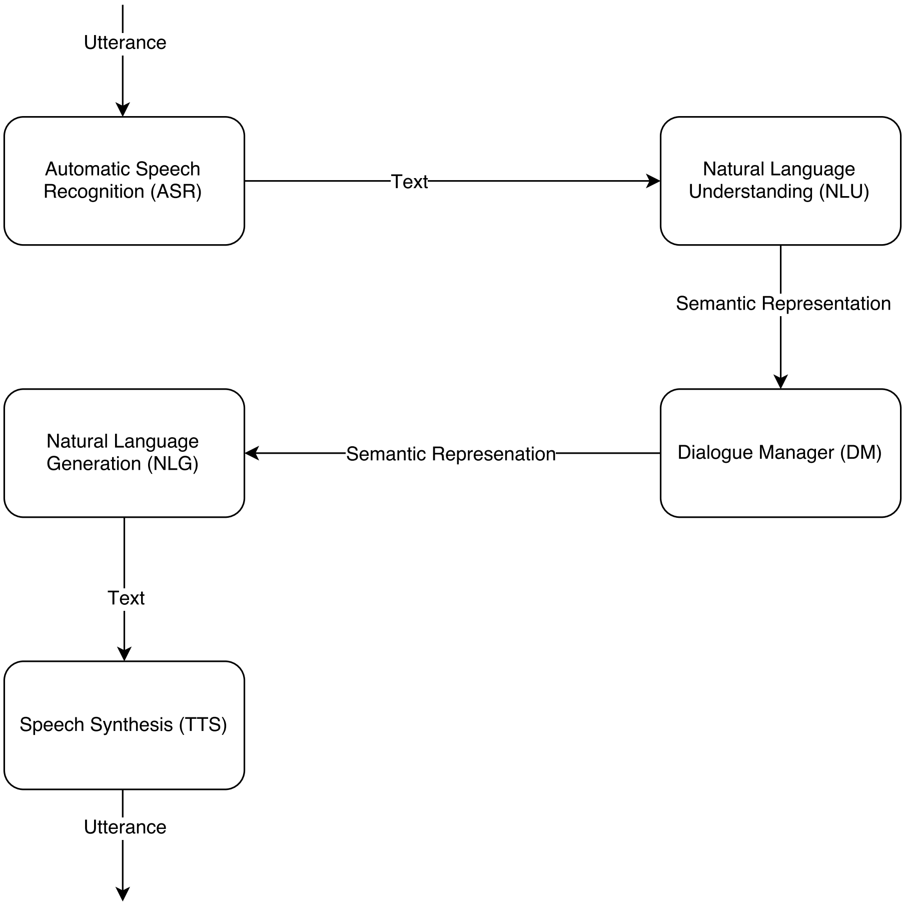


Figure 1.1: Generic Pipeline Spoken Dialogue System architecture.

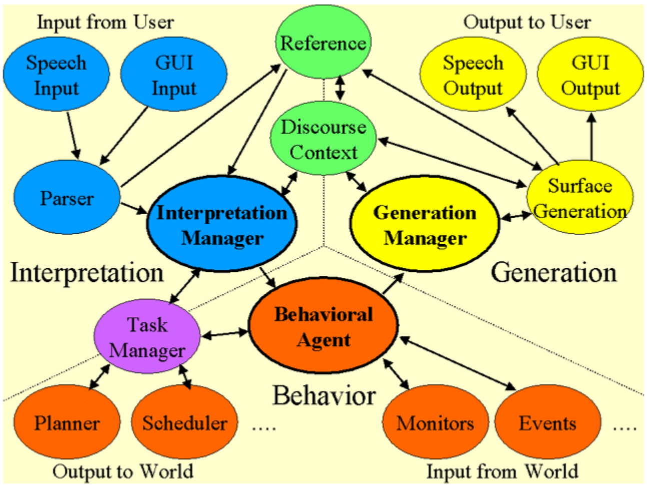


Figure 1.2: TRIPS architecture (Ferguson and Allen 1999).

1.2 Document structure

The document outline is as follows. In chapter 2 we provide the background for DSs, topic models, and deep learning. In chapter 3 we discuss the state of the art in NLG and the framing of NLG in the open domain context, as well as which approaches are suitable for addressing the open domain problem. Next, in chapter 4 we describe our corpus and how we build our own corpus from subtitles, as well as analysing the results.

In chapter 5, we describe an overview of our approach to open domain in the generation process and focus on the sentence planner module. Moreover, we describe what is our planner, present the experimental setup and present and discuss the results of sentence planning. In chapter 6, we focus on micro-planning and surface realization. In addition, we describe both components, present the experimental setup, and discuss the results of the realization. Finally, in chapter 7, we present our conclusions and directions for future work.

2 Background

In this chapter, we provide the reader the necessary background in deep learning, topic models, and dialogue systems, focusing on Dialogue Management (DM) in the latter. The NLG component is, usually, inserted in a DS and is the module responsible for deciding which is the utterance that best represents what does the system want to say. Thus, we start by discussing what is a dialogue systems and how it is, usually, built. Then, we focus on DM, as this component has a direct interaction with the generation module and is relevant background, specially considering an open domain environment.

Furthermore, we discuss what is a topic model and a specific topic model, LDA. This is relevant background for our approach to the domain-independent communication, namely to refine utterances with the context of the domain where it is inserted. Finally, we discuss what is deep learning and which approaches exist. This background is relevant in the context of open domain, as recent methods using deep learning allow to use the power of learning arbitrarily long sequences, which can be regarded as learning the sequence of words in a sentence for instance.

2.1 Dialogue Modelling Background

In this section, we address the concept of dialogue system, the components that constitute these systems and which are addressed in this work, as well as the existing approaches to build a DS (section 2.1.1). Then, we discuss dialogue management (section 2.1.3) and the approaches to build a dialogue manager (sections 2.1.3.1 to 2.1.3.6). We also present three well-known architectures, relevant in this field (section 2.1.2). Finally, the approaches to dialogue management in the context of open domain are discussed (section 2.1.4).

A DS is, roughly, a system whose goal is to interact with humans in a natural way. Usually, a DS is built with the goal of providing a way of human-machine interaction to be possible and natural (as fluid as possible). A DS is often characterized by its modality (text-based, spoken, graphical, multi-modal), and its initiative (system initiative, user initiative, mixed), device (where it is deployed), among others, and traditionally it is composed by five modules (these are illustrated in Figure 2.1). These modules are responsible for their own tasks and can run asynchronous from each other. Nonetheless, usually, these modules communicate with their "neighbours" in the architecture, this way providing their output as input of other modules.

These modules are, usually, an input decoder, a language understanding unit, a DM, an output generator, and an output renderer. Although the focus of this work is not the whole DS, this architecture is relevant as the context of this thesis.

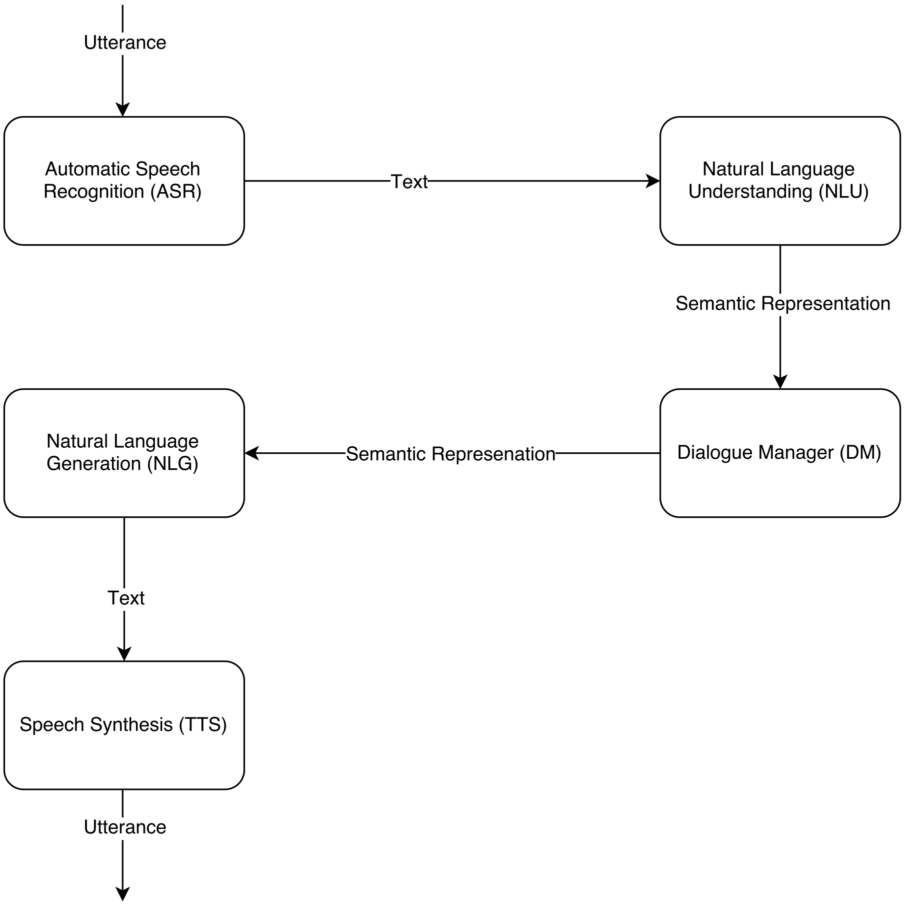


Figure 2.1: Generic Pipeline Spoken Dialogue System architecture.

Dialogue systems have a vast number of applications, as already mentioned (section 1.1), whether they are academic or industrial, including providing information, tutoring, service providing, advising, conversational partner, narrating. Usually, these systems are deployed in commercial systems, such as voice menus or call centers, mobile applications, such as Siri (Apple) or Cortana (Microsoft).

As previously mentioned, one of the characteristics of these systems, is its initiative. Regarding this characteristic, one can argue that a system that interacts with humans, especially if an open domain one, must be reactive and proactive, but, more importantly, it must be able to know when to switch between giving the user the initiative or taking itself the initiative. A system that always has the initiative, or always gives the initiative to the user, will not communicate in a natural way. In fact, if the initiative is the system's, usually there are constraints of vocabulary and grammar as the system is expecting a fixed dialogue flow (Lee et al. 2010), which implies that the dialogue is not natural and, thus, not well suited for open domain. On the other hand, a user initiative design, although it has advantages over system initiative and the system may sometimes ask questions to confirm slots, this initiative constraints the system to be reactive, since the proactive part is only to confirm slots. As a conclusion, if one wants a system that is able to "talk" about any concept, it must not be constrained as it would be in these two cases.

As a consequence, mixed initiative is the one that resembles human interaction, as it allows, not only taking turns, but also the initiative in the dialogue to be shared between humans and machines. This way, an open domain system considering human-machine interaction must be capable of, at different times, being proactive and reactive, among other characteristics.

A DS is an agent, as it is situated in an environment, it receives sensory input from the environment, and it produces actions that affect the environment. Thus, it is important to note that one of the key concepts of these systems is the DM, as this component is responsible for a number of tasks essential to the interaction with the environment. The DM is the component responsible for deciding "what to say" and its tasks consist of maintaining and updating the underlying context of the dialogue, deciding what should be the next action, providing an interface with back-end/task model. Another task the DM is responsible for, is providing an expectation for interpretation of communicative behaviour (perceived sensory input), as well as deciding what should be the next action. Thus, the DM has a significant role in these systems and, consequently, it is further discussed in section 2.1.3.

The different approaches to build dialogue systems are presented in section 2.1.1.

2.1.1 Building Dialogue Systems

Usually, dialogue systems are divided into two different approaches regarding the system's purpose (Larsson 2005). The first, is to consider that the system's purpose is one of interface engineering (as Larsson notes, the "engineering" view), having task-specific functionality and user-focused design, analogously to what command-line or menu-based graphical interfaces do. Briefly, it coordinates with the user in order to accomplish a task by deciding what should be the action for the current turn, given the context, input, and goals. The second, is to consider the DS as a simulation, i.e., DS is simply an "interface" between the user and the back-end of the system. This way, the system completely fulfills the theory of human language use and understanding, able to communicate with the user. Considering, then, which utterances must be produced to be natural to users and to be understandable by the back-end.

Examples of these approaches are presented next. For the first approach, significant examples of systems developed are ARTIMIS (Sadek et al. 1997) and TRAINS (Ferguson et al. 1996). The first was developed in the context of cooperative spoken dialogue, implementing a formal theory of interaction. The second was focused in practical spoken dialogue (conversation towards accomplishing a specific task) and was developed in the trains domain, using mixed-initiative planning. For the second approach, significant examples of developed systems are MITRE's Midiki toolkit (MITRE Corporation 2005) and SUNDIAL project (McGlashan et al. 1992).

Concluding this subject, it is important to note one of the most important DSs architecture, TRIPS (Ferguson and Allen 1999). This architecture is further discussed in section 2.1.2.2. In addition, in section 2.1.2, architectures for DM are discussed in terms of the contributions of each approach and what was the motivation behind each of them.

2.1.2 Architectures

DS architectures that use different DM strategies are exemplified in TRINDI (Larsson and Traum 2000), TRIPS, and COLLAGENThese architectures also describe how the DM interacts with the generation module. These approaches are relevant in statistical, plan-based and agent-based approaches, respectively. TRIPS, in particular, is one of the best known architectures and one of the most important. In sections 2.1.2.1, 2.1.2.2, and 2.1.2.3, we present these architectures.

2.1.2.1 TRINDI

TRINDI (Larsson and Traum 2000) is an architecture, as well as a toolkit, for building DS, where the dialogue manager is based on the theory of information state and dialogue move Traum and Larsson (2003) (discussed in section 2.1.3.6). This approach provides a toolkit for the formalization of the concept of information state, which in turn allows dialogue theories to be formalized and implemented. It is important to note that the factor that distinguishes this approach is the update of the information state with information related to observations and the execution of dialogue moves. Furthermore, this approach is often associated with the statistical DM approach. Figure 2.2 presents the architecture of TRINDI.

2.1.2.2 TRIPS

The TRIPS architecture (Ferguson and Allen 1999) is one of the most well known architectures and successful system in DS. This system is the successor of TRAINS (section 2.1.3.3) and is a continuation to the dialogue-based approach (M. Ferguson et al. 1996) to build a collaborative planning assistant, adding complex scenarios so that humans require the system's help to effectively solve the problems. The components of TRIPS are divided into three modules: interpretation, generation, and behaviour. Each of these modules is controlled by a manager component: interpretation manager, generation manager, and behaviour agent, respectively. Each one of these components runs independently and asynchronously. This allows the system to interpret the input and plan a response at the same time. This division also allows the

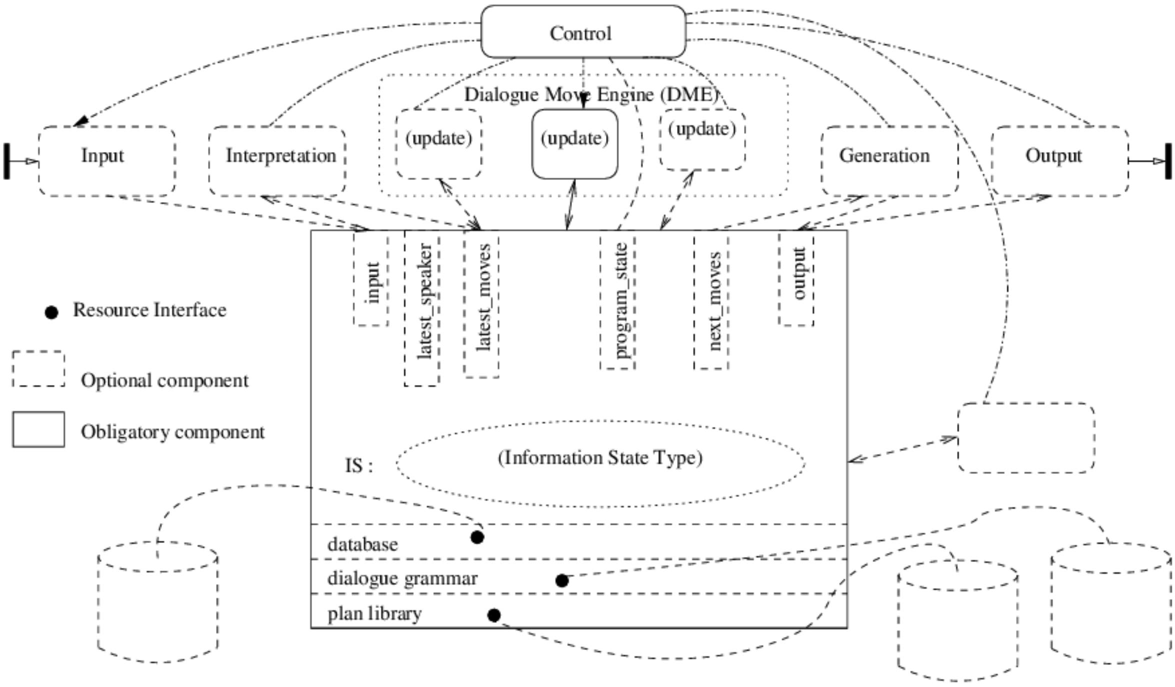


Figure 2.2: TRINDI architecture (Larsson and Traum 2000).

system to be formulated as a mixed-initiative one. An important note is that we focus on the generation module, without taking into consideration how the NLG module interacts with the behaviour and interpretation modules.

In addition, this approach is regarded as a collaborative plan-based (see section 2.1.3.3 for more regarding plan-based approaches) approach, also using agents. An illustrative figure of this architecture is presented in figure 2.3.

2.1.2.3 COLLAGEN

The COLLAGEN architecture (Rich and Sidner 1997) is an agent-based approach (see section 2.1.3.4) using well-established principles from computational linguistics, particularly shared plans (Grosz and Sidner 1990), discourse structure (Grosz and Sidner 1986), among others. This approach regards the problem-solving layer as an user-interface "middleware", using GUIs, and focusing on dialogue modelling (figure 2.4). Therefore, this approach can be regarded as both a (collaborative) agent-based and plan-based approach, as it uses principles of computational linguistics, an user-interface with an agent in the background, and planning. Additionally, this architecture supports mixed-initiative and has been applied to various areas, such as, air travel planning, email reading and answering, among others.

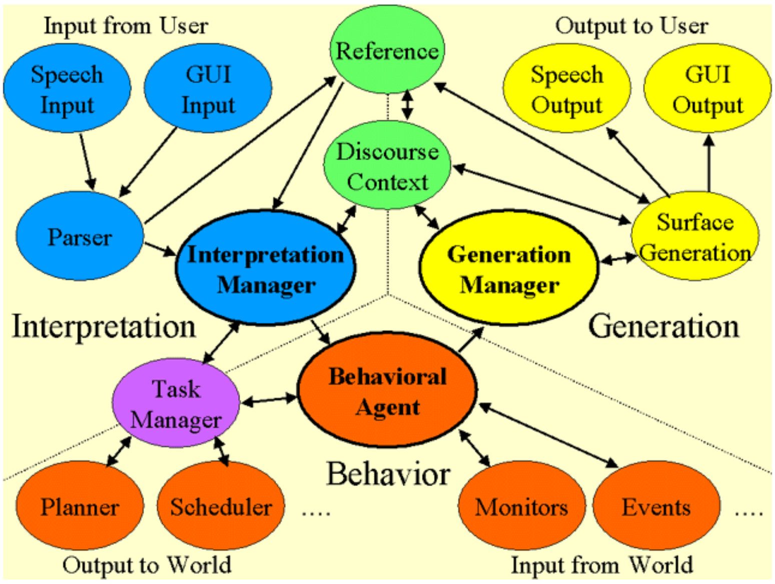
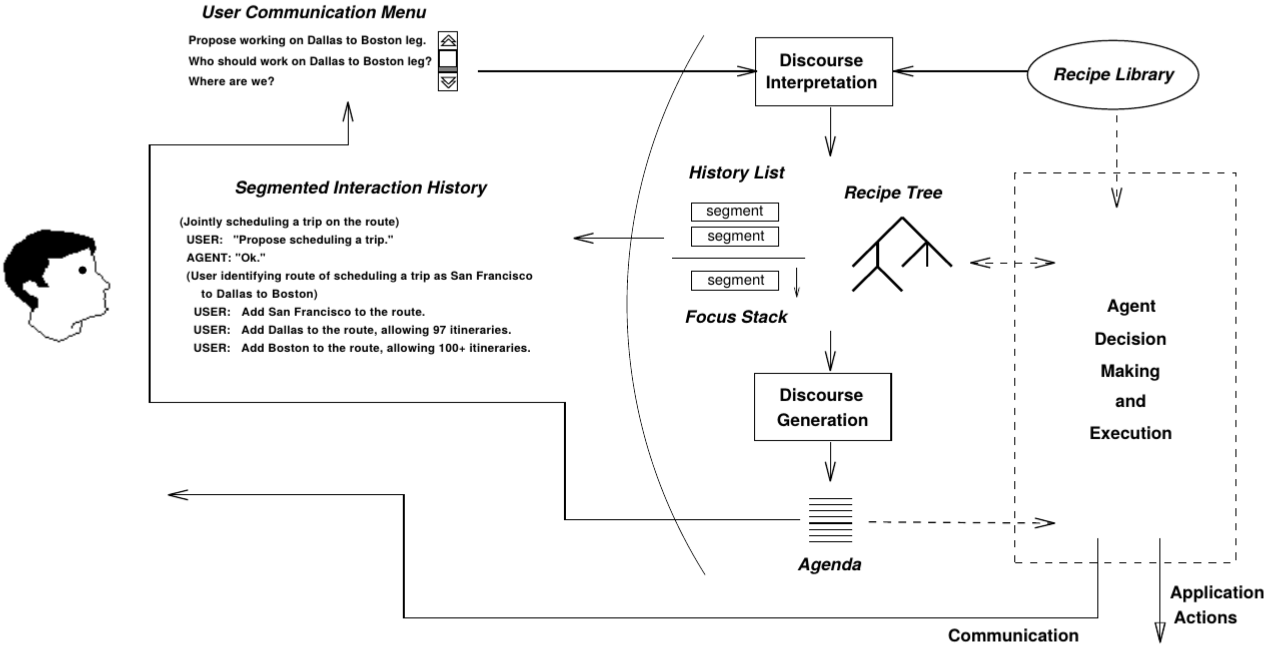


Figure 2.3: TRIPS architecture (Ferguson and Allen 1999).



(Jointly scheduling a trip on the route)
USER: "Propose scheduling a trip."
AGENT: "Ok."
(User identifying route of scheduling a trip as San Francisco to Dallas to Boston)
USER: Add San Francisco to the route.
USER: Add Dallas to the route, allowing 97 itineraries.
USER: Add Boston to the route, allowing 100+ itineraries.

Figure 2.4: COLLAGEN architecture (Rich and Sidner 1997).

2.1.3 Dialogue Management

As already mentioned, the dialogue manager has a very important role in a DS. The dialogue manager is responsible, primarily, for maintaining and updating the context (e.g. the history of the interactions), deciding what should be the next action (what should be said), provide an interface with the back-end model, and provide expectations for interpretation.

DM has, thus, a very important role when considering an open domain environment, even though it is not the focus of this work. Thus, understanding the different approaches to this problem is important, namely, state-based, frame-based, plan-based, collaborative agents, and statistical approaches (sections 2.1.3.1 to 2.1.3.6, respectively).

As noted by Bui (2006), there are several approaches to build models for DM. Among these approaches, one can distinguish state-based, frame-based, agent-based, plan-based, statistic, as discussed in the literature (McTear 2002; Xu et al. 2002). The aforementioned approaches are not mutually exclusive and often include common features from each other. In accordance with Bui (2006), the approaches that are described are state-based, frame-based, statistical, plan-based and agent-based.

2.1.3.1 State-based

This approach is the simplest one to model dialogue. In this approach, the dialogue consists of a finite sequence of predetermined steps, where the flow is to take the user through these steps in order to accomplish what is intended. This approach is, briefly, an automaton where the states represent the internal state of the dialogue and the transitions are the utterances produced by the user.

2.1.3.2 Frame-based

Frame-based approaches are, in a sense, extensions of the state-based approach and are developed to deal with some of the disadvantages of the latter, namely, their lack of flexibility. These approaches, instead of using a predefined sequence of steps, they build the model based on slot-filling, i.e., finding a predetermined set of information and fill the slots (similar to template-based approaches in NLG). Thus, the flow of the dialogue is not determined *a priori* (as in state-based), but depends of the input (human utterances) as its content is what determines how to fill the slots. Usually, these systems ask questions in order to fill slots.

In contrast with the state-based approach, frame-based approaches allow mixed-initiative (in fact, they allow it only to some degree) and allow more natural dialogues, as the dialogue is not fixed in a sequence of steps, rather the utterance depends of the human utterance.

These systems have been deployed in diverse applications, such as theatre booking systems (Hulstijn et al. 1996) and train timetable enquiry systems (van Zanten 1996). In order to allow more complex dialogues, other variations of this approach were developed, including versions that make use of schema and agendas, task structure graphs, blackboards, and others (Bui 2006). This way, attempts at achieving generic dialogue modelling capabilities were made. An example is the RavenClaw DM framework (Bohus and Rudnicky 2009).

2.1.3.3 Plan-based

These approaches, usually, aggregate features of various fields, such as, speech act theory, artificial intelligence planning, plan-based theory of speech acts – e.g. the work of Cohen and Perrault (1979) speech act theory – or theory of collaborative actions. Thus, considering these aspects, this approach regards human communication as the objective of achieving a given goal (through planning). Utterances are, thus, treated as speech acts, where those acts are part of a plan. Therefore, the listener must identify this plan (the speaker's intention) and give a response in accordance with this plan, applying the theories of communicative action and dialogue.

Two of the best known systems that follow this approach are the SUNDIAL project (McGlashan et al. 1992) and TRAINS-96 (M. Ferguson et al. 1996). These systems rely on the theoretical aspects already discussed and formulate the problem as planning. In addition, TRAINS supports mixed-initiative planning.

