Question Dependent Recurrent Entity Network for Question Answering

Candidate: **Andrea Madotto**

Supervisor: **Prof. Giuseppe Attardi**

Department of Computer Science
University of Pisa

This dissertation is submitted for the degree of
Master Degree in Computer Science

University of Pisa July 2017

Acknowledgements

A special thank goes to my family which always supported me, financially and mentally. Then, I would like to thank all of my friends. In particular my two flatmate Luca and Stefano. They have helped me a lot, in so many different ways that it would require an entire chapter to explain it, really thank you. And of course, I would like to thank my girlfriend, since she has been very closed to me in the last year.

Then, I would like to thank Antonio Carta, Federico Errica, and Ludovica Pannitto for the great collaboration we had in SemEval2017. Last but not least, I would like to express my most sincere gratitude to my supervisor, Prof. Giuseppe Attardi, for his knowledgeable guidance and supervision for my research work. I have learned a lot through interactions with him.

Finally, I would like to thank the University of Pisa that provided funding under the initiative Grandi Attrezzature 2016, that was used with partial co-founding from a group of professors of the Dipartimento di Informatica (Giuseppe Attardi, Marco Danelutto, Alessio Micheli, Dino Pedreschi, Rossano Venturini) to purchase a server equipped with 4 Nvidia Pascal 100 GPUs, that was used in running the experiments for this thesis.

Abstract

Question Answering is a task which requires building models that are able to automatically reply to questions given by humans. In the recent years, growing interest has been shown in tasks that require reasoning abilities in order to answer questions.

Thus, in this study, we propose an extensive literature review of the state-of-the-art, which is followed by the introduction of two models to accomplish different Question Answering tasks, which are: Community Question Answering, Reasoning Question Answering and Reading Comprehension.

Specially, we analysed and improved a novel model called Recurrent Entity Network, which follows the Memory Network framework, and it has shown a great potential in solving reasoning tasks. We named our model Question Dependent Recurrent Entity Network since our main contribution is to include the question into the memorization process.

Then we proposed a novel model called ThRee Embedding Recurrent Neural Network which has been used for the Community Question Answering task. In this case, we tried to embed information of a Dependency Parser in order to create an enhanced representation of inputs sentences.

All of our models have been validated by using both synthetic and real datasets. For the best of our knowledge, we achieved a new state-of-the-art in the Reasoning Question Answering task, and very promising results in Reading Comprehension and Community Question Answering ones.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

λ	Regularization Parameter
σ	Sigmoid Function
$L1$	Norm 1 Regularization Term
$L2$	Norm 2 Regularization Term
Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
BiGRU	Bi-directional GRU
BiLSTM	Bi-directional LSTM
BiRNN	Bi-directional RNN
BPTT	Backpropagation Through Time
CQA	Community Question Answer
Dr	Dropout
GRU	Gated Recurrent Unit
IR	Information Retrieval
LR	Learning Rate
LSTM	Long Short Term Memory
MAP	Mean Average Precision

NLP	Natural Language Processing
PCA	Principal Component Analysis
QDREN	Question Dependent Recurrent Entity Network
RC	Reading Comprehension
ReLU	Rectified Linear Unit
REN	Recurrent Entity Network
RNN	Recurrent Neural Network
RQA	Reasoning Question Answer
SGD	Stochastic Gradient Descent
SOTA	State-Of-The-Art
SVD	Singular Value Decomposition

Chapter 1

Introduction

Since the last two decades, people have started to interact with the machine in various ways, such as by using voice, text, physical touch etc. The public has a very high expectation on what a machine can achieve, and how it can react to human inputs. This is mostly because the current generation has been exposed to many spectacular Sci-Fi movies which idealise the power of an Artificial Intelligence [70].

The most primitive way to interact with machines was typing, and at the beginning, the only way was through machine language (e.g. assembly). As years go by, this interaction has become way easier mostly thanks to the high-level programming language and consequently by the introduction of user interfaces. Nowadays, instead, it is common to input a query in natural language (e.g. 'How is the weather in Rome?') to a searching engine (e.g., Google, Yahoo etc.) and obtain a correct answer.

In recent years, scientists have also explored the possibility of using voice as a more natural mean of input. Many companies like Google, Apple, Microsoft and Amazon, released their personal voice assistance which are able to capture requests of information or actions upon user requests. The techniques to solve such tasks use both speech recognition and text understanding, in order to translate from voice to text and to extract meaningful information from it. Researchers developed speech recognition algorithms since the 80s, and today the technology is already quite mature [42]. On the other hand, text understanding is still an open research problem.

Indeed, the ability to understand and process text is very challenging for a machine, in particular when the input is a human language text, thus very unstructured. Natural Language Processing (NLP) is the application of computational models aiming to overcome such difficulties. This topic is an active research area since 1950[1]. Some tasks in this field are still open research problems, for instance, some of the most important ones are: Part-Of-Speech Tagging, Chunking, Semantic Role Labelling, Named Entity Recognition, Tokenization (lexical analysis), Text Segmentation, Language Models, Machine Translation, and Question Answering.

Question Answering, for instance, represents one of the most interesting and challenging task in the NLP field[2]. This task requires building a system, which is able to automatically reply to question proposed by humans in natural language. As you can imagine, this task is very general and, absurdly, it can solve all the previous listed tasks. For example, a question could be: “Can you translate ‘Ciao mi chiamo Andrea’ in Chinese?” (Machine Translation), or “Could you find the entities inside the sentence ‘Frank is smart’?” (Named Entity Recognition). Obviously, with the current technology and the existing amount of data, such general question answering tasks are still impossible.

However, a specialisation of question answering task can be done. For instance, a task that specifies a piece of text (e.g., newspaper article) and proposes a question that can be answered directly with the information in the text (without external source). Even though this task may seem simple for a human, it is very hard for machines[3]. Indeed, both input text and question are given in natural language, which can be very ambiguous, and in many cases, it also requires to understand the underneath meaning of its content. The latter ability is not easy to acquire since it can involve a reasoning phase (chaining facts, basic deductions, etc.). Human has got this ability almost for ‘free’, instead, a machine can find it very challenging.

Recent progress in this field has been possible thanks to machine-learning algorithms [41] which have the ability to automatically learn through data. In particular, Deep Learning [49] techniques, which are Artificial Neural Networks (ANN), made a breakthrough thanks to the great amount of data available nowadays. Indeed, most of the techniques used today were already available since 2/3 decades ago, but they were not widely used since there were not enough data and processing power.

The mentioned techniques have shown great potential in solving language processing tasks, including Question Answering, but they do not always achieve human level accuracy. There-

[1]An interesting reference for the NLP history can be found in [40]

[2]It is also a well-known Information Retrieval task, however, our focus is more in the linguistic aspect of the input, rather than building a data structure to efficiently retrieve data.

[3]Sometimes also for humans, for example when the text is very long or the questions are very technical.

fore, in this thesis: (1) we first study the State-Of-The-Art in different Question Answering Tasks, such as Question Comment Similarity, Reasoning Task, and Reading Comprehension; (2) secondly, we propose two novel models able to solve the mentioned tasks achieving promising results.

1.1 Problem Statements

In this thesis, as mention before, we handled different kinds of Question Answering tasks. A description of such tasks is given in the following.

- **Community Question Answering** (CQA) offers new opportunities for users to provide, search and share knowledge (wiki). There, users post open questions and expect that someone in the internet community will answer properly. However, these forums are self-moderated and they do not always propose the best answer. The task is to automate the process of finding good answers among a group of 10 comments in the post (candidates).
- **Reasoning Question Answering** (RQA) is a task to test the ability of a machine to accomplish reasoning on a text. Differently, from a pure logic inference, facts and questions are given in Natural Language, and sometimes there are not enough information to reach a logical conclusion. However, human beings are still able to solve these tasks easily. The task is to search the answer of a given question among all the possible words presented in the text.
- **Reading Comprehension** (RC) is the ability to read a text, process it, and understand its meaning. This task is very close to the previous since it is tested by asking a question about facts in the text. Normally, the text to read can be very long and heterogeneous, thus quite difficult to process. As for the previous task, the possible candidates are words in the paragraph.

Note that all of these tasks do not use external resources, indeed the inference is done just by using the information presented in the text. On one hand, this increases the task difficulty since the given information to correctly answer the question may be very limited. On the other hand, it is possible to test the ability of a machine to automatically accomplish tasks, that humans are able to solve “easily”.

1.2 Our Contribution

In this thesis, we proposed two models, which are: ThRee Embeddings Recurrent Neural Network [2], used for the Community Question Answering task, and the Question Dependent Recurrent Entity Network[4], used for both Reasoning Question Answering and Reading Comprehension tasks. The first model has been proposed to participate at SemEval2017 [5], an international workshop on semantic evaluation, where the CQA task has been proposed. The second, instead, is an improvement of a recently proposed model called Recurrent Entity Network [32]. In the latter one, we redesign the core component of the model, and we improve the network infrastructure by adding a tighter regularisation scheme and by modifying several activation functions. These allowed us to achieve good results in both RQA and RC tasks.

1.3 Outline

The rest of the thesis is organised as follows: Chapter 2 introduces all the needed background, including a general introduction to Natural Language Processing and the techniques used in the field (Artificial Neural Networks, Word Embedding, etc.). Chapter 3 describes the State-Of-The-Art Techniques for the studied Question Answering tasks, in particular, we describe several benchmark datasets and results. Chapter 4 presents our two proposed models, in particular, we describe their mathematical formulation. Chapter 5 reports experimental results of our models in several benchmark datasets. Finally, Chapter 6 includes a short summary of the presented topics and future works.

[4]An implementation is available at https://github.com/andreamad8/QDREN
[5]http://alt.qcri.org/semeval2017/task3/

Chapter 2

Background

In this Chapter, we describe several Natural Language Processing (NLP) tasks and well-known methods to handle them. In particular, we are going to introduce techniques and terminology used in the NLP field. The level of details that is applied in this section is proportional to the pertinence with our work. In many cases we simply mention methods without going into many details, instead, in others (e.g. Word Embedding) we are giving a more detailed explanation.

2.1 NLP Tasks

The classical NLP tasks analyse both syntactic and semantics information of a text. Some of those have got a real word application, instead, others are used as a tool to solve more complicated tasks. In the following, we shortly describe the most common tasks, which we divide into two categories: Syntactic and Semantic.

2.1.1 Syntactic Tasks

The Syntactic tasks refer to the syntax of a sentence. Often, these tasks are related to the language used (e.g., Italian, English etc.), and they require a lot of linguistic knowledge. In this category, the major tasks are:

- **Stemming** the process to reduce a certain word to its word stem. For example: "training", "trained", and "trainer" are all reduced to "train".

- **Part-Of-Speech Tagging** (POS) aims at labelling each word with a unique tag that indicates its syntactic role, e.g. plural noun, adverb, and so on.

- **Parsing** is the process to determine the parse tree of a given sentence. Differently from programming languages, natural language has got a very ambiguous grammar, and indeed normally there are many parse trees, which represent the same sentence. In literature, two parsers have been commonly used: (1) Dependency parser, which connects words according to their relationships (e.g., is -> SUBJ -> It); (2) Constituency parser, on the other hand, breaks the text into sub-phrases and each internal node (Non-Terminal) describes the rule of such node (e.g., Verb Phrase, Noun Phrase, etc.). A more detailed explanation of such parsers is given in the next section since one of them has been used in our proposed model.

- **Chunking** also called shallow parsing, aims at labelling segments of a sentence with syntactic constituents such as noun or verb phrases (NP or VP). Each word is assigned to only one unique tag, often encoded as a begin-chunk (e.g. B-NP) or inside-chunk tag (e.g. I-NP).

- **Sentence breaking** divide a given chunk into sentences. Normally a period is used to mark the sentence end, but the same punctuation can be used elsewhere (e.g. marking abbreviations).

2.1.2 Semantic Tasks

The Semantic tasks are designed to test the ability of a machine in understanding the meaning of a text. In the following, we will shortly describe the most common tasks. Notably, there are many intersections between tasks even though we will describe them separately.

- **Named Entity Recognition** (NER) labels atomic elements in the sentence into categories such as “PERSON” or “LOCATION”. As in the chunking task, to each word is assigned a tag prefixed by an indicator of the beginning or the inside of an entity.

- **Natural Language Understanding** tries to understand the intended meaning, from the multiple possible semantics, of a sentence (or a paragraph). This task is also known as machine reading comprehension, and it is considered very difficult since it requires a phase of reasoning to fully understand the underline meaning of a given sentence.

- **Natural Language Generation** generates natural language text from a machine representation (e.g., database or logical proposition). This task normally requires learning a generative model able to reproduce natural language sentence. Its applications are unlimited (e.g., Joke generators [7]), and it is an open research area.

- **Machine Translation** automatically translates text from a language A (e.g., Italian) to a different language B (e.g., French). This was one of the first tasks to be proposed (on the 60th) and it is still one of the most difficult NLP tasks since it requires both text understanding and generation.
- **Automatic Summarization** produce a piece of text which summarises a given paragraph. In this case, a lot of gold labels (human annotated examples) are needed. But this task has a huge impact on the society since it can be very useful in many real word applications.
- **Sentiment Analysis** (a.k.a. opinion mining), extracts subjective information from a given paragraph, normally expressed with a coefficient called "polarity". The level of understanding can describe an emotional state, such as "angry", "sad", and "happy". The ability to understand opinion (sentiment) has been used to predict trends in public opinion, using social network platform (e.g., Facebook or Twitter).
- **Textual Entailment** takes two text in input, a text (t) and hypothesis (h), and decide whether t entails h. For example, a text like " A soccer game with multiple males playing" and a hypothesis like "Some men are playing a sport" is an entailment. There can be also negative and neutral entailment.
- **Question Answering** tries to create an application, which takes a question in natural language as input and it is able to answer it. This task is very general and it can theoretically solve all the other ones. For instance, when you ask to translate a sentence, you are actually solving a Machine Translation task. Normally both Information Retrieval (IR) and NLP are involved in solving this kind of task. The first (IR) is specialised in open domain question answering, and the second, instead, work in linguistic/semantical tasks. However, many differences between the two fields are present, in particular when the question is proposed in natural language rather than an explicit query.

In this study, we focus on Question Answering Tasks. In particular, the main tasks that we are going to solve are three: *Community Question Answering*, *Reasoning Question Answering* and *Reading Comprehension*. The first is related to Question Comments similarity, instead, the latter two, get a paragraph and a question in as input, and they produce an answer using a single (or multiple) word(s) from the vocabulary.

2.2 Techniques

In this section, we describe several techniques used to solve the aforementioned tasks. It is worth to mention that we are not going to show all the existing methods (e.g., Rule-based, Dictionary-based, SVM, etc.), but just methods based on Artificial Neural Network. We made this choice since in the recent years the State-Of-The-Art in Natural Language Processing tasks has been reached using such methods.

Moreover, Artificial Neural Networks have been a well-known tool for 30 years, but they were not so popular since there were not enough data and computational resources to make them work properly. Just recently, thanks mostly to the Internet, it has been possible to have access to millions of records (documents, images, and any kinds of digital data). Then, in combination with the progress of the GPUs (Graphical Process Units), it has been possible to overcome both limitations. Indeed, the combination of the latter two facts made possible to train models with billions of parameters using very large datasets.

In the following paragraphs, we first describe the standard Artificial Neural Networks models. Then, we introduce the concept of Language Model which is going to be used for the creation of Word Embedding. The latter is used as a building block in any NLP technique. Especially, we are going to describe their construction using three distinct methods. Finally, we introduce the Recurrent Neural Network model which is used to process sequence of words (or sentences).

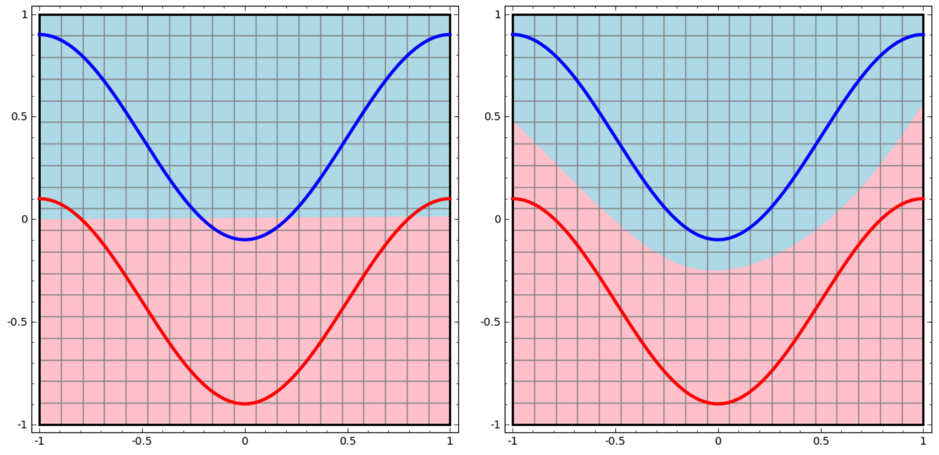


Fig. 2.1 Example Linear (Left) and Non-Linear (Right) decision boundary. Image is taken from Colah's blog (http://colah.github.io/).

