Adding semantic robustness to dialog agents

Felipe Salvatore

July 1, 2018

Abstract

Using the available dataset SICK (Sentence Involving Compositional Knowledge), we introduce a set of new question answering tasks **Entailment-QA** to measure how well a dialog system deals with abstract semantic knowledge. These tasks force the dialog agent to struggle with distinct notions that are at the intersection of text understanding and reasoning: boolean connectives, first-order quantifiers, synonymy/antinomy/hypernymy resolution, and paraphrase. Experimental results indicate that the current dialog systems present difficulties in solving these tasks.

Contents

Chapter 1

Introduction

One of the main goals in natural language processing (NLP) is to build agents capable of *understanding language and reasoning*, i.e., agents capable of carrying a conversation as if they were human. This goal is as old as the field of artificial intelligence (AI) itself [27], and it is a very ambitious one. We shall call this kind of agents "dialog systems".

But if we think more carefully about this subject, we can recognize that there is not a single type of conversation. A dialog between strangers in an elevator, a conversation between one psychologist and his patient, a salesman offering a service to a possible client, a scientific exchange in a conference; all these events can be classified as "dialog" but they present very different *dynamics* and *goals*. As a way of refining the analysis, the NLP literature on dialog made the distinction between *goal-driven dialog systems* and *non-goal-driven dialog systems*; the former includes *chatbots*, normally used in the industry for technical support services or information acquisition; the latter is a term used to refer to any conversational agent with no explicit purpose.

Each paradigm presents its advantages and disadvantages: on the one hand goal-driven dialog systems are useful systems but they demand a precise understanding (of the goal at hand) and there are not so much available data to train such systems. On the other hand, non-goal-driven dialog systems produce, at best, chit-chat conversation. Although we have a lot of available data to train such models (movie legends, social media conversations, etc.), it is not so easy to justify its relevancy.

Moreover, there is a central difference between these two paradigms: *the evaluation metric*. More precisely, *there is no good quantitative metric to compare non-goal-driven agents*, and as a consequence *there is also no stan-*

dardized benchmark. Given this limitations one alternative approach was proposed by [31]: developed a series of synthetic tasks in the form of question answering (QA) to test different capabilities of the competing models. Each task tries to assert one prerequisite to *full language understanding*.

Following this research decision, here we propose to expand the work done in [2, 31] by *adding new testbeds for complex semantic relationships*: **Entailment-QA**. The motivation behind this expansion is to guarantee that an end-to-end machine learning model can perform *complex linguistic inferences*. So far we have been focusing on two kinds of inferences: the ones defined by *logical operators* and others defined by *word knowledge*.

1.1 Motivation

Logic is not important by itself. But it can help us build agents capable of distinguishing between sentences that have a real informational content from sentences that do not. An intelligent agent should distinguish between *meaningful* and *nonsensical* speech. To do that the agent can make use of background knowledge, but it can also point out *gaps in the speech's rationality*. Although this importance, logical reasoning is one area that is often neglected by conversational AI researchers. This project tries to address this deficiency.

We often see a discussion inside the AI community between deep learning specialist and the classical AI researcher. This discussion can have a non-productive outcome: we heard that the former's engineering methodology is a kind of unscientific alchemy and that the results from the later has been made irrelevant by big data and deep models. Here we are trying to find a productive middle ground: combining the rich tradition of certain themes of classical AI (like logic reasoning) with the models and techniques of the deep learning community.

1.2 Objectives

The goal of this project is twofold: propose a new set of synthetic tasks in the same lines as [31] in order to help evaluating and building dialog systems that present reasonable text reasoning capabilities; and proposing new models for these tasks. Hence, we have defined the following specific objectives:

- Create a set of tasks that explicitly make use of complex logical forms.
- Perform a stress test in the existing *neural network based end-to-end dialog systems.*
- Integrate linguistic reasoning with visual references to create a new set of visual question answering (VQA) tasks.
- Define new models to achieve better results in the tasks proposed above.

1.3 Organization

Chapter 2 exposes the theory that is the basis for the research. Chapter 3 presents the models for dialog and our proposed task to evaluate logical reasoning. Chapter 4 discuss the possible next steps of this research.

Chapter 2

Background

We can make human language manageable to computers by using a sort of different methods and techniques. This is the bread and butter of any NLP researcher. Here, following the NLP community, we will focus only in the techniques derived from the machine learning field. In this chapter, we will present some abstract machine learning models follow by some applications in NLP.

2.1 Machine Learning

Machine learning is the branch of computer science that deals with programs that can improve with experience (i. e., learn). Machine learning is divided into three main subareas: *supervised learning*, *reinforcement learning* and *unsupervised learning.*

We will work with supervised learning only. In this setting, we assume that the process of interest is defined by an unknown function $g : X \to Y$. We try to approximate g by changing the parameters of a function f through an optimization process. The optimization is done by defining an *error function* that evaluates how well f approximates g in the part of g that we have access: the training data $D = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$ (where $g(\boldsymbol{x}^{(i)}) = \boldsymbol{y}^{(i)}$).

f can be a function from any family of models. One family that is having a lot of success for language tasks is the *neural network* family.

2.2 Neural Networks

A neural network is a non-linear function $f(\boldsymbol{x}; \boldsymbol{\theta})$. It is defined by a collection of parameters $\boldsymbol{\theta}$ and a collection of non-linear transformations. It is usual to represent f as a compositions of functions:

$$\begin{aligned} f(\boldsymbol{x}; \boldsymbol{\theta}) &= f^{(2)}(f^{(1)}(\boldsymbol{x}; \boldsymbol{W}_1, \boldsymbol{b}_1); \boldsymbol{W}_2, \boldsymbol{b}_2) && (2.1) \\ &= softmax(\boldsymbol{W}_2(\sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)) + \boldsymbol{b}_2) && (2.2) \end{aligned}$$

The output of these intermediary functions are referred as *layers*. So in the example above, $\boldsymbol{x}$ (the output of the identity function) is the *input layer*, $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}_1, \boldsymbol{b}_1)$ is the *hidden layer* and $f^{(2)}(f^{(1)}(\boldsymbol{x}; \boldsymbol{W}_1, \boldsymbol{b}_1); \boldsymbol{W}_2, \boldsymbol{b}_2)$ is the *output layer*. Since each layer is a vector, we normally speak about the *dimension* of a layer. For historical reasons we also say that each entry on a layer is a *node* or a *neuron*. Models with a large number of hidden layers are called *deep models*, for this reason the name *deep learning* is used.

A neural network is a function approximator: it can approximate any Borel measurable function from one finite dimensional space to another with any desired nonzero amount of error. This theoretical result is know as the *universal approximation theorem* [4]. Without entering in the theoretical concepts, it suffice to note that the family of Borel mensurable functions include all continuous functions on a closed and bounded subset of $\mathbb{R}^n$.

Different deep learning architectures are used in NLP: *convolutional architectures* have a good performance in tasks were it is required to find a linguistic indicator regardless of its position (e.g., document classification, short-text categorization, sentiment classification, etc); high quality word embeddings can be achieved with models that are a kind of *feedforward neural network* [17]. But for a variety of works in natural language we want to capture regularities and similarities in a text structure. That is way *recurrent* and *recursive* models have been widely used in the field. Here we are focused on generative models and since recurrent models have been producing very strong results for language modeling [5], we will concentrate on them.

2.3 Recurrent Neural Network

Recurrent neural network (RNN) is a family of neural network specialized in sequential data $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\tau)}$. As a neural network, a RNN is a parametrized function that we use to approximate one hidden function from the data. What makes RNN unique is a recurrent definition of one of its hidden layer:

$$\boldsymbol{h}^{(t)} = g(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \tag{2.3}$$

$\boldsymbol{h}^{(t)}$ is called *state*, *hidden state*, or *cell*.