2.1.3.4 Agent-based

Agent-based approaches regard dialogue as a collaboration between agents and humans in order to solve a given problem or complete a given task. In order to promote this collaboration, it is important to note that there must be an interaction between the user and the agent and that the latter is capable of reasoning about its own actions and beliefs. These approaches, usually, incorporate Bratman's theory of Belief, Desire and Intention (BDI) (Bratman 1987; Bratman, Israel, and Pollack 1988). In contrast with the previous approaches, instead of focusing in the structure of the task, this approach aims at capturing what motivates a dialogue and the existing mechanisms in dialogue. Thus, the beliefs of the participants in the collaboration are explicitly modeled. In addition, there are shared beliefs and, inserted in those beliefs, is the common goal that must be achieved (solve the problem or complete the task in question).

It is important to note that these approaches consider the context (e.g. dialogue history) when constructing the dialogue model. This way, they allow dialogue to dynamically flow (and evolve) considering a sequence of steps that are related and constructed taking into consideration the previous steps.

Two of the best known systems following this approach are COLLAGEN (Rich and Sidner 1997) and TRIPS (Ferguson and Allen 1999) (also plan-based).

2.1.3.5 Reinforcement Learning

Before diving into statistical approaches, there is an important concept to be understood, Reinforcement Learning. This learning method has been widely used in DS and, with more relevance, in NLG. Reinforcement Learning (RL) consists of learning how to map situations to actions so as to maximize a (numerical) reward signal (Sutton and Barto 1998). Thus the learner must discover which actions will give the best reward by trying them. In this sense, two of the most important features of RL are trial-and-error search and delayed reward. Moreover, traditionally RL is composed of four subelements, (i) a policy (how the agent behaves at a given time), (ii) a reward function (a, usually numerical, value mapped from the perceived state), (iii) a value function (in contrast with the reward, it specifies what is good in the long run), and (iv) a model (that of the behaviour of the environment). As noted by Sutton and Barto (1998), efficiently estimating values is essential for the RL algorithms, thus usually the methods for estimating values are focused in estimating value functions, although solving these problems with others methods (e.g. simulated annealing) is possible. Thus, RL is an approach to understand and automate goal-directed learning as well as decision-making. More formally, in this problem there is an agent and the environment with which the first interacts and there are three key concepts: (1) a set of environment states S, (2) a set of agent actions A, and (3) reinforcement signals or rewards R (typically numerical).

As such, the interaction between the agent and the environment can be defined as a sequence of time steps $t = 0, 1, 2, \ldots$ (here, these are discrete for the sake of argument). At each step t, the agent observes the environment and receives a state representation s_t, such that $s_t \in S$, and selects an action a_t, such that $a_t \in A$. In the next time step, as a part of the consequence of the action (another consequence is the change in the environment, not considering sensing acts), the agent receives a reward (traditionally numeric) r_{t+1}, such that $r_{t+1} \in R$ and is in a new state s_{t+1}. Thus, at each time step the agent must map states to probabilities of selecting each possible action and its goal is to maximize the reward received over the long-run. This mapping is formalized as the policy and is, commonly, denoted as π_t, where $\pi_t(a|s)$ is, in fact, the probability $P(a_t = a|s_t = s)$. Thus, the goal of RL methods is to determine how the agent changes its policy as a consequence of the interactions with the environment (its experience). As mentioned, the goal of the agent is to maximize, not the immediate reward, but the cumulative one in the long run, therefore it is necessary to determine the model of optimal behaviour. In this sense, the simplest one is thinking about optimising the reward for the next n steps

$$\Sigma_{t=0}^{n} r_t \tag{2.1}$$

However, this implies that the agent will always act according to the same policy, only limiting how far ahead it considers when choosing an action (the history of actions h). Thus, in order to take the long-run reward into account, the concept of discount, γ, is introduced and rewards received from the "future" are discounted according to such discount factor ($0 \leq \gamma \leq 1$). More formally, the model of optimal behaviour, considering the discount factor and the long-run reward, is defined as

$$\Sigma_{t=0}^{\infty} \gamma^t r_t \tag{2.2}$$

In RL, there is an important property an environment state can possess, the Markov property. An environment state has this property if and only if the conditional probability distribution of the future states dependents exclusively of the present state, this is, the transition from the current state to the next depends only of the current and none of the past states or actions (i.e., they are independent). One can argue that it must successfully comprise the previous states (no more than the total past history states) in a state without explicitly having such information. More formally,

$$P(r_{t+1} = r, s_{t+1} = s' | s_o, a_o, r_1, \ldots, s_{t-1}, a_{t-1}, r_t, s_t, a_t) \iff P(r_{t+1} = r, s_{t+1} = s' | s_t, a_t), \forall_{s',r,h} \tag{2.3}$$

Notice the right side of equation 2.3 is not conditioned by the last reward r_t: if the environment has the Markov property, the next state depends exclusively of current action and state. This property is important in RL due to decisions and values are, usually, a function of only the current state. Considering this, when a reinforcement learning task satisfies this property, it is considered an Markov Decision Process (MDP) (also has the delayed reinforcement property). More formally, an MDP is defined as a 4-tuple $< S, A, T, R >$, S is a set of states, A is a set of actions, T a state transition function $T : S \times A \rightarrow \Pi(S)$, where $\Pi(S)$ is a probability distribution over S and can be rewritten as $T(s', a, s)$ (the probability $P(s_{t+1} = s' | a_t = a, s_t = s)$ of the transition from state s to the next state s' with action a) and R is the reward function, formally $R : S \times A \rightarrow \mathbb{R}$. Now, and having defined the model for optimal behaviour previously, it is clear that the goal is to find a policy to maximise it. This is a well-studied topic and will be briefly addressed here, see Sutton and Barto (1998), Kaelbling, Littman, and Moore (1996) for more information. As mentioned before, one of the key elements of RL is the value function. This allows the value expected from the reward to be computed (recursively). Usually, the notation for the value function is $V^{\pi}(s)$. Moreover, given the policy π, it is the reward expected from the state s to the terminal one s_t, using the Bellman equations it is defined as

$$\begin{aligned} V^{\pi}(s) &= \sum_a \gamma\pi(s,a) \sum_{s'} T(s',a,s)[R(s',a,s) + V^{\pi}(s')] \\ &= \sum_a \gamma\pi(s,a) Q^{\pi}(s,a). \end{aligned} \tag{2.4}$$

This function is usually called state-value function for a given policy π and $Q(s, a)$ gives the reward for taking the action a in state s following the policy π, often called action-value function for a given policy π. Both these value functions can be estimated from the interaction with the environment. Thus, one can define the optimal state-value function and optimal action-value function, namely, using the Bellman optimality equation for the first and for the latter,

$$V^* = \max_a \sum_{s'} T(s', a, s)[R(s', a, s) + \gamma V^*(s')] \tag{2.5}$$

$$Q^* = \sum_{s'} T(s', a, s)[R(s', a, s) + \max_{a'} Q^*(s', a')] \tag{2.6}$$

Then, the optimal policy is defined as

$$\pi^* = \arg\max_a Q^*(s, a) \tag{2.7}$$

Next, it is necessary to learn (compute) the optimal policy. In order to do so, strategies to update the value-action and strategies for action selection are necessary. One of the issues that RL debates is *exploitation vs exploration*. This debate focuses on how the state space is traversed to select an action. This is well covered in the literature, e.g. Young (1999), Sutton and Barto (1998), and Kaelbling et al. (1996). Briefly, the problem is how to balance between the two approaches, *exploration*, searching the complete state space and, for each action in each state, the associated reward – e.g. dynamic programming based policy optimisation – and *exploitation*, focusing on the state sequences of optimal and near optimal estimate of the value-action – e.g. Sampling based policy optimisation (Monte Carlo methods).

Two of the best known strategies to update the value-action are State-Action-Reward-State-Action (SARSA) and Q-learning. In the first strategy, the learning derives from interacting with the environment and update the policy based on taken actions, an on-policy learning algorithm, i.e., the new state-action value depends on the reward received after taking an action, the current value of the state, and the value of the next state-action pair observed. Additionally, the Q-value for a state-action is updated by an error, adjusted by the learning rate α, more formally:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t]) - Q(s_t, a_t)] \tag{2.8}$$

In the second strategy, an off-policy learning strategy, the learning also derives from interacting with the environment, although it updates the policy based on the optimal actions (maximum reward of available actions). In contrast with SARSA, the new state-action value depends of the *optimal* state-action pair of the next state, rather than the next state-action taken. Therefore, SARSA learns the Q-values associated with taking the policy followed by itself, while Q-learning learns the Q-values associated with taking the exploitation policy together with following an exploration/exploitation policy.

Two of the best known strategies for action selection are ϵ-greedy and softmax. Here only the first is presented for the sake of argument, as it is one of the most well known strategies. Strategies for action selection can be pure exploitation (e.g. always select the action with the highest state-action value, greedy strategy), pure exploration (e.g. dynamic programming methods), and a balance between exploitation and exploration. These strategies aim to achieve the latter, the balance between exploitation and exploration. In this sense, ϵ-greedy is a variation of greedy selection (pure exploitation), this strategy also selects the action based on the best state-action value. However, rather than always following the best value selection, there is a probability ϵ of selecting an action from the remaining actions.

2.1.3.6 Statistical

Usually, commercial systems are built using state-based or frame-based approaches. However building a system with hand-crafted rules is expensive (if one wants to extend the scope of the system, new hand-crafted rules are required). In this regard, formulating the problem as having a dialogue policy to optimise and automatically learn that policy, to adapt to different domains or extend one, can be considered a suitable approach for open domain.

Statistical approaches are data-driven, i.e., these systems are trained from a dataset automatically, building stochastic statistical models, often almost without any human supervision and with few (or any) hand-crafted parameters. Moreover, these approaches are, usually, based on information state, rather than other structural dialogue state approaches. The differences between these approaches are, in the first case, that the information is the dialogue state itself, while in the latter, dialogue is regarded as behaviour according to some grammar, as noted by Traum and Larsson (2003). Information state approaches are composed of five components: (i) a description of informational components (e.g. common beliefs, intentions, linguistic structure), (ii) formal representation of the previous components, (iii) a set of dialogue moves (that trigger the update of the information state), (iv) a set of update rules (to update the information state), and (v) an update strategy (responsible for deciding which rules should be applied at a given point from a set of applicable rules).

Formulating the problem as data-driven and considering information state, motivated modelling dialogue as a stochastic model using RL: the most successful approaches are based on MDP and Partially Observable Markov Decision Process (POMDP). Although there was previous work on modelling dialogue strategy as a stochastic optimisation problem using RL, Walker et al. (1998) were the first to propose the statistical model considering the interactions with real users. Even tough this model was proposed as an off-line algorithm, it is important to note that the formulation by Walker et al. (1998) allowed the system to be evaluated in real-time with real users. In order to test with real users, Walker et al. used the PARADISE framework (Walker et al. 1997) to derive an empirical performance function, combining subjective user preferences and objective system performance measures. This framework is particularly important, as it proposes a method to specify reward functions in a data-driven manner. Moreover, PARADISE uses regression analysis to derive a linear combination of objective features that is predictive of subjective user ratings (Lemon 2011). However, the approach, to model dialogue strategy as a stochastic optimisation problem, was constrained to the dialogue manager being based on a state machine.

Levin et al. (1998) formulate the problem as an MDP, arguing that the problem of dialogue strategy can be defined as an optimisation problem and solved by a number of different methods, including RL. Thus, formulating the problem as an MDP allows dynamic changes to the dialogue strategy and the decision of the action (to be performed) to be based on optimizing rewards (or costs) given the current state. However, this approach has some disadvantages: for example, learning with large state spaces, learning from small initial data sets, learning the reward function. These disadvantages have already been addressed and for each one a solution has already been proposed (Singh et al. (1999), Young (1999), Levin and Pieraccini (1997), Litman et al. (2000), Walker et al. (1998)).

Another formulation proposed for this problem is a generalization of MDP, POMDP (Williams and Young 2007). This formulation is an extension of MDP that removes the requirement that the system always knows the current state precisely. This approach also assumes the Markov property and can also be solved via RL. However, the state s_t is not directly observable and reflects the uncertainty in the interpretation of user utterances, regarding the output of the Natural Language Understanding (NLU) module as a noisy observation (Young et al. 2013). More formally, a POMDP is a tuple $< S, A, T, R, O, Z, \gamma >$, where S is a set of states $s \in S$, A is a set of actions $a \in A$, T is the transition probability $P(s_t|s_{t-1}, a_{t-1})$, R is the expected reward $r(s_t, a_t)$, O is a set of observations with $o \in O$, Z is an observation probability $P(a_t|s_{t-1}, a_{t-1})$, and γ is a discount factor ($0 \leq \gamma \leq 1$). An important definition in POMDP is that, as s_t is not known precisely, the state is defined as belief state: this state is a distribution over all states, providing an explicit representation of uncertainty, as the observations are noisy. In turn, this representation allows building systems more robust to speech recognition errors, as well as, together with policy-derived action, it allows dialogue design criteria to be included through associating rewards with state-action pairs. More details regarding the belief state update and policy optimization can be found in Young et al. (2013).

As stated by Young et al. (2013), learning directly from corpora can be questionable, as the state space that prevailed during the collection of data may be different from the one used in policy optimisation. To address this, usually, the alternative is to build a model of the user that can interact directly with a DS, which, in turn, can itself be trained on corpora. This user simulator can be used for development, training, and evaluation purposes. The main applications of systems with this formulation are information inquiry applications, voice dialling, tourist information, car navigation, among others. In fact, Bayesian update of dialogue state (BUDS) was evaluated in a tourist information domain, where users could ask about hotels, restaurants, and bars in a fictitious town. Additionally, the dialogue supported mixed-initiative and the users could speak about nine concepts, in order to find a suitable venue. After finding the venue, the users could ask about four further concepts related to the venues. Another example in the same domain can be found in Young et al. (2010). This example is based on the Hidden Information State (HIS) model (not discussed here), an analogous model to BUDS to factorise the state space. However, even though the complexity of the update is not significantly different, in HIS the belief updates are calculated on the assumption that the user goal does not change. In contrast, the BUDS framework, using loopy belief propagation or grouped belief propagation, allows that the user goal changes (needs approximations to do so) and allows conditionally independent slots.

2.1.4 Open Domain and Dialogue Management

In order to be able to support open domain, techniques to extend a domain are crucial, i.e., when there is a previously unseen concept in a domain, the system must be able to learn it. The ability to extend a domain is very important, as it implies that a system with such ability is able to learn more about a domain, or even learn a new domain, and thus it must be able to learn any concept in an arbitrary domain. In order to be able to do the latter, the system must be built with techniques that allow it to reuse existing knowledge and to be adapted in an *on-line* fashion.

Recent work has been developed to address theses problems: extending domains and DM adaptation to those domains (Gašić et al. 2013; Gašić et al. 2014); policies to handle multi-domain statistical DM (Gašić et al. 2015); learning and transferring domain knowledge through ontology parameterisation (Wang et al. 2015). Gašić et al. (2013) suggest using policy adaptation in the form of transfer learning, as already used in RL, applied to DM, so that the dialogue manager supports domain adaptation. The key aspect of this approach is, after learning a domain, to improve the learning of others considering the expertise on the first. Thus, transfer learning addresses the problems of (i) given an arbitrary target domain, how to select the proper source domain from the set of source domains, (ii) given both arbitrary target and source domain, how to find the relationship between these domains, and (iii) having the relationship, how to effectively transfer knowledge from one to the other. In the aforementioned work, Gašić et al. models, in on-line RL, the Q-function as a Gaussian process (GP) (Engel et al. 2005),

$$Q(b, a) \sim GP(0, k((b, a), (b, a))) \tag{2.9}$$

, where the kernel k is factored into separate kernels over the belief state and action state $k_b(b, b')k_a(a, a')$. Thus, for policy optimisation, an updating strategy is required, as the Q-function is modelled as a GP, the strategy used is GP-SARSA. This strategy is inspired by the SARSA algorithm (section 2.1.3.5) and learns a Gaussian distribution over state-action values. Thus allowing model-free policy improvement performance. Formulating the Q-function as a GP derives from the need to explicitly obtain distributions of various derived quantities, as a GP is a normal distribution it implicitly inherits this property. In addition, in this approach, Gašić et al. use the BUDS framework.

Following this approach, Gašić et al. (2014), formulate the transfer learning technique so as to allow *on-line* repeated incremental domain extension. Also using GP to optimise the policy of the POMDP in the BUDS framework, this approach relies on ontologies (automatically generated using information from the web (Ben Mustapha et al. 2015)) to represent the knowledge of the domain. The domain is extended by learning new slots, new concepts, incrementally and on-line, and the adaptation is done through interaction with real users. This interaction was performed via the Amazon Mechanical Turk service (in a telephone-based DS), where the users were assigned specific tasks in different domains (e.g. to find restaurants with particular features) and after each dialogue they were asked if the dialogue was successful or not. This way, the policy was trained for a limited domain and then incrementally adapted to a larger domain without supervision. Additionally, as in the previous approach, it is suggested that learning the domain incrementally is faster than learning directly the full domain, i.e., starting with a simple system and successively extending and adapting it slot by slot has showed to achieve optimal performance faster than learning all the slots of the domain at once. More formally, decompose the complex domain into a series of domains with gradually increasing complexity and train the system in stage iteratively adapting to successfully more complex domains is faster than learning the full complex domain.

Gašić et al. (2015) also propose a multi-domain dialogue architecture, structuring dialogue policies in a class hierarchy aligned with an underlying knowledge graph. This graph is regarded as a tree-structure class hierarchy, where each class represents a domain or topic (using slots) and derived classes represent specialisations of that domain. The purpose of this formulation is to support scaling to multiple domains. Furthermore, the idea is to build generic policies with a small training dataset, then, when the system is deployed, and more data is collected, the specific policies will be trained for the derived classes. Therefore, refining the policy from generic domain policy to domain specific policy. In this approach, Gašić et al. also use ontologies to represent domain knowledge, in a hierarchy of domains and sub-domains, where the domains are associated with the generic policies and sub-domains with the specific ones. Furthermore, in accordance with the previous approaches, the problem is formulated as GP-based RL, also using GP-SARSA to model the Q-function as a GP and BUDS framework to factorise the state space.

2.2 Topic Modelling Background

Topic models are unsupervised models that aim to model words relationships inside a collection and group them by their corresponding topic, i.e., topic modelling is an unsupervised learning method (does not require any prior annotations) that aims at given a collection of documents find the group of words that best represent the collection, in other words, find the topics which best describe the collection. The number of topics is, usually, one of the parameters these algorithms require to be determined manually. These algorithms enable discovering topic patterns in the collection as they are based on the assumption that documents are mixtures of topics, where each topic is a probability distribution over the words (Steyvers and Griffiths 2007).

Furthermore, topic models are, usually, generative and only consider the number of times the word is seen in a document (although there are extensions which preserve word ordering). Statistical topic models require the number of topics to be defined manually and specifiy a multinomial distribution over words for topics and over topics for documents. Topic models do not make any assumption regarding the meaning of the words in a document, instead they make the bag-of-words assumption and aim at fitting the word in a topic. Bag of words assumption can be regarded a vector space model where each word in the bag is active in the vector representation of the terms. Briefly, a vector space models represent the importance of terms in a document through a vector representation. Usually, this model represent the documents as vectors and is used, usually, for information retrieval, relevance rankings, among others. Furthermore, in applications such as relevance ranking, the vector space model provides a method for comparing two different documents by computing the dot product and normalizing, which is in fact the cosine angle between those two vectors. The cosine angle is defined as

$$\cos\theta = \frac{D_1 \cdot D_2}{||D_1||_2||D_2||_2} \tag{2.10}$$

There are different topic models such as Hierarchical Dirichlet Processes (HDP) (Teh, Jordan, Beal, and Blei 2004) and LDA (Blei, Ng, and Jordan 2003). Although HDP are a generalization of LDA, where the number of topics can be learn from data, we only address here LDA, as although topic modelling is a relevant part of the work, it is not our concern.

LDA assumes that each document in a collection is made of multiple topics, where each topic is a distribution over a fixed vocabulary of terms (Steyvers and Griffiths 2007). Moreover, the model assumes there are K topics associated with the collection and each document is composed by different percentages of each topic. As a probabilistic topic model, LDA represent its hidden variables via topics, where the documents hidden variables are representative of the collection. Moreover, LDA is usually represented as a graphical model using the plate notation, see fFgure 2.5, where shaded nodes represent the observed variables, i.e., the variables which can be directly observed from the collection, and unshaded nodes the latent variables.

Furthermore, in Figure 2.5, each plate can be regarded as multiple sampling steps, where N is the collection of words in a document; M is the representation of all the documents in a collection. In addition, each arrow is a conditional dependency and the topics are represented by $\beta_{1:K}$, where K is the number of topics and β_K is a distribution over the vocabulary. Finally, θ_d is the variable representing the topic probabilities in document d, this variable has K dimen-

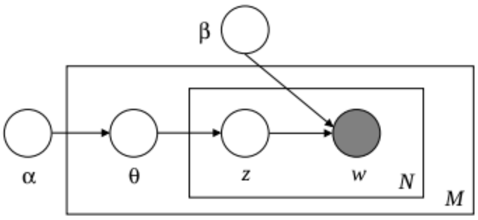


Figure 2.5: LDA plate notation (Blei, Ng, and Jordan 2003). Shaded and unshaded nodes represent the observed and latent variables, respectively.

sions, $z_{d:n}$ is nth' word topic in a document d, and $\omega_{d:n}$ are the observed words Both α and β parameters are symmetric Dirichlet priors which are constants in the model, as they are sampled only one time. While α is the prior weight of topic k in a document, β is a prior weight of word w in a topic, both are usually less than 1 and both can be interpreted as if the number is close to 0 then the model will prefer sparse topic distributions and sparse word distributions, which reflect few topics for each document and few words per topic, respectively.

Finally, the iterative process for document generation is as follows, for each document d in the collection:

(1) Choose θ_i, modeled by a Dirichlet distribution with the parameter α ($Dir(\alpha)$).

(2) For each word n in document d:

   (a) Retrieve topic z_N form the multinomial distribution $Multinomial(\theta_d)$;

   (b) Given topic z_n, retrieve a word from the multinomial distribution $Multinomial(\varphi)$, the probability over the vocabulary, where φ is modeled through a Dirichlet distribution with the parameter β ($Dir(\beta)$).

2.3 Deep Learning

Deep learning is a statistical method for machine learning and can be defined as simply using a Artificial Neural Networks (ANNs), which are a universal approximator, with a greater number of hidden layers. Thus, the concept of deep learning can be regarded as a redefinition of the previous neural networks by using architectures where the number of layers is significantly greater than traditional neural networks.

2.3.1 Deep Neural Network

A Deep Neural Network is an ANN containing multiple hidden layers between the input and output layers, thus inheriting the approximation capability of the ANN. Artificial Neural Networks are known for their ability to approximate any function in a compact set with just one hidden layer and the sigmoidal activation function (Cybenko 1989). Therefore, Deep Neural Networks (DNNs) can perform a mapping between the input and output, i.e., a deep network learns a function to perform a mapping from the input space to the output space. There are different architectures for these networks, namely feedforward and recurrent (see section 2.3.3).

Feedforward networks perform a forward pass, i.e., feed the network an input sample and predict the output, for each training example x_i, the network is usually trained with the standard backpropagation algorithm, where the weights of the hidden layers in the network are updated using an optimization algorithm, for instance stochastic gradient descent:

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \eta \frac{\partial C}{\partial \omega_{i,j}} \tag{2.11}$$

In Equation 2.11, η is the learning rate and C is the cost function, which measures how good were the predictions. Other methods for optimization exist and can yield better performance than the gradient descent, such as Adaptive Moment Estimation (ADAM) (Kingma and Ba 2014) and Root Mean Square Propagation (RMSProp) (Tieleman and Hinton 2012), which use adaptive methods instead of fixed learning rates, however we won't detail the optimizers as they are not our main concern. Two important notes: first, the partial derivatives are calculated via chain rule; and, second, for the sake of argument we only present one of the methods to train a network via backpropagation, which is online-learning. The other methods, batch update and minibatching, are further discussed in chapter 6.

DNN networks also have another important aspect, the hidden layers activation. This activation ranges from sigmoidal to hyperbolic tangent, softmax to radial basis, among others. The activation and cost function are chosen according to the task. Another important aspect is that these networks have a tendency to overfit, this is, to learn the mapping between the input and output so well for the training data that they do not generalize for new unseen samples. Thus, in order to suppress this disadvantage, these networks are usually trained using regularization methods such as $l2$ norm, and, more recently the dropout technique (Srivastava et al. 2014).

Finally, in feedforward DNN the layers have all their neurons connected to the previous and next layer and for that reason this type of layers are usually addressed as fully connected.

2.3.2 Convolutional Neural Network

Convolutional neural networks are very similar to regular neural networks, discussed in the previous section, as the same methods for learning still apply, as well as hidden layers made of neurons with the same dot product and activation functions, and a cost function. However, instead of regarding the input as simple feature vectors x_i, these networks make the explicit assumption that the input has a spatial relation and assumes that there is are 3D volumes of neurons. This is particularly useful for the application for which this type of networks was developed, image processing. The reason why this approach is better than the DNN is that not all the neurons are connect between layers, instead they are only connected to a small region of the precedent layer.

Convolutional Neural Networks (CNNs) have, usually, three different types of layers: convolutional, pooling, and fully connected (this one was already discussed). Briefly, the convolutional layers computes the output of the neurons with respect to a local region in the input, a filter (also known as kernel or feature detector), while pooling layers perform a downsampling along the spatial dimensions.

Moreover, convolutional layers apply the convolutional operation to learn the filters, i.e., the layer looks at a specific region in the input and through convolutions produces a new region in a different smaller space. Thus, this layers need a mechanism to look at different regions of the input, which is accomplish by three parameters: depth, stride, and zero-padding. The depth parameter corresponds to the number of filters the network will use, each looking at different regions of the input; the stride parameter defines how the convolution is applied, as it is responsible for determining how the many number of units the filters slide over the input matrix, and zero-padding allows to control the size of the kernels by padding with zeros in the border, allowing to filter border elements. Convolutional layers, also apply a non-linearity function to the output, usually Rectified Linear Unit (ReLU), which is defined as $f(x) = max(0, x)$. Pooling layers are responsible for reducing the dimensionality of each kernel, while keeping the most important information, this way reducing the number of parameters, size of the internal representation of the network, and preventing overfitting. Different types of pooling include max pooling, average pooling, among others.

Finally, these networks can be regarded as a feature detection network, in addition to the approximation power of DNN. These networks filter the features that best describe the mapping between the input and output. Therefore, during training the network learns the filters, this is, the network learns how to look at the 3D volume input and how optimise the spatial relation in the input (Karpathy 2015).