2.2.1 Artificial Neural Network

An Artificial Neural Networks (ANNs hereafter) is a computational model used for machine learning tasks. This model is a network of interconnected units (neurones) that can be trained iteratively with examples. A well-known and very important result shows that Artificial Neural Networks are universal function approximator [21, 37]. In practice, Artificial Neural Networks can be considered as a classifier (or a regressor) with non-linear decision boundary. An example of this concept is shown in Figure 2.1.

Artificial Neural Networks architecture is normally made by the following components: an input layer, with as many units as the input feature, one or more Hidden Layer, with a parametric number of units, and an output layer, with as many units as the output requires. Each unit, a part of the input ones, has got an activation function, later we will show the widely used one. Normally, there are two ways to see ANNs: using a tensorial notation or through a graphical (Network) representation. These two are used interchangeably since they represent the same concept.

Moreover, the network of interconnect units can be seen as a multi-dimensional non-linear function. The standard ANNs have got a single hidden layer, and its graphical representation is shown in Figure 2.2, and in what follows we introduce its corresponding tensorial notation. We denote the input feature with a vector $x \in \mathbb{R}^n$, the output vector $y \in \mathbb{R}^k$, then we have two weight matrix, one for each hidden layer, $W \in \mathbb{R}^{m \times n}$ and $U \in \mathbb{R}^{m \times k}$, and finally two bias vectors $b_W \in \mathbb{R}^m$ and $b_U \in \mathbb{R}^k$. The tensorial function defined by an ANN is:

$$s(x) = \phi(U^T \phi(Wx + b_W) + b_U)$$

where ϕ represent a point-wise activation function. The latter can be a linear or non-linear function, and the most common used one are: sigmoid (σ), hyperbolic tangent (Tanh), Linear, and Rectified Linear Unit (ReLU) [59]. Ones we define s we need to define a loss function to minimize. A classical choice is the least mean square error[1], however when there are multiple classes a cross entropy loss is preferred. This function indicates the cross

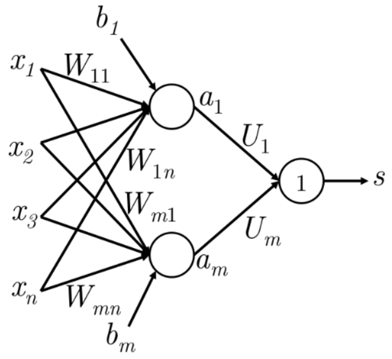


Fig. 2.2 Example of ANN, here we have $k = 1$, so just a single output, with a linear activation, and we denote the output of a neuron i with a_i.

[1]Which is defined as $E(\hat{y}, y) = \frac{1}{n}\sum_i^n(\hat{y}_i - y_i)^2$, where n are the number of samples, $\hat{y} = s(x)$, and y the expected output.

entropy between two probability distributions p and p', and since the expect output to learn is often a probability distribution over the different classes, this function fits well the scope.

Let define our data $\{(x_i, y_i)\}_{i=1}^{n}$, θ be the set of parameter to learn (e.g., W, U, b), and $\hat{y}_i = s(x_i)$. Then, the cross entropy loss function H, with K classes to predict, is defined as follow

$$H(\hat{y}, y) = -\sum_{i=1}^{n}\sum_{k=1}^{K} \hat{y}_i^{(k)} \; log(y_i^{(k)}) \tag{2.1}$$

The aim is to minimize H based on the parameters θ (a.k.a. training phase). The loss function, in the ANN case, is minimized (trained) using the Backpropagation Algorithms [69], which is a distributed gradient descent method. In practice, it is an iterative algorithm that at each step update the parameter θ. A simple formulation of this concept is:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_{\theta^{(t)}} H$$

where $\nabla_{\theta^{(t)}} H$ is the partial derivative of H w.r.t. $\theta^{(t)}$, and α represent the learning rate. The partial derivative for each element θ is obtained applying the chain rule formula. In its original version, this method uses all the contribution of each sample to compute the exact gradient value. However, the latter does not scale well with large datasets, since it can have millions of samples. To overcome this problematic, an online version, called Stochastic Gradient Descent (SGD), of the gradient method has been proposed. This uses a single sample or a batch of samples to compute an approximation of the real gradient value.

This approximation allows to scale with the batch size value. The choice of the latter can be critical for the model performance, and it should be properly selected[2]. Moreover, several methods have been proposed to improve the gradient algorithms[3], by modifying the learning rate while training, or by considering previous gradient updates. The most common used methods are the following: Momentum [66], Nesterov accelerated gradient [61], Adagrad [24], Adadelta [90], RMSprop [81], and Adam [44].

Activation Function

Let now describe some of the most used activation function. These functions take as input a generic vector $x \in \mathbb{R}^n$ and it returns a vector with the function applied to its elements. Normally, these functions are an elementwise application to the vector elements, thus the

[2]It can became an hyper-parameters

[3]Normally this methods are known as Optimizers

vector in output as the same as the input one. However, sometimes the activation function can change the output size. In the following, we list the widely used activation functions.

- **Identity** is the simplest activation function and its formulation is: $f(x) = x$.
- **Logistic** is most commonly used one and it is often represented with σ. Its formula is $\sigma(x) = \frac{1}{1+e^{-x}}$.
- **TanH** is the hyperbolic tangent function, and its formulation is $\text{TanH}(x) = \frac{2}{1+e^{-2x}} - 1$.
- **ReLU** stands for Rectified Linear Unit, and its formulation is $\text{ReLU}(x) = \max(0, x)$.
- **Softmax** is often used as last layer in combination with cross entropy loss, since it generates a probability distribution over K different possible outcomes. In this case the output of this layer has not the same dimensions as x, but instead as K (the number of classes). Its formula is $\text{Softmax}(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$ for $i = 1, \ldots, K$.

All the describe functions are differentiable, this is important since we are approximating our function using a gradient method [4]. Before we explain more advanced techniques, it is worth to briefly describe some statistical learning theory results and basic terminology used in the machine learning field (e.g., overfitting).

Terminology and Statistical Learning Theory

In this paragraph, we shortly introduce several Statistical Learning Theory results such as model capacity, over-fitting, and under-fitting. The main challenge for a machine learning algorithm is to minimise the *Generalization error*, which is the error in the prediction of unseen data. The latter can be estimated by using a set of data collected separately from the training set. This is usually called *test set*, and we assume that it follows the same distribution of the training set. In general, we define the *Training error* and the *Test error* (*generalization error*) separately by referring to the error obtaining using the training and the test set.

When we use a machine learning algorithm, we want to achieve the smallest test error, or better, we want to fill the gap between the training and the test error. This can be very hard since we do not train our model using the test data[5]. In particular, two factors are critical in this case:

- **Over-fitting** occurs when the gap between training and test error is too large. Indeed, by increasing the number of parameters in the model, we can achieve a perfect fitting

[4]Composition of differentiable function is still differentiable.

[5]In a sense, we force our model to learn general concept instead of specific instances

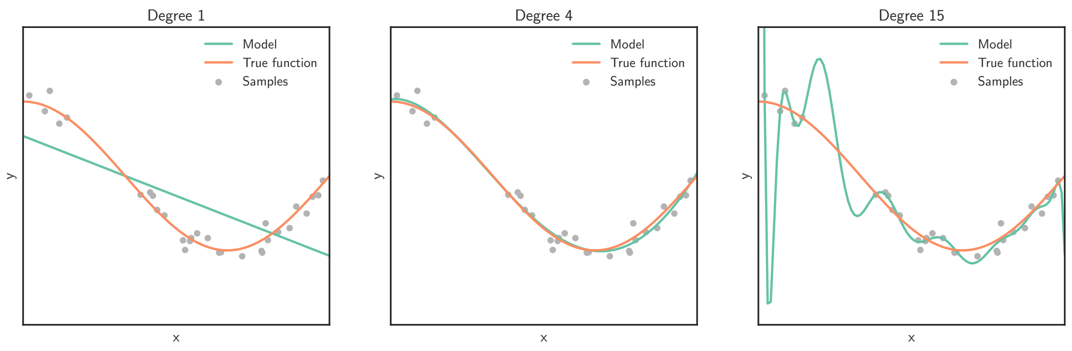


Fig. 2.3 Example of model over-fitting and under-fitting in regression task. The model is a polynomial function, and its capacity is tuned by changing its degree. The image on the left shows the under-fitting phenomenon, the middle one shows an optimal parametrization, and the right ones show the over-fitting case.

of the training data. However, in this way, the model is too tailored to the training set and then it is going to perform poorly with unseen data.

- **Under-fitting** occurs when the model cannot achieve sufficiently low error in the training set. Obviously, if the model cannot achieve small error in the training set, it won't for sure in the test set.

We can control the trade-off between underfitting and overfitting by tuning the models' *capacity*. Statistical Learning Theory provides a means to quantify the model capacity defining the Vapnik–Chervonenkis dimension (VC-dim) [83]. Base on such measures, several theoretical results have been proposed to bound model complexity. However, it is still an open research problem to find a theoretical bound for complex artificial neural networks. Therefore, other methods are normally used to find the right model capacity.

A widely used method to approximate the generalisation error is known as *cross validation*[6]. This method creates a validation set by sub-sampling the training set, and by using such samples, we can estimate the generalisation performance of different models. This procedure is normally called *model selection*, and it searches the best *Hyperparameters* (learning rate, a number of hidden units, etc.) using the error obtained in the validation set. The result of this step is the model that obtain the lowest error in the validation set.

Once we select the best model, we can finally estimate its generalisation error by using the test set. This last step is commonly called *model assessment*. It is very important to keep the latter two phases separate in order to have a real estimation of the model performance.

[6]Different cross-validation has been proposed, e.g. k-fold cross-validation, etc.

With the latter knowledge in mind, we can introduce the last concept in this section which is called *Regularization*, which is a method to improve the generalisation error.

Regularization

The generalisation power of a classifier (or a regressor) is tested by measuring its error in unseen data. A model with many parameters is prone to over-fitting in the training data, leading to a poor performance in the test data, as we discussed in the previous paragraph. Regularisation is used to overcome this problem. Many methods have been proposed, and in the following, we list some of the most used ones:

- **L2 norm** penalizes weights for being large in magnitude. As consequence, the flexibility of the model is reduced, and then it can avoid over-fitting phenomena. This regularisation is implemented by adding a penalization term into the loss function. For instance
$$H_R = H + \lambda \|\theta\|_2$$
where H is the loss function shown in Equation (2.1). The parameter λ is used to decide the regularisation strength, and it is normally tuned using the validation data (a sub-sample of the training data). Another powerful regularisation method uses L1 norm, which enforces a sparse activation of the network units. This method is also known as Lasso regularisation [80].

- **Dropout** [74] prevents complex co-adaptations on training data. In practice, it randomly disabling units during the training, allowing sparsity in the network. The randomness is guided by a parameter which could be tuned in validation.

- **Early Stopping** is one of the most naive forms of regularisation. It works by stopping the training when the validation error start increasing (or at any time). This early stopping breaks the weight grows, and then reduces the over-fitting phenomena.

In this short overview, we described the structure of an Artificial Neural Networks, and we briefly explain the basic machine learning concepts. Even though we did not go into many details, we covered the most important concepts of the subject. In particular, we included the techniques used in our proposed models.

In the next section, we introduce the concept of Language Model, which is fundamental for the Word Embedding construction. The latter is the building block for any semantical NLP task.

2.2.2 Language Model

The goal of a language model is to assign a probability to a sequence of words (e.g., sentence). If the sentence is well formulated, then we expect a high probability for such event, else a low one since it is considered hill formulated. For example, the sentence *"The pen is on the table"* has got a higher probability than *"stock boil fish is toy"*, which does not have any meaning. Mathematically, we define the probability of sequence of n words with

$$P(w_1, \ldots, w_n)$$

A simple way to approximate this probability is to use the **Unigram Model**. This model uses the occurrence of every word in the corpus, denoted as $P(w_i)$, and consider each w_i conditionally independent. Thus, it defines the probability of a sequence of words as:

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i)$$

Obviously, this model is too restrictive since the probability of a word is just related with its frequency. Hence, a more accurate model could be one that evaluates the probability of the word i considering the previous words $j < i$. For example, if we only consider the previous word, a **Bigram Model** (i.e. a pair of words), the probability becomes:

$$P(w_1, \ldots, w_n) = \prod_{i=2}^{n} P(w_i | w_{i-1})$$

If we consider more words to condition the probability of a word i, we would have a more accurate estimation. However, maintaining the co-occurrence count of every n-gram is both memory and computationally very expensive. Indeed, with a vocabulary V we need to keep in memory a table of size $|V|^2$.

It should be noticed that Language Models, as many other statistical methods, uses a symbolic representation of the word. This implies a rigid schema since each word is represented as a unique atom (for example using one-hot-encoding [7]). To overcome this issue, a distributed representation of the word has been introduced. The latter borrows the idea of the language model to build a vector representation of a word, which is commonly called Word Embedding.

[7] A vector of vocabulary size, with all zeros except for one that represents the word that we need.

2.2.3 Word Embeddings

Word Embedding refers to a technique that transforms a word into a vector (a.k.a. word space). These vectors encode semantic information of a word in its vector dimensions. For example, a dimension might indicate tense (past vs. present vs. future) or count (singular vs. plural). A very interesting aspect is that vectors of similar words (or synonymous) are close each other (in the metric space). Indeed, vectors are placed in the space based of their semantical meanings, and sometimes the vectors are also able to encode analogies between words. For example, $W(\text{king}) - W(\text{man}) + W(\text{woman}) \approx W(\text{queen})$, where $W(X)$ stand for the word vector of X. Basically, the word vector representation is used as building blocks of any NLP tasks, since it allows models to use a continuous representation for words (which are usually discrete).

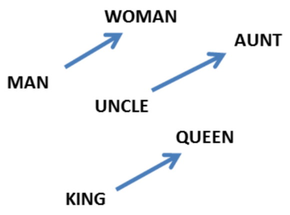


Fig. 2.4 Example of analogies in word embeddings. Image taken from [55].

In the next paragraph, we are going to describe how to create word embedding from a corpus. Among the year different methods have been designed to create word vectors using both statistical methods, like n-grams, and Artificial Neural Networks [4, 18]. There are basically three categories of methods: count based, context based, and their combination. The first method we describe is count based and it uses the Singular Value Decomposition [28] (SVD) on the co-occurrence matrix of the corpus.

Then, we describe two methods which are context based and they are called Continuous Bag of Words Model (CBOW) and Skip-gram model. The last category, it is combining count and context based methods and it is known as GloVe [65].

SVD

The first step of this method is the creation of a Word-to-Word Co-occurrence Matrix $X \in \mathbb{N}^{|V|\times|V|}$, that is a matrix where an element X_{ij} represents how many times a word w_j appears after a word w_i. Then we calculate the SVD of the matrix $X = U\Sigma V^T$ taking the first k singular vectors, where $U \in \mathbb{R}^{|V|\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$, and $V^T \in \mathbb{R}^{k\times|V|}$. The final embedding matrix is obtained by taking the matrix U. A graphical idea is shown in Figure 2.5.

A similar approach has been proposed using instead the Hellinger Principal Component Analysis (PCA) [48]. However, these methods have several drawbacks, such as a quadratic cost of the algorithm (i.e. SVD), matrix sparsity, and several problems with very low frequent

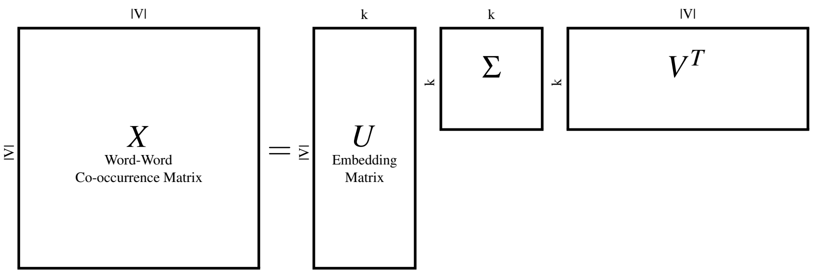


Fig. 2.5 Graphical representation of a Singular Value Decomposition.

words. The latter two can be solved using some kind of smoothing (e.g., Laplace), but the quadratic cost still remains an issue.

In what follows, we introduce three model that uses Artificial Neural Networks to generate word vectors. The idea is to use a Language Model, which predicts the probability of a certain word based on its context, to learn a dense vector able to represent such word in the space. The following methods can be considered as an Unsupervised Learning algorithm since no labels are involved in this process.

Continuous Bag of Words Model (CBOW)
The idea of this model is to predicting a central word from the surrounding context. The input of the model is the one-hot word vector of each word in the given context, where each words is denoted by $x^{(i)}$. The output, denoted with y, is a single vector representing the one-hot representation of the central word.