This recurrent equation can be unfolded for a finite number of steps τ. For example, when $\tau = 3$:

$$\begin{align}
\boldsymbol{h}^{(3)} &= g(\boldsymbol{h}^{(2)}, \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) \tag{2.4}\\
&= g(g(\boldsymbol{h}^{(1)}, \boldsymbol{x}^{(2)}; \boldsymbol{\theta}), \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) \tag{2.5}\\
&= g(g(g(\boldsymbol{h}^{(0)}, \boldsymbol{x}^{(1)}; \boldsymbol{\theta}), \boldsymbol{x}^{(2)}; \boldsymbol{\theta}), \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) \tag{2.6}\\
& \tag{2.7}
\end{align}$$

Using a concrete example consider the following classification model define by the equations:

$$f(\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}; \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{c}, \boldsymbol{b}) = \hat{\boldsymbol{y}}^{(t)} \tag{2.8}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}) \tag{2.9}$$

$$\boldsymbol{h}^{(t)} = g(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}) \tag{2.10}$$

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}) \tag{2.11}$$

This kind of model can create an output $\hat{\boldsymbol{y}}^{(t)}$ at each time t, or the model can produce a single output $\hat{\boldsymbol{y}}$ after processing an entire input sequence. This choice depends on the learning problem that is being modeled.

With the model's prediction at hand, we can use a loss function (like cross entropy for the classification problem) and apply the back-propagation algorithm to optimize the model. These models look complex, but it quite straightforward to compute the gradients [6, p. 374].

Although this kind of deep learning model is very useful, it presents a severe flaw. When computing the gradients there is a lot of repeated matrix multiplication using the recurrent weight matrix (in the example above, the matrix $\boldsymbol{W}$). Depending on some configurations of this matrix *the gradients may vanish or explode exponentially with respect to the number of time steps.*

Thus, handing long-term dependencies became a problem when using RNNs. Different solutions were proposed, the most effective results came from some modifications of this model. We will present the two most famous modifications: *the gated recurrent unit* and *the long short-term memory.*

2.4 Gated Recurrent Unit

To capture long-term dependencies on a RNN the authors of the paper [3] proposed a new architecture called *gated recurrent unit* (GRU). This model was constructed to make each hidden state $\boldsymbol{h}^{(t)}$ to adaptively capture dependencies of different time steps. It work as follows, at each step t one candidate for hidden state is formed:

$$\widetilde{\boldsymbol{h}}^{(t)} = tahn(\boldsymbol{W}(\boldsymbol{h}^{(t-1)} \odot \boldsymbol{r}^{(t)}) + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}) \tag{2.12}$$

where $\boldsymbol{r}^{(t)}$ is a vector with values in $[0,1]$ called a *reset gate*, i.e., a vector that at each entry outputs the probability of reseting the corresponding entry in the previous hidden state $\boldsymbol{h}^{(t-1)}$. Together with $\boldsymbol{r}^{(t)}$ we define an *update gate*, $\boldsymbol{u}^{(t)}$. It is also a vector with values in $[0,1]$. Intuitively we can say that this vector decides how much on each dimension we will use the candidate update. Both $\boldsymbol{r}^{(t)}$ and $\boldsymbol{u}^{(t)}$ are defined by $\boldsymbol{h}^{(t-1)}$ and $\boldsymbol{x}^{(t)}$; they also have specific parameters:

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}_r\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r\boldsymbol{x}^{(t)} + \boldsymbol{b}_r) \tag{2.13}$$

$$\boldsymbol{u}^{(t)} = \sigma(\boldsymbol{W}_u\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u\boldsymbol{x}^{(t)} + \boldsymbol{b}_u) \tag{2.14}$$

At the end the new hidden state $\boldsymbol{h}^{(t)}$ is defined by the recurrence:

$$\boldsymbol{h}^{(t)} = \boldsymbol{u}^{(t)} \odot \widetilde{\boldsymbol{h}}^{(t)} + (1 - \boldsymbol{u}^{(t)}) \odot \boldsymbol{h}^{(t-1)} \tag{2.15}$$

Note that the new hidden state combines the candidate hidden state $\widetilde{\boldsymbol{h}}^{(t)}$ with the past hidden state $\boldsymbol{h}^{(t-1)}$ using both $\boldsymbol{r}^{(t)}$ and $\boldsymbol{u}^{(t)}$ to adaptively copy and forget information.

2.5 Long Short-Term Memory

Long short-term memory (LSTM) is one of the most applied versions of the RNN family of models. Historically it was developed before the GRU model, but conceptually we can think in the LSTM as an expansion of the model presented in the last session. Because of notation differences they can look different. LSTM is also based on parametrized gates; in this case three: the *forget gate*, $\boldsymbol{f}^{(t)}$, the *input gate*, $\boldsymbol{i}^{(t)}$, and the *output gate*, $\boldsymbol{o}^{(t)}$. The gates are defined only by $\boldsymbol{h}^{(t-1)}$ and $\boldsymbol{x}^{(t)}$ with specific parameters:

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f) \tag{2.16}$$

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i) \tag{2.17}$$

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o) \tag{2.18}$$

Intuitively $\boldsymbol{f}^{(t)}$ should control how much informative will be discarded, $\boldsymbol{i}^{(t)}$ controls how much information will be updated, and $\boldsymbol{o}^{(t)}$ controls how munch each component should be outputted. A candidate cell, $\tilde{\boldsymbol{c}}^{(t)}$ is formed:

$$\tilde{\boldsymbol{c}}^{(t)} = tahn(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}) \tag{2.19}$$

And a new cell $\boldsymbol{c}^{(t)}$ is formed by forgetting some information of the previous cell $\tilde{\boldsymbol{c}}^{(t-1)}$ and by adding new values from $\tilde{\boldsymbol{c}}^{(t)}$ (scaled by the input gate)

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)} \tag{2.20}$$

The new hidden state, $\boldsymbol{h}^{(t)}$, is formed by filtering $\boldsymbol{c}^{(t)}$:

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot tanh(\boldsymbol{c}^{(t)}) \tag{2.21}$$

Until now we have presented general deep learning theory, now we will focus on the specificities of these models applied to natural language problems.

2.6 Language model

We call *language model* a probability distribution over sequences of tokens in a natural language.

$$P(x_1, x_2, x_3, x_4) = p$$

This model is used for different NLP tasks such as speech recognition, machine translation, text auto-completion, spell correction, question answering, summarization and many others.

The classical approach to a language model was to use the chain rule of probability and a Markovian assumption, i.e., for a specific n we assume that:

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_1, \ldots, x_{t-1}) = \prod_{t=1}^{T} P(x_t | x_{t-(n+1)}, \ldots, x_{t-1}) \quad (2.22)$$

This gave raise to models based on n-gram statistics. The choice of n yields different models; for example, the *unigram* language model ($n = 1$) is defined as:

$$P_{uni}(x_1, x_2, x_3, x_4) = P(x_1)P(x_2)P(x_3)P(x_4) \quad (2.23)$$

where $P(x_i) = count(x_i)$ and *count* is a function that counts tokens occurrence in a corpus.

Similarly the *bigram* language model ($n = 2$) is defined as:

$$P_{bi}(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3) \quad (2.24)$$

where

$$P(x_i|x_j) = \frac{count(x_i, x_j)}{count(x_j)} \quad (2.25)$$

With these basic statistics we can already define useful language models. It is observed that higher n-grams yields better performance. This comes with a price though, higher n-grams requires great amounts of memory [7]. For this motive n-grams based language models that are trained on large corpora uses at most 5-grams.

Since [18] the landscape has change, instead of using one approach that is specific for the language domain, we can use a general model for sequential data prediction, a RNN. The RNN's ability to deal with unrestricted size sequence input permits to abandon the n-gram model's restricted context assumption.

To understand the language model task as a machine learning task, some details should be clear.

First, the learning task is to estimate the probability distribution

$$P(x_n = \text{word}_{j^*} | x_1, \dots, x_{n-1}) \tag{2.26}$$

for any $(n-1)$-sequence of words $x_1, \dots, x_{n-1}$.

Second, the function f being used to approximate 2.26 is trained on a corpus in the following way: each word x_t of the corpus will be used as input to f (in the form of an one-hot vector), and the immediate subsequent word, say x_{t+1}, will be used as a target. The training is done by minimizing the cross entropy loss between the model's output and the probability distribution given by the target.