2.3.3 Recurrent Neural Network

An RNN is a type of neural network that has the ability to model and learn arbitrarily long sequences of data, this is, instead of considering a simple data point and predicting its output (like a regular DNN), this type of networks can model the sequential dependencies between inputs. These dependencies are often associated with a temporal dependency as the previous data point influences directly the current point. In other words let's consider the rationale behind reading a document: humans read documents word after word and each one conditions the meaning of the next one, instead of reading each word without considering the context

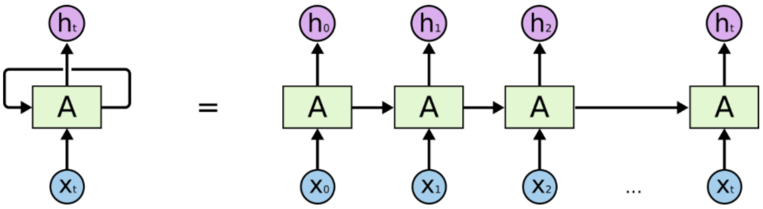


Figure 2.6: An unrolled RNN (Olah 2015).

where it is inserted. The same rationale can be applied to understand RNN, in contrast with feedforward neural networks, which predict each word without the context, an RNN has the ability to model the sequence by maintaining the context in a persistent state, instead of always forgetting.

Moreover, this persistence can be regarded as applying the same transformation at each step of the sequence, while considering the previous transformation. To illustrate this concept, an unrolled RNN is depicted in Figure 2.6.

Understanding the notation and mathematical background behind this type of networks is important, let us define the input sequence as $x = (x_1, \cdots, x_T)$, the hidden state vector as $h = (h_1, \cdots, h_T)$, and the predicted output vector as $y = (y_1, \cdots, y_T)$, where timestep t ranges from 1 to T. The RNN is defined by the following equations:

$$h_t = \sigma(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h) \tag{2.12}$$

$$y_t = W_y \cdot x_t + U_y \cdot h_{t-1} + b_y \tag{2.13}$$

In Equations 2.12 and 2.13, h_t is the network's hidden state, x_t is the input at step t, y_t is the output at step t, σ to a non-linear activation function and $W_{h,y}$, $U_{h,y}$ and b are weight matrices and biases, corresponding to the network's parameters.

Although in theory RNN can learn arbitrarily long sequences, in practice these networks have shown difficulties learning dependencies that have a very large span of information (Bengio et al. 1994). Another problem with these networks is how to train efficiently, as training with traditional Back Propagation Through Time (BPTT) algorithm has been proved to be extremely difficult due to the exploding and vanishing gradient problems (Bengio et al. 1994). Thus, to address these problems two different RNN-based methods were proposed, Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).

2.3.3.1 Long-Short Term Memory

To address the limitations of "vanilla" RNNs, the LSTM was proposed by Hochreiter and Schmidhuber (1997). This formulation was studied and further developed in following works

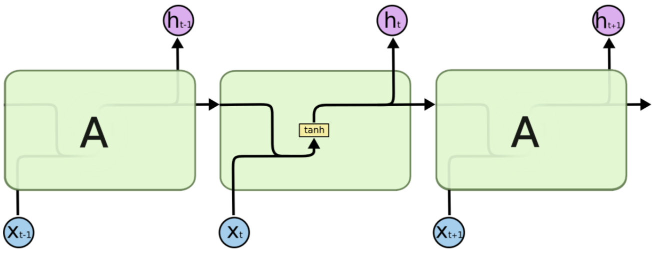


Figure 2.7: Internal RNN structure (Olah 2015).

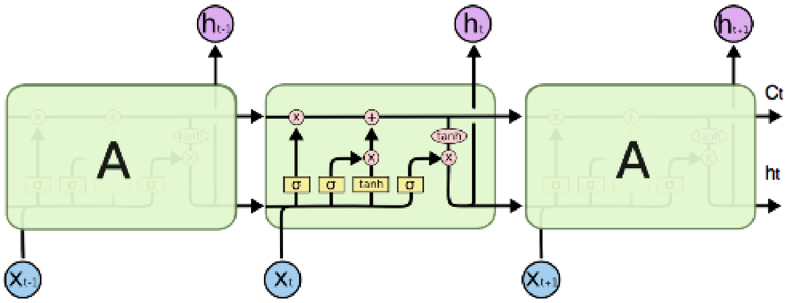


Figure 2.8: Internal LSTM structure (Olah 2015).

by the community. LSTMs have been widely applied in numerous Natural Language Processing (NLP) applications with success, as the tasks in NLP are easily formulated as sequential problems.

In contrast with RNNs, which possess a cell structure consisting mainly of a single neural network (see Figure 2.7), LSTMs' cell structure is more complex, as it uses different gates to control how the internal morphing of the information is processed. Thus, these gates interact with each other to control how the information is morphed in the cell's internal state. This is accomplished via "selective writing, reading, and forgetting" (R2RT 2016). The cell's structure is depicted in Figure 2.8, where each line represents the flow of a vector representation, each yellow square represents a regular neural network layer, and pink circles represent a elmentwise operation. In addition, in each divergence in the lines the vector is copied, while each convergence represents a concatenation.

The mathematical background for LSTM is analogous to the one in Equation 2.12. However, the output and hidden states take into consideration the internal gates. Therefore, an LSTM is, usually, defined as follows:

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \tag{2.14}$$

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \tag{2.15}$$

$$\tilde{C_t} = \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c) \tag{2.16}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.17}$$

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \tag{2.18}$$

$$h_t = o_t * \tanh(C_t) \tag{2.19}$$

In equations 2.14 to 2.19, σ is an activation function (usually the sigmoid function, as it is differentiable and produces continues values between 0 and 1), x_t is the input at step t, h_{t-1} is the previous hidden state, C_{t-1} is the previous cell state, f_t is the forget gate, i_t is the input (write) gate, $\tilde{C}_t$ is the candidate state value, C_t is the current cell state, o_t is the output (read) gate, h_t is the LSTM current hidden state, and $W_{i,f,c,o}$, $U_{i,f,c,o}$ and b are weight matrices and bias vectors, respectively. This is just one of the possible formulation for LSTM, as there are other approaches. However, for the sake of argument we only present here a well known version, for more variants the reader is encouraged to read Greff et al. (2015).

The decision process of an LSTM network can be described in the following steps: first the network decides which information should be "forgotten" via the forget gate, while deciding which values should be updated via input gate. Then, the network updates its cell state, this is accomplished by first computing the candidates values that could be added to the state, and conjugating which information should be forgotten (forget gate decides what should be forgotten from the previous cell state) and what new information should be written into the cell state (combining the candidates values and the input gate). The final step is deciding what to output. The output gate decides which parts of the cell are going to be outputted and is followed by a non-linearity activation ($\tanh$) to push the values between -1 and 1 and multiplied by the output gate. This product will be the cell hidden state h_t (Olah 2015; R2RT 2016).

2.3.3.2 Gated Recurrent Unit

The GRU was proposed by Cho et al. (2014) as a variation of the LSTM, where instead of coordinating the writes and forgets, the GRU links them explicitly into one gate, called the update gate (which acts as a "do-not-update"). Furthermore, the GRU replaces the "selective writes" and "selective forgets" by a single "selective overwrites". This is accomplished by setting the forget gate to 1 minus the input (write) gate (which is in fact specifying how much of the previous state the cell should not overwrite). The GRU cell internal architecture is depicted in figure 2.9.

The fundamental equations of the GRU are the following:

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z) \tag{2.20}$$

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r) \tag{2.21}$$

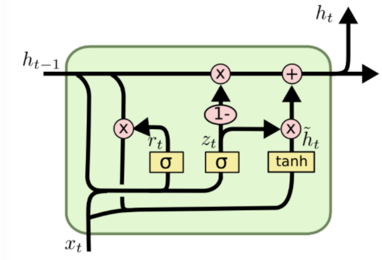


Figure 2.9: Internal GRU structure. Extracted from (Olah 2015).

$$\tilde{h_t} = \tanh(W \cdot (r_t * h_{t-1}) + U \cdot x_t + b) \tag{2.22}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t} \tag{2.23}$$

In Equations 2.20 to 2.23, x_t is the input at step t, h_{t-1} is the previous hidden state, z_t is the update gate, r_t is the reset gate, $\tilde{h_t}$ is the shadow gate, h_t is the current hidden state, and $W, W_{z,r}$, $U, U_{z,r}$, and b are the weight matrices and bias vectors, respectively. Note that the original formulation defined r_t as the reset gate, however this gate functions more as a read gate.

Both GRU and LSTM address the limitations of the vanilla RNN, however which one is better is yet not clear. Different studies concluded that the architectures work best in different problems (Chung et al. 2014). An anecdotal comparison is that, usually, GRUs converge faster and with less training examples, however LSTMs produce better results (again the architectures work best in different problems).

2.4 Summary

In this chapter we presented the background for deep learning, topic models, and DS, as they are all relevant for the context of open domain. A DS can be regarded as an agent with a communicative goal with different initiatives, proactive, reactive, and mixed. Moreover, DSs are usually built using one of two views: as an interface; as a simulation. The DM has an important role in the open domain context as it interacts directly with the generation module. In addition, the approaches to open domain in DM are, mainly, statistical and focus on POMDP and different policy optimizations to refine domains in an ontology graph.

A topic model is an unsupervised generative method for modelling the relation between words in different documents and cluster in different topics. A well known topic model is the LDA, which makes a bag-of-words assumption over the data and clusters different words into different topics.

Finally, ANN can map any function in a compact set and DNNs are just ANNs with multiple layers. Thus, deep learning is using deeper neural networks architectures. We described what is a DNN, how the network is trained and the power of ANN. CNNs are widely applied in computer vision and can be regarded as a network that learns which are the features that best represent the input. Moreover, RNNs can model arbitrarily long sequences but suffer from practical problems, namely vanishing and exploding gradients. However, different specializations have been proposed, namely LSTM and GRU, which use gates to control how the information flows inside the cell.

In the next chapter we discuss the approaches to our main focus, NLG, and contextualize them in the open domain problem, where we will use the terminology of reinforcement and deep learning, we do not discuss the mathematical background further.

3 Related Work

In this chapter, we provide the reader the related work on NLG, which is the main focus of this work. We start by presenting the different approaches in the literature, namely template-based approach, conventional pipeline approach, and statistical, and discuss the differences between them. Then, we contextualize NLG and open domain and discuss the best approaches to address open domain in the generation module.

In the previous chapter we provided the required background in topic modelling, deep learning, and DSs. The background in DS is relevant to understand where the generation module is integrated, as well as some approaches to DM and generation by jointly optimizing both components. Nevertheless, the management module is not our focus in this work, thus, we do not discuss more details of DM in this chapter. The deep learning and topic modelling background is relevant in the context of statistical approaches and open domain in NLG.

Regardless of the approach, the process of language generation is often viewed as goal-driven communication (Reiter and Dale 2000). Consequently, a *communicative goal* or *communicative intention* is attempted to be satisfied in order to produce an utterance. This communicative goal or intention is abstract, its interpretation is not fixed, and is often to inform the listener, request or persuade the listener to do something or obtain information from the listener. To satisfy those goals, a *communicative act* is decided and performed, so that the listener understands the speaker's intention. Furthermore, the interaction between the speaker and listeners can be viewed as reasoning about intentions (Stone 2005), i.e., first, the speaker must utter in such way that his intention is recognizable and, second, the listener must determine what the speaker was trying to do by reconstructing the speaker's intention.

Considering language use as an action underlies *speech act theory* (Philip R. Cohen and Pollack 1993), thus philosophical and linguistic work in this area has a great influence in NLP. This theory is not adequate for multisentential texts since it usually concerns only single sentences. The most influential work to consider multisentential texts from the perspective of the underlying intentions of the speakeris Rhetorical Structure Theory (RST) (Mann 1984; Mann and Thompson 1988), which is concerned with both inter-sentence and intra-sentence relations.

NLG is characterized as choosing "how" to say what is intended, i.e., after "what to say" has been determined, NLG is responsible for determining "how to say it". Therefore, NLG is, usually, composed by a variety of decisions. These decisions are, for example, content determination, referring expression generation, anaphora generation, ellipsis, aggregation, among others (each of this aspects, by itself is a research subject, e.g. Dale and Haddock (1991)). The "what" is usually determined by a DM, regardless of the approach (section 2.1.3), and the "how" is usually defined by templates or hand-coded rules. The aforementioned approach is the first one of the possible approaches, as generation has three different main approaches. First and most used is generation with *templates or hand-coded rules*. This is the main approach in industrial dialogue systems. Second, the *conventional* approach, as described in the literature (Reiter and Dale 2000), follows a pipeline architecture with three components (text planner, sentence planner, and surface realizer). Third and last, *trainable* generation, uses automatic techniques to train NLG modules and adapt the system to specific domains and/or users. Each of these approaches is described in detail in the following sections.

3.1 Templates and Hand-coded Rules

Template-based or hand-coded rules are often developed as systems that map their non-linguistic input directly to the linguistic surface structure. This linguistic structure may contain gaps and these are filled with the linguistic input. Thus, a well-formed output results when those gaps are all filled, more precisely, as noted by Van Deemter et al. (2005), when all the gaps have been replace by linguistic structures that do not have gaps. A well-known example is *D2S* (Theune et al. 2001), this system consists of two modules, (i) a language generation module (LGM) and (ii) a speech generation module (SGM), which turns the generated text into a speech signal.

3.2 Pipeline Conventional Approach

Reiter and Dale (2000) propose that in order to develop a NLG system, one should consider dividing the problem into modules, where each one performs one or more tasks. The input of a NLG system is formalised as a four-tuple $< k, c, u, d >$, where k is the knowledge base, c is the communicative goal to be achieved, u is a user's model and d is the discourse history. In this sense, the conventional approach, usually, consists of three modules (i) document planner, (ii) sentence planner and (iii) surface realizer. Therefore, each one of these modules is responsible for some of the tasks in "how to say it". This particular approach is also responsible for "what to say", as the first module is responsible for deciding it. Moreover, this approach can be decomposed into the last two modules and integrated in a dialogue system with the DM (responsible for "what to say" and decide the communicative goal). The first module – document planner (also knwon as sentence planner) – is, usually, responsible for content selection and discourse structuring, this is, its purpose is to determine the informational content and the structure of the document that meets the communicative goal, "what to say"; the second module – microplanner – is responsible for the selection of attributes for referring expressions, aggregation and selection of lexical items, this is, its purpose is to refine the document plan in order to produce a more fully-specified text specification and, last, the third module – surface realizer – is responsible for the linguistic realization as well as structure realization, i.e., its purpose is to convert the text specification (sentence plan) into natural language.

This approach has also been addressed as a problem of planning, treating, this way, the construction of sentences as a rational activity and using planning mechanisms to construct those sentences. Appelt (1992) was one of the first to propose such system and focused on the high-level structure of the task, such as speech acts (Cohen and Perrault (1979) work, which is also based on planning) and discourse theories, among others. However due to the inefficiency of the planners available at the time, this approach was not tractable for real-world systems. In this sense, recent approaches have also addressed this problem, namely, the systems CRISP (Koller and Stone 2007), SPUD (Stone, Doran, Webber, Bleam, and Palmer 2003), PCRISP (Bauer and Koller 2010), among others. These approaches use Tree Adjoining Grammars (TAG) , and Lexicalised Tree Adjoining Grammars (LTAG), in order to achieve what they propose. A TAG is, roughly, a finite set of elementary trees, for LTAG the trees are lexicalised. Additionally the CRISP system distinguishes itself from the approach by Appelt and Cohen and Perrault by focusing on the grammatically specified contributions of each individual word to the syntactic, semantic, and local pragmatic goals of the generator.

The SPUD system (Stone, Doran, Webber, Bleam, and Palmer 2003) was the first to approach the problem with TAG. The system assumes the existence of a lexicon of entries consisting of a TAG elementary tree annotated with semantic and pragmatic information, and its main goal was to merge the determination of referring expressions with the realization step as these steps perform well together. However, SPUD had to explore a search space that was too large and resorted to a greedy search (not complete nor optimal, with both time and space complexity $O(b^d)$, where d is the maximum depth of the search). To address this limitation, Koller and Stone propose the CRISP system (Koller and Stone 2007). Koller and Stone system regards sentence planning and surface realization as one step, and differs from the previous in, first, encoding the problem in Planning Domain Definition Language (PDDL), the input language for recent planning systems, and, second, the manner how they formalize the semantic content of a lexicalised elementary tree t, using the semantic roles present in the tree as arguments to the finite set of atoms that define tree t. In addition, CRISP depends of a grammar in order to solve the problem. However the problem of deciding whether given a communicative goal if it can be achieved with a given grammar is NP-complete.

3.3 Trainable Generation

The trainable approach is, currently, the one being further developed as its goal is to avoid any hand-craft decision – this way overcoming domain-dependent generation. There are several distinct methods of approaching the problem, for instance, automatic techniques to train NLG modules (Stent, Prasad, and Walker 2004), or using RL to model the optimisation and planning of sequence of actions-in-context (Lemon 2011), or Hierarchical Reinforcement Learning (HRL) (Dethlefs and Cuayáhuitl 2010). Early work in this approach focused on trainable modules within the framework of Reiter and Dale, i.e., using supervised learning to build models that were then used in decision processes of the modules and over-generate and rank. Such systems are exemplified in HALLOGEN (Langkilde and Knight 1998) and SPOT (Walker, Rambow, and Rogati 2001). The first had the premise of using n-gram language models to re-rank a set of candidates generated by a handcrafted generator and the latter the premise of over-generation and ranking, specifically, ranking rules learned from a corpus of manually ranked training examples. However, the latter was unable to produce utterances with variation in the rhetorical structure and was limited to learning to optimise speech-act ordering and sentence structure choices.

Furthermore, there are other systems with trainable modules within the generation framework to allow the model to adapt to different domains and also based in over-generation and ranking, particularly, the approach by Walker et al. (2007) of treating generation as a ranking problem, over-generate with domain-independent rules and then ranking based on user corpora or feedback, and SPaRKy system (Stent, Prasad, and Walker 2004), an extension of SPOT to sentence-planning and showing that the training technique used for that system can be extended to a new domain. Analogously, other approaches are in this scope, particularly corpus-based approach that aim to learn to generate from data also adopting over-generation and ranking, Oh and Rudnicky (2002) used bigram language models, and more recently Wen et al. (2015), used RNN based on Recurrent Neural Networks Language Model (RNNLM) (Mikolov et al. 2010) and using CNN to validate the semantic consistency of candidates during re-ranking, where the final response derives from re-ranking a set of candidates created by a stochastic generator. Moreover, Wen et al. (2015) use LSTM to learn from unaligned data and jointly optimise the sentence planning and surface realisation components (they treat the problem as one step analogously to what Koller and Stone proposed, as described in section 3.2), without using any heuristics, learning the control of gates and surface realisation jointly – this implies even less manually defined parameters. Serban et al. (2015), Li and Jurafsky (2016) propose learning generative models, namely variational latent models in two different contexts: the first models predicting the next utterance in an hierarchical model, while the second proposes learning topic context directly from data. These approaches are further discussed in section 3.4.

Another approach to the problem is using RL (section 2.1.3.5) to jointly optimise some (or all) of the steps in the conventional framework. The main reasons of approaching generation as a RL problem are: (i) reducing and eliminating all hand-crafted rules or thresholds, and (ii) treating generation as an optimisation problem. Moreover, an important reason to use RL in this problem is that this method is concerned with the problem of learning from the interaction between an agent and the environment, in order to achieve a goal. Although there were approaches that jointly optimise DM and NLG (section 2.1), in this section only approaches to NLG are presented. Of the recent approaches, we emphasize Rieser and Lemon (2010) in the context of DS, Lemon (2008) treating NLG as statistical planning and Dethlefs et al. (2011), Dethlefs and Cuayáhuitl (2011) dividing the problem into sub-problems and applying HRL to optimise (and solve) it.

Furthermore, Lemon (2008), in the context of information presentation, formulates the problem as statistical planning and optimisation using an MDP to achieve what is intended. Thus, trying to take advantage of an MDP-based planning and learning framework, analogously to what has been developed successfully in dialogue management, for instance Walker et al. (1998), Young (1999). Additionally, Lemon proposes the joint optimisation of the process of generation for information presentation, i.e., optimise the process as one step, training the policy with hierarchical SARSA algorithm. Lemon's work focused in a simple information presentation context, listing items about a restaurant domain, nonetheless obtaining fine-grained adaptations to context and automatic optimisation. Furthermore, the author suggests that this method could be applied to each of the decisions of generation separately.

Rieser and Lemon (2010), also in the context of information presentation, formulate the problem as an MDP and use the SARSA algorithm to train the policy, achieving (as the latter) fine-grained adaptations to context and automatic optimisation. As already mentioned and suggested by Lemon (2008), one can optimise each step of the process of generation, as such Dethlefs et al. (2011) formulate the problem as HRL to jointly optimise some of those steps, namely content selection, choice of text structure, referring expressions, and surface structure.

HRL is RL that optimises a hierarchy of policies on a hierarchy of agents. This hierarchy consists of L levels and N models per level and each agent is defined as an Semi-Markov Decision Process (SMDP). An SMDP is structured in a similar way to an MDP, however the reward is defined as the one received from taking an action a in state s and that lasts τ steps, this random variable represents the number of steps to complete an action. Thus, HRL is useful for tasks where the search space is too large, in contrast with "flat" RL as it may not be applicable to such task.

Using HRL, Dethlefs et al. approach the generation problem as an optimisation one, particularly to some of the steps of the conventional approach, training the policy with the hierarchical Q-learning algorithm, this is a hierarchical perspective of the traditional Q-learning algorithm. Formulating the problem as a hierarchy of sub-problems and decomposing the problem, allows having context-independent policies. Thus, the possibility of policy reuse and facilitating state-action abstraction. Additionally, the hierarchy is defined by 15 agents, where each agent is responsible for some subtask inside a "cluster" task to which it belongs. The number of agents is proportional to the task. This approach was developed in the context of giving navigation instructions (in GIVE domain) and each agent is responsible for a subtask of that domain. However, not every step of the process is fully automatic, as there is prior knowledge in the agents: leaf agents in the hierarchy were initiated with values from a hand-crafted language model, this prior knowledge is used by the reward function. Besides this approach, and using a similar one, Dethlefs and Cuayáhuitl formulate the problem also as an HRL to jointly optimise the generation process, this time only content selection, utterance planning, and surface realization, using a Bayesian Network (BN) for surface realisation. The BN was manually constructed, taking into account two dependencies (i) information need, must be a major influence in including all optional semantic constituents and the utterance's process, and (ii) the believe that there is a sequence of dependencies that spans from the verb to the end of the utterance (formulating the hypothesis that each constituent can be estimated based on the previous, as a BN allows). As an HRL problem, a reward function was defined based on the PARADISE framework (section 2.1.3) for content selection and utterance planning; for surface realization, a reward based on data collected from users was proposed, learning this way to balance the most likely surface form.

Finally, recent efforts to address statistical NLG in the context of open domain have been develop, namely by Li and Jurafsky (2016). Li and Jurafsky proposes instead of training a discriminative model, training generative models, namely Seq2Seq (Sutskever, Vinyals, and Le 2014) models to address coherence in open domain. In addition, Li and Jurafsky first propose using a vanilla Seq2Seq model, however conclude that using a vanilla Seq2Seq model does not address relevant features of discourse, namely topics. Moreover, Li and Jurafsky propose using a Markov Topic model (an extension to LDA), where the topic influences the prediction of the next word, and conclude that using a topic model for open domain is too generic, missing the fine-grained relations in each topic. Finally, the authors propose a variational latent model, which learns the global discourse information (topics) directly from data, instead of from a model like LDA.

3.4 Open Domain and Natural Language Generation

The NLG component is very important to achieve an open domain interaction as it is responsible for generating utterances that depend on the current context. Thus, this component must be capable of generating a valid utterance regardless of the domain, i.e., this component must be domain-independent.

Recent approaches to address open domain in NLG are mainly statistical. However, there have been approaches to create an hybrid approach, merging statistical and template-based approaches (Angeli, Liang, and Klein 2010): using templates to perform surface realization and statistical learning for the sentence planning component. Particularly, Angeli et al. (2010) divide the generation process into a sequence of local decisions, using domain-independent features responsible for macro content selection, micro content selection, and surface realization. Furthermore, this approach builds generative models for each type of decision previously mentioned, where each model has a specification of the sequence of records chosen (records are responsible for macro content selection), the sequence of fields chosen (fields are responsible for micro content selection, which fields of the records are mentioned), and which words in the text were spanned by the chosen records and fields. Then, the models parameters are learnt in an unsupervised approach using the Expectation Maximization (EM) algorithm, resulting in an alignment that specifies the record decisions and a set of fields for each record. Finally, the templates are extracted from the information acquired in the previous step. There are two templates in this approach: (i) a template that abstracts fields (replacing the words spanned by a field by the field itself), and (ii) a template which only abstracts a subset of the fields. The latter template was necessary due to the noise that can occur in the alignments.

Other approaches treat NLG as an optimisation problem, formulating the problem as an MDP (Lemon 2008) or SMDP (Dethlefs and Cuayáhuitl 2010) and solving it via RL (or HRL), or formulating it as a RNN (Wen et al. 2015) or LSTM (Wen et al. 2015). In the first case, the premise is to optimise some of the steps or tasks of the conventional approach jointly, solving through RL or HRL (usually, Q-learning is the method to optimise the policy, discussed in section 2.1.3.5). In the latter case, the premise is to apply the well-known techniques of Artificial Neural Networks, particularly RNN and LSTM, to the generation process.

Lemon and Dethlefs and Cuayáhuitl formulate the problem as an MDP and as a SMDP, respectively, with the premise of optimising some steps of the conventional approach. In the first case, Lemon's proposal is to jointly optimise surface planning and surface realisation, i.e., solving the problem in one step. In the second case, only some of the steps are optimised, solving the problem with SMDP (solved with HRL) to address the limitation of large state spaces in MDP formulation. Using these methods, the system is able to learn directly from the interaction with the environment.