Let's define $W_{in} \in \mathbb{N}^{d\times|V|}$ the input word matrix and $W_{out} \in \mathbb{N}^{|V|\times d}$ the output one, where $|V|$ is the cardinality of the vocabulary V, and d is the vector length for the word representation. We define the i-th word of the word matrix W_{in} with $u^{(i)}$, and the j-th word of W_{out} with $v^{(j)}$. Both vectors have got the same dimension $1 \times d$. The model can be built through the following steps:

- Generate a context window, that is a concatenation of one-hot vectors, cantered at the word of interest i. Thus, we have $(x^{(i-C)},\dots,x^{(i-1)},x^{(i+1)},\dots,x^{(i+C)})$, where C represents the context size.
- Then, it converts these vectors to their embedding vectors u using the W_{in} matrix. That is, $(u^{(i-C)} = W_{in}x^{(i-C)},\dots,u^{(i-1)} = W_{in}x^{(i-1)},u^{(i+1)} = W_{in}x^{(i+1)},\dots,u^{(i+C)} = W_{in}x^{(i+C)})$.

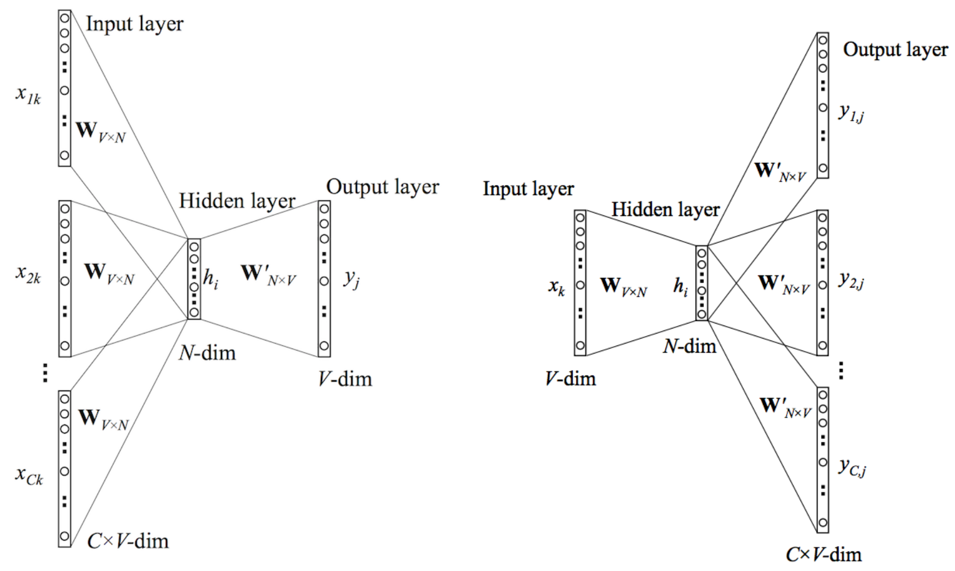


Fig. 2.6 Graphical model of CBOW (Left) and Skip-gram (Right). The image has been taken from Wikipedia (https://en.wikipedia.org/wiki/Word_embedding).

- Next, we take the average of the resulting vectors to obtain a single hidden vector denotes by h. Hence, we have $h = \frac{u^{(i-C)}+\dots+u^{(i-C+1)}+\dots+u^{(i+C)}}{2C}$.
- Finally, the model generates the final prediction using a Softmax function and the previously computed steps. Thus we have: $\hat{y} = Softmax(W_{out}h)$.

The model uses a cross entropy loss function $H(\hat{y}, y)$ (defined in Eq 2.1), and by minimizing H we learn the matrix W_{in} and W_{out}. To do so, the Backpropagation algorithm, introduced in the previous section, is employed. A graphical representation of the model is shown in Figure 2.6. Finally, the embedding matrix is obtained by using one of the learned matrix W_{in} and W_{out}.

Skip-gram

The Skip-gram model takes a word and tries to learn the surrounding words context. The model is very similar to the CBOW, it just inverts input and output. Hence, the input of the model became the one-hot encoding of the word of interest denoted by x, and the output is the concatenation of the one-hot encoded vectors of the words context, denoted by $y^{(i)}$. Considering W_{in} and W_{out} defined as before, the steps to build the model are:

- Using the one-hot encoding of x, we generate h by using the W_{in} matrix. That is simply: $h = W_{in}x$.

- Then, it generates $2C$ vectors $v^{(i-C)}, \dots, v^{(i-1)}, v^{(i+1)}, \dots, v^{(i+C)}$ by using the matrix W_{out}. Which is for instance $v = W_{out}h$.
- Each generated vector u is then converted into probabilities by using a softmax function. Thus, we have $\hat{y}^{(i-C)} = \text{softmax}(v^{(i-C)})$ and so on.

As before we use a cross entropy loss function, and we train the model by Backpropagation.

Both models use a cross entropy loss function, which implies to compute a summation over all the vocabulary V, that is potentially very big (1M words). A successful idea has been to approximate the softmax computation by sampling few words from the vocabulary. This allows to have a good approximation of the loss function and at the same time a very scalable model. Both, CBOW and Skip-gram[8], have been proposed by Mikolov et. al. [54] and they are computationally efficient compared to other count-based models.

Another successful way to build word embedding combines count-based and context-based methods. This model is called GloVe [65]. The latter tries to minimise the dot product of two words vector and the logarithm of their number of co-occurrences (still using a co-occurrence word matrix as the SVD model). Furthermore, this model scales well since for each word only uses a single row of the co-occurrence matrix.

In the next subsection, we discuss a particular Artificial Neural Network that is used to process sequence, e.g. a sequence of Word embedding. This model is called Recurrent Neural Network and it is very powerful since can handle input with a non-fixed dimension (e.g. lists, tree, graphs).

2.2.4 Recurrent Neural Network

The Recurrent Neural Networks (RNNs) [87] are a family of Artificial Neural Networks for processing sequential data. These models are similar to the standard ANN, but in addition they have a self-loop in their hidden state. This allows to have an internal state with temporal properties (dynamics through time).

Given an input sequence $x = x^{(1)}, \dots, x^{(n)}$, where $x^{(i)} \in \mathbb{R}^n$, the equations to calculate the hidden state and the output at the time t, denoted as $s^{(t)} \in \mathbb{R}^h$ and $o^{(t)} \in \mathbb{R}^k$ respectively, of a

[8]This method is normally called Word2Vec.

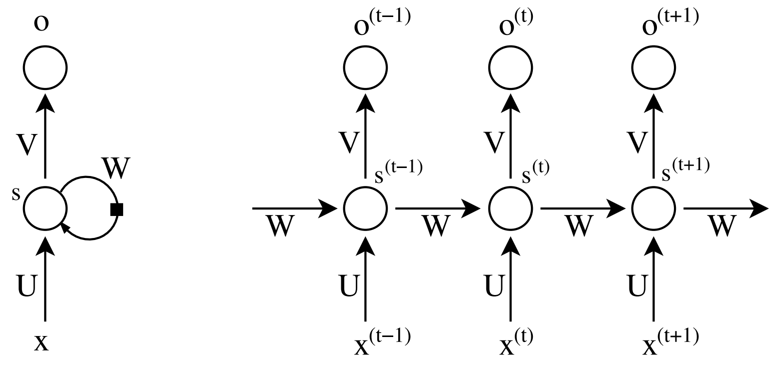


Fig. 2.7 Unfolding of a RNN model. On the left, the conceptual representation, and on the right, the respective unfolded version.

standard RNN [87] are the following:

$$
\begin{aligned}
s^{(t)} &= \sigma(Ux^{(t)} + Ws^{(t-1)} + b_W) \\
o^{(t)} &= \sigma(Vs^{(t)} + b)
\end{aligned}
$$

Where $W \in \mathbb{R}^{h\times h}$ is the hidden to hidden matrix, $U \in \mathbb{R}^{h\times n}$ is the input matrix, $V \in \mathbb{R}^{k\times h}$ the output one, and $b \in \mathbb{R}^k, b_W \in \mathbb{R}^h$ the relative bias vectors. The parameter h denotes the hidden dimension, and it used to tune the capacity of the model. To obtain the hidden state $s^{(t)}$ both the previous state $s^{(t-1)}$ and the input $x^{(t)}$ are used. Indeed, an RNN represents a recurrent equation over the hidden states, thus it can be unfolded to obtain an ANNs with as many layers as the sequence length. The unfold structure is shown in Figure 2.7.

In the unfolded structure the matrix involved in the calculation are shared between different layers, this allows to have a fixed structure able to handle arbitrary long sequence. Moreover, the RNN model can be trained by considering the unfolded structure as a normal ANN, thus it can still use the Backpropagation algorithm. This particular version of the algorithm is called Backpropagation Through Time (BPTT) [58, 68], and it is a bit different from the standard one since it considers the shared weight schema of the layers.

Training RNNs is not as simple as a normal ANNs since several problems occur during training, such as vanishing and exploding gradient [5]. In fact, when the Backpropagation Through Time algorithm updates the parameters weights two things can happen: the gradient may become very small when the important error is at the very beginning of the sentence, and the gradient value can explode if the error summed at each time-step became too big. The gradient explosion can be solved [63] by clipping the gradient norm of a certain threshold.

On the other hand, gradient vanishing can be solved by using a ReLU activation function, instead of a sigmoid one, and by initializing the matrix W with the identity matrix. However, better techniques to overcome these issues are still an open research problem.

Different recurrent models have been proposed to reduce the problematics of the standard RNN. Normally, the self-loop part is called memory cell, since at each time step the network memorises information into its hidden state. Based on this idea of memorization, several models have been designed to improve the recognition of long-term dependency in the input sequences. In the next two paragraphs, we describe two particular types of RNNs: Long Short-Term Memory (LSTM) [36] and Gated Recurrent Unit (GRU) [13].

LSTM

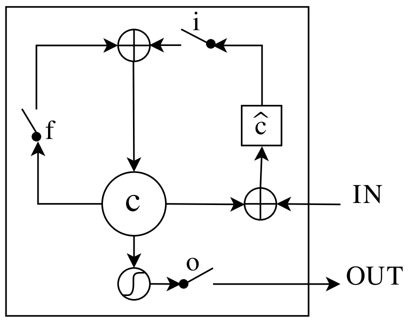
Long Short Term Memory is a particular type of RNN which use a more complex function to decide what to store in its hidden state. This infrastructure ensures to learn longer term dependency compared to the standard model. In particular is made by five components: Input Gate, Forget Gate, Output Gate, New Memory Cell, and Final Memory Cell. Basically, there are three gate function which manage the amount of information to store, a candidate memory, and the final memory that became the new hidden state. The equations that define an LSTM are the following:

$$\begin{aligned}
i^{(t)} &= \sigma(U^{(i)}x^{(t)} + W^{(i)}s^{(t-1)}) && \text{(Input Gate)}\\
f^{(t)} &= \sigma(U^{(f)}x^{(t)} + W^{(f)}s^{(t-1)}) && \text{(Forget Gate)}\\
o^{(t)} &= \sigma(U^{(o)}x^{(t)} + W^{(o)}s^{(t-1)}) && \text{(Output Gate)}\\
\hat{c}^{(t)} &= \text{TanH}(U^{(c)}x^{(t)} + W^{(c)}s^{(t-1)}) && \text{(New Memory Cell)}\\
c^{(t)} &= f^{(t)} \odot \hat{c}^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)} && \text{(Final Memory Cell)}\\
s^{(t)} &= o^{(t)} \odot \text{TanH}(c^{(t)})
\end{aligned}$$

Where $x^{(t)}$ is the input at the step t, W (i, f, o, c) are used for the previous hidden state, and U (i, f, o, c) are used for the current input. In the formulas, we reported the TanH activation function, however different activation functions can be used. Even though, this cell is very popular it is also very slow to train. This is due to the very complex infrastructure. Hence, a simpler but still effective cell is defined in the following.

GRU

A Gated Recurrent Unit is a simplification of the LSTM model. It has been proposed to speed up the training phase and to still capture long term dependency in the input sequence. The



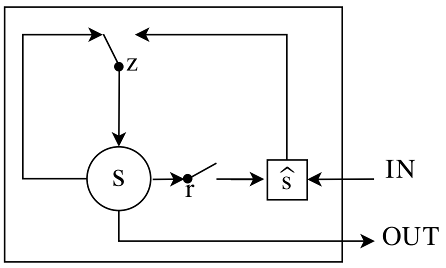


Fig. 2.8 LSTM graphical model (on the left) and GRU model (on the right).

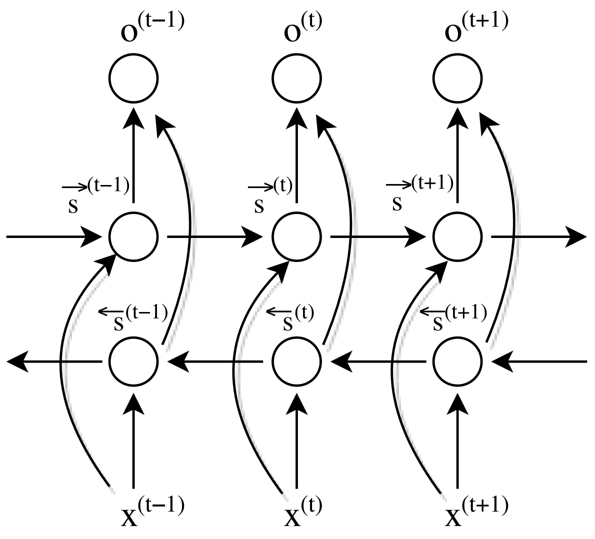
GRU cell has got three main components: Reset and Update Gates, New Memory Generation, and Hidden state. As in the LSTM model, the gate decides how important is the current state and how much of the current input should be stored. This information is then merged to obtain the new hidden state. The formulation of a GRU cell is:

$$
\begin{aligned}
z^{(t)} &= \sigma(U^{(z)}x^{(t)} + W^{(z)}s^{(t-1)}) && \text{(Update Gate)}\\
r^{(t)} &= \sigma(U^{(r)}x^{(t)} + W^{(r)}s^{(t-1)}) && \text{(Reset Gate)}\\
\hat{s}^{(t)} &= \text{TanH}(r^{(t)} \odot Ws^{(t-1)} + Ux^{(t)}) && \text{(New Memory)}\\
s^{(t)} &= (1 - z^{(t)}) \odot \hat{s}^{(t)} + z^{(t)} \odot s^{(t-1)} && \text{(Hidden State)}
\end{aligned}
$$

As in the LSTM equations, the W matrix is used to manage the previous hidden states, and U, instead, to manage the inputs sequence. Notice that the bias terms are present in every equation but are not shown in the former equations for readability purpose. The GRU cells [15] have shown comparable results to the LSTM model, achieving a notable speed up in the training phase.

Bi-Directional

The last model we present combines two RNN cells, one that examines the sequence from left to right, and the other from right to left. This model has been call Bi-Directional RNNs (BiRNNs), and basically it keeps updating two hidden states, $\overrightarrow{s}_t$ and $\overleftarrow{s}_t$ respectively, which represents a certain output at the instant t. For simplicity, we describe the bi-directional equation using the standard RNN model. However, it is possible to use any other cell, for example we can have a biLSTM and biGRU models, which uses LSTM and GRU cell



$o^{(t-1)}$
$o^{(t)}$
$o^{(t+1)}$
$\overrightarrow{s}^{(t-1)}$
$\overrightarrow{s}^{(t)}$
$\overrightarrow{s}^{(t+1)}$
$\overleftarrow{s}^{(t-1)}$
$\overleftarrow{s}^{(t)}$
$\overleftarrow{s}^{(t+1)}$
$x^{(t-1)}$
$x^{(t)}$
$x^{(t+1)}$

Fig. 2.9 Bi-directional RNN graphical representation.

respectively. The equation that describe a biRNN model are the following:

$$
\begin{aligned}
\overrightarrow{s}^{(t)} &= \sigma(\overrightarrow{U}x^{(t)} + \overrightarrow{W}\overrightarrow{s}^{(t-1)} + \overrightarrow{b}_W) \\
\overleftarrow{s}^{(t)} &= \sigma(\overleftarrow{U}x^{(t)} + \overleftarrow{W}\overleftarrow{s}^{(t+1)} + \overleftarrow{b}_W) \\
o^{(t)} &= \sigma(V[\overrightarrow{s}^{(t)}; \overleftarrow{s}^{(t)}] + b)
\end{aligned}
$$

Where ";" stand for concatenation, and the arrow is used to distinguish the direction of the unfolding. Differently from the original RNN, the output matrix $V \in \mathbb{R}^{k\times 2h}$, so its hidden state is doubled. A graphical representation of the model is shown in Figure 2.9.

Final Consideration

The model we described in the last three paragraphs can be extended to become deep models, which means to stack several layers of hidden units before the final output. In this way, we are able to extend the capacity of our model. Moreover, increasing in the deepness of the model (in general) allows creating a hierarchical representation of the input features, which could improve the performance of several tasks.

2.2.5 Other Models

In this subsection, we describe different models used to handle NLP tasks, but that we do not mention or directly used in our thesis. In the following, we shortly introduce two of them:

- **Convolutional Neural Networks** (ConvNet): is an extension of the ANN model, that it is specialised in processing data that has a known grid-like topology [51, 50]. The ConvNet uses the convolution process to get important features from the input. This method has been widely applied in Computer Vision, and also in NLP [17] tasks.
- **AutoEncoders**: is a particular ANN trained to copy its input to its output. The network can be divided into two modules: an Encoder that produces a sparse or compact representation of the input, and a Decoder that uses such representation to recreate the input. This architecture, with two RNNs as Encoder and Decoder, has been used as a sequence generator, as in the case of the Neural Machine Translation models [76].