One example is in order. Suppose our corpus $\mathbb{C}$ is compose only by the lines below:

Yes, here we go again, give you more, nothing lesser
Back on the mic is the anti-depressor
Ad-Rock, the pressure, yes, we need this
The best is yet to come, and yes, believe this

From this corpus we can construct a vocabulary list $\mathbb{V}$ as follows: after preprocessing the text we create a list of the most frequent words (in this case we can take all words) with the size V (here $V = 27$). Hence, we can treat each word in the text either by an index referring to the word position on $\mathbb{V}$ or as a one-hot vector that codifies this index (e.g., "the" would be identified with 0, "yes" with 1, "we" with 2, and so on).

Then, the dataset is the collection of words

$$D = \{(< eos >, \text{Yes}), (\text{Yes}, \text{here}), (\text{here}, \text{we}), \dots, (\text{believe}, \text{this}), (\text{this}, < eos >)\}$$

where $< eos >$ is the "end of sentence" token (also a member of $\mathbb{V}$).

A simple recurrent language model $f(\boldsymbol{x}^{(t)}, \boldsymbol{\theta})$ is defined by the following equations:

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)} \tag{2.27}$$

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{e}^{(t)} + \boldsymbol{b}) \tag{2.28}$$

$$f(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}) = \hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}) \tag{2.29}$$

where $\boldsymbol{E} \in \mathbb{R}^{d,V}$ is the matrix of word embeddings, $\boldsymbol{x}^{(t)} \in \mathbb{R}^{V}$ is one-hot word vector at time step t, $\boldsymbol{y}^{(t)} \in \mathbb{R}^{V}$ is the ground truth at time step t (also a one-hot word vector) and d is the size of the word embeddings.

For each word x_t let j_t be the index of the subsequent word, so at each time t the point-wise loss is:

$$\begin{aligned}
L^{(t)}(\boldsymbol{\theta}) &= CrossEntropy(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) && (2.30)\\
&= -\log(\hat{\boldsymbol{y}}^{(t)}_{j_t}) && (2.31)\\
&= -\log P(x_{t+1} = \text{word}_{j_t} | x_1, \dots, x_t) && (2.32)\\
&= -\log P(x_{t+1} | x_1, \dots, x_t) && (2.33)
\end{aligned}$$

The loss L is the mean of all point-wise losses

$$L(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} L^{(t)}(\boldsymbol{\theta}) \tag{2.34}$$

With the loss function defined, we apply some optimization algorithm like *stochastic gradient descent* to choose the optimal parameters for the language model:

$$\boldsymbol{\theta}^{*} = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{2.35}$$

Because of the historical connections with information theory, the *perplexity* (PP) metric is often used to evaluate a language model. This metric can be thought as the weighted average branching factor of a language.

Given $\mathbb{C} = x_1, x_2, \dots, x_T$, we define the perplexity of $\mathbb{C}$ ($PP(\mathbb{C})$) as:

$$\begin{aligned} PP(\mathbb{C}) &= P(x_1, x_2, \ldots, x_T)^{-\frac{1}{T}} && (2.36)\\ &= \sqrt[T]{\frac{1}{P(x_1, x_2, \ldots, x_T)}} && (2.37)\\ &= \sqrt[T]{\prod_{i=1}^{T} \frac{1}{P(x_i|x_1, \ldots, x_{i-1})}} && (2.38) \end{aligned}$$

Using 2.30 we can relate cross entropy loss and perplexity:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \frac{1}{T}\sum_{t=1}^{T} L^{(t)}(\boldsymbol{\theta}) && (2.39)\\ &= \frac{1}{T}\sum_{t=1}^{T} -\log P(x_{t+1}|x_1, \ldots, x_t) && (2.40)\\ &= \frac{1}{T}\sum_{t=1}^{T} \log((\frac{1}{P(x_{t+1}|x_1, \ldots, x_t)}) && (2.41)\\ &= \log\left(\sqrt[T]{\prod_{i=1}^{T} \frac{1}{P(x_i|x_1, \ldots, x_{i-1})}}\right) && (2.42)\\ &= \log(PP(\mathbb{C})) && (2.43) \end{aligned}$$

Hence,

$$2^{L(\boldsymbol{\theta})} = PP(\mathbb{C}) \qquad (2.44)$$

Thus, by finding the parameters that minimize the cross entropy error we are also minimizing the perplexity of the language model.

2.7 Sequence-to-Sequence

There is one powerful application of RNN based language model. The authors from [26] used two RNNs to create an end-to-end translation model that is now know as *the encoder-decoder* or *the sequence-to-sequence* (seq2seq) architecture. This architecture is define as follows: let $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$ be a source sentence in the one-hot representation, let $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}$ be the target sentence also in the one-hot format. f_{enc} (the *encoder*) is a RNN with the sole purpose of creating a vector representation of input language's sequences. f_{dec} (the *decoder*) is a language model for the target language. These models are trained together mapping source sentences to target sentences.

For example, suppose the training pair $(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}, \boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(m)})$ is ("Nas tardes de fazenda há muito azul demais", "In the farm's afternoons there is too much blue"). Here we want the model to translate one Portuguese sentence to an English one. We first encode the Portuguese sentence in the vector $\boldsymbol{s}$, i.e.,

$$\boldsymbol{s} = f_{enc}(\boldsymbol{x}^{(n)}, \boldsymbol{h}^{(n-1)}) \tag{2.45}$$

Then using the control English sentence as a target, at each time t we compute the cross entropy error between the decoder prediction $f_{dec}(\boldsymbol{y}^{(t)}, \tilde{\boldsymbol{h}}^{(t-1)})$ and the target $\boldsymbol{y}^{(t+1)}$ ($\boldsymbol{y}^{(0)}$ is the $< eos >$ token). The decoder uses the vector representation of the source sentence $\boldsymbol{s}$ as an initial hidden state (i.e., $\tilde{\boldsymbol{h}}^{(0)} = \boldsymbol{s}$). The goal of this model is to to approximate the probability distribution over the tokens from the target language given the sentence of the source language, i.e.,

$$\tilde{\boldsymbol{h}}^{(t)} = f_{dec}(\boldsymbol{y}^{(t)}, \tilde{\boldsymbol{h}}^{(t-1)}) \tag{2.46}$$

$$p(y_t|y_1, \dots, y_{t-1}, x_1, \dots, x_n) = softmax(\boldsymbol{W}_s \tilde{\boldsymbol{h}}^{(t)} + \boldsymbol{b}_s) \tag{2.47}$$

One limitation of this architecture is that the source sentence, in some cases, has more features than the decoder embedding $\boldsymbol{s}$ can properly summarize. To address that some attention mechanisms were introduced.

2.8 Attention

The attention-based models are models built on top of the seq2seq architecture. The encoding part continues the same as before, but now at each time t a context vector $\boldsymbol{c}^{(t)}$ is defined to capture relevant source-side information to help the prediction of the current target word $\boldsymbol{y}^{(t)}$. Once $\boldsymbol{c}^{(t)}$ is constructed the attention hidden state is defined as:

$$\tilde{\boldsymbol{h}}^{(t)} = tahn(\boldsymbol{W}_c[\boldsymbol{c}^{(t)}; \boldsymbol{h}^{(t)}]) \tag{2.48}$$

With the attention hidden state defined, the model's prediction is the same as the one defined in 2.47.

The core of this technique is the definition of $\boldsymbol{c}^{(t)}$. There are different strategies available, here we will focus only on one: *global attention*.

Let $\boldsymbol{a} \in \mathbb{R}^{m,n}$. We will use this matrix as an alignment matrix, i.e., at the end of the training $\boldsymbol{a}_{ts}$ should reflect the probability of the source representation $\boldsymbol{h}^{(s)}$ be relevant for the output $\hat{y}^{(t)}$. We define $\boldsymbol{a}_{ts}$ as

$$\boldsymbol{a}_{ts} = \frac{exp(score(\tilde{\boldsymbol{h}}_t, \boldsymbol{h}_s))}{\sum_j exp(score(\tilde{\boldsymbol{h}}_t, \boldsymbol{h}_j))} \tag{2.49}$$

Where *score* is a content-based function that can have different implementations:

$$score(\tilde{\boldsymbol{h}}_t, \boldsymbol{h}_s) = \begin{cases} \tilde{\boldsymbol{h}}_t^\top \boldsymbol{h}_s \\ \tilde{\boldsymbol{h}}_t^\top \boldsymbol{W}_a \boldsymbol{h}_s \\ \boldsymbol{v}_a^\top tahn(\boldsymbol{W}_a[\tilde{\boldsymbol{h}}_t; \boldsymbol{h}_s]) \end{cases} \tag{2.50}$$

At the end, a global context vector $\boldsymbol{c}^{(t)}$ is computed as the weighted average, according to $\boldsymbol{a}_t$ over all source states:

$$\boldsymbol{c}^{(t)} = \sum_s \boldsymbol{a}_{ts} \boldsymbol{h}^{(s)} \tag{2.51}$$

Chapter 3

Dialog Systems

In this chapter, we will review some techniques to produce dialogs together with some evaluation strategies. For brevity's sake, we decided not to give a full historic presentation of the field of dialog generation. Hence, our review of the techniques will be deep learning oriented.