The main reason to formulate the generation approach as an RNN or LSTM is due to Mikolov et al. (2010) RNNLM formulation, which demonstrated the value of distributed representations and the ability to model arbitrarily long sequences (although it was in speech recognition). This formulation is, briefly, using RNN for language modelling. Mikolov et al. trained the network using the simple feedforward method and, later, backpropagation through time (Mikolov et al. 2011), which outperformed all the other methods in statistical language modelling. A well-known problem of RNN is the vanishing gradient problem, each of the neural network's weights is updated proportionally to the gradient of the error function (regarding the current weight in each iteration of training) and, usually, gradient computation is performed through the chain rule. This implies multiplying n of small numbers in order to compute gradients of the "front" layers in an n-layer network, thus the gradient (error signal) decreases exponentially with n and the front layers train very slowly.

Formulating the problem as an LSTM solves the vanishing gradient problem in RNN: using gate units to avoid input conflicts and controlling the error flow, i.e., the net can use the input to decide when to keep or override information in the memory cell and the output to decide when to access memory cell c_j and prevent other units from being perturbed by cell c_j. Therefore, these gates are responsible for controlling how information is stored, forgotten, and explored. Both Wen et al. 2015 and Wen et al. 2015 approaches have, recently, suggested the ability to easily transfer the generation capability from one domain to a new one, although a smaller set of adaptation data is required. However, in Wen et al. (2015), the behaviour of the CNN was somewhat arbitrary, which implies that the sentence reranking was not as good as was intended. Additionally, this approach still used some hand-crafted heuristics to control the gates.

To address these limitations, Wen et al. (2015) approach does not require the usage of hand-crafted heuristics, as the control is already implicitly done. In addition, the network develop by Wen et al. was composed by a backward SC-LSTM and a forward SC-LSTM, instead of using a bidirectional as the generation process is sequential in time. The backward SC-LSTM was needed because there are sentence forms that depend of the backward context instead of only the preceding history. Thus, the backward network is referred as a reranker and the forward as a generator. Both networks were trained with back propagation through time and all costs and gradients were computed. Moreover, the method to optimise the parameters was the stochastic gradient descent. This approach, as other corpus-based approaches, adopts over-generation and reranking to learn generation directly from data. Wen et al. suggests that due to the ability to model arbitrarily long sequences and a single compact parameter encoding of the information to be conveyed, conditioning the generation with dialogue features is possible (e.g. dialogue information, social cues, among others).

Finally, Serban et al. (2015) proposes learning a variational latent model in an hierarchical model by learning the next utterance in a turn taking dialogue. Moreover, following this approach, Li and Jurafsky (2016) propose using Seq2Seq models using topics as a global information for discourse coherence in an open domain set. Li and Jurafsky's variational latent model addresses the limitations of using a generic topic model in open domain, the loss of fine-grained relations, and learns the global information directly from data with DNN.

The approaches that regard NLG as an optimisation problem are, usually, data-driven, i.e., use training data to build models and optimise parameters and then validate those models with test data. In this sense, the construction of the dataset is important, as the dataset is one of the most important components of data-driven approaches. For both Wen et al. 2015 and Wen et al. 2015 approaches, the target domain is information about restaurants and venues in San Francisco, respectively, using 8 dialogue act types, such as inform, confirm, reject, among others, and 12 slots (these are represented through ontologies). Both training corpora were constructed using dialogues collected from previous user trials, then randomly sampled and shown to workers via Amazon Mechanical Turk service. In turn, for each each dialogue, the workers were asked to enter an appropriate response for the system. Finally, the dialogue acts filled with slots were processed and group to obtain distinct dialogue acts. For RL formulation problems, Lemon addresses dialogues within the domain of information presentation, namely listing items, and Dethlefs and Cuayáhuitl uses a dataset of navigation instructions domain in a virtual 3D world in the GIVE scenario.

3.5 Discussion

Hand-coded rules are not sufficient for open domain generation, as it would require that for each new domain, the creation of a new set of rules, even though automatically extract rules bootstrapped from a corpus (Song, Howald, and Schilder 2015) is possible. However, template-based generation, although often associated with the hand-coded rules approach, generates acceptable utterances. This approach is often regarded as inferior to others, but template-based and standard NLG systems have been proven as being "Turing equivalent" (Reiter and Dale 1997). both can generate all recursively enumerable languages (Van Deemter, Krahmer, and Theune 2005). Furthermore, although template-based approaches have been claimed as being inferior to others when considering their maintainability, linguistic well-foundedness and the quality of generated utterances (Reiter and Dale state that these systems are more difficult to maintain and update and produce poorer and less varied output than standard NLG), van Deemter argues that this approach is not as inferior as claimed. This approach alone is not suitable for an open domain approach, however recent approaches that use template-based and trainable generation together, namely Angeli et al. (2010), have approached open domain generation. This approach was already discussed in section 3.4. Nevertheless, it is important to mention that Angeli et al. divide the generation process into a sequence of local decisions, using domain-independent features responsible for macro content selection, micro content selection and surface realization. Furthermore, for each of those decisions a model is built and the parameters are learned unsupervised using EM algorithm. Therefore template-based generation by itself is not as adaptive as the trainable approach: it does not adapt to context nor the environment, e.g. user. Even though template-based approaches have had an amount of success when used in different domains, the scalability across domains was not that successful. On the other hand, the trainable approach has shown better adaptability and scalability (Rieser and Lemon 2010; Lemon 2011; Wen et al. 2015; Wen et al. 2015; Walker et al. 2001). In this adaptability, it is included that template-based approaches tend to be excessively repetitive, which sustains the lack of adaptability to the context (user, environment, among others). Therefore, a hybrid approach where the realization is performed by templates and the sentence planning is trained and both are jointly optimised, may be a possible approach for open domain generation.

In the conventional approach, rules are, usually, developed (hand-crafted) for a particular domain. Then, the rules are tuned for that domain and, as such, the ability to adapt to fine-grained changes (whether its dialogue context or user behaviour, among others) is limited. This suggests that this approach may not be suitable to open domain, as it is generally developed for a domain. Although the planning approach is promising, as noted by Koller and Petrick the modern planners used for NLG suffer from practical problems (complexity of the algorithms may lead to be intractable for real-world systems). This way, limiting their performance (for real-world systems), even if the techniques to control the search in a sophisticated way proved to improve the tractability. Thus, even if this approach promises portability across domains and uses general rules for each generation module, the quality of generated utterances,f or a particular domain, may be inferior than the ones generated by a template-based approach.

In the trainable generation approach, the corpus-based approaches are the ones with fewer (if any) parameters or hand-crafted rules to be tuned. This is important because to be able to build an open-domain approach the fewer the parameters, the closer to an open-domain the system is, otherwise it suffers from the same limitations as template-based approaches. Although these approaches have limitations, e.g. data availability, they also have showed promising results in adaptability to changes in the environment and previously unseen concepts, important features in the scope of this work, regardless of how the problem is formulated. As such one can argue that this approach might be suitable for an open domain environment as there is no need to manually design the system for a specific domain. However, as pointed out by Walker et al. (2001), even with the increasing amount of available data, this approach suffers from the need for annotated training data. In addition, as pointed out by Wen et al. (2015), the approach has weakness in the utterance naturalness and training data efficiently and accurately, even though Wen et al. approach addresses these weakness and generates more natural utterances than previous approaches, more efficiently.

Concluding, one can argue that among the advantages in using the trainable approach to generation are the adaptability to fine-grained changes in the context of the dialogue (e.g. user, environment), being a data-driven approach, being a precise mathematical approach, being capable of generalizing to unseen utterances, among others (Lemon 2008). Formulating the problem as an MDP and using RL to solve it has those advantages. Additionally, learning is done through interaction with the environment, a desired characteristic in the scope of this work. The possible problems of using an MDP and RL to solve the generation problem are, among others, the time to converge, choosing the proper method to train the policy, the large state space (HRL can be used to address this limitation) and the need for larger initial training set. Even with these disadvantages, this approach has key advantages when comparing to template-based and conventional approaches. Additionally, as demonstrated by Dethlefs and Cuayáhuitl, jointly optimising the process provides better results than optimizing some of the steps and using a BN for surface realization generates language similar to human (better than other approaches). The authors also suggests that this work might be portable to other domains (transfer learning might be successful), which may be helpful in the context of this work.

Formulating the problem as an LSTM, as Wen et al. (2015) recently proposed, also has the aforementioned advantages, not relying on any manual handcraft parameter and exploring the ability to model arbitrary long sequences that all RNN-based architectures possess. This ability and a single compact parameter encoding of the information to be conveyed, it is possible to condition the generation with dialogue features (e.g. dialogue information, social cues, among others). Although this approach has the potential to scale across domains, as a corpus-based approach it also suffers from the need of data availability. Nonetheless, this approach, as well as MDP ones, are more suitable for an open domain approach than the template-based or conventional ones. Consequently, this approach is, possibly, the best suited approach for an open domain interaction.

Furthermore, formulating the problem in a similar way as Li and Jurafsky (2016) allows to address open domain in a statistical way. By providing the DNN the global information, conditioning the prediction with the topic information, it is possible to take advantage of the RNN sequence power for generation while improving the coherence of the generated utterances.

3.6 Summary

In this chapter we presented the state of art in NLG, as well as the existent approaches, namely template-based, conventional pipeline, and statistical. The generation process is, regardless of the approach, a goal- driven communication, as it maps goals into utterances.

Template-based approaches rely on hand-crafted rules tuned for the domain in question and do not generalize to new domains, as the templates fill empty slots. The conventional pipeline approach proposes dividing the problem into three modules: document planner, sentence planner, and surface realization. Each one of the modules is responsible for different tasks in the generation process. The data-driven approaches regard the pipeline architecture as a statistical optimization problem, usually addressing one or more components, as well as jointly optimizing the components. Moreover, the statistical are usually based on reinforcement learning or using supervised methods, in recent times deep learning. Although reinforcement learning, or even deep reinforcement learning, is an interesting approach, deep learning approaches have shown their potential to model arbitrarily large sequences by using RNNs.

Therefore, although there are hybrid approaches between template and statistical approaches, the best suitable approach for open domain is using statistical methods. The recent methods used for the generation process rely on deep learning and are the baseline for our approach described in chapters 5 and 6.

4 Open Domain and Domain-specific Corpora

In order to study open domain generation, especially using a data-driven method, our dataset must meet the following requirements: coherent discourse structure, domain dependent discourse, and a large domain independent vocabulary. Therefore, to meet these requirements we have two different corpus in our dataset. The first corpus, documentaries subtitles, provides the first and second requirement, while the second, Google n-gram (Brants and Franz 2006) provides the last requirement. Furthermore, our documentaries subtitles focus on a specific domain, physics, and their nature provide a domain dependent discourse, as the subtitles address the domain in question, thus having a more domain dependent and limited vocabulary. In addition, the subtitles provide a coherent discourse structure by being coherent with respect to itself and to the documentary, respecting the documentary temporal nature. Finally, the Google n-gram corpus provides a large domain independent vocabulary. The need of a large vocabulary comes from, being able to generate utterances in open domain, as the vocabulary addresses most of the language.

In this chapter, we describe our large domain independent vocabulary and representation, using word representations. We describe our corpora and our approach to build a corpus from subtitles. Our original corpus is an extension of the documentaries subtitles set described by Aparício et al. (2016). The corpus was gathered for abstract summarization tasks and was also used for topic modelling, which is a relevant part of this work (even if it is not the focus). Moreover, this corpus consists of 265 subtitles documentaries of the physics domain, where almost all the documentaries are monologues and the narrator presents the different aspects of the documentaries' subject. Therefore, this corpus is, potentially, adequate for the focus of this thesis: on the one hand, the corpus can be used for to topic modelling and, thus, is suitable for the domain part of the work, and, on the other hand, it enables the narrator/explainer to demonstrate the purpose of this work.

Furthermore, we exploit the temporal nature of documentaries to build our own corpus. We detect scene's boundaries by interpreting the subtitle artifact and collapsing subtitles temporal alignments that meet a criteria, while preserving the original temporal alignment with the documentary. From the narrator perspective, on one hand, we have the natural explanation from the documentary narrator, and, on the other, we could, potentially, exploit the scene in the video documentary to further explain the concepts.

We also describe and analyse our approach to topic modelling, for each we use the LDA.

In addition, we also analyse how does the corpus we build affects the topic models.

4.1 Large Vocabulary Corpus

Having a large vocabulary is a very important requirement to address generation in an open domain context, as the vocabulary is not limited and covers most of the words in the language. This way, when generating an utterance, it is possible to cover, potentially, most of the words from any specific domain.

Thus, we use the Google's ngram dataset (Brants and Franz 2006) as our domain independent corpus, as its vocabulary covers must of the English language. Moreover, how we represent the words is also a relevant aspect of the large vocabulary, as we want to have a generic space where words preserve the relations between one another. Thus, we represent the words using word2vec (Mikolov et al. 2013), i.e., we transform the discrete word space into a continuous dense space that preserves words relations.

Briefly, there are two architectures for generating the representations for the words, Continuous Bag of Words (CBOW) and Skip-gram. Both have advantages and disadvantages with respect to each other, as well as the reason behind using one or another. The CBOW focuses on, given a context (words), predict the next word, while the Skip-gram focuses on, given a word, predict its context (words). While Skip-gram can produce a good representation with a small amount of training data, even for infrequent words, CBOW needs a larger amount of data. However, CBOW is faster to train and can produce better accuracy for the frequent words. Another aspect, which we will not further developed, is how these models are trained, namely the normalized hierarchical softmax and the un-normalized negative sampling. Mikolov et al. suggested that using Skip-gram with negative sampling is the best approach, as it outperformed the other architectures on different tasks.

Therefore, we only consider the Skip-gram model with negative sampling and use Google's ngram dataset (Brants and Franz 2006) to train the word2vec model (Ginter and Kanerva 2014), where the size of an embedding is a 200 dimensional vector, the text is normalized using the default normalization provided by the word2vec toolkit, and we use a window of 5 context words for the Skip-gram.

4.2 Building the Domain Dependent Corpus

Although our main focus is open domain, we study how does conditioning generation with a domain dependent coherence is possible. As previously described, we require domain dependent structure and coherent discourse structure. Thus, we use a collection of documentaries of the physics domain for our domain dependent part.

Furthermore, documentaries provide domain dependent structure and coherent discourse implicitly in the scenes that compose the documentary. Therefore, we consider that each scene can be regarded as an individual document, as each scene not only compresses enough information regarding the subject of the documentary and can be more detailed about a particular context of the documentary's subject, but also, provides a coherent domain dependent discourse structuring, each scene is coherent with respect to itself and to the documentary's subject. To achieve this, we propose an approach for detecting scenes in subtitles and extracting them from the original documentary.

4.2.1 Scene detection

Documentaries are, intrinsically, segmented into scenes: some scenes are longer and, others, are shorter. However, regardless of the duration, scenes can be distinguished by their transitions, this is, the time between one scene and another is an explicit marker of the scene. Although the boundary of each scene can be softer or crisper, we propose dividing the scene by using parameters that measure the time between subtitles items (each time alignment in a subtitle) and the distance between different scenes .

Therefore, we exploit the nature of documentaries to collapse the subtitle script (srt) into a new one with merged items. We achieve this through 3three functions, $find\_subtitle$ (algorithm 1), $merge\_subtitles$ (algorithm 2), and $clean\_text$, which removes tags from the text. The algorithm consists of finding a minimum length subtitle item and then merge until the criteria for stopping is met, the distance between the current merged item and the next item exceeds a threshold. Note that each new subtitle preserves the original time frames.

Finally, we perform this merge procedure with a fixed minimum length of $500ms$ and with a distance ranging from $100ms$ to $950ms$. This step was done to analyse how the boundaries vary and how significant is the parameter.

Algorithm 1: Find nearest Subtitle

```
Input: input_file,from_t, to_t, lo
Output: scene_text, index_merge
$i \leftarrow lo$;
while $i < size(input\_file)$ do
    $scene_i \leftarrow input\_file[i]$;
    if $scene_i\ start >= to_t$ then
        break;
    if $(scene_i\ start <= from_t)\ and\ (to_t <= scene_i\ end)$ then
        return $scene_i\ text, i$;
    $i \leftarrow i + 1$;
return "", $i$;
```

Algorithm 2: Merge Subtitle

```
Input: input_file, min_length, distance
Output: subtitle
$begs \leftarrow$ item start for item in input_file;
$ends \leftarrow$ item end for item in input_file;
$j \leftarrow 0$;
$o\_start \leftarrow 0$;
$o\_text \leftarrow$ “”;
for $i \leftarrow 0$ upto size(begs) do
    $start \leftarrow begs_i$;
    if $o\_start == 0$ then
        $o\_start \leftarrow start$;
    $end \leftarrow ends_i$;
    if $end - start > min\_length$ then
        $text, j \leftarrow find\_subtitle(input\_file, start, end, j)$;
        $text \leftarrow$ clean_text(text);
        if size(text) $== 0$ then
            if $o\_text > 0$ then
                $item \leftarrow Subtitle\_Item(0, o\_start, end, o\_text)$;
                add item to out;
            $o\_start \leftarrow 0$;
            $o\_text \leftarrow$ “”;
            continue;
        if size(o_text) $== 0$ then
            $o\_text \leftarrow o\_text +$ “ ” $+ text$;
        else
            $o\_text \leftarrow text$;
        if $i + 1 <$ size(begs) then
            $next\_start \leftarrow begs_{i+1}$;
            if $(next\_start - end) > distance$ then
                if size(o_text) $> 0$ then
                    $item \leftarrow Subtitle\_Item(0, o\_start, end, o\_text)$;
                    $o\_start \leftarrow 0$;
                    $o\_text \leftarrow$ ””;
                    add item to out;
        else
            if $len(o_text) > 0$ then
                $item \leftarrow Subtitle\_Item(0, o\_start, end, o\_text)$;
                $o\_start \leftarrow 0$;
                $o\_text \leftarrow$ ””;
                add item to out;
clean_indexes(out);
return out;
```

4.2.2 Extracting scenes from documentaries

After all the scenes are merged for all different parameters, we have to extract the scenes and create new documents containing only each scene. The process of extracting scenes depends of three parameters, minimum number of words, minimum duration and maximum duration. The first two parameters ensure that leftover text without any relevant contribution from the previous process is eliminated. For instance, sometimes in documentaries instead of spoken language the authors use sounds, such as crackling, to illustrate a concept; those sounds are maintained as they usually fulfils the time constraints. The last parameter is optional and is responsible for ignoring scenes whose boundaries were so crisp they end up collapsing most (if not all) of the original documentary. To understand the effect of max parameter we analysed the previous step (merging scenes) and how many documents were poorly merged. This parameter was set to $2000ms$, while the first and second were set to 5 and none or $100000ms$, respectively.

This process can be disk intensive (the minimum number of documents is now 30000) and we create a new corpus for each of the parameters from this step and from the previous one, which are highly parallelizable. Therefore, we used Condor to parallelize the process of building the corpus. Briefly, Condor is a software framework for managing the workload on a dedicated cluster of computers, i.e., Condor manages submitted jobs and executes them in a cluster of computers by assigning each job to each own computer.

4.3 Corpus Analysis

Creating the corpus influences all the remaining experiments, as such we evaluated how the scene detection, and consequently the scene extraction, performs with different parameters. In Figure 4.1, we depict how using different gap ms parameters, varying from 100 to $950ms$, affects scenes duration distribution, where the x coordinate represents the length in minutes and the y coordinate represents the frequency of occurrence. Moreover, we can observe that the behaviour of the gap is the expected one, as when the parameter is a small value lesser scenes are grouped, which leads to having a greater number of small scenes. However, there is a pattern in the frequency: from $100ms$ to $200ms$ the gap is not stable and the number of small scenes tend to oscillate from one value to the next, then from $225ms$ until $950ms$ the decay is smoother, which leads to conclude that the gap stabilizes and the decay of frequencies is just a natural consequence of the gradual increment of less distant scenes (more scene collapse).

Furthermore, although the frequency analysis leads us to reason that this parameter should be at least $200ms$, we look at the mass of the curve (the integral) to further understand how the parameters influence the boundary detection. Thus, in Figure 4.2, we depict how the variation of the parameters affect the mass of the curve, where the x and y coordinate still have the same

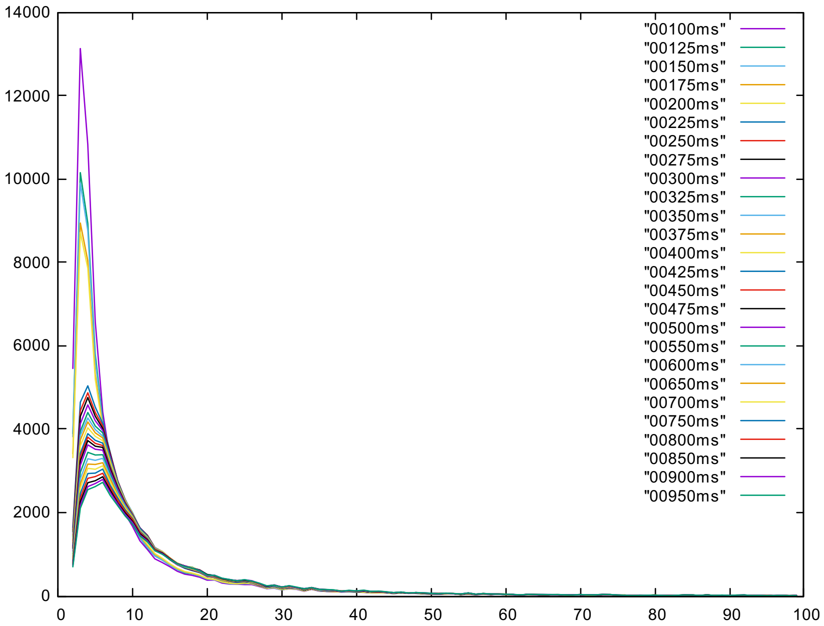


Figure 4.1: Corpus frequencies distribution with different gap parameters.

representation as before but now we present the mass of the frequency. Analysing the previous Figure, not only, the range from $100ms$ to $200ms$ presents the same result, very distant from the other intervals, but also, the mass of the curve concentrates in shorter scenes and decays very fast. In contrast with this range, the $225ms$ to $950ms$ range proves to concentrate the mass in a very similar way, with short to medium size scenes, while decaying smoother than the previous range.

4.3.1 Discussion

The results of the analysis of the scenes merging provide a way of detecting how well we are finding the boundary between the scenes. After analysing Figures 4.1 and 4.2, we conclude that using intervals with a value less than $200ms$ finds very soft boundaries and leads to shorter scenes, while intervals greater than $200ms$ lead to incrementally crisper boundaries and more scenes merged. Therefore, we test all gap scenes corpus to understand the effects of this boundary detection in the topic models, and consequently in the NLG modules.

Finally, for the scenes extraction, we set the parameters for minimum length to $2000ms$ and minimum words to 3, to clean noisy scenes. Another relevant aspect, is that in Figure 4.2 there is a clear cut after 100 scenes, we do not present documents longer than 100 scenes as the mass is negligible. Moreover, the maximum duration parameter of the extraction is tested using the maximum duration of $100000ms$. This parameter is further discussed in section 4.4.

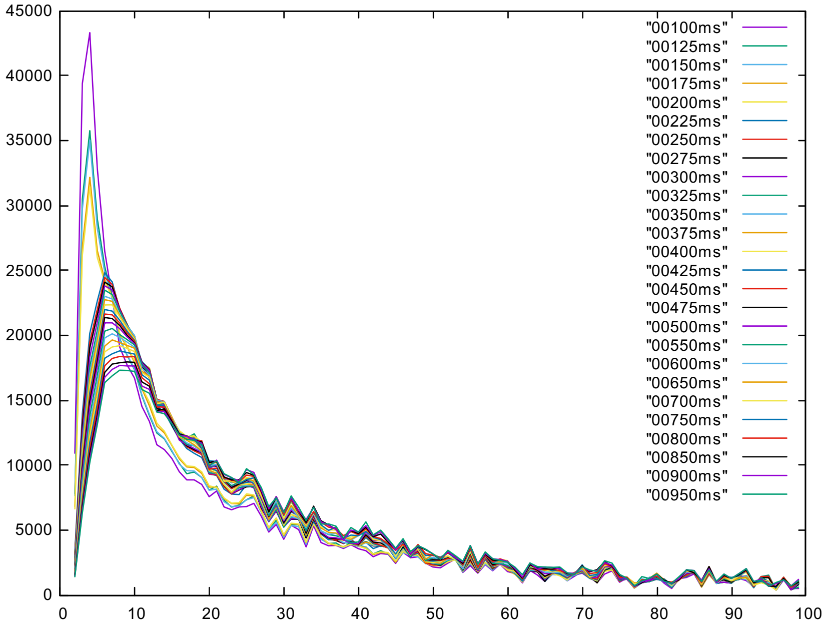


Figure 4.2: Corpus mass distribution with different gap parameters.

4.4 Topic Modelling

Using a topic model provides a natural method for grouping concepts in a collection. This way, we can model the fine-grained relations of our domain dependent corpus. From the different topic models we chose LDA due to its wide application in the literature and the previous application of this model in the subtitles domain (Aparício et al. 2016). Furthermore, we use the LDA implementation provided by Blei et al. (2003).

To perform an LDA estimation over our collection we need to create the vocabulary of the collection, ignoring stop words, and, then, create for each document of the collection the bag-of-words of the scene's vocabulary. After creating the vocabulary, we estimate the model using a random topic initialization, an α of 0.3, and vary the number of topics the model should estimate. We perform the same conditions for 50, 100, and 200 topic models. Finally, we perform inference over a collection of documents from a test set. All experiments were conducted using Condor to speed up the process of automatically obtain the results.

Furthermore, we then produce the top N words for each topic for each topic model. In addition, we also produce a top N words conditioned on the word frequency in the vocabulary (this way, ignoring very frequent words which may pollute the topics quality by being too frequent). An illustrative example is depicted in table 4.1.

Gap	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
$100ms$	sky center night radio bright supernova	star hot death closer white extremely	years billion ago millions ancestors astronomy	built person areas crew signals laboratory	distance kilometers means total completely path

Table 4.1: Latent Dirichlet Allocation 6 words from top 20 example, from a model with 100 topics with 300ms gap.