Several other models have been proposed in the literature, which can handle sequence input or generate data. However, we do not use it in our work, but it still worth to at least mention it, as a short list in the following: Echo State Network [39] used to handle sequence like an RNN, and Deep Believe Network [35], etc.

2.3 Application

In this section, we describe how the presented techniques can be applied to NLP tasks. Here, we just include notable results about semantic and syntactic tasks excluding Question Answering, since we reserved a chapter for that. In the following, we first present a Neural Dependency Parser, since we also use it in one of our model. Then, we present a Recursive Neural Network model that uses the parse tree for the prediction of sentiment in a sentence (Sentiment Analysis). Finally, we present an End-to-End model able to jointly solve many tasks, such as part-of-speech tagging, chunking, named entity recognition, semantic roles labelling, and semantically similar checking. The description of these topics is very informal since these are not the main topic of this thesis but we just discuss it for completeness.

2.3.1 Neural Dependency Parser

In the task section, we discussed the importance of Parsing. Indeed, the latter allows a machine to analyse the linguistic structure of a sentence, and to understand the role of each word in it. As mention before there are two types of parsers commonly used in NLP tasks, such as Constituency and Dependency parsers. An example of the result from this two parsers is shown in Figure 2.10. The Dependency Parser has been widely used leveraging on Neural Network Architecture [11, 3].

Moreover, it is usually called Neural Dependency Parser and it tries to predict the correct transition of an SR-Parser (i.e. based on the features extracted from the parser configuration). The transaction decides how to create the parse tree and how to assign labels (NMOD, SUBJ, etc.) to each edge. There are many kinds of Tag Sets for the labels, the most commonly used ones are Universal Dependencies [9], Stanford Dependencies [22] and CoNLL Dependencies [9]. An efficient implementation of a neural dependency parser is the Parsey McParseface by Google. They release an open source implementation of their parser [10] based on DRAGNN [46].

2.3.2 Recursive Neural Network

A Parse Tree (Dependency or Constituent) has been widely used for Sentimental Analysis tasks [72, 77, 71] achieving excellent results. The idea is to use the parse tree as an infrastructure to compose words in the sentence. This allows obtaining a single, and more

[9]http://universaldependencies.org/

[10]https://github.com/tensorflow/models/tree/master/syntaxnet

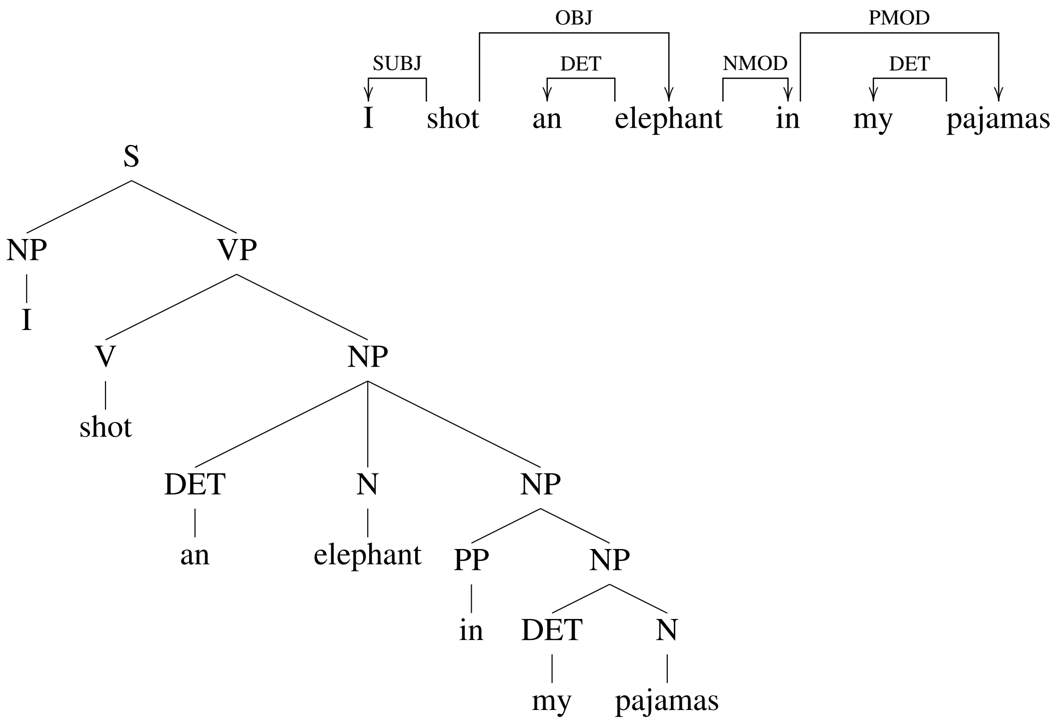


Fig. 2.10 Resulting trees from a Constituency parser (on the bottom left) and a Dependency Parser (on the top right).

semantical, representation of a sentence, useful to better accomplish the task. The model uses a particular type of ANN called Recursive Neural Network [27, 73]. The idea is to compose Word Embedding (present as a leaf of the tree) starting from the bottom. At each tree level, two nodes are merged by learning a hidden representation which keeps important information.

The same idea of using a parsing tree to improve the overall representation of a sentence has been used in one in our model called ThRee Embedding Recurrent Neural Network, achieving promising results. In Chapter 4, we describe in more detailed this model.

2.3.3 Unified Architecture for Natural Language Processing

This last model is reported for completeness, and because with a single pipeline can solve many of the previously listed tasks (part-of-speech tags, chunks, named entity tags, semantic roles, and semantically similar). This model [17] jointly learns all the tasks using a multi-task [79] learning approach. Moreover, by using a weight-sharing schema, like in ConvNets and in RNNs, it is able to share the same infrastructure between different tasks.

In the next Chapter, we describe the State-Of-The-Art in Question Answering Tasks, with a particular focus on Reasoning Question Answering, Reading Comprehension, and Community Question Answering.

Chapter 3

Related Works

In this Chapter, we describe the State-Of-The-Art in different Question Answering (QA) tasks. In particular, we analyse three different sub-tasks: Reasoning Question Answering (RQA), Read Comprehension (RC), and Community Question Answering (CQA). The former two tasks are tested by formulating questions which are answerable directly by reading the piece of text in input (e.g. a document or a paragraph). Normally, the first sub-task is included in the second. Indeed reading (and understanding) a paragraph requires connecting facts regarding entities present in the text. Such ability is not trivial for a machine since facts are written in natural language and often there could not be enough information to make a pure logical inference.

At the end of this discussion, we introduce two models used for the Community Question Answers task. We do not describe as many models as for RQA and RC since a good summary is already available in the SemEval organisers' paper.

Since we are describing the State-Of-The-Art in tasks presented in the last 2/3 years, many papers and methodologies that we present are just available in arXiv[1], an e-Print archive for scientific publications. Nerveless, most of the presented articles have already reached a high number of citations.

3.1 Reasoning Question Answers

In this section, we focus on text understanding and reasoning. A series of synthetic tasks, called bAbI, have been recently proposed [85] to test the ability of a machine in chaining

[1]https://arxiv.org/ Open access to 1,257,656 e-prints in Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance and Statistics

Table 3.1 Summary of the 20 bAbI tasks. The description includes the type of task, and their expected difficulties.

Type	**Task number**	**Difficulty**
Single Supporting Fact	1	Easy
Two or Three Supporting Facts	2-3	Hard
Two or Three Argument Relations	4-5	Medium
Yes/No Questions	6	Easy
Counting and Lists/Sets	7-8	Medium
Simple Negation and Indefinite Knowledge	9-10	Hard
Basic Coreference, Conjunctions and Compound Coreference	11-12-13	Medium
Time Reasoning	14	Medium
Basic Deduction and Induction	15-16	Medium
Positional and Size Reasoning	17-18	Hard
Path Finding	19	Medium
Agent's Motivations	20	Easy

facts, simple induction, deduction and many more. The dataset is available at http://fb.ai/babi, and it includes 20 tasks, each of which is studied to test a particular reasoning ability. It is proposed in two different settings, one with 1K samples and another one with 10K samples. The types of task are summarised in Table 3.1. In what follows, we are going to briefly describe how the latter tasks have been solved. Most of the solutions are based on Artificial Neural Networks with an explicit usage of external memory. This seems necessary to solve such tasks since the system needs to store relevant facts and to retrieve it based on the given question.

Moreover, we do not report the detailed results when we present different models, instead, at the end of the section, we are going to show a summary table, with all the results.

3.1.1 Memory Network

Memory Network [86] is one of the first models to introduce the ability to explicitly store facts in memory. In their original paper, the authors proposed a general framework to build an Artificial Neural Network with the ability to explicit use an addressable memory. The main components[2] of this framework are the following:

m: (memory) set of memories, e.g, represented as element in an array.

[2]We keep the same notation used in the original paper

I: (input feature map) which transform the input into an internal representation, e.g., using a Word Embedding matrix.

G: (Generalization) updates the current memories given the input.

O: (output feature map) produce an output given an input (e.g. a question) and the current memories.

R: (response) convert the output in the desired format, e.g., text or actions.

Basically, the **G** component allocates memory slots given an input text, encoded it by the **I** module, and then the **O** and **R** components infer the correct answer by using the allocated memories (and eventually a question). A graphical representation of the framework is shown in Figure 3.1. This framework has been used to solve the 20 bAbI tasks by using word embedding as input module and by taking the memory which matches the most (i.e. using the inner product as a score) the embedded query. This model was one of the first to achieve good results in bAbI tasks. However, one of the main drawbacks of this system is that each

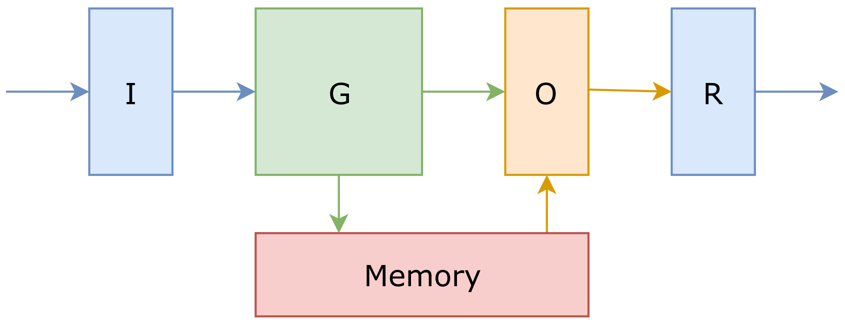


Fig. 3.1 Diagram representing the components, and their interactions, of the Memory Networks framework.

component must be trained separately. To overcome this problem an End to End Memory Network [75] has been proposed[3]. This model uses a different embedding matrix to encode respectively story, memories and query, and a Softmax layer to find the most likely memory that can answer the embedded query. Then the output module, which is quite similar to the previous version, uses a Softmax layer to assign a probability to all the possible candidate answers[4]. This model can be also extended to multi-hop, that is a way to score the memories several times based on the given question. This extension can further improve the accuracy in many tasks, including bAbI and in other Reading Comprehension tasks. A schematic of the model is shown in the Figure 3.2. On the same line, Deep Mind group[5] proposed

[3]End to End refers to a system that can train all of its components together.

[4]All the word in the vocabulary.

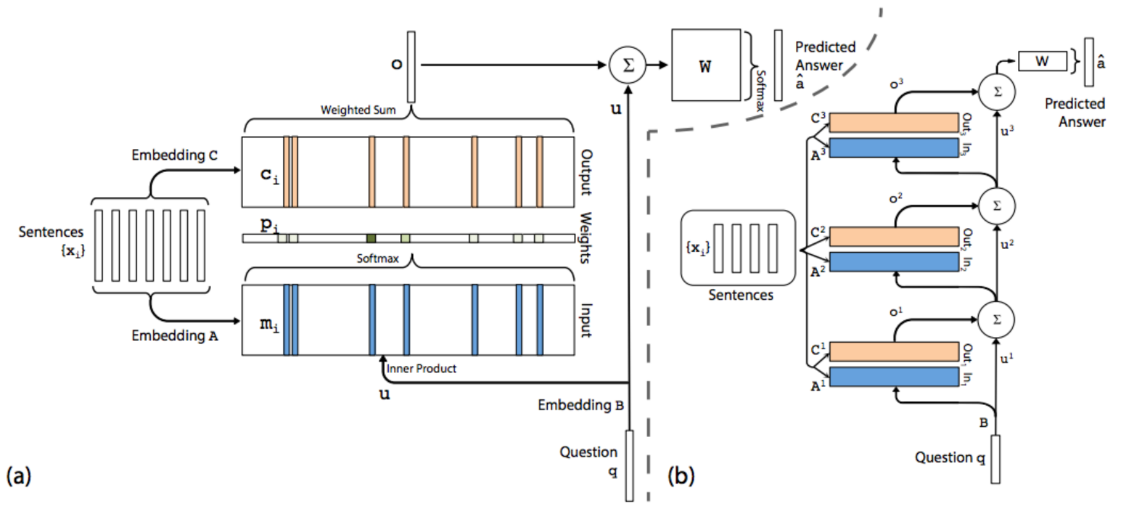


Fig. 3.2 End To End Memory Network architecture. Image taken from the original paper [75].

their version of an ANN able to address memory. They proposed two models, which are: Neural Turing Machine [29] and Differentiable Neural Computer [30]. These two have been tested in the bAbI task achieving better results than Memory Networks. However, these architectures are more suitable to implement general algorithms, like sorting or shortest path, instead of pure Question Answering, therefore we won't give many details.

The model which is able to solve all the 20 bAbI tasks, in the 10K setting, is the Recurrent Entity Network [32]. This model is going to be discussed in the next Chapter since our model starts from the same network infrastructure.

3.1.2 Dynamic Memory Network

Dynamic Memory Network [47, 88] is another model that explicitly memorise facts in order to answer a given question. Differently from the previous, this model does not create an array of memories, instead, it builds up memory states based on the input sentence, the question, and the previous history[6]. To be more specific, it has got four modules (quite similar to the previous ones): **Input** and **Question** modules which respectively encode the sentence in input and the question into a single embedding. Then, it has got an **Episodic Memory Module** that takes as input the encoded sentence, the encoded question and its previous state, and it creates a new memory state. Notice that, the memory state is initialized with

[5]https://deepmind.com/

[6]This model is similar to an RNN model.

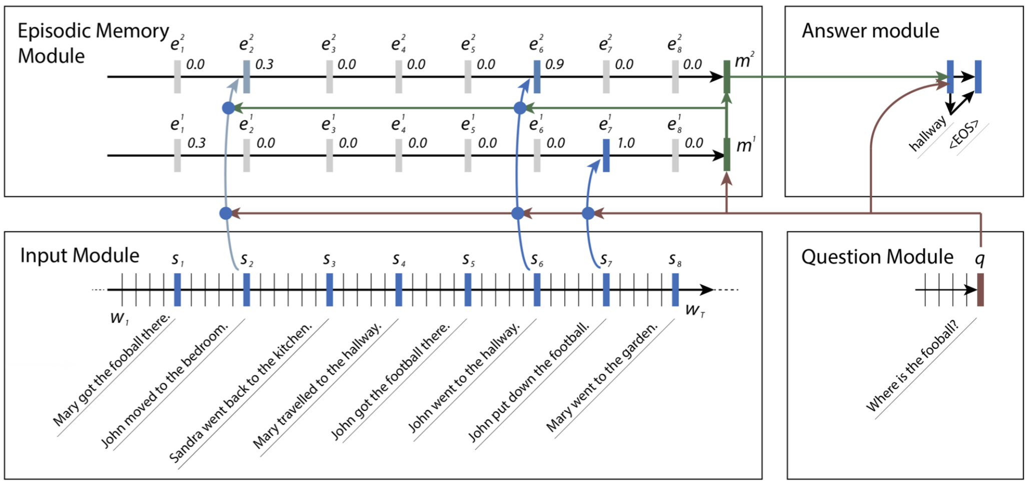

Episodic Memory Module
Answer module
Input Module
Question Module
Mary got the fooball there.
John moved to the bedroom.
Sandra went back to the kitchen.
Mary travelled to the hallway.
John got the football there.
John went to the hallway.
John put down the football.
Mary went to the garden.
Where is the fooball?
hallway
<EOS>

Fig. 3.3 Dynamic Memory Network. The image is taken from the original paper [47].

the encoded question. When all the sentences are passed through the latter module, the final memory state is used by the **Answer Module**, which is a Softmax layer, that assigns probabilities to all the candidate answers.

As in the End To End Memory Network, this model allows passing multiple times through the Episodic Memory Module, which let the model read the input sentences multiple times. This model has passed all tasks except for one. A schematic of the model is shown in the Figure 3.3.