3.1 Using neural networks to generate dialog

For some time the main paradigm in AI was the so called *knowledge base approach*: the main goal of AI was to develop intelligent systems capable of solving certain classes of problems by having a *representation* or *model* of the world. In this view, a decision by a machine is synonymous to *inferring in a formal language* [16]. This was applied to the dialog system ELIZA developed at MIT in 1964 [29]. The ELIZA system was built with a very precise dialog model, it tried to simulate a Rogerian psychologist. So in a conversation with a human the human's statements were reflected back at them based on an algorithm that transforms the human's sentence using a keyword rank system.

A different point of view is that AI should construct systems that acquire their knowledge via *data observation*. More useful than programming hand-coded rules in an AI agent, we should enable that agent to extract patters from the complexity of data. This is the *machine learning approach* and, regarding dialog generation, the models based on this point of view are the most important models today [2, 14, 21, 22, 23, 30].

This point of view makes sense when we frame the dialog generation problem as a translation problem. As in the field of automatic translation, we can use the already available human-made translations as data to train a machine learning model. So all the complex translation rules will be learned from the data. In a similar way, we can also use already available dialog as data. To frame the dialog task between two agents A and B as a translation problem we need to see the dialog act as follows: the source language is the set of utterances spoken by A, the target language is the sentences of B, and so the dialog system is just a program translating massages from A to B. Hence, the dialog generation problem can be viewed as a simple *context-to-response mapping*.

3.1.1 The seq2seq approach

The authors of [28] were the first to use this model to generate dialog, since then this approach has deeply influenced the field of dialog generation. As we have seen, a seq2seq model encodes the source sentence in a vector $\boldsymbol{h}$ and uses this vector to initiate a language model. This can be used to model a dialog as follows: suppose we have recorded two agents A and B. We will use this dialog as the training data for our model. The model will try to predict how B would respond. Let $x = x_1, x_2, \ldots, x_n$ be an utterance from A and $y = y_1, y_2, \ldots, y_m$ be the response from B (both x and y are a sequence of words). x is the input to an encoder network, which is a RNN. The encodes summarizes the input utterance into a vector, say $\boldsymbol{s}$, and this vector is used to initiate the decoder model, which is another RNN. The decoder takes as input y and at each time t the model tries to predict the word $t + 1$ using $y_1, y_2, ..., y_{t-1}, \boldsymbol{s}$. This became a multiclass classification task, and, as usual, we train the model by minimizing the cross entropy loss.

After the model is trained we can use the same model to generate a new dialog as follows: the agent A gives an utterance and we use the encoder model to summarize this utterance in a vector $\boldsymbol{s}$ as before, but now we use the probability distribution learned by the decoder model (initialized by $\boldsymbol{s}$) to sample words until it generates an end-of-speech <eos> token (or until a upper bound for the utterance size is reached). This approach lead to interesting results, for example in [28] after they have trained the model on the OpenSubtitles dataset (a dataset compose of subtitles from different movies) the follow dialog was produced:

- **Human**: What is moral?

- **Machine**: What empowered humanity, what intellectual the essence is.
- **Human**: What is immoral?
- **Machine**: The fact that you have a child.
- **Human**: What is morality?
- **Machine**: What is altruism?
- **Human**: What is the definition of altruism?
- **Machine**: If you don't believe in god, then you don't know.
- **Human**: Ok ... so what is the definition of morality?
- **Machine**: Well, the truth is, you're not a believer in God almighty.
- **Human**: Tell me the definition of morality, I am quite upset now!
- **Machine**: I'm not ashamed of being a philosopher!

The model is clearly learning some interesting responses, but we can observe that some answers are *generic* (for example, when the model answer the question "What is morality?" with another question "What is altruism?"). We can also see that the model lacks *coherence on the responses.* It seems that the model fails to generate meaningful responses taking the dialog context into account. Solving this problem was the focus of the subsequent models.

3.1.2 The hierarchical recurrent encoder-decoder architecture

For example, take the *hierarchical recurrent encoder-decoder architecture* (HRED) model proposed in [22]. In this setting a dialogue is viewed as a sequence of utterances $D = U_1, \dots, U_M$ involving two interlocutors such that each U_i contains a sequence of N_i tokens, $U_i = x_{i,1}, \dots, x_{i,N_i}$. Each $x_{i,j}$ is a random variable taking values in $\mathbb{V} \cup \mathcal{A}$ where $\mathbb{V}$ is the vocabulary and $\mathcal{A}$ is the set of *speech acts* (for example "pause" and "end of turn" are speech acts). In this case the learning task is the same as the one from language modeling: we want to estimate the probability of one token (speech acts included) conditioned on the previously tokens:

$$P(x_{i,j}|x_{i,1},\ldots,x_{i,j-1},U_1,\ldots U_{i-1}) \tag{3.1}$$

The different utterances $U_1,\ldots,U_M$ will serve as a dialog corpus. The HRED model is based on three RNNs: an *encoder* RNN, a *context* RNN and a *decoder* RNN. To understand the workings of this model, suppose we have the following dialog:

- U_1: Mom, I don't feel so good.

- U_2: What's wrong?

- U_3: I feel like I'm going to pass out.

First the encoder RNN will encode the sentence U_1 in a vector $\boldsymbol{U}_1$, then the context RNN will compute a hidden state, say $\boldsymbol{C_1}$ using $\boldsymbol{U}_1$ and all past sentences in the dialog. After that the decoder RNN will use U_2 as an input to predict the next word using $\boldsymbol{C_1}$. The same procedure is repeated for U_2 and for the following utterances.

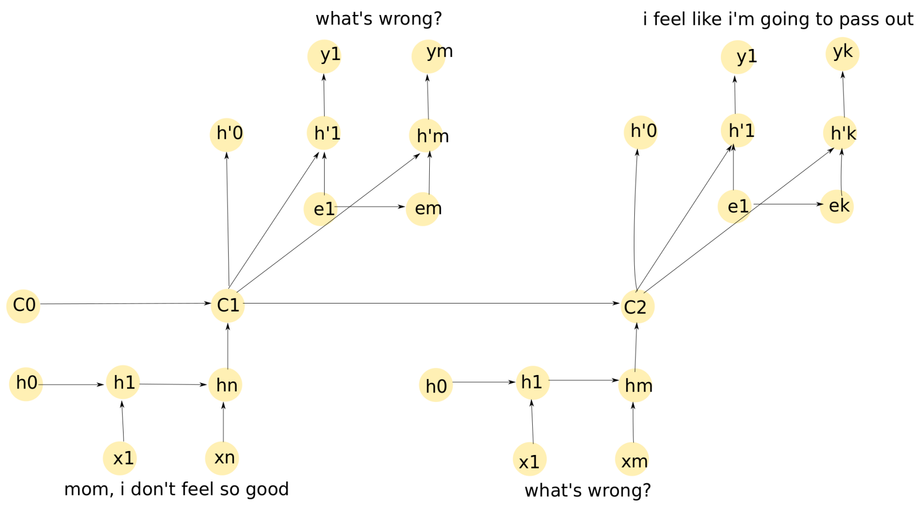

Figure 3.1: HRED model

As can be seen in Figure 3.1, the prediction from the decoder model is conditioned on the hidden state of the context RNN.

3.1.3 The memory model

One version of the RNN model that found success on the dialog task is the Memory Network model [25]. In a memory model the recurrence reads from a possibly large external memory multiple times before outputting a token. To explain this model we will use a very simplified version of dialog: a question-answer example where the answer is composed of one word only.