4.4.1 Analysis

Section 4.3 lead us to analyse how does the merging and extracting parameters affect the vocabulary and the number of scenes. In Table 4.2 we depict how the vocabulary size and number of scenes differ when the boundary is incrementally crisper. Moreover, the number of scenes decreases with the increment of the merging parameters, even more accentuated when the $10000ms$ cutoff is applied. This is explained by boundaries being incrementally crisper, which leads to shorter scenes being merged more frequently. Furthermore, the vocabulary size oscillates, which can be explained by the extracting parameters: softer boundaries will have shorter scenes that when extract may be discarded; while crisper boundaries will have short to medium scenes preserving more vocabulary. In addition, when the cutoff is applied, the number of scenes discarded increases but not significantly, in contrast with the vocabulary, which decreases significantly when the cutoff is applied.

To understand the effect of the vocabulary and the number of scenes, we estimate LDA models for all the boundaries and analyse the quality of the topics by computing the top 10 and 20 words for the different 50, 100, and 200 topics models. In Tables 4.4, 4.5, and 4.6 we depict the top 10 words of the first five topics for four different topic models.

Furthermore, we evaluate how many overlapping words there are between topics, so as to understand whether the topic model is performing a softer or crisp boundary between topics. Table 4.3 presents max number of word co-occurrence in different topic models. From Tables 4.3 to 4.6, we can conclude that each of the models have a crisp boundary, as the maximum word topic co-occurrence is 14 for 50 topics in scenes merged with a gap of $500ms$ with cutoff and $900ms$. Although each model can be used for further experiences with sentence planning and surface realization, we only consider the $300ms$ gap merged scenes with 100 topics, as its behaviour is regular both in the corpus analysis and in the LDA topic model analysis.

4.5 Summary

We described our dataset requirements: coherent discourse structure, domain discourse structure, and a large domain independent vocabulary, and which corpora meets these require-

Scenes Gap (ms)	Vocabulary Size	Number of scenes
100	9574	62035
100 *	9052	61635
200	9635	52654
200 *	8993	52597
300	9660	44326
300 *	8987	44058
400	9655	42571
400 *	8984	42107
500	9658	40832
500 *	8965	40357
600	9660	39510
600 *	8945	39025
700	9660	38389
700 *	8938	37895
800	9661	37178
800 *	8919	36665
900	9662	36087
900 *	8896	35563

Table 4.2: Merged Scenes Statistics (* cutoff of $100000ms$).

Gap (ms)	50 topics	100 topics	200 topics
100	2	2	2
100 *	2	2	2
200	4	2	2
200 *	3	2	2
300	12	3	**4**
300 *	5	4	2
400	13	3	3
400 *	9	2	2
500	10	3	2
500 *	**14**	2	3
600	10	3	2
600 *	5	2	2
700	12	**6**	2
700 *	12	2	3
800	13	4	2
800 *	10	2	2
900	**14**	**6**	3
900 *	13	3	2

Table 4.3: Word topic co-occurrence (* cutoff of $100000ms$).

Scenes Gaps (ms)	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
100	galaxy	science	live	kind	matter
	large	idea	ways	sky	made
	center	lead	search	night	place
	milky	blood	longer	exist	dark
	picture	ideas	happening	making	call
	scale	open	food	built	existence
	radio	rest	carbon	discover	elements
	group	bad	result	machines	birth
	region	minute	intelligent	remains	extremely
	telescopes	carry	recently	model	hold
100 *	universe	kind	back	galaxies	thing
	end	spacecraft	important	cosmic	great
	call	living	process	narrator	waves
	beginning	nasa	days	happened	distance
	god	molecules	natural	asteroids	build
	worlds	vast	eyes	comets	sound
	existence	chemistry	research	distant	learn
	birth	kinds	food	violent	fly
	simply	book	absolutely	orbits	middle
	places	engineers	difference	asteroid	knowledge
200	matter	brain	water	human	star
	dark	thought	surface	species	called
	amount	information	rock	death	massive
	enormous	moment	liquid	evolution	bigger
	crater	radio	deep	animals	turned
	ordinary	brains	cold	dinosaurs	named
	antimatter	room	inside	plants	rest
	station	hear	air	creatures	challenge
	straight	freeman	molecules	beings	zone
	wide	activity	titan	humans	rapidly
200 *	lot	nasa	today	things	inside
	asteroid	spacecraft	high	understand	deep
	happened	mission	created	change	ocean
	evidence	apollo	telescope	effect	vast
	event	flight	looked	simple	exist
	land	degrees	interesting	begin	structure
	dinosaurs	finally	chance	low	happening
	clear	minutes	state	slowly	image
	named	crew	city	run	sunlight
	person	rocket	sugar	level	hidden

Table 4.4: Top 10 words of Latent Dirichlet Allocation 50 topic models (first five topics) for different gaps (* cutoff of $100000ms$).

Scenes Gaps (ms)	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
100	galaxy	world	result	kind	place
	milky	science	detect	scale	call
	astronauts	true	nebula	evolved	atoms
	changed	lead	suddenly	model	elements
	group	decades	spinning	stand	extremely
	starting	range	died	unlike	hold
	andromeda	carry	occurred	producing	heavy
	proposed	straight	fine	microscopic	special
	powered	fiction	dying	creature	pattern
	copernicus	war	device	proved	pass
100 *	universe	kind	process	jupiter	understanding
	place	complex	days	cosmic	waves
	call	molecules	order	happened	fast
	beginning	university	crew	asteroids	key
	existence	chemistry	weather	comets	sound
	birth	organic	absolutely	distant	amazing
	expanding	nucleus	difference	neptune	knowledge
	giving	failed	changing	uranus	fit
	creation	grand	desert	reaches	slightly
	imagined	europe	engine	giants	archimedes
200	measure	looked	water	million	scientists
	camera	control	scale	evolution	found
	minute	top	volcanoes	search	ball
	path	radio	active	dinosaurs	named
	straight	room	volcanic	intelligent	major
	familiar	send	lava	including	bacteria
	cut	hear	craters	cats	challenge
	tons	bottom	molten	rare	zone
	trees	signal	volcano	extinction	determined
	lower	signals	flows	fate	community
200 *	asteroid	spacecraft	fact	things	deep
	happened	rocket	created	understand	millions
	event	seconds	interesting	change	ocean
	land	launch	direction	effect	vast
	dinosaurs	shuttle	gave	simple	liquid
	named	active	possibility	begin	happening
	brought	program	satellite	control	sunlight
	including	engine	mountain	low	oceans
	belt	meant	paper	remember	hidden
	extinction	color	fraction	antimatter	beneath

Table 4.5: Top 10 words of Latent Dirichlet Allocation 100 topic models (first five topics) for different gaps (* cutoff of $100000ms$).

ments. We used two corpora, a documentaries subtitles corpus, which provides the first two requirements, and a large domain independent corpus, which provides the last requirement. To represent our large vocabulary we chose using word2vec, which provides dense word representation. Furthermore, we train the word embedding using an ngram model (Brants and Franz 2006), which have demonstrated the ability to maintain the good performance for modelling words distributions (Ginter and Kanerva 2014).

To address the domain dependent requirements, we built our own corpus by exploiting the temporal nature of documentaries: detecting scenes. Scenes provide a natural way of having domain dependent discourse structure, as they address a concept of the documentary, as well as a coherent discourse struture, with respect to yhemselves and the documentary. Furthermore, we describe our approach in algorithm 2, where we detect the gap (in ms) between the subtile's time alignments and collapse items which are at a distance greater than the gap from each other, until the gap is met or there are not more items. We analyse how the parameters affect the corpus by looking at the frequencies and mass frequencies of the scenes durations.

Finally, we estimated different topic models using the subtitle's scenes and evaluate how does the domain dependent corpus affects the vocabulary and number of scenes available for performing the model's estimate. Moreover, we analysed how does the corpus affect the quality of the LDA models and concluded that the topic models have a crisp boundary, where the maximum word co-ocurrence is not significant, specially for 100 and 200 topic models.

Scenes Gaps (ms)	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
100	grow	lead	imagine	bigger	fact
	word	straight	death	unlike	god
	starting	recently	oceans	bones	hand
	laser	growing	explanation	tons	special
	ship	involved	zone	creature	easy
	proposed	floating	dying	smallest	cut
	photons	tissue	habitable	searching	steve
	quasar	lucy	odds	experienced	chris
	powered	mix	crab	peak	bound
	stranger	fortunately	earthlike	expanded	suggest
100 *	narrator	thing	back	happened	key
	began	greater	reveal	asteroids	sound
	orbits	missions	kill	comets	intelligence
	extreme	arrived	sign	neptune	compared
	creation	blasts	battle	uranus	knowledge
	kepler	stuck	jim	giants	language
	destruction	blew	british	lucky	written
	quest	lhc	sail	stretched	limit
	cycle	accelerator	luminous	odd	gods
	managed	quarks	frank	stretching	era
200	major	top	rock	fact	called
	camera	bottom	volcanoes	evolution	key
	familiar	smell	active	dinosaurs	essentially
	fate	train	volcanic	couple	zone
	cost	sleep	craters	catch	immediately
	concentrated	subject	lava	wonders	dramatic
	dollars	sees	molten	village	habitable
	alternative	response	recent	debate	balls
	service	sit	volcano	genome	technical
	pile	moche	geological	garden	claims
200 *	sense	ocean	finally	worked	ice
	slowly	eye	months	solution	cold
	patterns	sunlight	pieces	vision	molecules
	profound	visible	explanation	driving	face
	deal	trillion	provided	secret	gases
	situation	naked	national	recent	sight
	groups	observable	foundation	presence	frozen
	molecular	rainbow	park	patients	polar
	focus	fastest	equally	reaching	freeze
	mine	heats	matters	achievement	snow

Table 4.6: Top 10 words of Latent Dirichlet Allocation 200 topic models (first five topics) for different gaps (* cutoff of $100000ms$).

5 Sentence Planning

In this chapter and the following, we describe our approach to the generation problem, namely our approach to sentence planning, micro-content planning, and realization. Our concern in this chapter is the sentence planner, while in the next chapter we focus on the micro-content planning and realization.

Sentence planning is, usually, responsible for determining the content and structure of the response. As noted by Reiter and Dale (2000), this component is the most important module for many applications, as determining the content (and structure) of the response can be more important to users than the naturalness of the response. Moreover, we approach the content determination problem and structure of the response in different modules, not exclusive to the sentence planner, i.e., we use the sentence planner to determine the content of the response and, implicitly, the structure. However, we explicitly determine the real structure in a micro-content planner which is described in the next chapter.

In this chapter, we present our approach towards natural language generation and open domain, namely for the sentence planner module. We approach the generation problem in a statistical fashion considering the conventional pipeline components as two distinct problems, optimizing each one separately instead of end to end. The reason for approaching the problem this way was to evaluate how each module behaves, more concretely to study if the planner can learn how to determine the content of the response in the domain, while the micro-planner and surface realizer can learn to structure a coherent response and present the response using a generic vocabulary conditioned by the domain. The conventional architecture of Reiter and Dale (2000) is depicted in Figure 6.1, with the contributions for each of the modules.

We approach the problem in the context of building an explainer, given a question about a concept in a domain produce the necessary utterances to explain that concept. This can be situated in a question-answering problem. However this is not the case as the explainer does not simply search for facts in a knowledge base and present them, rather, the explainer should provide an explanation for the questions using a coherent domain dependent discourse, yet addressing the facts as well.

The architecture proposed can be regarded as an encoder-decoder architecture (Cho et al. 2014) where the sentence planner is responsible for encoding what should be said in a context vector, determining the content of the response, which is represented by a topic multinomial distribution, and the micro-planner and surface realizer are responsible for decoding this distribution into utterances by determining the content structure and perform the linguistic realization.

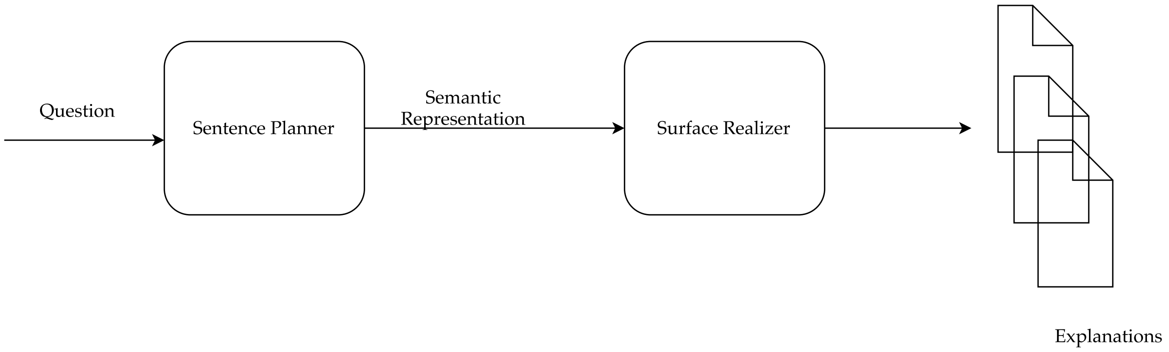


Figure 5.1: Full architecture.

Our sentence planner, learns how to map questions, in a large vocabulary, into a domain specific domain, i.e., our sentence planner has one of the tasks from the pipeline conventional architecture, determining the content in the domain. The sentence planner architecture is depicted in Figure 5.2. In addition, for the domain independent vocabulary we use word embeddings, while for the domain part of the planning we use topic models, Figure 5.3.

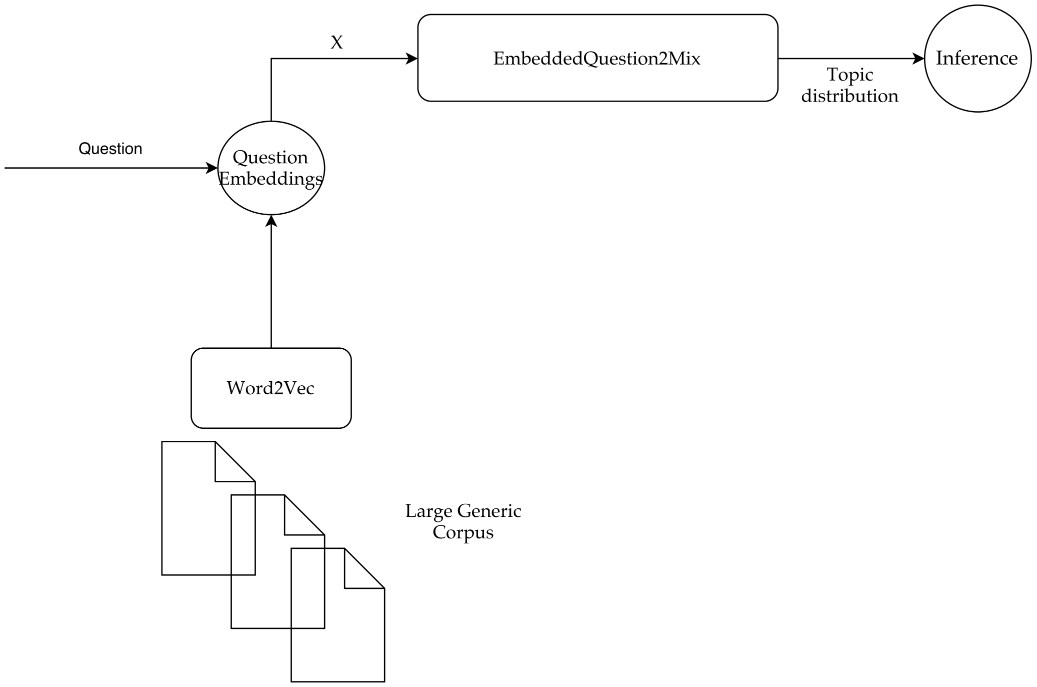


Figure 5.2: Sentence Planner architecture.

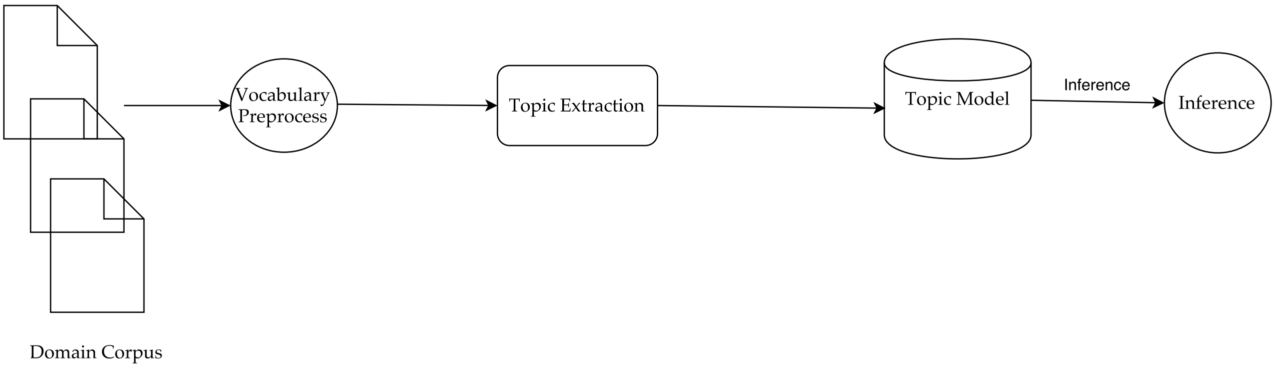


Figure 5.3: Topic model. Domain dependent part of the sentence planning.

5.1 Mapping Generic Embeddings to Topics

The sentence planner must determine the content of the response, and implicitly the structure by being able to map from a communication objective into a non-linguistic representation. Therefore, in our approach we consider the communication objective is implicit in the question and the module has to determine the content, the topic distribution, that characterizes the adequate response to the question.

Our sentence planner is a generic mapping from word representations in a generic space into topic distributions, i.e., the sentence planner learns how words should be encoded into a topic distribution (analogously to what a topic model does). Therefore, our focus on the sentence planner is how to map generic word representations into a topic distribution, i.e., how to perform a mapping from an already dense space which preserves relationships between words into an even denser space which models the distribution of words in a topic space. This can be viewed as lowering the dimensionality of one space to fit into another one and determining the response content in a specific domain.

The contribution of this mapping is to refine the relationship of the words in the higher dimension into one containing more fine-grained relations, while preserving relationships of the words in the given domain. Therefore, one can simply replace a given domain by another, thus, using a domain-independent method for refining words relationships. Furthermore, approaching the sentence planner using a statistical approach, allows to model fine-grained relations with more flexibility than using a template-based or the pipeline conventional approach planning.

Learning how to map from the question into the topic distribution can be regarded as determining the response content: the response must be constrained by the topic distribution as it provides a way to determine which parts are important for the response. While performing reshaping the content in the domain space back to the original space means that the words in the generic space are conditioned by the topic model and preserve the fine-grained relationships of the domain. This is addressed in the next chapter 6.

Finally, in Figure 5.2 the question is transformed from the discrete space of words into the domain-independent space, which is achieved by transforming each word using word2vec trained over a large generic corpus. Next, the embedded question models, implicitly, the communicative goal and the EmbeddedQuestion2Mix is our sentence planner, which maps from the question space (word embeddings space) into the topic, domain specific, space. Thus, determining the content of the response in the domain space.

5.2 Experimental Setup

As already mentioned in section 2.3.1, DNNs can learn arbitrary mappings between two different spaces. Thus, they fit perfectly in this problem, where we want to map from a generic word embedding space into the topic latent space. Thus, we approach the sentence planning using deep learning, namely using feedfoward DNNs and CNNs, to learn a one-to-one mapping, more concretly, to perform the mapping between the generic space into the topic space. In addition, we use word2vec, for the generic space, and LDA for the topic space. We evaluate three different approaches to perform the mapping from the word embeddings (question) into the topic distribution (content determination): a deep feedforward network and two convolutional networks (both 1D ConvNets).

In order to train the networks we conducted two experiments, which are influenced directly by the LDA results. First, we divide the document collection into train and test and perform the model estimate over the training set exclusively, which is then used on the sentence planner as the training set (10% of the training set is used as validation), and evaluate with respect to the LDA inference, comparing this way previously unseen document for both models. Second, we use LDA to infer the topic distributions of the previous estimate model and train the planner with the inferred topic distributions (10% of the training set is used as validation), i.e., instead of using the internal LDA model directly we train with an approximation given by the inference, which is closer to what the planner should predict as it is closer to the LDA inference. Furthermore, we evaluate the models by using the test set which both LDA and the network have never seen. The collection division into train and test is performed randomly.

Summarizing, in the first experiment, the planner learns a mapping from the embeddings space to the LDA estimate space and predicts distributions never seen in the LDA estimation (nor by the network). In the second experiment, the planner learns a mapping from the embeddings space to the LDA inference space, as the train set is the inference of the LDA estimate from the second experiment.

All models are trained to optimize the mean squared error and are compared with respect to the LDA predictions using the cosine similarity:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y} - Y)^2 \tag{5.1}$$

$$cos(\theta) = \frac{Y \cdot \hat{Y}}{\|Y\|_2 \cdot \|\hat{Y}\|_2}. \tag{5.2}$$

Both experiments are evaluated with different models: first, we map a representation of the document into the topic distribution by summing all the word embeddings of the vocabulary of the document; and, second, we use a concatenation of the word embeddings of the vocabulary to map into the topic distribution. For first scenario we used the feedforward network and one of the convolutional networks and for the second we used the remaining convolutional network.

For the first scenario, the feedforward train is straightforward, we feed the network the representation of the document and predict its distribution. However, for the convolutional network we have to set the hyper-parameters to guarantee that the stride is equal to 1, so that the network looks at the only region of space available, the intuition is that the network will learn which dimensions of the word embeddings better describe the sentence's representation.

In contrast with the convolutional approach from the first scenario, in the second scenario we feed the network a concatenation of the vocabulary words. Thus, in this scenario, the network learns which features best represent the vocabulary words with respect to the topic distribution. In addition, this approach requires the zero-padding discussed in section 2.3.2, so that the network can look at different regions including near the border.

Finally, all models were trained using the minibatching technique, which is further discussed in 6.2.1. In addition, we also apply the dropout technique to prevent overfitting. We use a CNN which takes as input the bag of words of the question and an DNN and a CNN which take as input a representation of the bag of words (the sum of all word embeddings). Moreover, we synthesize the questions by using the vocabulary words in the scene and map into their corresponding topic distribution from the LDA. This can be regarded as performing the topic model's inference using neural networks, while mapping from the embeddings space, domain independent, to a topic space, domain dependent.

Furthermore, we experiment different DNNs, namely feedforward and convolutional networks, which are tested with different hyper-parameters, we vary the number of layers and hidden units per layer, as well as the number of epochs (note we use early stopping but it depends on the number of epochs). In addition, all the networks run on Graphics Processing Unit (GPU), namely GeForce GTX TITAN X (Nvidia Corporation 2015), and were developed using the machine learning toolkit keras (Chollet 2015).

5.3 Experimental Results and Discussion

Our sentence planning approach relies on the power of DNN to learn a mapping between an input space and an output space, i.e., our planning approach maps from a domain independent domain into a domain dependent domain. This can be regarded as projecting the generic language vocabulary into a more fine-grained vocabulary, which possess more specific words relations than the generic vocabulary. As already described, we perform this mapping by learning word embeddings into a topic model space.

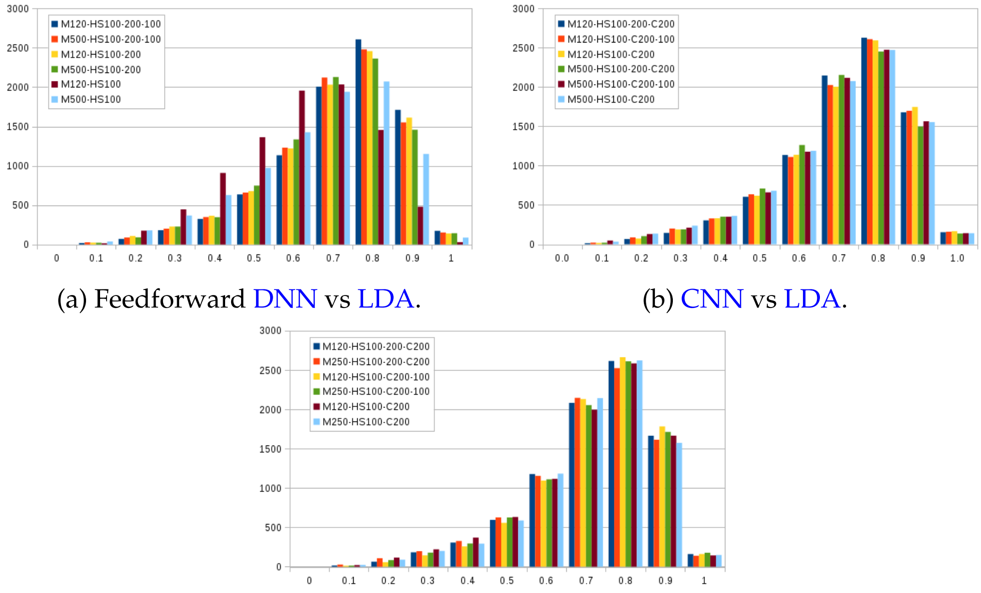
To learn the mapping we conducted two experiments: first, the network learns the topic model estimate and infers never seen documents; second, the network learns the topic model inference and infers never seen documents. This two scenarios are an approximation of the best possible scenario: partitioning the dataset into three parts and train the topic model with the first part, train the network with inferred distributions from the second part, and evaluate both with the third partition. Nonetheless, we show that using inferred estimate distributions provides an approximation for learning the topic model.

The sentence planner was trained to predict the topic model distributions from word embeddings, which synthesise a question. All experiences map from the word embeddings of the vocabulary used to train the LDA models into the topic distribution space. However, LDA models can perform inference over the topic models, which leads us to test how does the network behaves compared to the LDA inference and, further, how do the hyper parameters affect the network's prediction.

We perform the LDA estimation over the training set and the inference over the test set, while the network learns the model from the training set (where 10% is for validation) and predicts the test set. In Figures 5.4a, 5.4b, and 5.4c we depict an histogram of the cosine similarities frequency, while in Figure 5.5, we depict the best results for each one of the networks. We can regard these results as the network trying to learn to estimate the topic model and predict distributions close to the estimates.

In order to mimic the LDA inference, the network must be able to predict the inference space of the topic model, instead of its estimate – although the estimate provides a ceiling of the topic distributions. Thus, we train and evaluate how do the networks perform when learning the inference of the LDA model, even if it is the model's estimate inference. This way, we can understand how does the network learns the topic model's inference by comparing the distributions predicted with the expected distributions of the inference. Moreover, this can be regarded as the network learning how to sample from the topic model, i.e., the network learns a mapping between the embeddings space and the LDA inference space. In Figures 5.6a, 5.6b, and 5.6c we depict an histogram of the cosine similarities frequency.