3.1.3 Neural Reasoner

The last model that we present does not use any external memory. The Neural Reasoner [64] takes as input all the sentences (called facts in the article), the question, and it encodes each of them using an RNN (i.e. GRU cells). After that, it pairs each encoded fact with the encoded question, to create a score between them. Then, each couple is passed to an Artificial Neural Network which outputs a single vector for each pair.

The latter set of vectors is passed through an Average or Max-Pooling layer which selects the final vector representation. The described process can be iterated multiple times using as question the output of the previous steps. The last layer, called Answering Layer in the article, takes the output of the last pooling step and it passes it to a Softmax layer which generates the final answer.

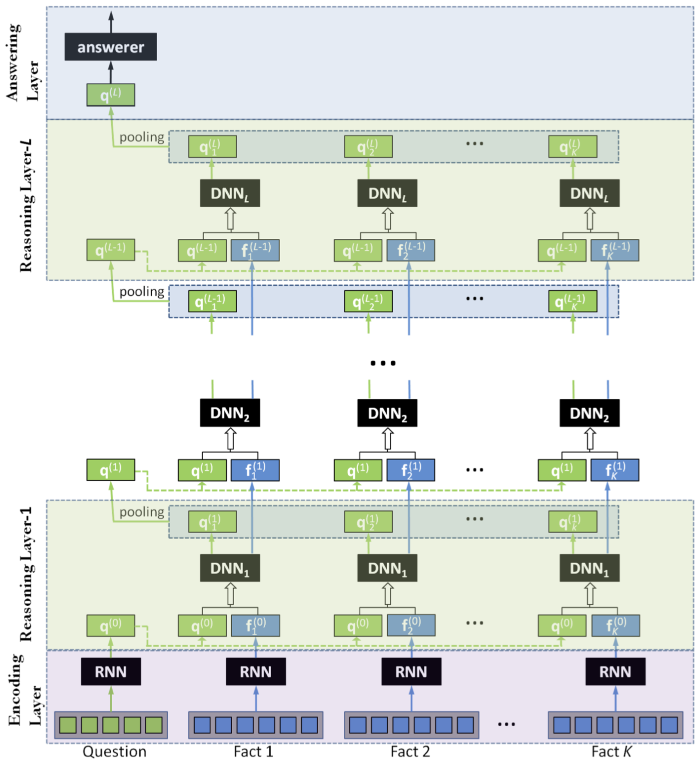


Fig. 3.4 Neural Reasoner. The image is taken from the original paper [64].

This model is not very scalable when there are long paragraphs, but it works well for bAbI tasks 17 and 19 (in particular with 1k sample setting). However, all the other tasks are not actually reported in this paper. A schematic of the model is shown in the Figure 3.4.

3.1.4 Other Approaches

Several others baseline approach have been proposed [85] such as: $n-gram$ [67] model, LSTM reader and an SVM model. The latter and the original Memory Network requires a Strong Supervision, thus the training set should include the sentence which gave the right answer. This obviously is a great advantage for the learner, but it is too far from the reality, since normally this information is never present in a real dataset. For this reason, this two models are not included in the following results, which does not use such information.

Table 3.2 shows the result in the 20 bAbI tasks of the described models. All the results shown in the Table are taken from the dataset with 10K samples. To pass a task the model should have an error lower than 5%. The results using 1K setting are shown later in the experiment Chapter. Using the latter setting, there is no model that pass all the tasks, and therefore it is still considered an open problem.

Table 3.2 Test error rates (%) on the 20 bAbI tasks for models using 10k training examples. NTM stands for Neural Turing Machine, MemN2N for End To End Memory Network, DNC for Differentiable Neural Computer, DMN+ for Dynamic Memory Network, and EntNet for Entity Recurrent Network.

	Tasks	**NTM**	**MemN2N**	**DNC**	**DMN+**	**EntNet**
1	1 supporting fact	31.5	0	0	0	0
2	2 supporting facts	54.5	0.3	0.4	0.3	0.1
3	3 supporting facts	43.9	2.1	1.8	1.1	4.1
4	2 argument,relations	0	0	0	0	0
5	3 argument,relations	0.8	0.8	0.8	0.5	0.3
6	yes/no questions	17.1	0.1	0	0	0.2
7	counting	17.8	2	0.6	2.4	0
8	lists/sets	13.8	0.9	0.3	0	0.5
9	simple negation	16.4	0.3	0.2	0	0.1
10	indefinite knowledge	16.6	0	0.2	0	0.6
11	basic coreference	15.2	0	0	0	0.3
12	conjunction	8.9	0	0	0.2	0
13	compound coreference	7.4	0	0	0	1.3
14	time reasoning	24.2	0.2	0.4	0.2	0
15	basic deduction	47	0	0	0	0
16	basic induction	53.6	51.8	55.1	45.3	0.2
17	positional reasoning	25.5	18.6	12	4.2	0.5
18	size reasoning	2.2	5.3	0.8	2.1	0.3
19	path finding	4.3	2.3	3.9	0	2.3
20	agent‘s motivation	1.5	0	0	0	0
Failed	Tasks (>5% error):	16	3	2	1	0
Mean	Error:	20.1	4.2	3.8	2.8	0.5

3.2 Reading Comprehension

In the previous section, we discussed several Question Answering tasks which require reasoning over a sequence of (synthetic) facts. However, these tasks are simplistic comparing to a real word text. For instance Read Comprehension (RC hereafter), which is defined as the ability to read a text, process it, and understand its meaning[7], it is a more realistic scenario. Indeed, such task not only implies a reasoning process but it also requires to process important facts mixed in raw text, e.g., a paragraph with more than a thousand words.

Reading Comprehension is considered as QA task since it is tested by asking questions which require reasoning from the paragraph. A big problem among the years was to find large datasets with human annotated sample. For example, the MC test [67] offered a few thousand human annotated sample (also said gold label), but still, it was too small for building powerful statistical models. This shortcoming has been solved by finding a clever way to create big datasets from unlabelled data. In the following, we list three of the most popular datasets and we briefly explain their creation.

CBT Children's Text Books [34] dataset has been created from books that are freely available in the Project Gutenberg[8]. Each data sample is made of 21 sentences, where 20 of those represents the paragraph to read, and the 21st becomes the query. In the latter, a word is removed and it became the expected answer. Thus, the model must be able to identify the correct word among a selection of 10 candidate answers. This dataset allows evaluating four classes of questions by removing distinct types of word: Named Entities, (Common) Nouns, Verbs and Prepositions.

CNN & DM These two datasets [33] comes from a large collection of news articles from CNN and Daily Mail. Each article is accompanied by a bullet point, which summarises aspects of the information contained in the article. From these short summary sentences, one entity is replaced with a placeholder. The summary together with the unknown entity become respectively the question and the answer for an article. Once the triples text, question, and answer have been created, all the entities that appear in it are anonymized (by using a marker @entity-i) and shuffled.

WdW Who did What [62] is a dataset created from LDC English Gigaword newswire corpus (https://catalog.ldc.upenn.edu/ldc2011t07). It has been created in the same way as the CNN & DM datasets, but instead of predicting among all the possible entities in

[7]https://en.wikipedia.org/wiki/Reading_comprehension
[8]https://www.gutenberg.org/

the text, the choice is limited to a smaller set of candidates. Another key difference is that entities are not anonymised which allows to building systems that can better exploit semantical information. The dataset is available in two settings: a smaller but cleaner Strict version and a larger but noisier called Relaxed.

These three datasets can be categorized as fill-in-the-gap Cloze [78] style question answer, and they are often used as benchmark for Reading Comprehension. Some statistics about these datesets are reported in Table 3.3.

Many different models have been proposed to achieve good results in these datasets, using many different architectures. For instance, Memory Network, described in the previous section, has been tested [34] in CNN and CBT datasets, achieving good results. In the following we list the most effective models present in literature and as before we summarize their performance at the end of the chapter.

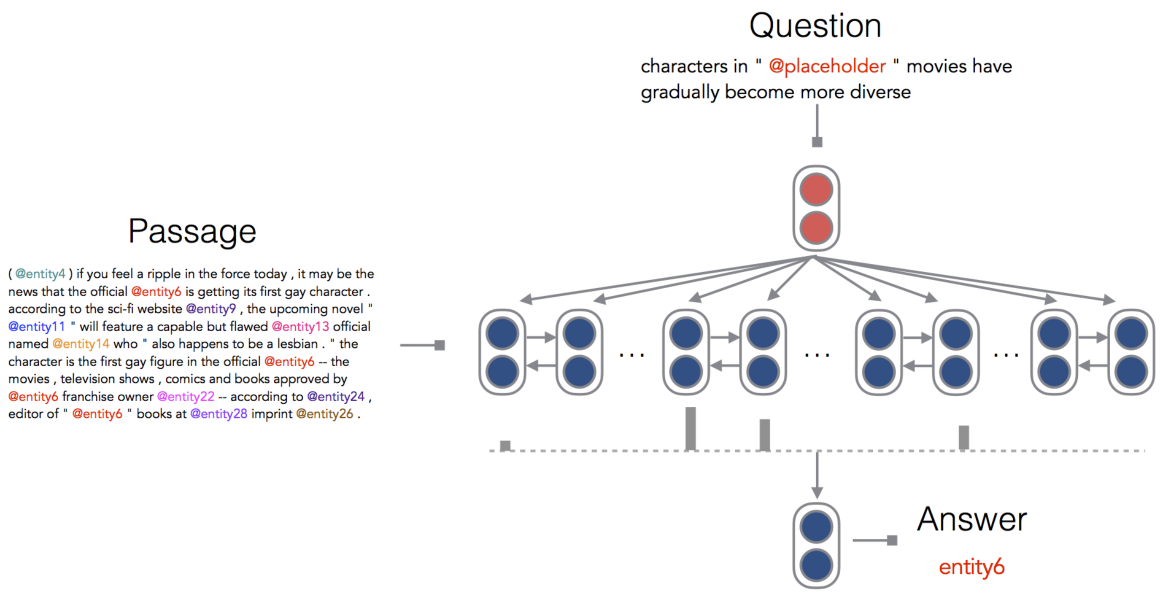
3.2.1 Attentive and Impatient Reader

Attentive and Impatient Reader [33] have been proposed as the first model for CNN and Daily Mail datasets, and they are often used as a baseline. These models have got a framework similar to the Memory Network, indeed the following modules are present: Input, Attention, and Output. The Input module, for both Attentive and Impatient Reader, encodes separately the question and the paragraph using a bi-directional single layer LSTMs (BiLSTM). The Attention module assigns a score to each word (represented by the concatenation of its BiLSTM states) based on the relevance to the question (by a liner layer with a TanH activation).

This creates a distribution over the words in the text, that combined with the encoded representation of the question, is used by the Output module (a Softmax over the vocabulary) to generate the correct answer. The Impatient Reader has got the ability to reread the document as each query token is read, similarly to the multiple hops in the End To End Memory

Table 3.3 Dataset statistics. NE stands for Named Entity, and CN for common noun.

	CNN	**Daily Mail**	**CBT-NE**	**CBT-CN**	**WdW-Strict**	**WdW-Relaxed**
Train	380298	879450	108719	120769	127786	185978
Validation	3924	64835	2000	2000	10000	10000
Test	3198	53182	2500	2500	10000	10000
Vocab	118497	208045	53063	53185	347406	308602
Max doc length	2000	2000	1338	1338	3085	3085



characters in " @placeholder " movies have gradually become more diverse

(@entity4) if you feel a ripple in the force today , it may be the news that the official @entity6 is getting its first gay character . according to the sci-fi website @entity9 , the upcoming novel " @entity11 " will feature a capable but flawed @entity13 official named @entity14 who " also happens to be a lesbian . " the character is the first gay figure in the official @entity6 -- the movies , television shows , comics and books approved by @entity6 franchise owner @entity22 -- according to @entity24 , editor of " @entity6 " books at @entity28 imprint @entity26 .

Fig. 3.5 The Standford version of Attentive Reader. The infrastructure is the same as the one originally [33]. The image has been taken from the original paper [10].

Network. The latter should empower the attention mechanism; however, it did not further improve the accuracy of the system. The Attention module is similar to the Generalisation (**G**) step in the Memory Network framework. Differently from such module, the Attentive Reader does not store and update facts in memory; but instead it assigns a score to the RNN hidden state of each word based on the given question.

Almost one year later, Danqi Chen from Standford proposed a few modifications to the Attentive Reader [10] which increased its accuracy by 10%. This was very surprising since they just changed the activation function of the attention layer and they only predict among entities which appear in the passage, instead of the entire vocabulary. A schematic of the model is shown in the Figure 3.5.

3.2.2 Pointer Network

The model in this section is inspired by a novel Artificial Neural Network architecture called Pointer Network [84]. The latter showed a great ability to handle very long sequences using a fixed length infrastructure (still by using RNNs). In facts, this model explicitly uses the words in input (using an index which represents their position) to generate the answer.

Attentive Sum Reader [43] uses such idea to handle Reading Comprehension tasks. The model acts in three phases: Firstly, it computes an embedded representation of the question

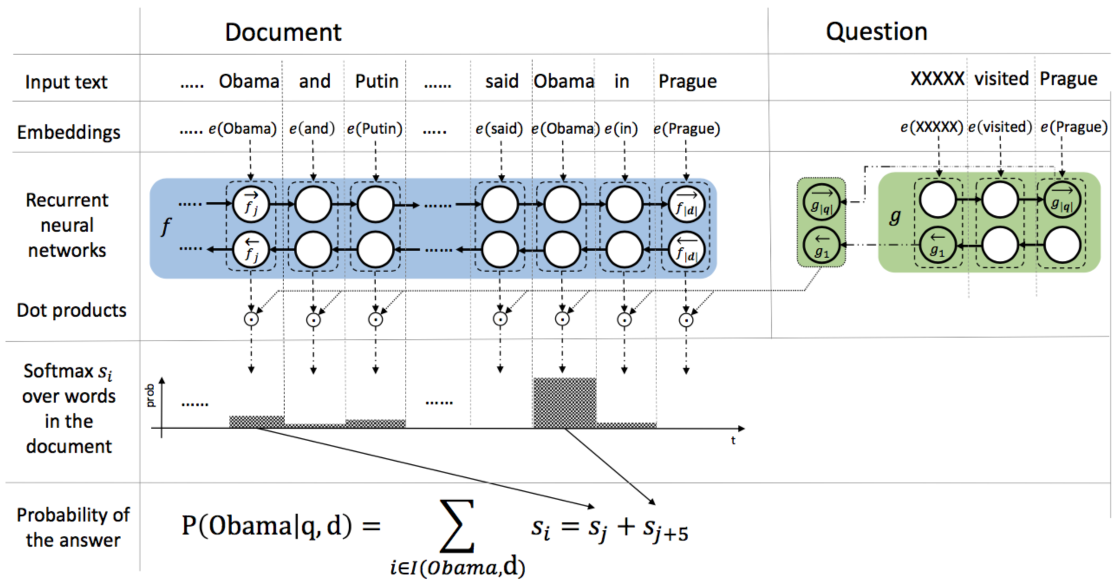

Document
Question
Input text
..... Obama and Putin ...... said Obama in Prague
XXXXX visited Prague
Embeddings
..... e(Obama) e(and) e(Putin) ..... e(said) e(Obama) e(in) e(Prague)
e(XXXXX) e(visited) e(Prague)
Recurrent neural networks
f
g
Dot products
Softmax s_i over words in the document
prob
t
Probability of the answer
$P(\text{Obama}|q,d) = \sum_{i \in I(Obama,d)} s_i = s_j + s_{j+5}$

Fig. 3.6 Graphical representation of the Attentive Sum Reader. The image has been taken from the original paper [43].

using a bidirectional GRUs (biGRU). Secondly, it creates the context embedding, using the whole document, for each word. This is done by multiplying the embedding of a word with a bi-directional representation (its hidden stats) of the document until that moment. Thirdly, the model selects the most likely answer by multiplying the embedded question and the contextual embedding of each occurrence of a candidate answer in the document. Basically, the idea of this model is to use the contextual embedding to give an improved semantical representation to each word. The graphical schema of the model is shown in Figure 3.6.

Another model which uses a similar idea is called Attention over Attention [20]. This model uses a Pair-wise Matching matrix (which is a matching between each context word and each question word) together with a Bidirectional RNN (bi-GRU in this case) over the document to create a context embedding for each word. After that, the final prediction is made in the same ways as the Attentive Sum Reader [43]. Essentially, this model tries to compare the given question versus the document and vice-versa in order to obtain a double attention mechanism. The graphical schema of the model is shown in Figure 3.7.