Let V be the vocabulary size, $U_1, ..., U_n$ be the context utterances (where $U_i = x_{i,1}, \ldots, x_{i,N_i}$), $q = w_1, \ldots, w_l$ a question and a the answer. Here we use w and $x_{i,j}$ both as a token and as an one-hot vector representing the same token.

The memory model is defined by k memory layers, each layer is compose of the following parts:

- $\{\boldsymbol{m}^k{}_i\}$ is an n-sequence of *memory vectors.* Where $i = 1, \ldots, n$ and $\boldsymbol{m}_i^k = \sum_j \boldsymbol{A}^k x_{i,j}$.
- $\boldsymbol{u}^k$ is the *input vector*, where

$$\boldsymbol{u}^k = \begin{cases} \sum_j \boldsymbol{B}^k w_j & \text{if } k = 1, \\ \boldsymbol{u}^{k-1} + \boldsymbol{o}^{k-1} & \text{otherwise} \end{cases} \tag{3.2}$$

- $\boldsymbol{p}^k \in \mathbb{R}^n$ is the *match between the input vector* $\boldsymbol{u}^k$ *and each memory vector* $\boldsymbol{m}^k{}_i$. $\boldsymbol{p}^k$ is defined as

$$\boldsymbol{p}^k{}_i = softmax(\boldsymbol{u}^{k^\top} \boldsymbol{m}^k{}_i) \tag{3.3}$$

- $\{\boldsymbol{c}^k{}_i\}$ is another representation of the context $U_1, ..., U_n$ defined by another embedding matrix $\boldsymbol{C}$, i.e., $\boldsymbol{c}_i^k = \sum_j \boldsymbol{C}^k x_{i,j}$.
- $\boldsymbol{o}^k$ is the memory layer's *output.* It is a sum over the transformed context $\{\boldsymbol{c}^k{}_i\}$ weighted by the probability vector $\boldsymbol{p}^k$ defined by the input $\boldsymbol{u}^k$, i.e., $\boldsymbol{o}^k = \sum_i \boldsymbol{p}^k{}_i \boldsymbol{c}^k{}_i$.

Note that each memory layer k is defined by three embedding matrices $\boldsymbol{A}^k, \boldsymbol{B}^k, \boldsymbol{C}^k \in \mathbb{R}^{d,V}$ where d, the size of the embedding, is a hyperparameter.

After all the K memory layers have computed their output, the model's prediction is a probability distribution over all the tokens in the vocabulary, remember here the answer is composed of one word only:

$$\hat{\boldsymbol{a}} = softmax(\boldsymbol{W}(\boldsymbol{o}^K)) \tag{3.4}$$

3.2 How to evaluate dialogs?

In the seminal paper by Alan Turing [27] it is proposed a game to evaluate a dialog system. The idea behind the game was simple: three players A, B and C can interact only by nonpersonal communication. C knows that he will interact with a dialog system (A) and a person (B), but C does not know which is which. C should exchange some conversation both with A and B; after some time C should guess who is the dialog system and who is the human. The goal of A is to be as human as possible in order to fool C; and the goal of B is to help C come to the right answer.

This is the famous *Turing Test*. If the dialog system wins this game we say that *it has passed the Turing Test*. For many people this is the holy grail of AI. Although this is an interesting test for a philosophical investigation, for many applications we do not want a dialog system to emulate an human being. We just want that the system solve a very specific task (booking an airplane seat, asking for user information, etc.). So we will avoid such complex tests and focus on more simple approaches.

3.2.1 Human evaluation

The most basic form to check if a generated sentence is sound is by using human evaluations. Hence is normal to see researches making use of crowd-sourced judges (using services like Amazon Mechanical Turk). These judges can, in an informal way, say that a generated sentence is just good or bad. But we can also use a more rigorous score system.

In [24] the authors enumerate three criteria to judge a generated sentence:

1. *Adequacy*: the meaning equivalence between the generated and control sentence.

2. *Fluency*: the syntactic correctness of the generated sequence.

3. *Readability*: efficacy of the generated sentence in a particular context.

Adequacy and fluency are obvious criteria, but readability is also important. Take, for example, the generated response "If you don't believe in god, then you don't know." for the question "What is the definition of altruism?". It is a syntactic correct answer that is completely out of context.

Regarding text generation, there should always be some kind of human evaluation, but an evaluation system that relies only on humans presents some severe disadvantages: i) the research results became non-reproducible; and ii) the costs for running experiments increases dramatically. Because of the problems presented in using human judges, you can find in the literature the use of automatic metrics from the field of machine translation and automatic summarization.

3.2.2 Automatic evaluation

BLUE

One popular metric for automatic translation is called BLUE (bilingual evaluation understudy) [20]. It compares n-grams of the candidate translation with the n-grams of the reference translation and counts the number of matches. Let x be a source sentence, $\hat{y}$ the hypothesis translation and y a reference translations produced by a professional human translator.

To define the BLUE score for each translation we need to define first the n-gram precision P_n:

$$P_n = \frac{\text{number of } n\text{-grams in both } \hat{y} \text{ and } y}{\text{number of } n\text{-grams appearing in } \hat{y}} \tag{3.5}$$

We also define a brevity penalty (BP) for translations that are too short. Let $len(a)$ be the length of the sentence a, we define BP as:

$$BP = \begin{cases} 1 & \text{if } len(\hat{y}) > len(y) \\ \exp\left(1 - \frac{len(y)}{len(\hat{y})}\right) & \text{otherwise} \end{cases} \tag{3.6}$$

Hence, the BLEU score is define as:

$$BLEU = BP \ \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log P_n\right) \tag{3.7}$$

There are some details about the definition above: to avoid computing $\log 0$ all precisions are smoothed to ensure that they are positive (one technique of smoothing is to use a small positive value to replace any $P_n = 0$); each n-gram in the candidate $\hat{y}$ is used at most once; and since it is expensive to calculate higher n-grams we usually take $N = 4$.

It is useful to see one application of this metric. For example, suppose s is the Portuguese sentence 'Nas tardes de fazenda há muito azul demais', y is the reference English translation 'In the farm's afternoons there is too much blue' and suppose we also have three models producing three distinct English translations

- $\hat{y}_1$: "In the afternoons of the farm there is too much blue"
- $\hat{y}_2$: "In the farm's afternoons there is a lot of blue"
- $\hat{y}_3$: "The farm's afternoons are blue"

The table below shows all the values to compute the BLUE score for this example.

Sentence	P_1	P_2	P_3	P_4	BP	$BLUE$
$\hat{y}_1$	8/11	5/10	3/9	2/8	1	.42
$\hat{y}_2$	7/10	5/9	4/8	3/7	1	.52
$\hat{y}_3$	3/5	1/4	0	0	.45	.28

Figure 3.2: Example of calculating P_n, BP and $BLUE$ score

METEOR

METEOR (Metric for Evaluation of Translation with Explicit ORdering) was created to address the weakness in BLUE [11]. This metric is based on an alignment between unigrams from the hypothesis translation $\hat{y}$ and unigrams from the reference translation y. This alignment can be based on exact matches, matching between stemmed unigrams, or matching by synonym.

After the alignment is done we compute the unigram precision P and unigram recall R as:

$$P = \frac{\text{number of unigrams in both } \hat{y} \text{ and } y}{\text{number of unigrams appearing in } \hat{y}} \tag{3.8}$$

$$R = \frac{\text{number of unigrams in both } \hat{y} \text{ and } y}{\text{number of unigrams appearing in } y} \tag{3.9}$$

We them combine these values using the harmonic mean:

$$F_{mean} = \frac{10PR}{R + 9P} \tag{3.10}$$

To take into account longer matches, this metric computes a penalty for the "chunkiness" of the alignment. The idea is to group the unigrams in fewest possible chunks. Here chunk is defined as a set of unigrams that are adjacent in $\hat{y}$ and in y. Hence the penalty is defined as

$$penalty = 0.5 \left(\frac{c}{um} \right)^3 \tag{3.11}$$

where c is the number of *chunks* and um is the number of matched unigrams. Hence the METEOR score is defined as

$$METEOR = F_{mean}(1 - penalty) \tag{3.12}$$

ROUGE

ROUGE stands for (Recall Oriented Understudy for Gisting Evaluation), it is a set of metrics for automatic summarization [32]. Following the dialog generation literature [14], we will consider only the $ROUGE_L$ metric. This one is just an F-measure based on the Longest Common Subsequence (LCS), i.e., the longest non-contiguous set of words that occurs in two sentences in the same order. So, let $lcs(a, b)$ be the lenght of the LCS between the sentences a and b, the $ROUGE_L$ metric is defined as follows:

$$P_{lcs} = \frac{lcs(\hat{y}, y)}{len(\hat{y})} \tag{3.13}$$

$$R_{lcs} = \frac{lcs(\hat{y}, y)}{len(y)} \tag{3.14}$$

$$ROUGE_L = \frac{(1 + \beta^2) P_{lcs} R_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \tag{3.15}$$

where β is usually set to favour recal ($\beta = 1.2$).