In Figures 5.7a, 5.7b, and 5.7c we depict a zoom in the range of $0.7 - 09$ of the similarities curve. As expected, the networks' predictions yield better results when the number of layers is



(a) Feedforward DNN vs LDA.

(b) CNN vs LDA.

(c) CNN with vocabulary vs LDA.

Figure 5.4: Cosine similarity frequency of sentence planner predictions with respect to LDA inference. Mapping embeddings to LDA estimate and performing inference over previously unseen scenes. Sentence planner learns the estimate model. M denotes the max epochs, HS the FeedFoward hidden sizes, and C the convolutional filters size.

incremented, as all the best results for the mean and area under the curve are from the deepest. This behaviour is more evident for the feedfoward DNN and CNN with "sequence" (vocabulary), while the CNN has a very regular behaviour for all the parameters and it is not easy to differentiate which is better. Thus, the number of convolutional layers in the CNN without vocabulary did not yield a very significant effect, which can be explained by the number of filters in the first layer already selecting the best features. In contrast, the CNN with vocabulary has the worst individual performance of all the networks, where the number of convolutional layers affects negatively when the number of epochs, and consequently early stopping, increases. Furthermore, in Figures 5.8a, 5.8b, and 5.8 we depict the mass of the frequencies, the area under the curve. From the previous Figures, we can conclude that, again, deeper networks yield better results, even if all have a similar behaviour, except for the CNN with vocabulary, where we find the worst results.

Finally, in Figure 5.9 we, depict the best results for the three networks learning the inference. In addition, we also depict the masses and zoom in the top similarity curve for the best approaches in Figures 5.10 and 5.11, respectively. As can be observed, every approach is close to the others, as the frequencies are very similar, which implies that all the methods are learning the mapping from the embeddings space to the topic's inference space. Again, as expected, deeper networks perform better when learning the mapping from the embedding space to the topic space. In addition, although the CNN with the vocabulary has the maximum number of

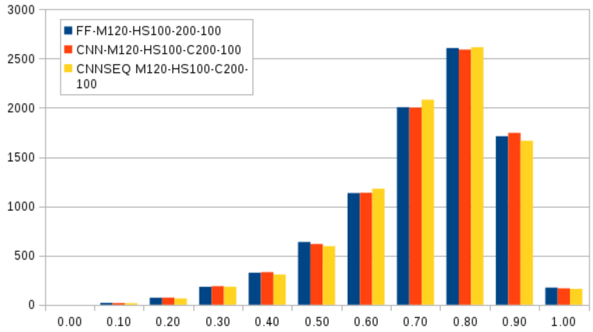


Figure 5.5: Sentence Planner vs LDA Inference, cosine similarity frequency. Best networks results for estimation learning, where FF is a feedfoward network, cnn is a CNN, and CNNSEQ is a CNN with vocabulary "sequence".

cosine frequencies of the expected value, it decays faster than the other approaches. The best approach is the CNN approach, even if not significantly better.

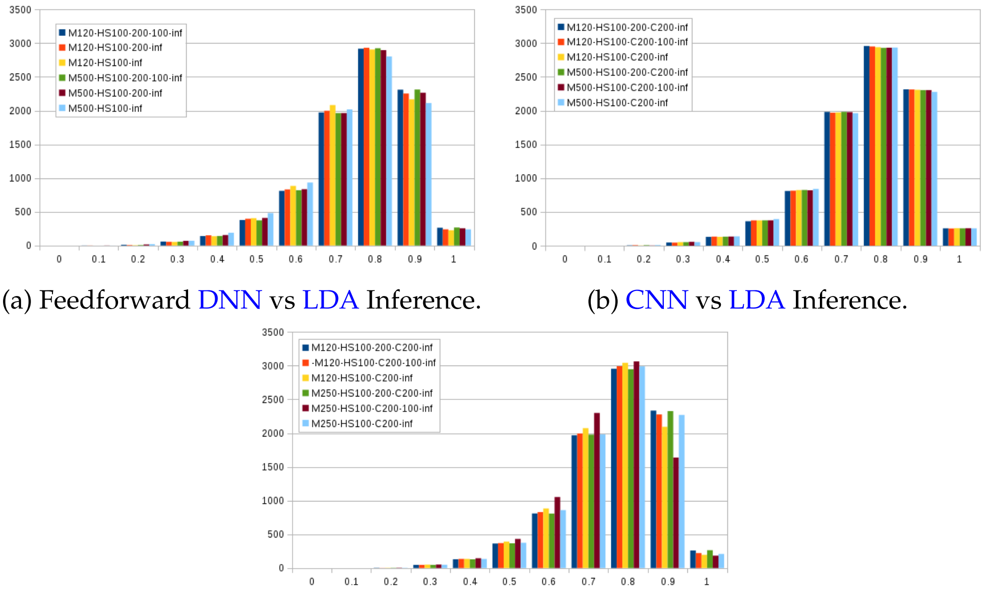
A comparison between the worst performing network learning the inference and the best performing learning the inference is depicted in Figure 5.12. As expected, using the inference as train leads to better performances, as the model learns directly the inference model and predicts as close as it can to the inference space, while the first scenario approach predicts closer to estimates. Furthermore, the second scenario shows that is possible to train with an approximation of the inference space, without yield worse results.

Moreover, learning to map from the embeddings space to the topic space does not yield worse results than performing directly the LDA inference. This way, the method we use, not only, is more flexible, as the word embeddings are a continuous dense space and the vocabulary in the LDA model is a discrete word space, but also, provides a way for mapping a generic embedding space into a lower dimension topic space.

5.3.1 Discussion

The main task of the sentence planner is to determine the response content and the structure of the response. However, the explicit structure of the response is addressed by the micro-content planner (chapter 6.1). Therefore, our approach to sentence planning has as its main task modelling the communicative objective and determine the content of the response in a domain specific space by mapping a (synthesized) question from a domain-independent question into a topic distribution (domain-dependent), i.e., we approach the planning as determining the domain content by mapping from a question in a embedding space into a response in the topic model space. Therefore, the sentence planner maps from a generic embedding space into a more fine-grained domain space by mapping word representations into topic distributions.

Therefore, the sentence planner maps from a generic embedding space into a more fine-



(a) Feedforward DNN vs LDA Inference.

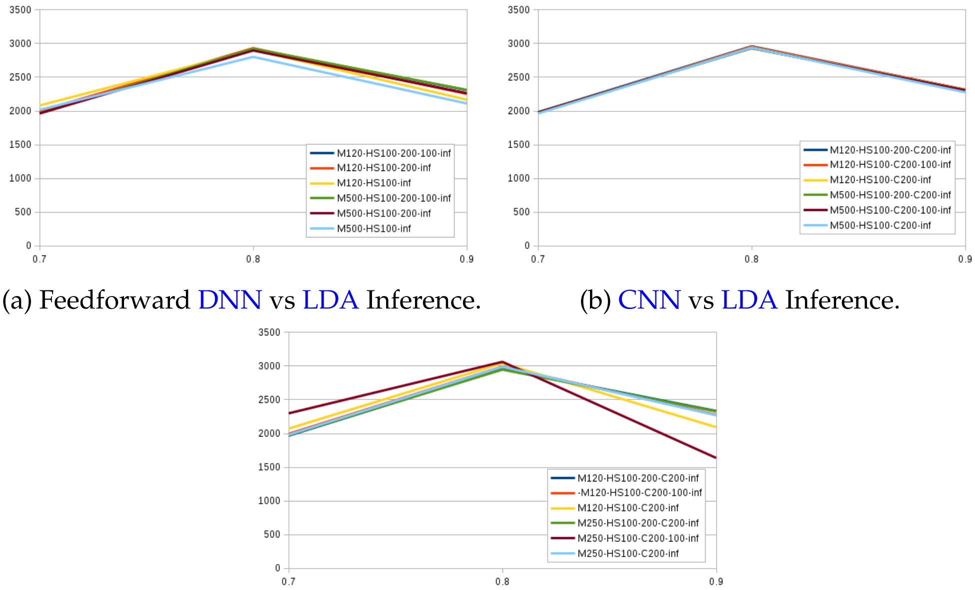
(b) CNN vs LDA Inference.

(c) CNN with vocabulary vs LDA Inference.

Figure 5.6: Cosine similarity frequency of sentence planner predictions with respect to LDA inference. Mapping embeddings to LDA inference and performing inference over previously unseen scenes. Sentence planner learns the inference model.

grained domain space by mapping word representations of a (synthesized) question into topic distributions. The main advantage of performing the inference via DNN instead of LDA is the mapping from the generic word representations into a more fined-grained representation, which the LDA can not provide as it is constrained on the collection's vocabulary. One can argue that the LDA model could be trained using a larger corpus with more vocabulary, however by doing so we would lose the fine-grained relations that the LDA provides in the specific domain. Our claim is that this method is domain independent and given any collection of any domain it is possible to train both the LDA and DNN models modularly. Furthermore, using transfer learning methods, ensemble methods, or learning the context directly from data could, potentially, allow to better generalize to new unseen domains.

There are limitations in the experiments conducted, first, and foremost, we use two scenarios that approximate the best possible scenario: divide the collection into three partitions, train the topic model with a representative set of the collection (first partition), infer the second partition of from previously unseen documents and train the network with the inferred distributions, and, finally, a held out set to evaluate the planner's performance. This would be the fairest approach. However, due to the size limitation of the domain dependent part of the dataset, we used an approximation, infer the model's estimate. Second, during the experiments we did not use wider networks and privileged a deeper architecture. Exploring all the space of hyper parameters is not feasible, therefore we limited our approach by evaluating deeper



(a) Feedforward DNN vs LDA Inference.

(b) CNN vs LDA Inference.

(c) CNN with vocabulary vs LDA.

Figure 5.7: Zoom in the range 0.7-0.9 of the cosine similarities curve, sentence planner vs LDA inference. Sentence planner learns the inference model.

networks.

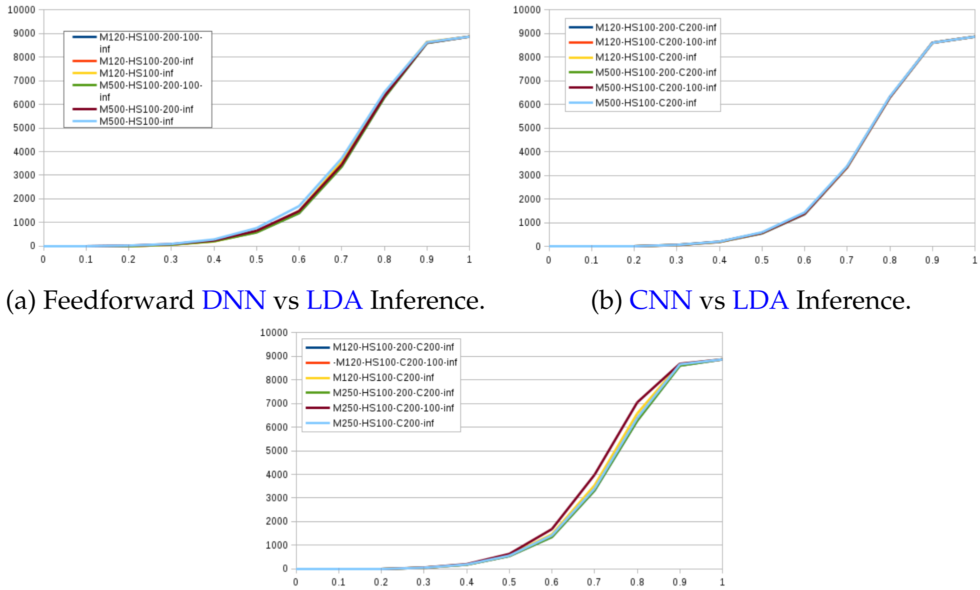
5.4 Summary

We described our approach to the generation problem to address the open domain task. Moreover, we focus in this chapter the sentence planning, while the next chapter focus the micro-content planning and realization.

Furthermore, we described our sentence planner main task: determining the content of the response in a statistical way, i.e., we use statistical methods for mapping from a generic, large vocabulary, space into a domain specific space, in our case from generic word embeddings into a physics domain. Moreover we addressed generation in a statistical way, to reduce the number of hand-crafted rules and parameters, and approach sentence planning using DNN and CNNs.

Ou planner depends of two different spaces, first a generic large vocabulary space and a domain specific space. For the large vocabulary we used the Google's ngram (Brants and Franz 2006), which covers a large domain independent space, while for the domain specific requisite we used a topic model, namely LDA.

Finally, we evaluated the performance of the mapping between the synthesized question embedding space into the domain topic distribution, comparing the cosine similarity between predicted, never seen, documents. To achieve this, we used two scenarios: first, the networks learn the model estimate space; second, the networks learn the inference space, an approxi-



(a) Feedforward DNN vs LDA Inference.

(b) CNN vs LDA Inference.

(c) CNN with vocabulary vs LDA Inference.

Figure 5.8: Mass of the cosine similarity frequency of figures 5.6a, 5.6b, and 5.6c. This is the mass frequency of the cosine similarity between the sentence planner and LDA Inference, thus can be regarded as area under the curve.

mation. We concluded that using the networks to perform this mapping did not yield worse results than using the LDA inference directly, which implies that learning a specific domain from a domain-independent vocabulary is possible and provides more flexibility than using the LDA inference directly.

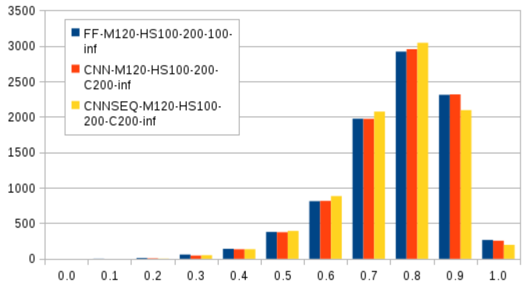


Figure 5.9: Sentence Planner vs LDA Inference, cosine similarity frequency. Planner learns the inference model. Best performing networks learning inference.

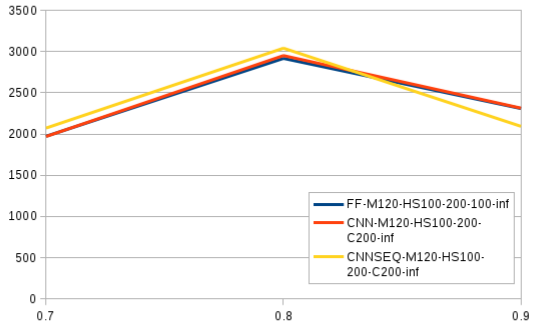


Figure 5.10: Zoom in the range 0.7-0.9 of the cosine similarities curve, sentence planner vs LDA inference, from Figure 5.9. Sentence planner learns the inference model.

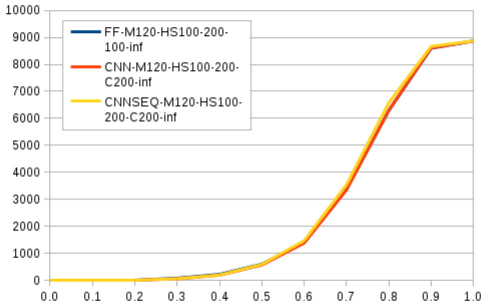


Figure 5.11: Mass of the cosine similarity frequency of Figure 5.9. This is the mass frequency of the cosine similarity between the sentence planner and LDA Inference, thus can be regarded as area under the curve.

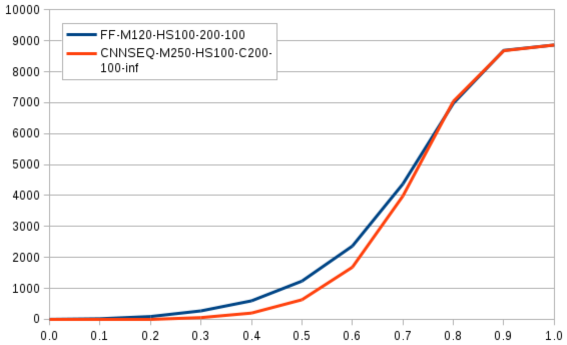


Figure 5.12: Comparison between the best estimation and worst inference learning.

6 Micro Planning and Realization

In the previous chapter we described our approach to the generation process in open domain, namely the general overview and the sentence planning. As described in the previous chapter, our sentence planner only defines explicitly the response content, thus, in this chapter we focus on micro-content planning, determining the structure of the response, and realization, deciding which words should be presented.

We consider here that the micro-content planner is responsible for determining the response structure, i.e., the micro planner determines which sentences should be generated in a non-linguistic representation. Thus, the micro planner is a continuation of the sentence planning and still represents the problem in a non-linguistic and domain specific way, in fact, the micro-planner refines the determined content to specify more fine-grained relations.

Furthermore, the surface realization is responsible for given the abstract representation from the previous steps, transform that representation into words. Thus, the realization is responsible for deciding which lexical items should be chosen to map from the abstract representation to the words.

As previously mentioned, we address the generation process in a statistical way, thus, both micro-content planner and realization are regarded as an optimization problem. Furthermore, we consider that these modules should be discussed together as we addressed the components both jointly and individually. The conventional architecture of Reiter and Dale (2000) is depicted in Figure 6.1, with the contributions for each of the modules.

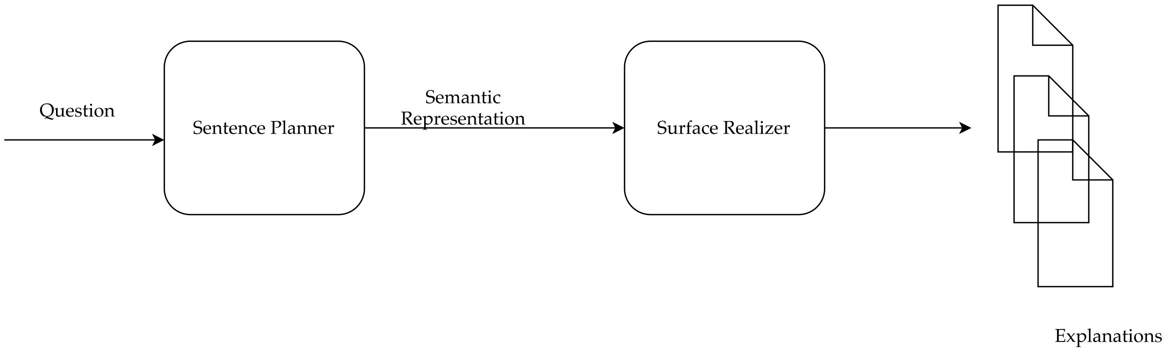


Figure 6.1: Full architecture.

6.1 Micro Planning and Realization

Our micro-planner is responsible for determining the structure of the response explicitly, while modelling fine-grained relationships by learning a mapping from the document topic distribution into its sentences topic distributions. Surface realization determines linguistic content from non-linguistic content, in our case, the component should learn a mapping between a topic distribution and the words which best realize that distribution, i.e., after shaping the generic space into the domain space, we want to realize the words from the generic space conditioned on the domain fine-grained relationships.

We use two methods for micro-planning and surface realization, which perform the mapping at document and word level: the first jointly optimizes the process; the second divides the problem into two sub-problems. This way, these approaches learn how to map the context into a sequence of sentences which in turn maps into a sequence of words. To address the inter-sentence relations of a document we learn the mapping between the topic distribution that the sentence planner should produce and model. This way, not only, the learning the structure of the response, by determining which topic distributions best represent the sentences that should be produced, but also, learning which words best suit the topic distributions. Therefore, we use two approaches: jointly optimize both document and sentence structure using a hierarchical approach; splitting the problem into two parts, first, optimizing the content planning, by learning a sequence of topic distributions, and, second, we optimize the words in each of the previous sequence.

6.1.1 Micro-Planning and Realization

In this approach, we divide the realization into two steps: first, decide the structure of the final answer, addressing the inter-sentence relations; and, second, given the structure, i.e., the content, perform the realization into a linguistic structure.

Figure 6.2 depicts the first step, while figure 6.3 depicts the second step. Dividing the problem into two steps has as its main advantage allowing to scrutinize the result of the content planning before performing the word realization. This way, we can understand if the inter-sentence relations are addressing the content selection as they should and only then perform the word realization.

6.1.2 Hierarchical Naive Approach

This approach models the micro-planning and realization problem as jointly learning the mapping between the topic distribution and the response structure, as well as mapping from the response structure into the words. Thus, the hierarchical naive approach models the micro-planner and realization by determining which sentences should be realized and which words

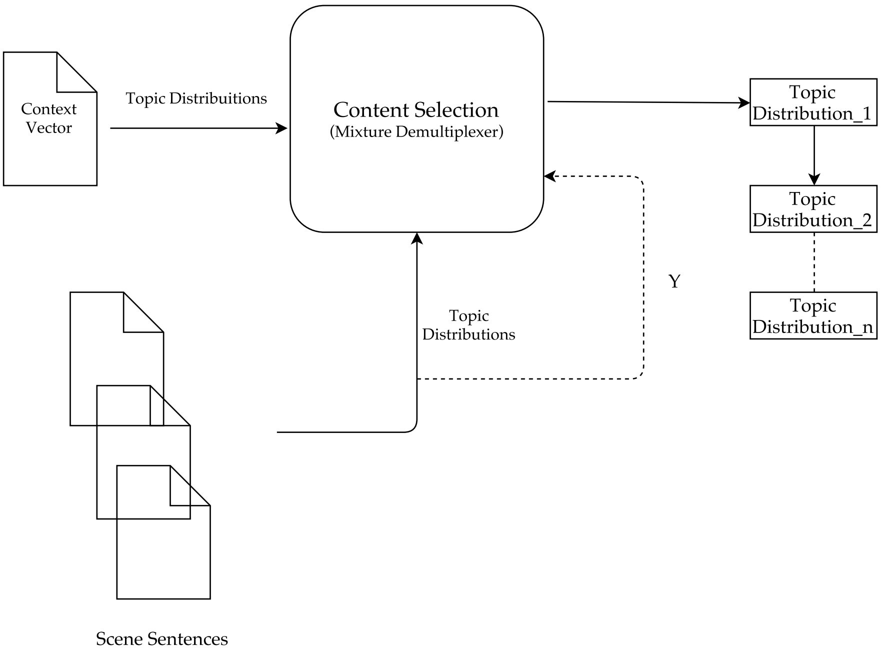


Figure 6.2: Content planning. Determining the structure of the explanation (Mixture Demultiplexer).

should be realized in each sentence. Moreover, this approach is the approach in Figures 6.2 and 6.3, where both steps are performed jointly and is conditioned with the sentence representation.

6.1.3 Naive Surface Realization

We also use a naive approach which does not perform micro-planning and tries to decodes on word level, not taking into consideration the document structure. Therefore, the model learns how to generate arbitrarily long sentences by learning how to map from the topic distribution into words. This approach main advantage is its simplicity, approaching the realization in a naive way by simply optimizing the sequence of words with respect to a context vector. However, this approach lacks the ability to model the document structure and the inter-sentence relations, which is relevant for generating better utterances. Furthermore, this approach tries to learn arbitrarily long sequences, which can lead to worse results, as there are practical limitations in the statistical learning approach. The surface realizer module for the naive approach is depicted in Figure 6.4.

6.2 Experimental Setup

Our micro-planner is responsible for determining the structure of the response explicitly, while modelling fine-grained relationships by learning a mapping from the document topic distribution into its sentences topic distributions. Surface realization determines linguistic content from

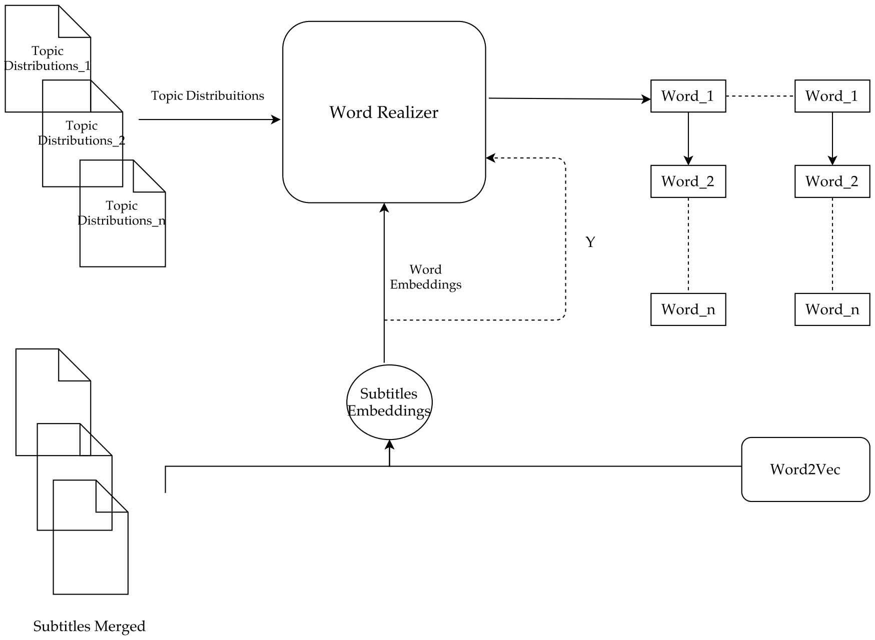


Figure 6.3: Word realization. Realize the non linguistic content with linguistic content.

non-linguistic content, as such we propose an approach which relies on deep learning, with emphasis on recurrent neural networks, to decode the representation of the sentence planner into words. Considering the encoder-decoder (Cho et al. 2014) architecture, this is the decoding part.