An even more complex architecture, which also implements the idea of a Pointer Network, is the EpiReader [82]. This model is made by two components: Extractor and Reasoner. The first one selects a small subset of answers in the text using the same attention mechanism of the Attentive Sum Reader. The second, instead, plug the short-listed candidates into the

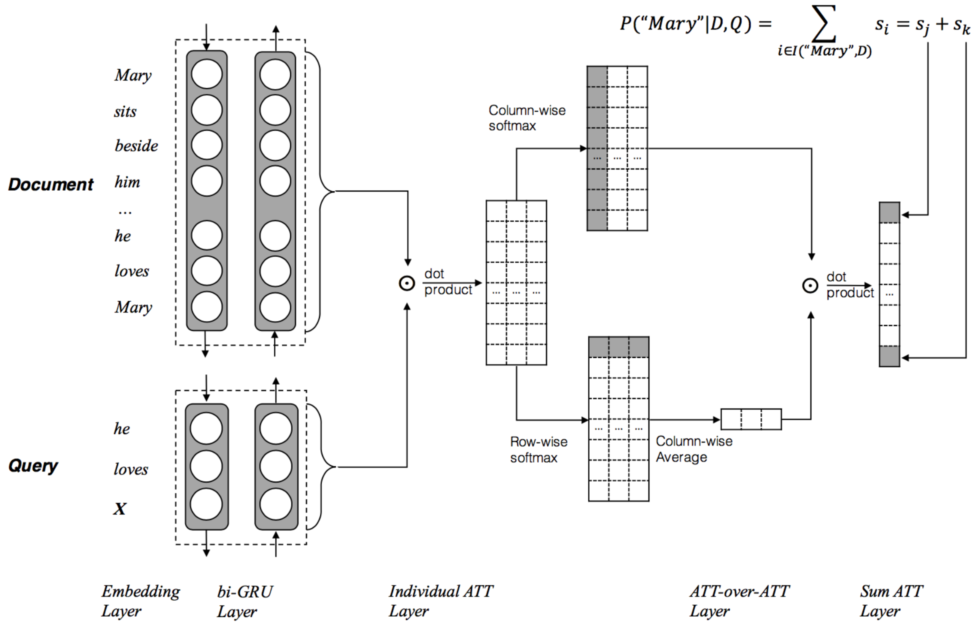


Fig. 3.7 Attention Over Attention model. Image taken from the original paper [20].

question (substituting the placeholder) and then estimate the concordance of each generated candidate sentence with the words in the paragraph. This final match is then used in parallel with the Extractor to produce a final ranking of the answer candidates. The graphical schema of the model is shown in Figure 3.8.

3.2.3 Dynamic Entity Representation

The last model we present is called Dynamic Entity Representation [45]. Differently from the previous, this model tries to accumulate information about entities in the document. Indeed, it takes any sentence in which a determined entity e appears and, using a bi-directional RNN (BiLSTM), it creates a compact representation. To create a single vector which represents all the knowledge of a given entity e, the model applies a weighted sum to all the vectors. The weights used in the sum are the matching score of the sentence versus the given question[9].

Finally, the answer is generated by using a Softmax layer over the entities in the text. This model has shown good results, in particular by adding a max-pool layer to the vectors of a given entity e. A max-pool layer takes the max value of each dimension in the vectors that represent the encoded sentence.

[9]This step can be considered as an Attention step

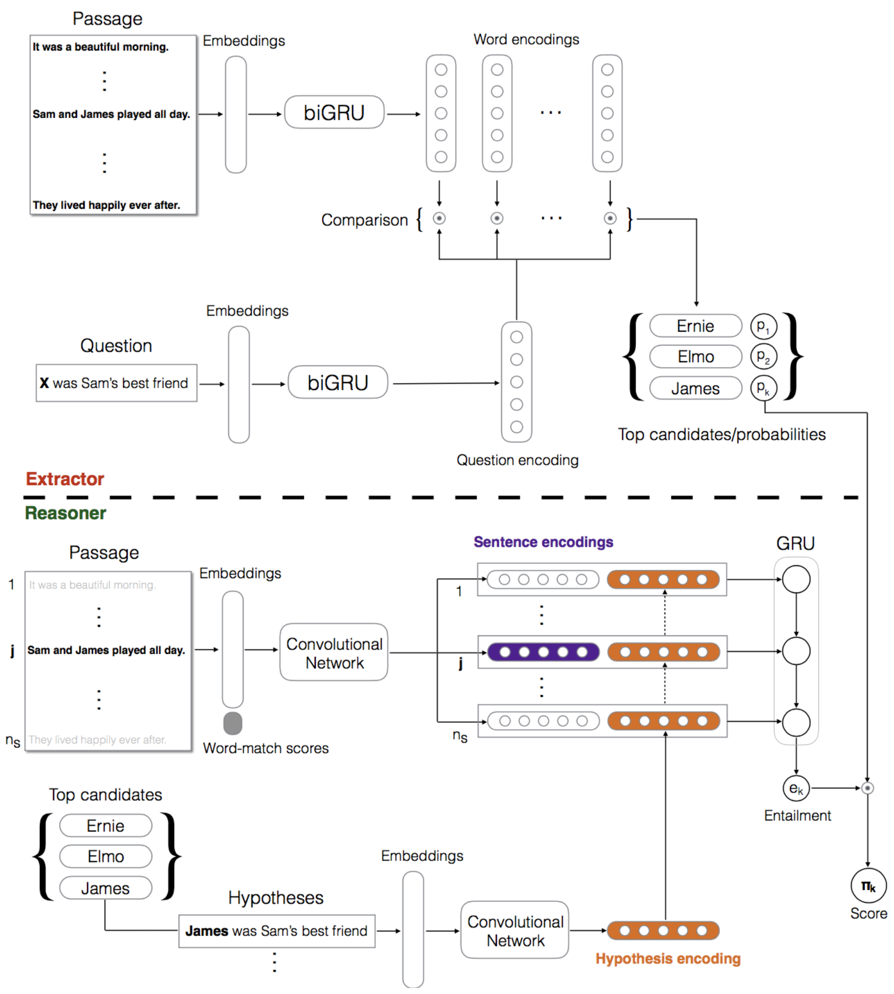


Fig. 3.8 EpiReader. Image taken from the original paper [82].

3.2.4 Learning and Non-Learning Baseline

In Reading Comprehension tasks, several Learning and Non-Learning baseline have been proposed, thus we list the most used ones. In the Learning baseline, we have the LSTM Reader [33]. This model simply concatenates document words and question words, and it uses as input for a deep LSTM. In this way, it creates an encoded representation of the sentence that, in combination with an encoded representation of the question, predicts the most likely answer. There is also a variation of this method, such as sliding window and window + position [34], which uses different embedding matrix and ways to encoded the sentences. The latter gives a stronger baseline; however, they do not reach the same performance as the model which uses the Attention mechanism explained in the previous paragraphs.

The Non-Learning solution uses methods that do not involve a learning process. Frame-Semantic and Word distance models [33] are two model that follow this line. The first, identify predicates (triples (e_1, V, e_2) “who did what to whom”) from the document and tries to finds a matching (exact or using some heuristic approach) with the predicate in the question. The second instead, align the placeholder of the question with every entity in the document. Then extract a window context from both and it sums the exact word matching between them. Both of this baseline have been evaluated in the CNN and Daily Mail dataset.

In Table 3.4 we report the accuracy of each considered models among the four datasets. The results shown in the table are taken from the original article where each model was presented. We do not report results from ensemble (results averaged on multiple models) since we are interested in studying models itself and not their composition.

Table 3.4 Validation/Test accuracy (%) on CNN, Daily Mail, CBT and WdW dataset. In the list AR stands for Attentive Reader, AS for Attentive Sum, AoA for Attention over Attention, and DER for Dynamic Entity Representation. In bolt the model with the best performance in the Test set.

Model	**CNN**		**Daily Mail**		**CBT-NE**		**CBT-CN**		**WdW-S**	**WdW-R**
	Val	Test	Val	Test	Val	Test	Val	Test	Test	Test
Human	-	-	-	-	-	81.6	-	81.6	84.0	-
Max Freq.	30.5	33.2	25.6	25.5	-	33.5	-	28.1	0.32	-
Frame-semantic	36.3	40.2	35.5	35.5	-	-	-	-	-	-
Word distance	50.5	50.9	56.4	55.5	-	39.8	-	36.4	46.0	-
Language Model	-	-	-	-	-	43.9	-	57.7	-	-
LSTM Reader	55.0	57.0	63.3	62.2	-	40.8	-	56.1	-	-
Memory Net	63.4	66.8	-	-	70.4	66.6	64.2	63.0	-	-
Uniform Reader	39.0	39.4	34.6	34.4	-	-	-	-	-	-
Attentive Reader	61.6	63.0	70.5	69.0	-	-	-	-	53.0	55.0
Impatient Reader	61.8	63.8	69.0	68.0	-	-	-	-	-	-
Stanford (AR)	72.5	72.7	76.9	**76.0**	-	-	-	-	64.0	**65.0**
AS Reader	68.6	69.5	75.0	73.9	73.8	68.6	68.8	63.4	57.0	59.0
AoA	73.1	**74.4**	-	-	77.8	**72.0**	72.2	**69.4**	-	-
EpiReader	73.4	74.0	-	-	75.3	69.7	71.5	67.4	-	-
DER	71.3	72.9	-	-	-	-	-	-	-	-

3.3 Community Question Answers

In this section, we describe several models used for Community Question Answers task of SemEval2017. This task required to reorder 10 comments given a question. Moreover, the systems need to assign a label ('good' or 'bad') and a score value to each comment. The performance of different models is compared using the Mean Average Precision (MAP) with respect to the real ranking position. In this task, of the SemEval2017 competition, participated 14 teams, plus the organiser proposed 2 baseline models (random, and chronological order).

In what follows, we report just the winner and the second runner models, since all the others competitors used similar approaches. In fact, most of the models, including the winner, used a Support Vector Machine (SVM) [19] classifier on several features extracted from the text, or an Artificial Neural Network directly on raw data.

3.3.1 KeLP

KeLP [26] is the model that won SemEval2017 Community Question Answers task (A), and it used an SVM classifier. In this model, the authors proposed several features for each couple question comment, that are then fed into an SVM. They divided these features into three categories: Intra-pair similarities, Inter-pair kernel, and Task Specific Features. The Intra-pair set of features tries to collect similarities between pair question comment, and it includes Lexical, Syntactic, and Semantic similarities. The Inter-pair similarity uses a Tree Kernel [57] that computes the shared substructures between parse trees of question and comment. The Task-specific features includes: Ranking Features, Heuristic (comment characteristics such as its length, its category, whether it includes URLs, emails, or other particular words, etc.), Thread-based features (e.g., whether a comment is followed by an acknowledgment from the user who asked the question), and Stacking features. All of these features are then fed into an SVM model, which classified the pairs question comment as 'Good' or 'Bad'.

Many other competitors used such scheme, feature collection and then SVM classifier. We do not report all of them since they have just made a slightly different feature selection.

3.3.2 MSRA

The MSRA [25] is the second runner of the competition and it has achieved a very close MAP (0.2% less) compare to the winner. This model uses a combination of Artificial Neu-

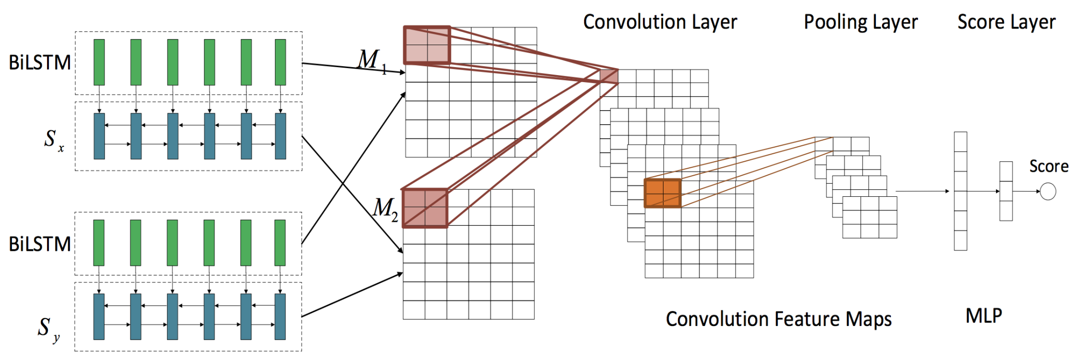


Fig. 3.9 MSRA. The image has been taken from the original paper [25].

ral Network and classical handcrafted features (e.g., Longest common subsequence, Word overlap, etc.). These features are then combined as a ranking model by a gradient boosted regression tree, implemented by Xgboost [12].

The ANN is used to generate a similarity score between the pairs question comment. This is done by using two Bidirectional LSTM (BiLSTM), one for the question and one for the comment, and then by using a Convolution Neural Network on the resulting hidden states. The result is then passed to a fully connected layer that makes the final score for the labels ('Good' or 'Bad'). A graphical representation of the model can be found in Figure 3.9.

Other models, proposed in the competition, used ANNs to generate a similarity score. A quite original one is called Siamese CNN [23] that cleverly generate an attention mechanism from two distinct CNN networks. We do not report the results of all the teams that participated in the competition, but interested reader may find it in the organiser paper [60].

Chapter 4

Proposed Models

In this chapter, we describe the models produced to solve the three Question Answering tasks studied in this thesis: *Community Question Answer* (CQA), *Reasoning Question Answering* (RQA), and *Reading Comprehension* (RC). The first task is tested using the CQA data proposed at SemEval2017 [1]. For the second, we used the bAbI tasks described in the previous section, and in the last, we used the CNN news article dataset, also previously described. In the next Chapter, we are going to describe in deep the data splitting and the eventual re-sampling. In this, instead, we describe the general formulation of our proposed models.

In what follows, we first describe the ThRee Embedding Recurrent Neural Network [2], and then the Question Dependent Recurrent Entity Network. At the beginning of each section, we provide a more formal description of each task.

4.1 ThRee Embeddings Recurrent Neural Network

This model has been designed to solve the CQA task[2], which can be formulated as follows:

CQA- Given a question q and 10 comments $c_1,\dots,c_{10}$, rank such comments from the most relevant to the least one with respect to q, and assign to each one a label "Good" if the comment answers the question and "Bad" vice versa.

Our model creates pair question comment and tries to assign a label and a score to each of them. The proposed model exploits both syntactic and semantic information to build a single

[1] http://alt.qcri.org/semeval2017/task3/

[2] Another sub-task proposed at SemEval2017 was to rank similar questions. However, we do not describe such task since it is more related to a question to question similarity rather than a question answer task.

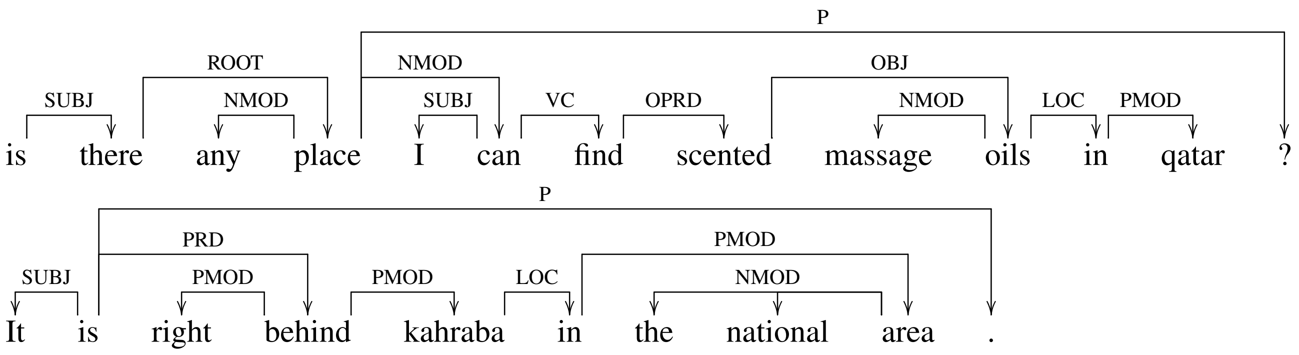

P
ROOT
NMOD
OBJ
SUBJ
NMOD
SUBJ
VC
OPRD
NMOD
LOC
PMOD
is there any place I can find scented massage oils in qatar ?
P
PRD
PMOD
SUBJ
PMOD
PMOD
LOC
NMOD
It is right behind kahraba in the national area .

Fig. 4.1 Dependency parsing tree of two sentences taken from a question and a comment in the training set. In this example the first input $x(t)$ of the RNN encoder is going to be: <"is",SUBJ,"there","is",SUBJ,"It">.

and meaningful embedding space. This idea has been frequently used in this field [38, 89], in particular by properly combine Word Embedding in order to encode semantic information into a single vector. In our model, we exploit both semantic and syntactic information, through the use of Word Embedding and a Dependency Parser, that are used to build a single vector. The latter becomes the input of an Artificial Neural Network, that gives the final score. The CQA data of SemEval2017 were quite dirty, then we first pass it through a series of pre-processing steps. Then we derived several additional features that are used together with the raw input. We formally define the aforementioned process in the next sections.

4.1.1 Data Preprocessing

We applied standard pre-processing to question and comment body. This also helped to achieve better performance during syntactic parsing and better alignment of our vocabulary to the pre-trained embedding one. Pre-processing included the following steps:

- Portions of text that include HTML tags and special sequences were removed or substituted with simpler strings.
- Using a set of regular expressions, we replaced URLs, nicknames, email addresses with a placeholder for each category.
- Too long repetitions of characters inside tokens were replaced by a single character (e.g. *loooot* became *lot*). Indeed, in the language spoken on community forums, letters are often repeated to emphasise words; with our approach, we were able to reconstruct their standard form. Moreover, multiple punctuations were also collapsed.