3.2.3 The mismatch between human and automatic evaluation

Complaints about the mismatch between the automatic metrics and the human judgments are not new. In [24] the authors have observed that i) there are positive, but not strong, correlations between the scores of the automatic evaluation metrics and human judgments of adequacy; and ii) there are negative correlations between the scores of the automatic metrics and human judgments of fluency. As can be seen on the table below:

Automatic metric	Adequacy	Fluency
BLEU	0.39	-0.49

Figure 3.3: Correlation between $BLUE$ score and human judgments of adequacy and fluency (extract from [24])

In a more recent paper [13], the authors have trained two neural generative models for dialog (a LSTM language model and the HRED model) on two different datasets: the chit chat oriented Twitter dataset and the technical Ubuntu Dialogue Corpus.

After the models were training they were used to generate some sentences, these sentences were evaluated using both the automatic metrics and human judgments.

The main experiment of the paper was to test for significance of correlation between each automatic metric and human judgments. The tables below show the results from this experiment:

metric	Spearman	p-value	Pearson	p-value
BLEU	0.34	< 0.01	0.14	0.17
METEOR	0.19	0.06	0.19	0.05
ROUGE	0.12	0.22	0.1	0.34

Table 3.1: Correlation between automatic metrics and human judgments based on dialog generated on Twitter (extract from [13])

What this paper indicates is that these metrics correlate very weakly with human judgements in the non-technical Twitter domain, and not at all in the technical Ubuntu domain. These results do not show that automatic metrics

metric	Spearman	p-value	Pearson	p-value
BLEU	0.12	0.23	0.11	0.26
METEOR	0.06	0.53	0.14	0.16
ROUGE	0.05	0.59	0.06	0.53

Table 3.2: Correlation between automatic metrics and human judgments based on dialog generated on Ubuntu (extract from [13])

should be avoided at all cost, but they indicate the danger of relying solely on those metrics.

3.3 Creating simplified tasks as tests

Given all these limitations, one interesting strategy proposed in [31] is to create a set of QA synthetic tasks to test different capabilities of a dialog agent. QA tasks are easy to evaluate, so the difficulty lies on creating the right tasks. They should be simple, diverse, self-contained and relevant. At the same time, any adult person with no specialized knowledge should be able to achieve 100% accuracy on each task.

With that purpose the authors of [31] have created a set of 20 simple tasks, they believe that those tasks are a "prerequisite to full language understanding and reasoning". These tasks were called the bAbI dataset. Without entering in the details of each task, it is sufficient to know that each task was designed to cover a variety of skills: from counting to positional reasoning. Here are some examples:

Task 1: Simple Supporting Fact

Mary went to the bathroom.

John moved to the hallway.

Mary travelled to the office.

Where is Mary? A: office

Task 7: Counting

Daniel picked up the football.

Daniel dropped the football.

Daniel got the milk.

Daniel took the apple.

How many objects is Daniel holding? A: two

By running a set of experiments with different models they observed that by using a variation of the memory network they achieve 95% accuracy in most of the tasks (the mean performance was 93%).

Here we will focus only in how the *logic relations* were formulated in this framework and how they can be expanded.

3.4 The logical entailment problem inside the bAbI tasks

In [31], among a variety of useful tasks the authors introduce two tasks that deals with logical inference – Task 15: *basic deduction* and Task 16: *basic induction*. The basic deduction task offers questions of the form:

P^1 are afraid of Q^1
P^2 are afraid of Q^2
P^3 are afraid of Q^3
P^4 are afraid of Q^4
c^1 is a P^1
c^2 is a P^2
c^3 is a P^3
c^4 is a P^4
What is c^j afraid of? A: P^j

where P^j and Q^j are animals (e.g., "cats", "mice", "wolfs", etc.) and c^j are names (e.g., "Jessica", "Gertrud", "Emily", etc.). The underlying relation under focus here is the *membership* relation: if Emily is a cat and cats are afraid of wolfs then Emily is afraid of wolfs. This task is solvable by the current models, in [31] the best accuracy for this task is 100%.

The basic induction task is composed by questions of the form:

c^1 is a P^1
c^1 is C^1
c^2 is a P^2
c^2 is C^2
c^3 is a P^3
c^3 is C^3
c^4 is a P^4
c^4 is C^4
c is a P^j
What color is c? A: C^j

Where P^j is a animal, C^j is a color, and c^j is a name. Here the agent is asked to remember the relation between animal and color, and when presented a new name of an animal in the example, the agent should infer the color using past examples. Similar to the deduction task, in [31] is reported that this task is completely solved.

3.5 Entailment-QA

These two tasks from the bAbI dataset present a first step towards *a complete set of tasks to tests inference capabilities.* One aspect that is missing though is the use of logical operators such as *boolean connectives* and *first-order quantifiers.* Those operators play a big role on everyday speech, hence a natural way of expanding this work is by creating a set of tasks that make agents learn *valid logic structures.* In this section we will give the details for such tasks.

To highlight the speech structure we will make use of an artificial language, but this is just an exposition tool. We are concerned in logical structures *only used in everyday speech.*

Task 1: Boolean Connectives. The first task is focused only on some propositional connectives $\wedge$ (and), $\vee$ (or), $\neg$ (not). The agent is given two sentences s_1 and s_2, and he is asked if s_1 entails s_2 or not (a yes/no question). The position of the sentences is important here. We look at only six general cases:

- Entailment
  - $\underbrace{P^1a^1 \wedge \cdots \wedge P^na^n}_{s_1}, \underbrace{P^ja^j}_{s_2}$

  - $\underbrace{P^j a^j}_{s_1}, \underbrace{P^1 a^1 \vee \cdots \vee P^n a^n}_{s_2}$

  - $\underbrace{Pa}_{s_1}, \underbrace{\neg\neg Pa}_{s_2}$

- Not entailment

  - $\underbrace{P^j a^j}_{s_1}, \underbrace{P^1 a^1 \wedge \cdots \wedge P^n a^n}_{s_2}$

  - $\underbrace{P^1 a^1 \vee \cdots \vee P^n a^n}_{s_1}, \underbrace{P^j a^j}_{s_2}$

  - $\underbrace{Pa}_{s_1}, \underbrace{\neg Pa}_{s_2}$

Where P is a predicate and a is a name. The point here is not to demand that the agent learn complex logical forms, but to recognize which forms are sound and which are not. This should be achieved independently from the content, i.e., the different predicates and names that appear on the sentences. So from the abstract forms above we have only simple examples like:

> Ashley is fit
>
> Ashley is not fit
>
> The first sentence implies the second sentence? A: no
>
> Avery is nice and Avery is obedient
>
> Avery is nice
>
> The first sentence implies the second sentence? A: yes
>
> Elbert is handsome or Elbert is long

Elbert is handsome

The first sentence implies the second sentence? A: no

Task 2: First-Order Quantifiers. Task 2 tries to capture the use of some basic *quantifiers*. In this dataset we are trying to predict the entailment relationship between two sentences s_1 and s_2. We say that there is an *entailment* relationship if s_1 implies s_2, we say that there is a *contradiction* if the combination of sentences s_1 and s_2 implies an absurdity and if the combination of sentences does not imply an absurdity we say that they are *neutral*. We have used six forms of logical relations for the quantifiers $\forall$ (for every) and $\exists$ (exists):

- Entailment
  - $\forall x Px, Pa$
  - $Pa, \exists x Px$
- Contradiction
  - $\forall x Px, \neg Pa$
  - $\forall x Px, \exists x \neg Px$
- Neutral
  - Pa, Qa
  - $\forall x Px, \neg Qa$

where P and Q are non-related predicates and a is a name. So we have examples like:

Every person is lively

Belden is lively

What is the semantic relation? A: entailment

Every person is short

There is one person that is not short

What is the semantic relation? A: contradiction

Every person is beautiful

Abilene is not blue

What is the semantic relation? A: neutral

The tasks above force the dialog system to predict entailment independently of the specific meaning of nouns, verbs and adjectives presented on speech.