We approach the problem as a sequence-to-sequence problem, namely a one-to-many and a one-to-many-to-many problem – naive realization and micro content and realization, respectively. We address the problem with statistical optimization, namely deep learning, due to RNNs having the ability to model arbitrarily long sequences. Furthermore, RNNs also provide a method to increase the flexibility and, arguably, naturalness of the generated utterance as it avoids repetitiveness. The decoders considered in this work are similar to the one proposed by Bahdanau et al. (2014), where the embedding of target word $e(y_t)$ at step t is a peek of the output, and the additional weight matrices $C, C_{r,z}$ compute the context vector c_t at each step:

$$z_t = \sigma(W_z \cdot e(y_t) + U_z \cdot h_{t-1} + C_z \cdot c_t + b_z) \tag{6.1}$$

$$r_t = \sigma(W_r \cdot e(y_t) + U_r \cdot h_{t-1} + C_r \cdot c_t + b_r) \tag{6.2}$$

$$\tilde{h}_t = \tanh(W \cdot e(y_t) + U \cdot (r_t * h_{t-1}) + C \cdot c_t + b) \tag{6.3}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{6.4}$$

Furthermore, the hierarchical naive approach is similar to the one used by Li et al. (2015), with one decoder decoding on document level (sentences) and another decoder conditioned by the first one decoding on sentence level (words), using a “static” attention mechanism (Yan

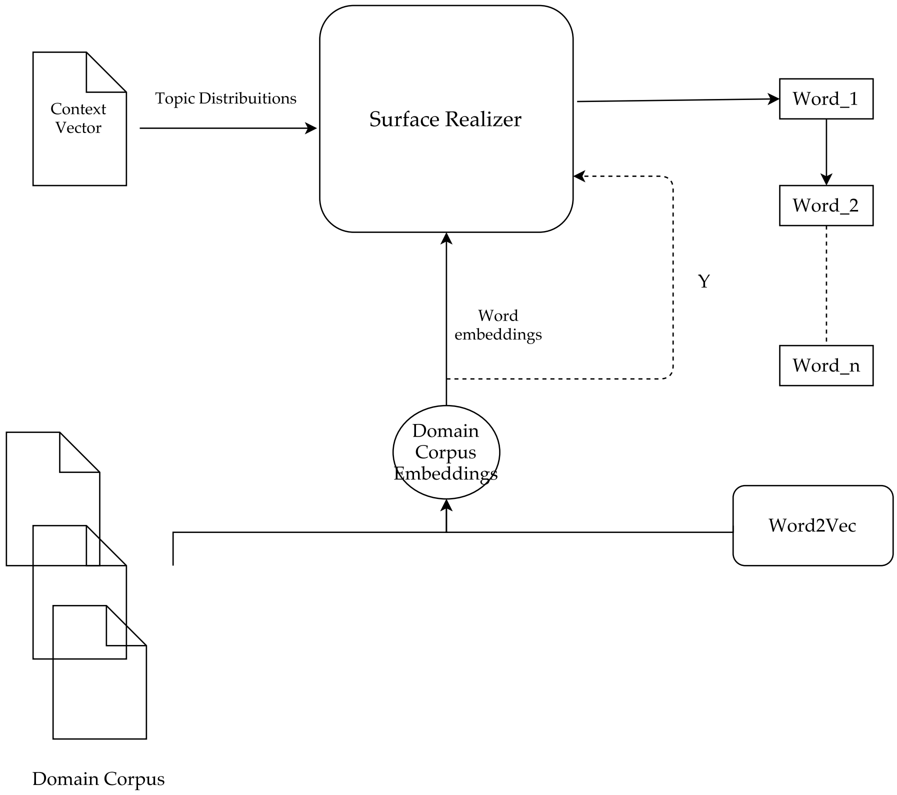


Figure 6.4: Surface Realizer.

2016). Therefore, this decoder operates on inter-sentence and intra-sentence level, preserving the document (response) structure and the sentence structure of each sentence. The micro planning and realization divide this approach into two different steps, the first models a one-to-many sequence, unrolling a topic distribution into its composing topic distributions, while the second is for each topic distribution realize a sequence of words. Figure 6.5 depicts the hierarchical decoder.

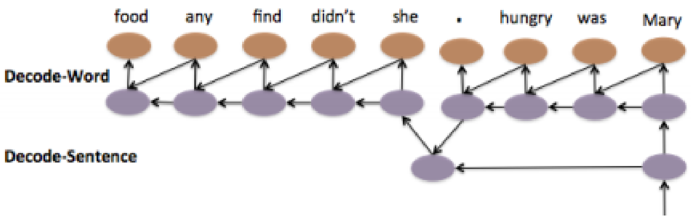


Figure 6.5: Hierarchical decoder. Extracted from Li, Luong, and Jurafsky (2015).

Both the naiver realization and micro planning and realization models were develop using Theano (Theano Development Team 2016) and the framework developed during the implementation depicted in the appendix A, where more details about the implementation are provided. Both use the same corpus and the same normalization (the same normalization used for the word embeddings). In addition, both models run on a GeForce GTX TITAN X (Nvidia

Corporation 2015).

Before addressing the decoders and their setup, there are some concepts that must be described, which we implemented and used: minibatching update which is the method used for training the network 6.2.1, zero padding for different size training examples 6.2.2, masking 6.2.3, bucketing 6.2.4, and early stopping 6.2.5.

6.2.1 Minibatch Update

ANNs are usually trained using backpropagation and there are three mainstream methods for training the network: on-line update, batch update, and minibatch update. On-line update consists of applying a forward pass for a single training example, calculate the error with the cost function (loss) and backpropagate by updating the parameters of the network (using an optimization algorithm, which we will assume to be Stochastic Gradient Descent (SGD) for the sake of argument). Thus, this method is very slow as the network has to converge by looking one example at each forward pass, i.e., by feeding one example in each iteration the network's error can be very distinct with respect to the last one, as the examples can be very distinct, and this will imply that the SGD will take longer to converge, note the step may be in the opposite direction of the previous one.

To address the variations in the network's error, the batch update technique applies all the training examples in each forward pass (which is in fact an epoch), calculating the error and updating all the parameters taking into consideration all the examples at once, trying to converge faster. However, batch update suffers from trying to learn everything from everything, i.e., as the network is fed all the training examples, it will try to learn the mapping at once, which may cause the network to converge to a local minima.

Minibatch update is a combination of the previous two methods and consists of instead of assuming one or all, the network is fed minibatches, which are fixed size batches of examples. Therefore, the network will look at a percentage of the examples at each forward pass and backpropagate the error considering those examples. A note, if the minibatch size is 1 then the behaviour is the same as on-line update and if the minibatch size is the size of the training set then it behaves as batch update.

6.2.2 Zero Padding

The problem we are considering is a sequence-to-sequence and, as such, we have to perform a zero padding so that all the sequences have the same size when fed to the network. This guarantees that the recurrent network works as expected.

6.2.3 Masking

Using padding implies that there are a lot of steps which are irrelevant for the sequence, the ones represented by zeros. This implies that the network will look at these steps and just perform a reset instead of continuing to learn, with the additional problem of computing the cost. To address this, the masking technique consists of building a matrix that determines which steps should be taken into account when computing the recurrent steps and the cost.

6.2.4 Bucketing

Although zero padding is crucial for sequence to sequence tasks, there is a clear disadvantage: padding without any context will lead to shorter size samples being represented almost by zeros, as the longer size samples will force the size of the padding. Thus, in order to address this limitation, the bucketing technique divides the samples by their size to avoid too much padding and thus the padding is performed with respect to the biggest size of the bucket instead of the whole collection.

6.2.5 Early Stopping

In addition to all the machine learning techniques already discussed, we also implement and use early stopping, this is, we use the validation set after each epoch and if the network is not improving after n iterations then we stop training as the network already converged, to a local minima at least.

6.2.6 Training Details

The naive surface realizer is trained using a peek of the target word, therefore, not only, the decoder is trained conditioned by the context vector (the topic distribution), but also, it is conditioned by the word which should be predicted. This is accomplished by training the network with a peek of the output.

The hierarchical naive approach is trained using a peek of the target sentence and target word, both in different levels. The first level decodes on document (sentence) level and decides what information should the sentences have, while the second level decodes on sentence (words) level and decides which words should constitute each sentence. Finally, the content selection and realization are trained separately, the first one is trained as a demultiplexer of topic distributions in a one-to-many problem, while the second, is trained using sequences of topic distributions into sequences of words, in a sequence-to-sequence approach.

Furthermore the experiments conducted for the naive realization approach consist of a one layer decoder with 1024 units and the word representations the word embeddings previously discussed. While the hyper parameters defined for the hierarchical decoders and micro-planning and realization were 512 units for all hidden layers. All the decoders are trained using an adaptive optimizer, namely ADAM. Briefly, ADAM is an extension of the SGD and RMSProp and uses a running average of both algorithms along with the second moments of the gradient.

During the realization of these experiments we faced a constraint regarding the available GPU memory, as when performing softmax over the vocabulary it exceed the available memory (a distribution for each time step for each training example and the vocabulary is in the million of words). In addition, loading all the dataset into the GPU memory also proved to be insuperable, both problems are even more evident in the hierarchical decoder. Therefore to address these limitations, in all experiments we have a sparse representation in the Central Processing Unit (CPU) memory and only transfer to the GPU the minibatch for each minibatch of the dataset. Moreover, to overcome the memory limitation, we don't perform a softmax at the end of the network, predicting only the word embeddings or topic distributions. Finally, we perform greedy best first search until an end-of-sentence or end-of-document is found and we define the probability of a word as the multinomial distribution of the cosine similarity to the words in the word embeddings matrix.

More details of all the models are further described in appendix A.

6.3 Experimental Results and Discussion

Evaluating a natural language generator is a hard task, as different utterances can encode the same meaning. In fact, the community is often divided when the subject is evaluating a generation system, as the existent objective measures only take into consideration the frequency of ngrams in the generated utterance with respect to the expected utterance, the same applies for fields such as machine translation. Thus, subjective measures where independent judges classify the system's response are one of the methods to evaluate the system's performance, for instance considering the fluency and naturalness of the generated utterance.

Due to the hard nature of evaluation and, mostly, because the results presented here are still preliminary results, we can not perform an objective analysis, similar to the one for the sentence planning (section 5), nor a subjective analysis of the quality of the generation. However, we depict a sample of our results in Table 6.1.

From the previous Table, we can observe that both the naive surface realization and hierarchical naive approach produce words without a coherent structure and their semantic interpretation is very hard to interpret, as some of the words are part of the domain-specific corpus, while others are very far from what we were expecting.

Naive Realization	always united dc important potential music phone be likely child get bed user before category everyone month according employment wrote life resources professional deal what very must table author air
Hierarchical Realization	record testing supported planning accept budget not club rd ca provides rates essential upper amateur california think open names multi lots correct follow cool updated minister steps latest calls easily be i union sure than prior money programming research prevent key mini received been hot teen probably rock aug benefit page pro space just date teen enter applied this compare reported outside photo even thought name support central supplies pussy www technologies church limited format remember win come located win end behind high being relevant fucking quote percent about movies amateur president anything output they store not educational around do so last feedback party feet editor started camera reference those force property february simply why cd none technologies option
Expected	solar flares are planet-sized eruptions of boiling gas, prominences that break free of the sun. they been seen from Earth, but not in such detail and quantity.

Table 6.1: Surface Realization Preliminar Results.

6.3.1 Discussion

The results depicted here are preliminar and require further study to conclude whether the approach is feasible. The results of the Micro-planning and realization as two steps are not depicted here, as the micro-planner could not map the topic distribution to a sequence of topic distributions. Furthermore, the behaviour of the networks was predicting the same distributions for different initial distributions. Thus, the realization using the micro-planner results was not approached.

All the experiments performed suffer from the same defect, which could be improved: the number of layers in the architecture is not sufficient to model from the domain-specific content into the word generic representation. This requires further study by increasing the number of layers.

While conducting the experiments, we faced an insuperable obstacle: the GPU memory limitations. We would perform softmax, or an approximation, at the end of the network and use as loss function the categorical cross entropy. Therefore, we used an approximation by using as loss function the mean squared error for the flat model and hierarchical model, predicting directly the embeddings, and the cosine distance (and mean squared error) for the content planning.

Finally, dividing the realization into two steps, where the first predicts content words and the second adds functional words, is a study to be conducted.

6.4 Summary

We described our approach to micro-planning and surface realization using deep learning, namely RNNs. This way, we exploit the ability to model arbitrarily large sequences, which are reflected on determining the structure of the answer (document level, inter-sentence relations, and sentence level, intra-sentence relations). To achieve this, we use a naive approach, which operates on intra-sentence level, and a micro content and surface realization approach, which operates on two levels, document level, and sentence level. Moreover we formulate the naive approach as a decoder with a peek of the target and the micro content and realization as a joint optimization (hierarchical decoding) and a decomposing into two separate tasks, micro planning to plan the structure of the realization and a realizer to determine the words, where the first part is responsible for determining which sentences best reflect the answer and how many (micro planning), while the second decoder is responsible for determining which words fit best in each sentence (realization).

Evaluating a generation system is hard due to the subjective nature of the utterance generated. Furthermore, we depict our preliminary results, which did yield good results. In fact, our approaches did not exploit the full deep architecture of neural networks and the results were not the ones we expected. Thus, the approach requires further study to understand if this approach is feasible.

7 Conclusions and Future Work

Approaching the generation problem as open domain is a hard task, in fact, its not solved yet. There are different approaches for the generation problem, namely template-based, conventional pipeline, and statistical approach, however the open domain problem has only been addressed recently by Li and Jurafsky (2016). Thus, approaching the problem in the context of open domain implies a number of hard tasks. The statistical approaches to the generation problem are the ones, arguably, closer to an open domain interaction, as they allow to reduce hand-crafted rules for the domain dependent parts. With this in mind, the statistical approaches, usually, optimize one, or both, components of the conventional approach. Thus, we approached the generation problem addressing both components as different tasks.

Approaching the problem as end-to-end problem is possible, as Li and Jurafsky (2016) showed. However, we build a generation system in the context of a narrator and use subtitles as our domain dependent corpus. This corpus is limited in size, therefore we want to minimize the error an end-to-end approach would yield and divide into different components.

We explored different corpus constructions to understand the nature of the subtitles corpus and evaluate if this corpus is suitable for the domain-dependent part that even an open domain system must comprise. We perform this evaluation by building different LDA topic models, which are build in an unsupervised way. In this step we faced constraints regarding the quality of the subtitles, as we had to perform specific normalization for the subtitles and build our own corpus from the initial subtitles. The corpus construction is a very important part of the generation process, especially using data-driven methods, as it conditions all the process.

Furthermore, in order to consider open domain generation, we identified the following challenges, (1) how to tackle open domain in the NLG module, (2) how do neural networks (deep feedforward and recurrent) can be combined with topic modeling to address open domain.

The first challenge is described in chapter 3 where we provide the state of the art and discuss which alternatives are suitable for an open domain generation. However, open domain generation is still an unsolved problem and we study methods to condition the generation with domain-specific content. Thus, in chapters 5 and 6 we describe our approach to the generation process, addressing the second challenge. Moreover, we address the generation problem by regarding each component as a statistical problem, namely the sentence planner learns how to refine a question generic space into a domain-specific space, and the micro-planner and surface realizer learn how to map back to the generic space conditioned on the domain-specific content.

Our sentence planner models the communication objective by refining a generic (synthesized) question space into a topic space, which is representative of a domain. Thus, the planner determines the content for the response by determining which topics should be realized. The mapping between the two spaces is accomplished using DNN, which can learn a mapping between different spaces. This could lead to, potently, a flexible and modular planning that can be learn via DNN. We showed that the planner can learn how to lower the dimensionality of the word space, by learning the mapping from a domain independent space (word embeddings) with a large vocabulary into a lower denser space (topic inference) with a small vocabulary. Moreover, comparing unsupervised models is a hard task, even more when comparing a supervised model that learns the unsupervised model. The evaluation of a topic model is not easy, as the interpretation, usually, is taken directly from the quality of the estimated bag of words. We compare how the DNN compare with the LDA inference using the cosine similarity and show that the results suggest that mapping from the generic embedding space into the inference space is possible and does not yield worse results.

We address the surface realization as performing the inverse operation from the sentence planner, i.e., the realization maps from the domain space and conditions the generation in the domain independent space. To achieve this, we use sequence to sequence models by exploiting the power of RNN to model arbitrarily long sequences. Formulating the realization as a RNN allows to exploit the temporal relations of language However, our results are far from the ones we expected. We approached the surface realization using a naive realization approach and two micro-planning and realization approaches, a naive jointly optimization and a divided approach. All the approaches require more study as the results are all preliminar.

Addressing the surface realization module with deep learning provides a flexible and powerful method to model sequence learning. However, learning a very large vocabulary is a very hard task, as there are memory and temporal limitations. The memory limits lead us to a sub-optimal approach, where we are not optimizing the softmax probability, rather we predict the embeddings directly. This limitation leads to more noise being added to the model.

Finally, our main contribution is both the corpus construction, as well as addressing the sentence planning as a mapping from a generic space into a domain space, through a domain-independent method. Moreover, our approach to the realization is also a contribution from the abstract perspective, even if in practice the results are far from the ones expected.

7.1 Future Work

Our future work will address using Gaussian LDA (Das, Zaheer, and Dyer) for the sentence planning, as the mapping from word embeddings to topic distributions is natural for the method. Another approach is formulating the sentence planning as a RNN with real questions instead of synthesized ones. Furthermore we will approach the problem as an end-to-end task and perform a comparison between the two systems. We will also study how to transfer the domain learnt to another, as currently method is domain independent from a plugable perspective.

Finally, we will also resort to machine learning techniques such as the soft attention mechanism and scheduled sampling to improve the surface realization. Moreover, we will also study dividing the realization problem into one more step, where the first step predicts content words and the second adds functions words to the content ones.

Bibliography

Abadi, M., 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 (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Angeli, G., P. Liang, and D. Klein (2010). A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 502–512.

Aparício, M., P. Figueiredo, F. Raposo, D. Martins de Matos, R. Ribeiro, and L. Marujo (2016, April). Summarization of films and documentaries based on subtitles and scripts. *Pattern Recogn. Lett. 73*(C), 7–12.

Appelt, D. E. (1992). *Planning English sentences*. Studies in natural language processing. Cambridge University Press.

Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learning to align and translate. *CoRR abs/1409.0473*.

Bauer, D. and A. Koller (2010). Sentence generation as planning with probabilistic LTAG. In *Proceedings of the 10th TAG+ Workshop*, New Haven.

Ben Mustapha, N., M.-A. Aufaure, H. Baazaoui Zghal, and H. Ben Ghezala (2015, August). Query-driven approach of contextual ontology module learning using web snippets. *J. Intell. Inf. Syst. 45*(1), 61–94.

Bengio, Y., P. Simard, and P. Frasconi (1994, March). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw. 5*(2), 157–166.

Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. *J. Mach. Learn. Res. 3*, 993–1022.

Bohus, D. and A. I. Rudnicky (2009, July). The RavenClaw dialog management framework: Architecture and systems. *Comput. Speech Lang. 23*(3), 332–361.

Brants, T. and A. Franz (2006). Web 1t 5-gram version 1.

Bratman, M. E. (1987). *Intention, plans, and practical reason*. Harvard University Press.

Bratman, M. E., D. J. Israel, and M. E. Pollack (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence 4*, 349–355.

Bui, T. H. (2006). Multimodal dialogue management-state of the art.

Cho, K., B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR abs/1406.1078*.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Chung, J., Ç. Gülçehre, K. Cho, and Y. Bengio (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR abs/1412.3555*.

Cohen, P. R. and C. R. Perrault (1979). Elements of a plan-based theory of speech acts. *Cognitive Science 3*(3), 177–212.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS) 2*(4), 303–314.

Dale, R. and N. J. Haddock (1991). Generating referring expressions involving relations. In *EACL 1991, 5th Conference of the European Chapter of the Association for Computational Linguistics, April 9-11, 1991, Congress Hall, Alexanderplatz, Berlin, Germany*, pp. 161–166.

Das, R., M. Zaheer, and C. Dyer. Gaussian lda for topic models with word embeddings. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics.

Dethlefs, N. and H. Cuayáhuitl (2010). Hierarchical reinforcement learning for adaptive text generation. In *INLG 2010 - Proceedings of the Sixth International Natural Language Generation Conference, July 7-9, 2010, Trim, Co. Meath, Ireland*.

Dethlefs, N. and H. Cuayáhuitl (2011). Combining hierarchical reinforcement learning and bayesian networks for natural language generation in situated dialogue. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pp. 110–120. Association for Computational Linguistics.

Dethlefs, N., H. Cuayáhuitl, and J. Viethen (2011). Optimising natural language generation decision making for situated dialogue. In *Proceedings of the SIGDIAL 2011 Conference, The 12th Annual Meeting of the Special Interest Group on Discourse and Dialogue, June 17-18, 2011, Oregon Science & Health University, Portland, Oregon, USA*, pp. 78–87.

Engel, Y., S. Mannor, and R. Meir (2005). Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pp. 201–208. ACM.

Ferguson, G. M. and J. F. Allen (1999). TRIPS: the rochester interactive planning system. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pp. 906–907.

Ferguson, G. M., J. F. Allen, and B. W. Miller (1996). TRAINS-95: towards a mixed-initiative planning assistant. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, Edinburgh, Scotland, May 29-31, 1996*, pp. 70–77.

Gašić, M., C. Breslin, M. Henderson, D. Kim, M. Szummer, B. Thomson, P. Tsiakoulis, and S. J. Young (2013). Pomdp-based dialogue manager adaptation to extended domains. In *Proceedings of the SIGDIAL 2013 Conference, 14th Annual Conference, Metz, France, August 2013*, pp. 214–222. Association for Computational Linguistics.

Gašić, M., D. Kim, P. Tsiakoulis, C. Breslin, M. Henderson, M. Szummer, B. Thomson, and S. J. Young (2014). Incremental on-line adaptation of POMDP-based dialogue managers to extended domains. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pp. 140–144.

Gašić, M., D. Kim, P. Tsiakoulis, and S. J. Young (2015). Distributed dialogue policies for multi-domain statistical dialogue management. In *ICASSP 2015, 40th IEEE Internation Conference on Accoustic, Speech and Signal Processing, Brisbane, Austraila, April 19-24, 2015*.

Ginter, F. and J. Kanerva (2014). Fast training of word2vec representations using n-gram corpora.

Greff, K., R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber (2015). LSTM: A search space odyssey. *CoRR abs/1503.04069*.

Grosz, B. J. and C. L. Sidner (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics 12*(3), 175–204.

Grosz, B. J. and C. L. Sidner (1990). Plans for discourse. In M. Cohen and M. E. Pollack (Eds.), *Intentions in Communications*, Chapter 20, pp. 417–444. MIT Press.

Hochreiter, S. and J. Schmidhuber (1997, November). Long short-term memory. *Neural Comput. 9*(8), 1735–1780.

Hulstijn, J., R. Steetskamp, H. T. Doest, S. van de Burgt, and A. Nijholt (1996). Topics in schisma dialogues.

Kaelbling, L. P., M. L. Littman, and A. W. Moore (1996). Reinforcement learning: A survey. *J. Artif. Intell. Res. (JAIR) 4*, 237–285.

Karpathy, A. (2015). Cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *CoRR abs/1412.6980*.

Koller, A. and R. P. A. Petrick (2011). Experiences with planning for natural language generation. *Computational Intelligence 27*(1), 23–40.

Koller, A. and M. Stone (2007). Sentence generation as a planning problem. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*.

Langkilde, I. and K. Knight (1998). Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98, Stroudsburg, PA, USA, pp. 704–710. Association for Computational Linguistics.

Larsson, S. (2005). Dialogue systems: Simulations or interfaces. In *in Dialor'05: Proceedings of the ninth workshop on the semantics and pragmatics of dialogue, C. Gardent and*, pp. 4552.

Larsson, S. and D. R. Traum (2000, September). Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering 6*(3-4), 323–340.

Lee, C., S. Jung, K. Kim, D. Lee, and G. G. Lee (2010). Recent approaches to dialog management for spoken dialog systems. *JCSE 4*(1), 1–22.

Lemon, O. (2008). Adaptive natural language generation in dialogue using reinforcement learning. *Proc. SEM-dial*, 141–148.

Lemon, O. (2011). Learning what to say and how to say it: Joint optimisation of spoken dialogue management and natural language generation. *Computer Speech & Language 25*(2), 210–221.

Levin, E. and R. Pieraccini (1997). A stochastic model of computer-human interaction for learning dialogue strategies. In *Fifth European Conference on Speech Communication and Technology, EUROSPEECH 1997, Rhodes, Greece, September 22-25, 1997*.

Levin, E., R. Pieraccini, and W. Eckert (1998). Using markov decision process for learning dialogue strategies. In *Proceedings of the 1998 IEEE International Conference on*

Acoustics, Speech and Signal Processing, ICASSP '98, Seattle, Washington, USA, May 12-15, 1998, pp. 201–204.

Li, J. and D. Jurafsky (2016). Neural net models for open-domain discourse coherence. *CoRR abs/1606.01545*.

Li, J., M. Luong, and D. Jurafsky (2015). A hierarchical neural autoencoder for paragraphs and documents. *CoRR abs/1506.01057*.

Litman, D. J., M. S. Kearns, S. P. Singh, and M. A. Walker (2000). Automatic optimization of dialogue management. In *COLING 2000, 18th International Conference on Computational Linguistics, Proceedings of the Conference, 2 Volumes, July 31 - August 4, 2000, Universität des Saarlandes, Saarbrücken, Germany*, pp. 502–508.

Litman, D. J. and S. Silliman (2004). ITSPOKE: An intelligent tutoring spoken dialogue system. In *Demonstration Papers at HLT-NAACL 2004*, HLT-NAACL–Demonstrations 2004, Stroudsburg, PA, USA, pp. 5–8. Association for Computational Linguistics.

M. Ferguson, G., J. F. Allen, B. W. Miller, and E. K. Ringger (1996). The design and implementation of the trains-96 system: A prototype mixed-initiative planning assistant. Technical report, University of Rochester, Rochester, NY, USA.

Mann, W. C. (1984). Discourse structures for text generation. In *10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics, Proceedings of COLING '84, July 2-6, 1984, Stanford University, California, USA.*, pp. 367–375.

Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text 8*(3), 243–281.

McGlashan, S., N. Fraser, N. Gilbert, E. Bilange, P. Heisterkamp, and N. Youd (1992). Dialogue management for telephone information systems. In *Proceedings of the Third Conference on Applied Natural Language Processing*, ANLC '92, Stroudsburg, PA, USA, pp. 245–246. Association for Computational Linguistics.

McTear, M. F. (2002). Spoken dialogue technology: enabling the conversational user interface. *ACM Comput. Surv. 34*(1), 90–169.

Mikolov, T., M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur (2010). Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pp. 1045–1048.