- Standard use of spacing after punctuation was restored, in order to avoid problems during tokenization.
- Using a hand-written dictionary, the most common abbreviations were replaced with the corresponding extended form.

We then performed sentence splitting and tokenization using *nltk* [8]. During the tokenization step, we performed spelling corrections.

Finally, texts were analysed using Tanl pipeline [3], adding morpho-syntactic and syntactic information (i.e., part of speech tagging and dependency parsing). Figure 4.1 shows an example of question and comment which are parsed accordingly.

4.1.2 Additional Features

After that, we generated several features, representing both metadata and properties of the couple Question-Comment. These features have been commonly used in literature, both using Neural Networks [56], and SVM models [53]. The latter can include additional and potentially relevant information not easily conveyed through semantic representations. In our case, they are used as additional input besides the last hidden state of the RNN encoder. Features can be grouped as follows:

- Features encoding information about the standard similarity between question and comment (all measures are expressed in terms of the number of tokens):
  - size of intersection between question and comment
  - Jaccard Coefficient (ratio between intersection size and union size of question and comment)
  - comment length
  - ratio between comment length and question length
  - length of the longest common subsequence between question and comment
- Features encoding metadata information, in particular:
  - number of the comment in default ordering
  - whether the comment was posted by the same user asking the question
  - whether the user posting the comment had already posted a comment for the same question

- Features encoding presence of certain elements in comment body, in particular, we looked for:
  - presence of question marks
  - presence of URLs (through regex)
  - presence of username (through regex)
  - presence of a username among those that are authors of comments preceding the considered one

4.1.3 Model

The proposed model[3] makes use of the previous steps (i.e. dependency parser) whose output is a tree, to generate a sequence of triples. The *i*th triple is made of $< W_i, rel, W_r >$, where W_i is the *i*th word of the text and W_r is the word associated through *rel* (i.e. the dependency relation extracted by the parser). Then triples $< e_i, rel, e_r >$ are generated, where e_i and e_r are word-embeddings vectors for the two words, and *rel* is a 1-hot-encoding of the dependency relations. The *k*th input to be fed into the encoder is simply made by concatenating the *k*th embedding triple of the comment with the *k*th one of the question. Figure 4.1 shows an example of how to obtain a valid input for our model.

The goal is to let the system learn the correct composition rule through syntactic dependencies. Hence, the input becomes dual: a sequence of triples which represents the question and another sequence for the comments. In formulas, a sample x can be described as:

$$x^{(t)} = < e_{Qi}^{(t)}, rel_Q^{(t)}, e_{Qr}^{(t)}, e_{Ci}^{(t)}, rel_C^{(t)}, e_{Cr}^{(t)} >$$

where t represent the generic tuple included in the input x, Q the question and C the comment. These are then passed to a sentence encoder, which is a Recurrent Neural Network (RNNs), that is used to return a single output aiming to represent the entire sequences. The RNN equation and its variants are shown in Chapter 2.

Then the RNN output, that is its final hidden state, along with a vector made up of the additional features, become the inputs passed to the final feed-forward layers which perform the scoring. This is a standard multilayer ANN, described in the background Chapter, in

[3]An implementation is available at https://github.com/AntonioCarta/ThreeRNN

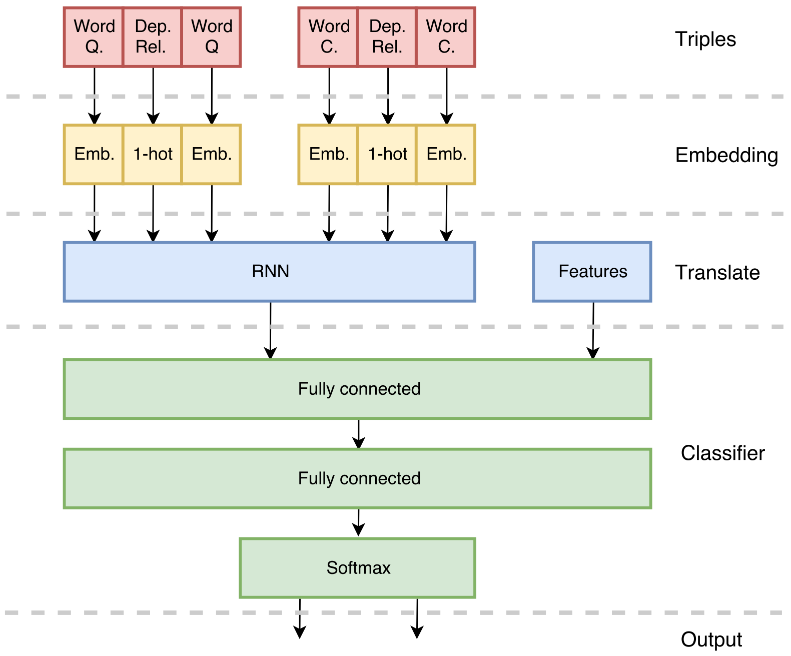


Fig. 4.2 Conceptual schema of the model used for the classification.

which the final layer is Softmax over two classes: ’Good’ and ’Bad’. To obtain the final ranking we took the probability of a given input to be labelled as the positive class (’Good’). The entire network is finally trained using Backpropagation Through Time (BPTT) algorithm using a cross-entropy loss function (H). Figure 4.2 shows the conceptual schema of the model.

The final re-rank, required by the task, has been done by sorting the 10 comments by their score. In Chapter 5, we discuss implementation details, conducted experiments, and obtained results. In the next section, instead, we describe the Question Dependent Recurrent Entity Network, used for the other two tasks.

4.2 Question Dependent Recurrent Entity Network

In this section, we describe the model used to solve RQA and RC tasks. These tasks are semantically different but they can be formulated in the same way, such as:

RQA/RC- the training data consists of tuples $\{(x_i, y_i)\}_{i=1}^{n}$, with n equal to the sample size, where:

- x_i is composed by a tuple (T_i, Q_i), where T_i is the text made of a set of sentences $\{s_{i1}, \ldots, s_{it}\}$ where t represent the number of sentence in T_i, and Q_i is a single sentence representing the question.
- y_i is a single word that represents the answer to the question Q_i. For instance, in the RC task, the set of possible answers is restricted to the entities (e.g., Persons, Organizations, etc.) present in T_i. Instead, in the RQA (i.e., bAbI tasks) the set of possible answers is selected among all the words in vocabulary V.

In Chapter 3, State-Of-The-Art, we have shown different models that achieve good results in both RQA and RC. In this section, we present an improvement of the Recurrent Entity Network (REN) [32] model. The latter is the only model able to pass all the 20 bAbI tasks, presented in the previous Chapter, with the 10k sample setting. However, this model fails many tasks with the 1K setting, and it has not been tried in a challenging RC datasets, like the CNN news article. In this thesis, we deeply studied this model, and we modified its core part in order to overcome that performance lacks. We called this elaborated model Question Dependent Recurrent Entity Network (QDREN)[4]. In the following, we introduce the model, and we motivate the architectural choice. In the description, we show the general structure of the REN model, and we specify the modification made in order to obtain the QDREN.

4.2.1 Model

The Recurrent Entity Network follows the Memory Network framework described in the previous Chapter. Indeed, it has the three classical components: Input Encoder, Dynamic Memory (G in the Memory Network framework), and the Output Module. In the following paragraphs, we provide a detailed description of each component.

[4]An implementation is available at https://github.com/andreamad8/QDREN

Input Encoder

The input of the model is a set of sentences[5] $\{s_{i1},\dots,s_{it}\}$ that represent the text T_i, where each sentence s_{it} is a sequence of m words $\{w_{i1},\dots,w_{im}\}$. The input encoder transforms the set of words of a sentence s_{it} into a single vector representation by using a multiplicative mask. Let's define $E \in \mathbb{R}^{|V|\times d}$ the embedding matrix, that is used to convert words to vectors, i.e. $E(w_{im}) = e_{im} \in \mathbb{R}^d$. Hence, $\{e_{i1},\dots,e_{im}\}$ are the word embedding of each word in the sentence s_{it}. The multiplicative masks for the sentences are defined as $f^{(s)} = \{f_1^{(s)},\dots,f_m^{(s)}\}$, where each $f_i \in \mathbb{R}^d$. The encoded vector of a sentence is defined as:

$$s_{it} = \sum_{r=1}^{m} e_{ir} \odot f_i^{(s)}$$

where $\odot$ is the Hadamard product (elementwise product). On the other hand, if we considered $\{e_1,\dots,e_k\}$ as the embedding of the question's words, and $f^{(q)} = \{f_1^{(q)},\dots,f_m^{(q)}\}$ as the multiplicative masks, then the encoded vector for the question can be defined as:

$$q = \sum_{r=1}^{k} e_{ir} \odot f_i^{(q)}$$

If we fix the generic f_i as the all one vector, then it becomes simply the sum of each word vector (a.k.a. Bag of Words). Moreover, there could be several options to explore for an input encoder such as: a RNN, Positional encoding [75], or any other ways to combine word vectors. The choice of the multiplicative mask is mainly guided by its simplicity and effectiveness. In the future, we plan to explore other alternatives. Now on, we denote a generic encoded sentence with s_t without indicating the sample index.

Dynamic Memory

This module is the core component of the model. Indeed, it stores information of entities present in the text T. This module is very similar to a Gated Recurrent Unit (GRU) with a hidden state divided into blocks. Moreover, each block ideally represents an entity, and it stores relevant facts about it.

Each block i is made by a hidden state $h_i \in \mathbb{R}^d$ and by a key $k_i \in \mathbb{R}^d$, where d is the embedding size. Therefore the Dynamic Memory module is made of a set of blocks, which can be represented by the following two sets: hidden states $\{h_1,\dots,h_z\}$, and a set of keys

[5]In the RC task, we used a set of windows as input instead of the sentence directly. But for the model does not make any difference.

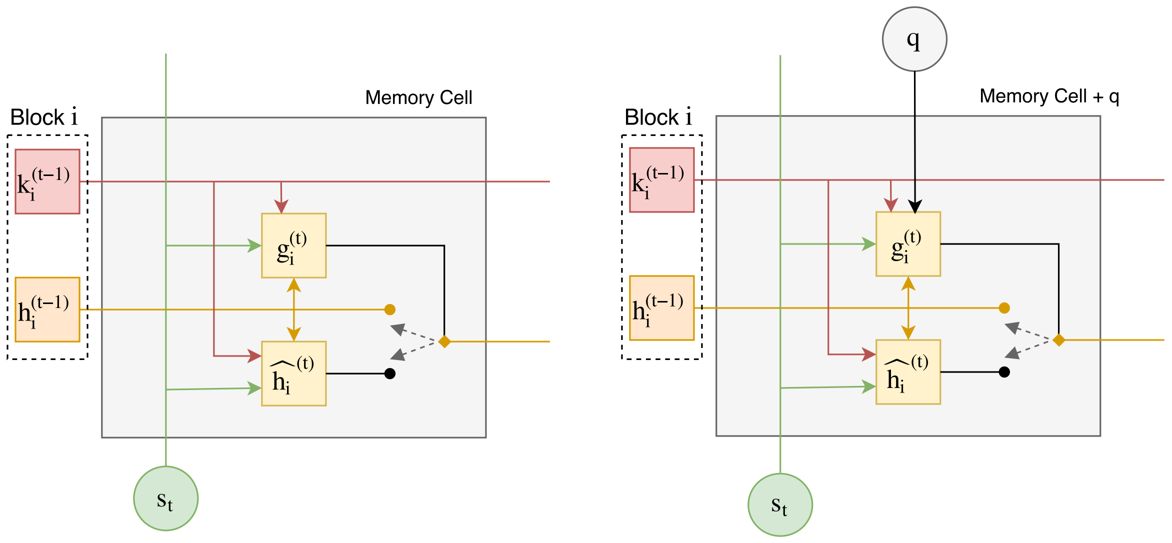


Fig. 4.3 Conceptual scheme of how a memory cell propagates its hidden state and its key. On the left-hand side the standard cell, and on the right the memory that considers also the question to open the gate.

$\{k_1,\dots,k_z\}$. Now, we formally define how to update the hidden states (h_i) of a generic block i depending on a sentence s_t, encoded into a single vector by the previous module. For a generic block i, we have the following equations:

$$\begin{aligned}
g_i^{(t)} &= \sigma(s_t^T h_i^{(t-1)} + s_t^T k_i^{(t-1)}) && \text{(Gating Function)}\\
\hat{h}_i^{(t)} &= \phi(U h_i^{(t-1)} + V k_i^{(t-1)} + W s_t) && \text{(Candidate Memory)}\\
h_i^{(t)} &= h_i^{(t-1)} + g_i^{(t)} \odot \hat{h}_i^{(t)} && \text{(New Memory)}\\
h_i^{(t)} &= \frac{h_i^{(t)}}{\|h_i^{(t)}\|} && \text{(Reset Memory)}
\end{aligned}$$

where σ represents a sigmoid function, ϕ a generic activation function which can be chose among a set (e.g. sigmoid, ReLU, etc.). $g_i^{(t)}$ is the gating function which determines how much the ith memory should be updated, and $\hat{h}_i^{(t)}$ is the new candidate value of the memory to be combined with the existing one $h_i^{(t-1)}$. The matrix $U \in \mathbb{R}^{d\times d}$, $V \in \mathbb{R}^{d\times d}$, $W \in \mathbb{R}^{d\times d}$ are **shared** among different blocks, and are trained together with the key vectors.

The final normalization step makes the model to forget previous information. To see this, note that since the memories lie on the unit sphere, all information is contained in their phase. Adding any vector to a given memory (other than the memory itself) will decrease the cosine distance between the original memory and the updated one. Therefore, as new information is added, old information is forgotten.

Our main contribution in this module is to the gating function. As we saw in other models described in the State-Of-The-Art Chapter, the question should be taken in consideration while we analyse the sentence in input. The gating function that we have introduced in the previous equation completely ignore the question. Furthermore, the model has been actually designed to accumulate information about all the facts independently from the question in input. Even though, this solution works well for generic tasks, it is not the best choice in the Question Answering case. To overcome this issue, we proposed a new gating function that take in consideration the question in input. Given the question $q \in \mathbb{R}^d$, encode as the sentences but with another set of masks, the new gating function is defined as:

$$g_i^{(t)} = \sigma(s_t^T h_i^{(t-1)} + s_t^T k_i^{(t-1)} + s_t^T q)$$

The addition of this last term can be considered as a specialization of the original model for question answering tasks. Hence, the name Question Dependent Recurrent Entity Network (QDREN). Figure 4.3 shows the graphical representation of the memory cell, with and without the question.

Output Module

This module is very similar to the one proposed in the End To End Memory Network [75]. Basically, it is doing a scoring of the memory (hidden states) using again the question q. This step is done by using a Softmax layer which creates a probability distribution over the memories (hidden states). Thus, the hidden states are summed up, using the probability as weight, to obtain a single state which represents the entire memory content. Finally, the network output is obtained by combining the final state with the question. The latter steps can be formalised by the following equations:

$$\begin{aligned} p_i &= \text{Softmax}(q^T h_i) \\ u &= \sum_{j=1}^{z} p_j h_j \\ \hat{y} &= R\phi(q + Hu) \end{aligned}$$

where $R \in \mathbb{R}^{|V| \times d}$, $H \in \mathbb{R}^{d \times d}$, $\hat{y} \in \mathbb{R}^{|V|}$, z is the number of blocks, and ϕ can be chosen among different activation functions.

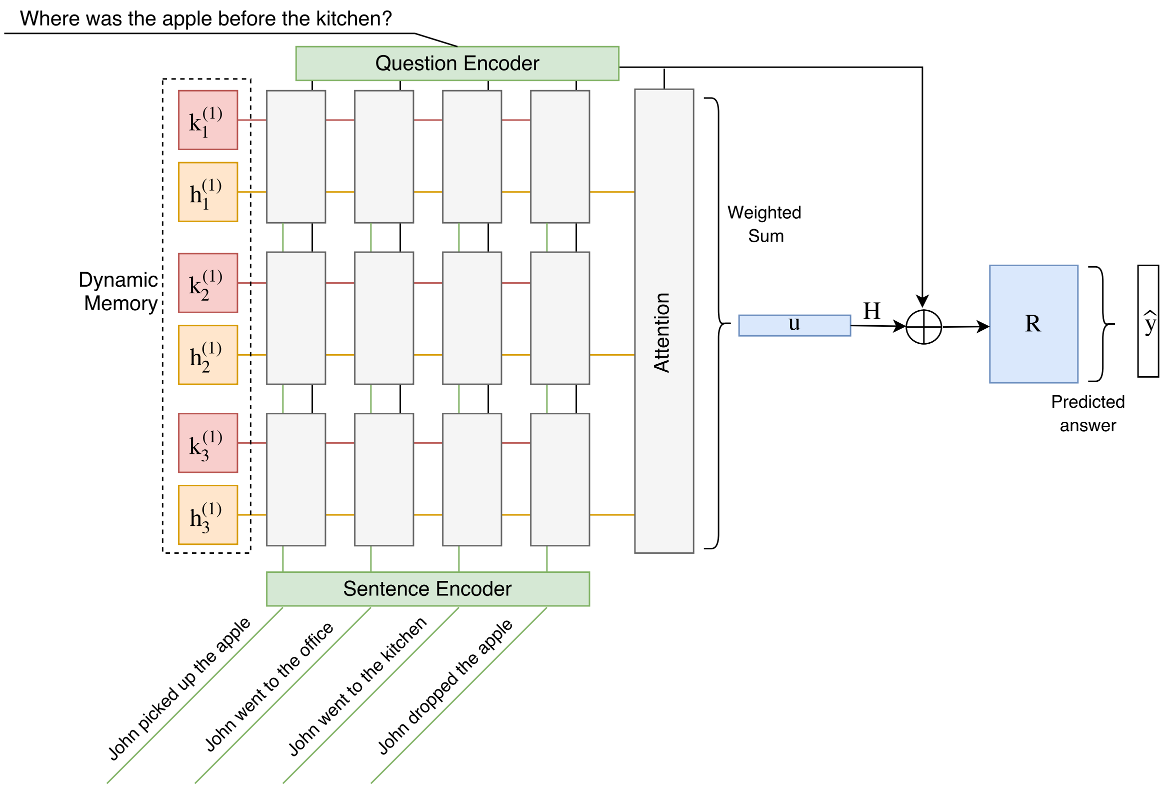


Fig. 4.4 Conceptual schema of the QDREN model, with three memory blocks. In input a sample taken from bAbI task dataset.