We have created a first version of tasks 1 and 2, now we intend to design four more tasks centered on *generic semantic knowledge*.

Task 3: Synonymy. This task tests if a dialog system can identify paraphrase caused by synonym use. Two supporting facts are presented, the second differs from the first by one noun, verb or adjective that can be a synonym or not, e.g., "*The girl is talking into the microphone. The girl is speaking. Are the above sentences duplicate?*".

Task 4: Antinomy. This task consists of recognizing contradictions by antinomy use. For example, the question "*John is not an old person. John is young. Are the above sentences a contradiction?*" should be answered "no" and the question "*Susan is happy. Susan is sad and she is crying. Are the above sentences a contradiction?*" should be answered "yes".

Task 5: Hypernymy. When one term is a specific instance of another we say that there is a hypernymy relationship between then. For example, we say that "anthem" is a hyponym and "song" a hypernym, because anthem is a kind of song. Task 5 tests the kind of linguistic entailment caused by the use of hyponyms and hypernyms, e.g., "*A woman is eating an apple. A woman is eating a fruit. Are the above sentences duplicate?*"

Task 6: Active/Passive voice. Finally the last task is centered on the paraphrases originated by the changes from active to passive voice. So two supporting facts are presented, although they have a different syntactic structure, they share the same meaning. For example, "*A man is playing the piano. The piano is being played by a man. Are the above sentences duplicate?*".

It should be noted that we can formulate the notion of paraphrase as an entailment relation: if s_1 is a paraphrase of s_2, it is natural to say that s_1 implies s_2 and vice-versa. This is done in [15]. Although this approach presents no problem it can alienate some people from the machine learning community: this community normally deals with problems of similarity between sentences, there is very little datasets available centered on the notion of entailment. So although we have mentioned only paraphrase detection in tasks 3–6 the entailment relationship is present.

3.6 Preliminary Results

To organize all experiment in an unified framework we decided to use the platform ParlAI [19]. Our criteria for a *semantic robust dialog agent* is not an agent that is only tuned for the Entailment-QA tasks mentioned above, but an agent that performs well across all established tasks (like the bAbI tasks [31]) and performs reasonably on the Entailment-QA tasks 1-6. Here "reasonably" means that the agent should exceed a random agent by a significant margin.

Right now we are in the process of creating the dataset. We have already built tasks 1 (boolean connectives) and 2 (first order quantifiers). They both are composed of 10000 questions for training and 1000 for testing.

We have used the dataset SICK (Sentence Involving Compositional Knowledge) [15] as a proxy for tasks 3-6, since all the structures presented on those tasks are presented in this dataset. The SICK data is composed of a pairs of sentences and a label describing the entailment relation between the sentences. To cast this classification dataset as a QA problem we have added a question and maintained the labels:

A group of people are marching

A group of people are walking

What is the semantic relation? A: entailment

There is no dog leaping in the air

A dog is leaping high in the air and another is watching

What is the semantic relation? A: contradiction

A man is exercising

A baby is laughing

What is the semantic relation? A: neutral

Some dogs are playing in a river

Some dogs are playing in a stream

What is the semantic relation? A: entailment

This QA task is composed of 23000 questions for training and 5900 for testing.

We have performed the first experiments using the seq2seq model (with and without attention: Seq2seq and Seq2seqAtt, respectively) and the memory network model (MenNN).

So far these models show unsatisfactory results, as can be seen in Figure 3.4.

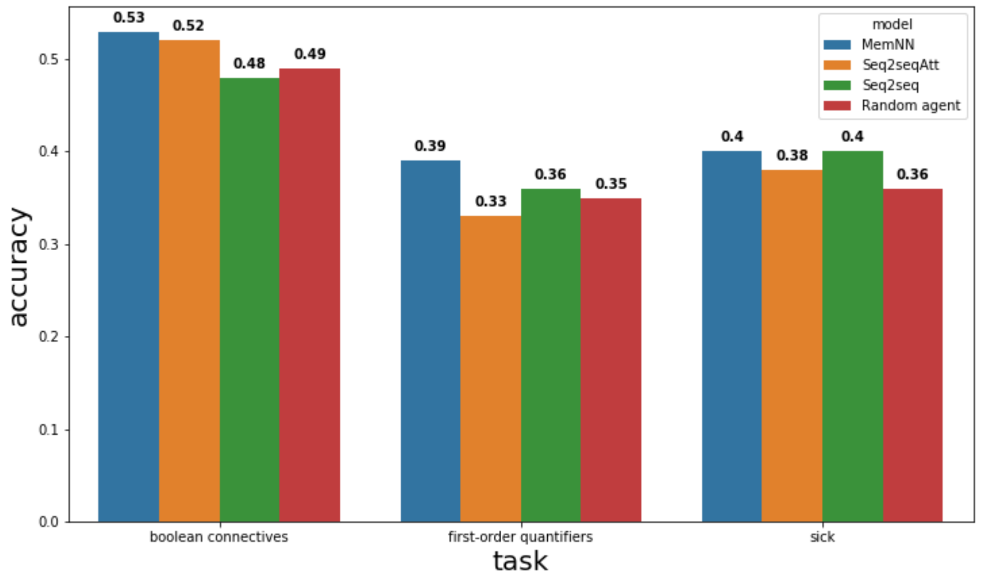

Figure 3.4: Accuracy results for the Entailment-QA tasks

Their overall performance is only slightly better when compared to the random agent. For example, when we look at the memory model, the model that often outperform the seq2seq model on QA tasks [31], we can see that regarding tasks 1 and 2 there is an overfitting problem: the model shows high accuracy on the train dataset (75% accuracy on task 1, and 86% accuracy on task 2) as can be seen in Figure 3.5. But when we take a closer look at the confusion matrix of the test data, Figure 3.6, the results are not impressive.[1]

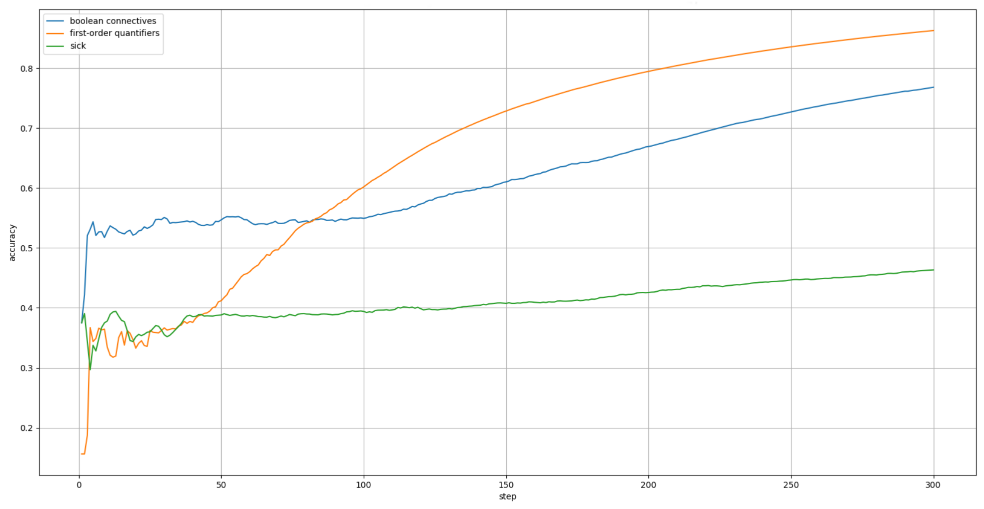

Figure 3.5: Training accuracy for the memory network

The results so far may seen grim, but they show also a positive side: *the Entailment-QA tasks are not a set of trivial tasks that can be completely solved by the current models.* These results indicated that it is worthwhile to explore this set of task, either by improving the training using the current models or by exploring new kinds of models. One thing should be clear, we are not concern in obtaining high accuracy on these tasks only (just as a comparison, in [10] is reported an accuracy of 87% on the SICK dataset by using feature engineering techniques). The main idea is to use an end-to-end model to obtain good results in the Entailment-QA task without compromising the accuracy on other well established QA tasks.