Mikolov, T., S. Kombrink, L. Burget, J. Cernocký, and S. Khudanpur (2011). Extensions of recurrent neural network language model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*, pp. 5528–5531.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119.

MITRE Corporation (2005). Midiki: Mitre dialogue kit user's manual. Technical report.

Nvidia Corporation (2015). Nvidia GeForce Titan X. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x.

Oh, A. and A. I. Rudnicky (2002). Stochastic natural language generation for spoken dialog systems. *Computer Speech & Language 16*(3-4), 387–407.

Olah, C. (2015). Understanding LSTM networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Philip R. Cohen, J. M. and M. E. Pollack (1993). Intentions in communication. *Artif. Intell. 63*(1-2), 511–520.

R2RT (2016). Written memories: Understanding, deriving and extending the LSTM. http://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html.

Reiter, E. and R. Dale (1997, 3). Building applied natural language generation systems. *Natural Language Engineering 3*, 57–87.

Reiter, E. and R. Dale (2000). *Building Natural Language Generation Systems*. New York, NY, USA: Cambridge University Press.

Rich, C. and C. L. Sidner (1997). COLLAGEN: when agents collaborate with people. In *Agents*, pp. 284–291.

Rieser, V. and O. Lemon (2010). Natural language generation as planning under uncertainty for spoken dialogue systems. In *Empirical Methods in Natural Language Generation: Data-oriented Methods and Empirical Evaluation*, pp. 105–120.

Sadek, M. D., P. Bretier, and F. Panaget (1997). Artimis: Natural dialogue meets rational agency. In *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence - Volume 2*, IJCAI'97, San Francisco, CA, USA, pp. 1030–1035. Morgan Kaufmann Publishers Inc.

Serban, I. V., A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau (2015). Hierarchical neural network generative models for movie dialogues. *CoRR abs/1507.04808*.

Singh, S. P., M. J. Kearns, D. J. Litman, and M. A. Walker (1999). Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pp. 956–962.

Song, D., B. Howald, and F. Schilder (2015, September 17). Template bootstrapping for domain-adaptable natural language generation. US Patent App. 14/726,119.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 1929–1958.

Stent, A., R. Prasad, and M. A. Walker (2004). Trainable sentence planning for complex information presentations in spoken dialog systems. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain.*, pp. 79–86.

Steyvers, M. and T. Griffiths (2007). Probabilistic topic models. *Handbook of latent semantic analysis 427*(7), 424–440.

Stone, M. (2005). Communicative intentions and conversational processes in human-human and humancomputer dialogue. *Approaches to studying world-situated language use*, 39–70.

Stone, M., C. Doran, B. L. Webber, T. Bleam, and M. Palmer (2003). Microplanning with communicative intentions: The SPUD system. *Computational Intelligence 19*(4), 311–381.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. *CoRR abs/1409.3215*.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction* (1st ed.). Cambridge, MA, USA: MIT Press.

Teh, Y. W., M. I. Jordan, M. J. Beal, and D. M. Blei (2004). Hierarchical dirichlet processes. *Journal of the American Statistical Association 101*.

Theano Development Team (2016, May). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints abs/1605.02688*.

Theune, M., E. Klabbers, J. R. De Pijper, E. Krahmer, and J. Odijk (2001, March). From data to speech: A general approach. *Nat. Lang. Eng. 7*(1), 47–86.

Tieleman, T. and G. Hinton (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning 4*(2).

Traum, D. R. and S. Larsson (2003). The information state approach to dialogue management. In J. van Kuppevelt and R. Smith (Eds.), *Current and New Directions in Discourse and Dialogue*, Volume 22 of *Text, Speech and Language Technology*, pp. 325–353. Springer Netherlands.

Van Deemter, K., E. Krahmer, and M. Theune (2005, March). Real versus template-based natural language generation: A false opposition? *Computational Linguistics 31*(1), 15–24.

van Merriënboer, B., D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio (2015). Blocks and fuel: Frameworks for deep learning. *CoRR abs/1506.00619*.

van Zanten, G. V. (1996). Pragmatic interpretation and dialogue management in spoken-language systems.

Walker, M. A., J. Frommer, and S. Narayanan (1998). Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, COLING-ACL '98, August 10-14, 1998, Université de Montréal, Montréal, Quebec, Canada. Proceedings of the Conference.*, pp. 1345–1351.

Walker, M. A., D. J. Litman, C. A. Kamm, and A. Abella (1997). PARADISE: A framework for evaluating spoken dialogue agents. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, 7-12 July 1997, Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain.*, pp. 271–280.

Walker, M. A., O. Rambow, and M. Rogati (2001). Spot: A trainable sentence planner. In *NAACL*.

Walker, M. A., A. Stent, F. Mairesse, and R. Prasad (2007). Individual and domain adaptation in sentence planning for dialogue. *J. Artif. Intell. Res. (JAIR) 30*, 413–456.

Wang, Z., T. Wen, P. Su, and Y. Stylianou (2015). Learning domain-independent dialogue policies via ontology parameterisation. In *Proceedings of the SIGDIAL 2015 Conference, 16th Annual Conference, Prague, Czech Republic, September 2015*, pp. 412–416. Association for Computational Linguistics.

Wen, T., M. Gašić, D. Kim, N. Mrksic, P. Su, D. Vandyke, and S. J. Young (2015). Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. *CoRR abs/1508.01755*.

Wen, T., M. Gašić, N. Mrksic, P. Su, D. Vandyke, and S. J. Young (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pp. 1711–1721.

Williams, J. D. and S. J. Young (2007). Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language 21*(2), 393–422.

Xu, W., B. Xu, T. Huang, and H. Xia (2002). Bridging the gap between dialogue management and dialogue models. In *Proceedings of the 3rd SIGdial workshop on Discourse and dialogue-Volume 2*, pp. 201–210. Association for Computational Linguistics.

Yan, R. (2016). I, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pp. 2238–2244. AAAI Press.

Young, S. J. (1999). Probabilistic methods in spoken dialogue systems. *Philosophical Transactions of the Royal Society (Series A 358*, 1389–1402.

Young, S. J., M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu (2010). The hidden information state model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language 24*(2), 150–174.

Young, S. J., M. Gašić, B. Thomson, and J. D. Williams (2013). POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE 101*(5), 1160–1179.

Appendix

A Deep Learning

In this appendix we describe the details regarding the Theano implementation of both flat and hierarchical decoders, as well as the L2F Deep Learning Framework.

A.1 Flat and Micro Planning Decoder

Both decoders were implemented using Theano and the developed framework during the realization of this work. This framework was developed with Miguel Varela Ramos and will be further described in section A.2. We developed different layers and models to address both sentence planning and surface realization with deep learning. Therefore, we developed a flat layer where the input is a batch of topic distributions and the target is the word embeddings; an hierarchical layer where the first part maps from a topic latent space into the embeddings sentence embeddings and second part maps from sentence representation into the word embeddings (an unroll of the space of the sentence); and a layer which decodes from one topic distribution to a sequence of topic distributions, as well as a layer witch decodes from a sequence of topic sequences into a sequence of word embeddings.

For each one of the problems, we also implement a different model, as the computational graph differ very much one from another. During the realization of this work, we faced an insuperable obstacle: we want to decode from the domain dependent vocabulary into the generic vocabulary and as such we tried to perform a softmax over the generic vocabulary (millions of words). This proved to be insuperable due to the memory limitations of the GPU, as well as the tractability of performing a softmax over a large vocabulary. Thus we use an approximation during train and test, we predict the embeddings directly at train time and at test we define the probability of the word as the $\arg\max$ of a multinomial distribution of the cosine distance to the embedding matrix. Thus, we implement a greedy best first sampler for all the models.

Therefore we implemented the following layers:

- FlatDecoder
- HierachicalDecoder
- Mix2MixesDecoder
- Mixes2WordsDecoder

and the following models:

- FlatModel
- HierarchicalModel
- Mix2MixesModel
- Mixes2WordsModel

Furthermore, we also faced another constraint during this work related to the available memory of the GPU: how many samples can be in the GPU's memory. Thus, to address this limitation we only transfer from the CPU memory to the GPU the minibatch that is being considered. We achieve this through a data iterator, which has a sparse representation in the CPU and for each of the minibatches produces the full representation to transfer to the GPU. In addition, we also implement bucketing, we sort the sentences by their size and the group them in batches to minimize the zero padding.

A.2 L2F Deep Learning Framework

There are numerous deep learning frameworks, such as Chollet (2015), van Merriënboer et al. (2015), among others, which provide most of the layers, optimizers, and cost functions required to train a DNN, whether recursive or feedforward. However, the flexibility we encountered in some of these frameworks did not meet our requirements, as the development of new layers, for instance, is constrained by how these frameworks are built. Thus, we developed a small framework for personal usage, which is still being developed.

The framework was developed over the well known mathematical framework Theano (Theano Development Team 2016), which provided all the background for the sound mathematical operations required by the models and layers. There are alternatives to Theano that operate on a similar level, namely Tensorflow (Abadi et al. 2015), but we choose to develop over Theano due to the flexibility it provided without any specific architecture for the layers. Moreover, developing using directly Theano provides a greater flexibility than the other frameworks, even if those frameworks are built using this mathematical framework. Nevertheless, we borrowed some of the features those frameworks already provide, for instance the optimizers, and we provide the reference and credit from where they were developed.

The framework is divided into four modules: *layers*, *models*, *optimization*, and *utils*. The first module, layers, contains

The *layers* module contains the template for all layers, the *GenericLayer*, that provides an opaque method for saving and loading a layer, and consequently a model, parameters. Moreover, in this module we also provide relevant layers used in deep learning and is further discussed in A.2.1. The *models* module contains all the developed models for this framework that provide a easy to use interface to train the models. The *optimization* module provides everything related to model optimization, such as optimizer algorithms (*e.g.* ADAM, RMSProp, SGD), and regularization techniques (dropout). Finally, the *utils* module provides utilities for different contexts, for instance decay functions for a curriculum learning train, data handlers template, among others. In addition, the framework also provides the activation functions that Theano provides, as well as the variables initialization.

A.2.1 Layers

A layer is a representation of a ANN hidden layer, namely a python class representation. There are numerous possible layers, feedforward, convolutional, recurrent, thus we only provide a few of those layers as they are the ones that met our requirements during the development of the framework, namely for RNNs and sequence to sequence. A note, although we did not use attention models and attention decoders, they were both developed. The implemented layers are:

- Attention mechanism
- Fully Connected Layer (Dense)
- Time Distributed Fully Connected Layer (Time Distributed Dense)
- GRU
- Bidirectional GRU
- LSTM
- Bidirectional LSTM
- GRU Decoder
- LSTM Decoder
- GRU Attention Decoder

As already described, this module contains the generic representation for the existent and future layers, named *GenericLayer*, which provides the super class abstraction for all layers. In addition, this class provides a way to load and save the layer parameters from or to a file.

An example of one of the simplest layers is the fully connected layer and is provided next – we provide the code.

```
class Dense(Layer):
    """
        This layer is responsible for performing a linear transformation from one dimension to
        another one.
    """

    def __init__(self, input_size, output_size, activation=linear, bias=True, name='dense',
    **kwargs):
        """
        Init method for Dense layer.
        :param input_size:
        :param output_size:
        :param activation:
        :param bias:
        :param name:
        :param kwargs:
        """
        self.input_size = input_size
        self.output_size = output_size
        self.activation = activation
        self.name = name
        self.bias = bias

        self.W = random_variable((self.input_size, self.output_size),
                                 name='{}_W'.format(name))

        params = [self.W]

        if self.bias:
            self.b = theano_zeros((self.output_size,),
                        name='{}_b'.format(name))
            params += [self.b]

        super(Dense, self).__init__(params)

    def forward(self, x, mask=None, mask_out=True, **kwargs):
        """
        Forward method for Dense.

        :param x:         2D Matrix, (batch_size, input_size)
        :param mask:      Matrix with binary mask for padding,
        (batch_size, nr_frames), not used.
        :param mask_out: Boolean value to indicate whether to
        perform masking to the hidden state, not used.
        :return: hidden state with the linear transformation
        """
        z = T.dot(x, self.W) + self.b
        return self.activation(z)
```

All layers must have two essential and mandatory components: a constructor, a *forward* method. Moreover, other layers can provide a *predict* method to differentiate the *forward* and *predict* operations.

In the previous example, we initialize all layer's parameters as shared variables, this is, the variables are loaded to the GPU (if it is available) and remain in the GPU memory. In addition, the constructor should define all the weight and bias matrix of the layers and call the super constructor with those parameters.

The *forward* method defines the layer's behaviour during training, and in some cases during test. In the previous case, the layer's behaviour is performing the transformation into the hidden space followed by a non-linearity

$$y = activation(W \cdot x + b) \tag{A.1}$$

Furthermore, the *predict* method should be present when the behaviour at test time is different from the one at training time.

An important note, n the current version, the framework does not support training a single a layer just as a layer, to train even a single layer the models should be used.

A.2.2 Models

A model is the abstract representation for the behaviour one or more layers define, i.e., the model allows to develop DNN with specified layers, for instance a stacking multiple layers where hidden states are propagated from one layer to another as input. The models developed are the following:

- DNN (feedforward, recurrent and recurrent bidirectional)
- Sequence to Sequence
- Attention Sequence to Sequence

Each model should combine several implemented layers, as one chooses, and must be, at least, a subclass of *GenericLayer*. Thus, each model should provide a forward and a predict method. We provide a generic model that already provides minibatch update, which behaves as online update for batches of size 1 and batch update when the batch size is the size of the training set, and that already provides the computational graph for a simple prediction model. Thus this model can be extended to define the variables of the graph and use as-is. Next we provided the code for the deep network which is a generic model, thus the computational graph and all that is required to build one is abstracted:

```
class StackedLayers(Model):
    """
    This class is the representation of a deep network.
    This way is easy to stack LSTM, GRU, or BiDirectional layers.
    The last step of this model is to perform a linear transformation
    from the last hidden size dimension to the output
    dimension.
    """
    def __init__(self, input_size, h_sizes, output_size, layer=LSTM, do_softmax=False):
        """
        Init method of the StackedLayers model.

        :param input_size:
        :param h_sizes: list, sizes. Should be of the
        form [input_size, hidden_size, ..., n_features].
        :param layer: class, layer to use
        """

        assert isinstance(h_sizes, list), "Please provide a list with the hidden sizes"
        assert len(h_sizes) > 0, "Empty hidden sizes list. Aborting."

        self.input_size = input_size
        self.output_size = output_size
        self.h_sizes = h_sizes
        # Weight Initialization
        self.stack = [layer(self.input_size, h_sizes[0], name='{}0'.format(layer.__name__))]

        if isinstance(self.stack[0], BiDirectionalLayer):
            self.is_bidir = True

        if len(h_sizes) > 1:
            self.stack = self.stack + [layer(h_sizes[i], size,
                                             name='{}{}'.format(layer.__name__, i+1))
                                                for i, size in enumerate(h_sizes[1:])]

        self.t_dense = TimeDistributedDense(output_size=self.output_size,
                                            input_size=h_sizes[-1], name='t_dense')

        # Model parameters
        layers_params = [layer.params for layer in self.stack]
        layers_params = [params for sub_list in layers_params for params in sub_list]
        params = layers_params + self.t_dense.params
        super(StackedLayers, self).__init__(params, do_softmax)

    def forward(self, x, y=None, epsilon=None, mx=None, my=None, dropout_rate=0.1,
                mask_out=True, train_phase=True,
                learning_rate=0.001, **kwargs):

        # Encoders Forward Pass
        h = self.stack[0].forward(x, mask=mx, mask_out=mask_out)

        if train_phase:
            if self.is_bidir:
                h = (dropout(h[0], dropout_rate),dropout(h[1], dropout_rate))
            else:
                h = dropout(h, dropout_rate)
        if self.is_bidir:
            h = h[0] + h[1]

        for layer in self.stack[1:]:
            h = layer.forward(h, mask=mx, mask_out=mask_out)

            if train_phase:
                if self.is_bidir:
                    h = (dropout(h[0], dropout_rate), dropout(h[1], dropout_rate))
                else:
                    h = dropout(h, dropout_rate)

            if self.is_bidir:
                h = h[0] + h[1]

        y_hat = self.t_dense.forward(h)

        if self.do_softmax:
            y_hat = softmax(y_hat)

        return y_hat
```

Furthermore, all models require a computational graph to be compiled with Theano variables and the parameters of each layer, next we calculate the loss function (which is also an argument of the train method) and apply an optimizer, for instance ADAM. With these steps, we compile two Theano functions, one for training and one for validation/testing, that allow to use the numerical values over the computational graph. In addition, we also allow techniques, such as masking and masked costs, to be performed during the compilation of the graph. An overview of the compilation process is depicted next, note that we omit other methods including the minibatch update which is straightfoward:

```
class Model(GenericLayer):

    def __init__(self, params, do_softmax=False):
        super(Model, self).__init__(params)
        self.debug = os.environ.get(Constant.DEBUG_STR) is not None

    def compile(self, mask=False, learning_rate=0.01, momentum=0.9, dropout_rate=0.1,
                mask_out=True, loss_function=mean_squared_error, optimizer=rmsprop,
                **kwargs):

        symbolic_variables = self._symbolic_variables(**kwargs)

        self.sym_vars = symbolic_variables

        assert "y" in symbolic_variables, "Symbolic y not defined. Aborting"
        assert "x" in symbolic_variables, "Symbolic x not defined. Aborting"
        y_hat, y_hat_val = self._compile(symbolic_variables, mask=mask, mask_out=mask_out,
                                         learning_rate=learning_rate,
                                         momentum=momentum, dropout_rate=dropout_rate, **kwargs)

        # (batch_size, nr_frames_out, output_dim) * mask_out.dimshuffle(0, 1, 'x')

        y = symbolic_variables["y"]
        my = symbolic_variables["mask_y"]

        cost = loss_function(y, y_hat)
        cost_val= loss_function(y, y_hat_val)

        if mask:
            cost = masked_cost(cost, my)
            cost_val = masked_cost(cost_val, my)
        else:
            cost = masked_cost(cost)
            cost_val = masked_cost(cost_val)
        if self.debug:
            import theano.printing
            theano.printing.pydotprint(cost, outfile="debug.png", var_with_name_simple=True)

        grads = T.grad(cost, self.params)

        updates = optimizer(grads, self.params, learning_rate=learning_rate)

        if self.debug:
            print("[+] Creating train and validation functions")

        self._functions(cost, cost_val, updates, symbolic_variables, mask=mask, **kwargs)

        return self._train_function, self._validation_function

    def forward(self, x, y=None, epsilon=None, mx=None, my=None, dropout_rate=0.1,
                mask_out=True, train_phase=True,
                learning_rate=0.001, **kwargs):
        """
        Interface definition for Model's forward. Every model MUST override this method.

        :param x:
        :param y:
        :param epsilon:
        :param mx:
        :param my:
        :param dropout_rate:
        :param mask_out:
        :param train_phase:
        :param learning_rate:
```

```
        :param kwargs:
        :return:
        """
        pass


    def _compile(self, symbolic_variables, mask_out=True, mask=False,
                 learning_rate=0.01, momentum=0.9,
                 dropout_rate=0.1, **kwargs):
        """
        Compile function by default.

        :param symbolic_variables:
        :param mask_out:
        :param mask:
        :param with_info:
        :param learning_rate:
        :param momentum:
        :param dropout_rate:
        :param kwargs:
        :return:
        """
        with_info = kwargs["with_info"] if kwargs.get("with_info") else None

        x = symbolic_variables["x"]
        y = symbolic_variables["y"]
        mx = symbolic_variables["mask_x"] if mask else None
        my = symbolic_variables["mask_y"] if mask else None
        n_steps = symbolic_variables["n_steps"] if symbolic_variables.get("n_steps") else None
        tmp_steps = None
        if kwargs.get("n_steps") is not None and n_steps is not None:
            tmp_steps = kwargs.get("n_steps")
            del kwargs["n_steps"]

        if mask:
            y_hat = self.forward(x, n_steps=n_steps, mx=mx, my=my,
                                 dropout_rate=dropout_rate, mask_out=mask_out,
                                 train_phase=True,
                                 learning_rate=learning_rate, momentum=momentum, **kwargs) \
                if n_steps is not None \
                else self.forward(x, y=y, mx=mx, my=my, dropout_rate=dropout_rate,
                                  mask_out=mask_out, train_phase=True,
                                  learning_rate=learning_rate, momentum=momentum, **kwargs)

            y_hat_val = self.forward(x, n_steps=n_steps, train_phase=False,
                                     learning_rate=learning_rate,
                                     momentum=momentum, **kwargs) if with_info \
                else self.forward(x, y=y, train_phase=False, learning_rate=learning_rate,
                                  momentum=momentum, **kwargs)
        else:
            if self.debug:
                print("[+] Warning: Masking not provided.")

            y_hat = self.forward(x, n_steps=n_steps, dropout_rate=dropout_rate,
                                 mask_out=mask_out, train_phase=True,
                                 learning_rate=learning_rate, momentum=momentum, **kwargs) \
                if n_steps is not None \
                else self.forward(x, y=y, dropout_rate=dropout_rate, mask_out=mask_out, l
                                  earning_rate=learning_rate,
                                  momentum=momentum, train_phase=True)

            y_hat_val = self.forward(x, n_steps=n_steps, train_phase=False,
                                     learning_rate=learning_rate,
                                     momentum=momentum) \
                if with_info else self.forward(x, y=y, train_phase=False,
                                               learning_rate=learning_rate,
                                               momentum=momentum, **kwargs)

        if kwargs.get("n_steps") is not None and n_steps is not None:
            kwargs["n_steps"] = tmp_steps

        return y_hat, y_hat_val

    def _functions(self, cost, cost_val, updates, symbolic_variables, mask=False, **kwargs):
        """
        Internal functions definition.
        By default each model is provided with functions with number of steps provided.

        :param cost:         Cost of train.
        :param cost_val:     Cost of validation.
        :param updates:      Optimizer updates.
        :param x:            Tensor3 symbolic.
        :param y:            Tensor3 symbolic.
        :param n_steps:      Integer Scalar symbolic.
        :param mx:           Matrix symbolic.
        :param my:           Matrix symbolic.
        :param mask:         Boolean. Mask.
        :param with_info:    Boolean. If with n_steps
        :param kwargs:       Remaining args.
        :return:
        """

        x = symbolic_variables["x"]
        y = symbolic_variables["y"]
        mx = symbolic_variables["mask_x"] if mask else None
        my = symbolic_variables["mask_y"] if mask else None
        n_steps = symbolic_variables["n_steps"] if symbolic_variables.get("n_steps") else None
        assert y is not None or n_steps is not None, "Nor y nor n_steps provided. Aborting."
        if mask:
            train_function = theano.function(inputs=[x, n_steps, y, mx, my],
                                             outputs=cost, updates=updates) \
                if n_steps is not None else theano.function(inputs=[x, y, mx, my],
                                                            utputs=cost, updates=updates)

            validation_function = theano.function(inputs=[x, n_steps, y, mx, my],
                                                  utputs=cost_val) \
                if n_steps is not None else theano.function(inputs=[x, y, mx, my],
                                                            outputs=cost_val,
                                                            on_unused_input='ignore')
        else:
            train_function = theano.function(inputs=[x, n_steps, y],
                                             outputs=cost, updates=updates) \
                if n_steps is not None else theano.function(inputs=[x, y],
                                                            outputs=cost, updates=updates)

            validation_function = theano.function(inputs=[x, n_steps, y], outputs=cost_val) \
                if n_steps is not None else theano.function(inputs=[x, y], outputs=cost_val)

        self._train_function = train_function
        self._validation_function = validation_function

    def _symbolic_variables(self, **kwargs):
        """
        Symbolic variables of the model.
        By default only x, y and masks are defined. If with_info is provided, then
        the number of steps is also added.

        :param kwargs:
        :return: dictionary with symbolic vars
        """
        vars = {}
        vars['x'] = T.tensor3('x')
        vars['y'] = T.tensor3('y')
        vars['mask_x'] = T.matrix('mask_x')
        vars['mask_y'] = T.matrix('mask_y')

        if "with_info" in kwargs:
            if(kwargs.get("with_info")):
                vars['n_steps'] = T.lscalar('target_timesteps')

        return vars
```

Finally, we also provide an early stopping mechanism to allow the user more flexibility while choosing the number of epochs for the training. Moreover, we also provide more advanced models such as sequence to sequence with and without the attention mechanism where the model allows the decoder to take a peek at the target during training. Thus, the decoders developed allow to take a peek at the target to increase the performance, while a more advanced decoder is equipped with an attention mechanism which allows to align the input with the output.

A current limitation of the framework is that the models does not, yet, support saving and loading its configuration.

A.2.3 Usage example

Using a model with our framework is extremely simple and requires very little code. An example of a single GRU layer to learn a sinusoidal signals is presented below:

```
from l2fdeepl.layers.gru import *
from l2fdeepl.layers.lstm import *
from l2fdeepl.layers.bidirectionallayer import *
from l2fdeepl.models.stacked import *
from l2fdeepl.layers.dense import TimeDistributedDense
from l2fdeepl.optimization.loss_functions import mean_squared_error as mse, masked_cost
from l2fdeepl.optimization.optimizers import adam



if __name__ == ”__main__”:
    import numpy as np
    n_feat = 1
    n_timesteps = 1200
    n_samples = 10
    n_feat_tgt = 64

    model = StackedLayers(n_feat, [n_feat, 256, 128, n_feat_tgt],
                          n_feat, layer=BiGRU)

    x = np.random.random((n_samples, n_timesteps, n_feat)).astype(theano.config.floatX)
    y = np.random.random((n_samples, n_timesteps, n_feat_tgt)).astype(theano.config.floatX)

    x_val = np.random.random((n_samples, n_timesteps, n_feat)).astype(theano.config.floatX)
    y_val = np.random.random((n_samples, n_timesteps, n_feat_tgt)).astype(theano.config.floatX)
    train_data = {}
    train_data[’x’] = x
    train_data[’y’] = y

    validation_data = {}
    validation_data[’x’] = x_val
    validation_data[’y’] = y_val
```

```
n_epochs = 5

model.minibatch_update(data, validation_data, n_epochs)
```

A.2.4 Future Development

A better version is currently being developed, cleaner and more abstract, as well as more flexible. This version will support more features, such as convolutional layers, better sampling (beam search), embeddings layers, among others. This is yet another deep learning framework which is in a very early stage.