[1]All the experiments and the respective results are available on GitHub: `https://github.com/felipessalvatore/DialogGym`.

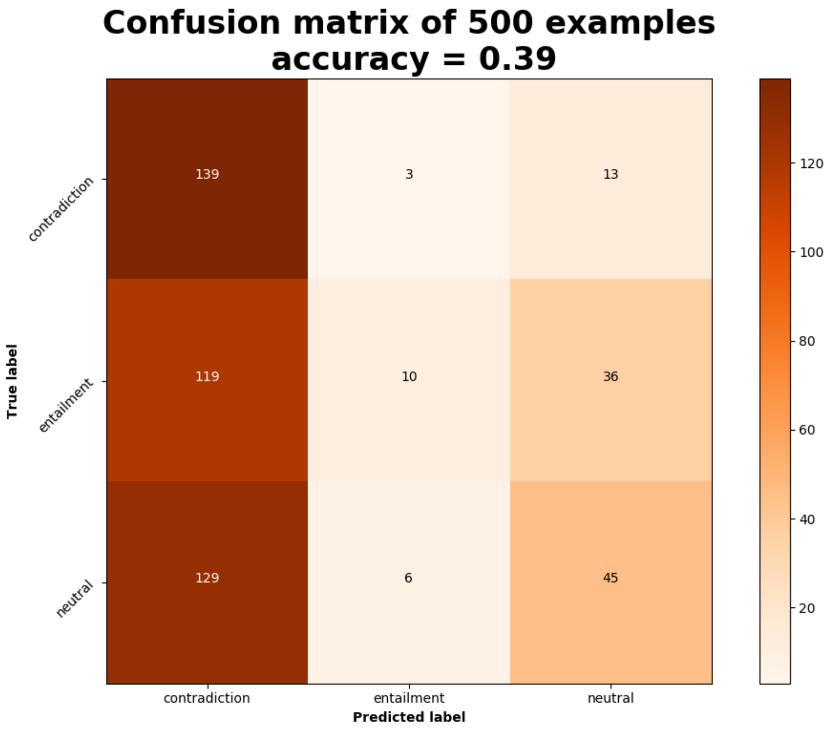

Figure 3.6: Confusion matrix for the memory network on task 2

Chapter 4

Future Steps

The previous chapters resume what we have done so far: we grasp the theoretical framework to formulate this NLP problem; we reviewed the literature on dialog generation; we built a software workbench to perform different experiments; and we have isolated a specific problem not sufficiently addressed in the literature: logical reasoning for dialog agents.

In July we will present this work at one summer school organized by the company DeepMind in Europe `https://tmlss.ro/` and so we expect to gather more feedback on our research.

To address our research proposal we decided to formulate the following next steps:

- Apply regularization strategies on the available models to overcome the reported overfitting problem.
- Finish the Entailment-QA corpus to have a fine grain analysis of the result that we are seeing on the SICK corpus.
- Explore the different extensions for all mentioned models.
- Explore new models not mentioned here, like Dynamic Memory Networks [9] and the models using the Memory Attention and Composition (MAC) cell [8].
- Create a visual version of the Entailment-QA to test logical inference with images.

- There is a different literature that frames the dialog problem as an MDP (Markovian Decision Process) and a POMDP (Partially Observable Markovian Decision Process) applying different techniques of reinforcement learning (a recent example is [12]). It is fruitful to investigate if these techniques can help our research.

- One of the main focused here is model comparison. It would be fruitful if we could use the available literature on the theory of comparing models (e.g., [1]) to refine our analysis.

4.1 Work Plan

Here, we use a visual tool to display the scheduling of future and past activities. This serve as a sanity check to verify if the proposed goals can be realistically achieved.

Activity	2016		2017		2018		2019		2020
	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st
Courses	🟩	🟩	🟩	🟩					
Teaching Assist. (PAE)					🟩				
Bibliographic Review	🟩	🟩	🟩	🟩	🟩				
Software Implementation				🟩	🟩	🟦	🟦	🟦	
Qualification Writing					🟩				
Qualification Exam					🟩				
Finishing Entailment-QA task						🟦			
Visual Entailment-QA task						🟦	🟦		
Improve Training						🟦			
Adding new models						🟦	🟦		
Reinforcement Learning Methods							🟦	🟦	
Model Comparison Theory							🟦	🟦	
Thesis Writing								🟦	
Thesis Defense									🟦

Table 4.1: Schedule for the PhD program. Completed activities are shown in green, while future activities are in blue.

Bibliography

[1] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *Journal of Machine Learning Research*, 18:77:1–77:36, 2017.

[2] A. Bordes and J. Weston. Learning end-to-end goal-oriented dialog. *CoRR*, abs/1605.07683, 2016.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

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

[5] Y. Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2017.

[7] K. Heafield. Scalable modified kneser-ney language model estimation. In *In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.

[8] D. A. Hudson and C. D. Manning. Compositional attention networks for machine reasoning. *CoRR*, abs/1803.03067, 2018.

[9] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, abs/1506.07285, 2015.

[10] A. Lai and J. Hockenmaier. Illinois-lh: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 329–334. Association for Computational Linguistics, 2014.

[11] A. Lavie and A. Agarwal. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 228–231, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[12] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky. Deep reinforcement learning for dialogue generation. *CoRR*, abs/1606.01541, 2016.

[13] C. Liu, R. Lowe, I. V. Serban, M. Noseworthy, L. Charlin, and J. Pineau. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *CoRR*, abs/1603.08023, 2016.

[14] R. Lowe, M. Noseworthy, I. V. Serban, N. Angelard-Gontier, Y. Bengio, and J. Pineau. Towards an automatic turing test: Learning to evaluate dialogue responses. *CoRR*, abs/1708.07149, 2017.

[15] M. Marelli, L. Bentivogl, M. Baroni, R. Bernardi, S. Menini, and R. Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. *SemEval 2014*, 2014.

[16] J. McCarthy and P. J. Hayes. Readings in nonmonotonic reasoning. chapter Some Philosophical Problems from the Standpoint of Artificial Intelligence, pages 26–45. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[18] T. Mikolov, S. Kombrink, L. Burget, J. Cernocký, and S. Khudanpur. Extensions of recurrent neural network language. *IEEE*, pages 5528–5531, 2011.

[19] A. H. Miller, W. Feng, A. Fisch, J. Lu, D. Batra, A. Bordes, D. Parikh, and J. Weston. Parlai: A dialog research software platform. *CoRR*, abs/1705.06476, 2017.

[20] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics, 2001.

[21] I. V. Serban, R. Lowe, L. Charlin, and J. Pineau. Generative deep neural networks for dialogue: a short review. *CoRR*, abs/1611.06216, 2016.

[22] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth AAAI Conference of Artificial Intelligence*, AAAI'16, pages 3776–3783. AAAI Press, 2016.

[23] Y. Shao, S. Gouws, D. Britz, A. Goldie, B. Strope, and R. Kurzweil. Generating high-quality and informative conversation responses with sequence-to-sequence models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2200–2209, 2017.

[24] A. Stent, M. Marge, and M. Singhai. Evaluating evaluation methods for generation in the presence of variation. *In Computational Linguistics and Intelligent Text Processing*, 2005.

[25] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks. *CoRR*, abs/1503.08895, 2015.

[26] I. Sustskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, 2014.

[27] A. Turing. Computing machinery and inteligence. *Mind*, pages 433–460, 1950.

[28] O. Vinyals and Q. V. Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015.

[29] J. Weizenbaum. Eliza&mdash;a computer program for the study of natural language communication between man and machine. *Commun. ACM*, pages 36–45, 1966.

[30] T. Wen, M. Gasic, N. Mrksic, L. M. Rojas-Barahona, P. Su, S. Ultes, D. Vandyke, and S. J. Young. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*, abs/1604.04562, 2016.

[31] J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

[32] C. yew Lin. Rouge: a package for automatic evaluation of summaries. pages 25–26, 2004.