Finally, the model uses a cross entropy loss $H(\hat{y}, y)$, where y is the one hot encoding of the answer. Overall, the trainable parameters of this model are the following:

$$\Theta = [E, f^{(s)}, f^{(q)}, U, V, W, k_1, \ldots, k_z, R, H]$$

where $f^{(s)}$ refers to the sentence multiplicative masks, $f^{(q)}$ to the question multiplicative masks, and each k_i to the key of a generic block i. The number of parameters is dominated by E and R, since they depend on the vocabulary size. However, R is normally is much smaller than E like for the RC task, in which the prediction is made on a restricted number of entities and not the entire vocabulary[6]. All the parameters are learned using the Backpropagation Through Time (BPTT) algorithm. A schematic representation of the model is shown in Figure 4.4.

[6]Therefore $R \in \mathbb{R}^{|entities| \times d}$

4.2.2 Details and Discussion

After this general description of the model, we discuss several details and modification we made to it. The key points are listed in the following.

- in all the equations the activation function ϕ can be freely chosen, in the original implementation they used a parametric ReLU function [31] in both memory and output equation. However, in the RC task, this activation function results to be hurtful for the performance. Thus, we preferred to use a simple sigmoid activation function (σ).
- the original model was designed mainly for bAbI tasks with a setting of 10K, hence it basically did not have any over-fitting problems. However, in both RC and bAbI 1K setting tasks, we added an L2 regularisation term to the loss function and a dropout layer in front of the embedding matrix. In several bAbI tasks, this helped the network to better generalise.
- the original paper proposed also an interesting setting of this network which is called tied scheme. In this case, the keys k_i were fixed (not trained) to the word embedding of an entity. In all the experiment of the original article, this scheme did not increase the performance of the model, it actually gives the worst result. Indeed, the model tries to build a distributed representation of the information among the memories, i.e. it stores the information of a given entity in all the memories.
- if needed, a simplified version of the model can be obtained by removing the output module, and by considering the candidate memory $\hat{h}_i^{(t)}$ simply as the sentence s_t, instead of as a combination of the previous hidden state. This version can work if there are a small number of candidates[7] since without output module we have already a distribution over the candidates. Obviously, this choice is not realistic since normally the task requires to find the answer among all the words in the text (or among its entities), and not from a shortlist of candidates.

[7]Like for the Children's Book Text, where 10 candidates are given.

Chapter 5

Experiments and Results

In this chapter, we present experiments and results of our proposed models. We conducted experiments in three datasets, one for each QA task discussed before, which are: SemEval2017 data (CQA), bAbI tasks (RQA), and CNN news article dataset (RC). Previously, we shortly described several datasets, either way, we give further details and statistics of the dataset used in the experiments. Moreover, in each section, we discussed the experiment setting and we analyse the obtained results.

The models we build required a lot of computational power since we basically need to learn high dimensional vectors. For instance, in the RC task, 5 million parameters must be updated, since for each word in the corpus (50K words) we have a 100-dimension vector. Moreover, the datasets we used have got a lot of samples, like in the case of CNN news article (380K samples). For the latter reasons, for running the experiments, we used two machines equipped with four GPUs each. The first has got four NVIDIA Tesla K80 Accelerator[1], and the second has got four NVIDIA Tesla P100 accelerators[2] based on the new NVIDIA Pascal GPU architecture[3].

In the next three sections, we describe the implementation details, experiments and results of all the models we built.

[1]http://www.nvidia.com/object/tesla-k80.html
[2]http://www.nvidia.com/object/tesla-p100.html
[3]http://www.nvidia.com/object/gpu-architecture.html

5.1 Community Question Answer: SemEval2017

SemEval2017 organizers already provided the training and the validation set[4] (called Dev set by them). However, the test set was not provided at the beginning of the competition. Indeed, each competitor has to submit their best model and the system automatically evaluate it. The evaluation score is based on the Mean Average Precision (MAP) achieved in the test set. The winner achieved a MAP equal to 88.431, compared to our model, ranked 7-th, which scored a MAP of 83.429. The organisers provided a description paper [60] that includes task challenge, participant models, and all the final scores. In this thesis, we do not report detailed results of all the participants, since it would just be a copy and paste from the organiser paper. Instead, we prefer to describe implementation details of our model. Interested readers can visit this website for more information: http://alt.qcri.org/semeval2017/

The provided data are summarised in Table 5.1, where we also show the sample subdivision by label kind. Each sample in the table represents a triple: question, comment, and label. Even though we used the label to train our model, the Mean Average Precision (MAP) score is used to test the performance of the model. The latter, it is considering the ranking position of the 10 comments. Thus, ones the model generate the prediction (probability of the 'Good' class), we re-ranked the 10-comment based on their probability score. In this way, we obtained the final order used to calculate the MAP.

Table 5.1 Summary of training, validation and test sets provided by SemEval2017 organisers. The Sample column represents the number of example size, Good and Bad, instead, the subdivision of the samples by their label.

	Sample	**Good**	**Bad**
Training	17900	6651	11249
Validation	2440	818	1622
Test	2930	1230	1700

5.1.1 Experiments

In this task, we used the ThRee Embeddings Recurrent Neural Network, described in the previous chapter. This model has been implemented using Keras [14], a python library which provides an efficient and easy-to-use deep learning utilities. To perform model selection, we merged training and validation samples provided by Semeval organisers. Then, we shuffled

[4]http://alt.qcri.org/semeval2017/task3/index.php?id=data-and-tools

Table 5.2 Hyper-parameters used during model selection. The selected parameters are in bold.

Parameter	**Values**
RNN	LSTM, **GRU**, SUM
RNN layer	1,**2**,3
Hidden layer	1,2,**3**
Embeddings size	**100**, 200, 300
Hidden size	**50**, 100, 200
Max length	5,**10**,25,50,75,100,150
Dropout	0, **0.1**, 0.2, 0.3, 0.4
L2	0.01, **0.001**, 0.0001, 0.00001
Activation	ReLU, tanh, **sigmoid**
Optimizer	**adam**, rmsprop

it and we extracted a training and a validation set (10% of the samples).

We selected various hyper-parameters, shown with their values in Table 5.2, such as learning rate, the number of hidden units and hidden layers for the recurrent and feed-forward layers, dropout, L2 regularization, activation functions (i.e. ReLU, sigmoid and hyperbolic tangent), optimization algorithms (i.e. Adam [44] and RMSprop [81]). We limit the number of triples in input to the RNN with a threshold, that is also part of the hyper-parameter (i.e., Max length); if the comment/question is shorter, it is filled up with zeros ("null triples"). Since each training required quite a large amount of time, we opted for a random search technique [6].

The Word Embedding matrix is initialized with pre-trained Embeddings[5], and it remains fixed during the training phase (not learned). We tried to update them together with the entire network during training but the resulting network always ended up to over-fit. Two different types of Word Embedding have been evaluated: GloVe [65], which are trained using Wikipedia, and another one trained directly with questions and answers extracted from Qatar Living forum [52]. However, in our model both Embedding worked fine; Indeed, using the latter, which was tailored for this particular task, we did not achieve any improvements.

To encode the input into a single embedding we compare three different approaches: SUM (which sums all the triples given as input), LSTM and GRU. This is also included in the hyper-parameters selection.

[5]This technique is also known as Transferring Learning.

Table 5.3 Test results of the submitted models. We also report the result from the winner model KeLP [26] and the second runner [25].

	MAP	**Acc**	**AvgRec**	**MRR**	**P**	**R**	**F1**
Baseline (IR)	72.61	-	79.32	82.37	-	-	-
Primary	83.42	68.02	89.90	90.32	73.82	59.62	65.96
Contrastive	82.87	68.67	89.64	89.98	77.28	56.27	65.12
KeLP	88.43	73.89	93.79	92.82	87.30	58.24	69.87
MSRA	88.24	51.98	93.87	92.34	51.98	100.00	68.40

5.1.2 Results

We selected our best model by taking the one that achieved the highest MAP in the validation set. The competition allowed also to make a contrastive submission, so we also tested our second-best model. The primary submission uses GRU, instead, the contrastive model uses SUM as aggregation. Using SUM, which is computationally less expensive than the RNN, we obtained just a slightly worst MAP (i.e. around 0.5%), which suggests we did not fully exploit the RNN potentiality. Moreover, there is a trade-off between representation length and computational costs, achieved with the use of the length threshold; this may be regarded as a crucial choice for our model. In Table B.1 in Appendix B we report the best and the worst MAP values achieved in the validation set.

Even though the competition used the Mean Average Precision (MAP) for the ranking, different measures have been used to evaluate the general performance of a model. For instance there are other considered measures in the evaluation: **Acc** that is the accuracy on the label, **AvgRec** that is the average recall[6], **MRR** is the mean reciprocal rank[7], **P** is the precision [8], **R** is the Recall, and **F1** is the F1-score[9] with respect to the 'Good' label. All the results obtained in the test set are summarised in Table 5.3. We also include the results of the winner (KeLP) and the second runner (MSRA). These models have been described in Chapter 3.

In conclusion, our model greatly surpasses the baseline models by 11 percent points more, and it achieve 5 percent points less compare to the competition winner. We have also noticed that achieving more accuracy in the label does not always help to obtain a higher MAP.

[6]Where recall is defined as Recall $= \frac{tp}{tp+fn}$, *tp* stand for true positive, and *fn* for false negative

[7]https://en.wikipedia.org/wiki/Mean_reciprocal_rank

[8]Where precision is defined as Precision $= \frac{tp}{tp+fp}$, *tp* stand for true positive, and *fp* for false positive.

[9]That is defined as $F = 2\frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$

Interested readers can have further details by reading the paper ‘FA3L at SemEval-2017 Task 3: A ThRee Embeddings Recurrent Neural Network for Question Answering’ [2] accepted in the SemEval2017 conference.

5.2 Reasoning Question Answering: bAbI tasks

As discussed, in the State-Of-The-Art chapter, the bAbI dataset includes 20 different tasks to test the reasoning ability of a machine. These tasks are synthetically created, and they are present in 2 different sample sizes: 1K and 10K. Table 5.4 shows an example taken from one of the bAbI task (number 3). In the in the previous chapter, we have shown the

Table 5.4 Sample from bAbI Task 3: Three Supporting Facts

Story	**Question**
John picked up the apple John went to the office John went to the kitchen John dropped the apple	Where was the apple before the kitchen?
	Answer
	office

results for the 10K setting, and we saw that Recurrent Entity Network (REN) was the only model able to pass it all (Error lower than 5% in each task). However, in the 1K setting the REN does not achieve good results, indeed it just passes 5 tasks. Moreover, End To End Memory Network [75] in the 1K setting represents the State-Of-The-Art, passing 9 tasks.

All the 20 tasks are present as a set of distinct tuples $\{(x_i, y_i)\}_{i=1}^{n}$, with n equal to the sample size, as explained in the previous Chapter. Each task provides the following amount of data (already split): 900 training samples, 100 validation samples, and 1000 Test samples. We keep using this splitting in order to compare our model with the existing ones. We also consider a task passed, if the model achieved an error in accuracy lower than 5%.

5.2.1 Implementation

For this task, we implemented the Question Dependent Recurrent Entity Network, described in the previous Chapter. The original REN model has been implemented, by the authors [32], using Torch 7 [16] a deep learning library written in Lua. However, we found the latter implementation hard to understand and to manipulate as we desired. Therefore, we decided to implement the QDREN model from scratch using TensorFlow v1.1 [1].

This library is an open source software for numerical computation that makes use of data flow graphs. We choose it, because it provides a class interface to implement a customized version of an RNN cell (`tf.contrib.rnn.RNNCell`[10]). Indeed, once the cell has been

[10]https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/RNNCell

created, TensorFlow allows to use it as a predefined RNN cell, and so it can be plugged wherever we needed. Another, advantages of using this library is the ability to perform a fully dynamic unrolling of the input, which means we do not need to set a maximum length size for a sentence. For doing so, we just need to specify the length of each sequence to the method `tf.nn.dynamic_rnn`[11].

Moreover, Tensorflow, and also other libraries based on computational graphs[12], implements automatic differentiation. Thus, it computes the gradient of the implemented function, and it automatically updates the weights of all the parameters accordingly. This feature is very useful because it allows to easily implement models, avoiding to implement everything from scratch. Last but not least, the code written using Tensorflow can be executed using GPUs, achieving great performance.

5.2.2 Experiments

To select the best model, we perform a model selection on the validation set of each bAbi task. Thus, we selected 20 different hyper-parameters, one for each task. We selected various hyper-parameters, shown in Table 5.5, such as: number of memory blocks, learning rate, L2 regularization (λ), and dropout. In this task, we fixed the batch size to 32, we do no used any pre-trained word embedding, and we used Adam [44] optimizer. We have also clipped the gradient to a maximum of 40 (to avoid gradient explosion), and we set the word embedding size to 100, as it has also been suggested in the original paper. We have also implemented an early stopping method, which stop the training ones the validation accuracy does not improve after 50 epochs.

Table 5.5 Hyper-parameters used in the model selection for each bAbI task.

Parameter	**Values**
Learning Rate	0.01,0.001,0.0001
Number of Blocks	20,30,40,50
λ	0,0.001,0.0001
Dropout	0.3,0.5,0.7

Some initialization led to very poor results (in the validation), thus we repeated the experiment several times. For each task, we selected the model that achieved the highest accuracy

[11] https://www.tensorflow.org/versions/r1.2/api_docs/python/tf/nn/dynamic_rnn

[12] Theano, Torch, Chainer etc.

in validation. Once we selected the best model, we estimate its generalization error using the provided Test set.

The best hyper-parameter for each task are shown in Table A.1 of Appendix A. In Table 5.6, we show the values of the loss function and the accuracy for the training, validation and test sets of the selected model. A common setting among different tasks is to use 20 blocks of memories; This also seams a good trade-off, since on average the vocabulary size is between 20 to 40 words (possible entity candidates). Just in few case the L2 regularization results to be useful, and then really improve the performance. However, in the failed tasks we are still over-fitting in the training data, even though in the model selection we tried a quite strong regularization (both L2 and dropout). This can be due to the fact that we are using just 1K samples, and also that our model does not have enough generalization power.

Table 5.6 Summary of the selected model for the 20 bAbI task. The elements in bolt show the task with less than 5% of error, thus considered as passed.

	Loss			**Accuracy**		
Task	Training	Validation	Test	Training	Validation	Test
1	0.008	0.000	0.000	1.00	1.00	**1.00**
2	1.519	1.735	1.665	0.40	0.38	0.32
3	0.942	1.588	1.775	0.64	0.48	0.39
4	0.048	0.011	0.021	0.99	1.00	**1.00**
5	0.107	0.228	0.083	0.96	0.96	**0.98**
6	0.097	1.044	1.031	0.97	0.74	0.71
7	0.017	0.002	0.032	0.99	1.00	**0.99**
8	0.151	0.354	0.221	1.00	0.95	**0.97**
9	0.131	0.176	0.222	0.98	0.98	**0.95**
10	0.077	0.067	0.112	0.98	0.99	**0.96**
11	0.013	0.004	0.016	1.00	1.00	**0.99**
12	0.106	0.035	0.036	0.98	1.00	**1.00**
13	0.161	0.157	0.159	1.00	1.00	**1.00**
14	0.132	0.467	0.611	0.99	0.91	0.84
15	0.050	0.041	0.011	0.98	0.99	**1.00**
16	0.965	1.223	1.211	0.65	0.56	0.48
17	0.589	0.692	0.734	0.74	0.69	0.63
18	0.112	0.317	0.251	0.97	0.91	0.90
19	2.276	2.451	2.450	0.23	0.17	0.15
20	0.007	0.000	0.005	1.00	1.00	**1.00**

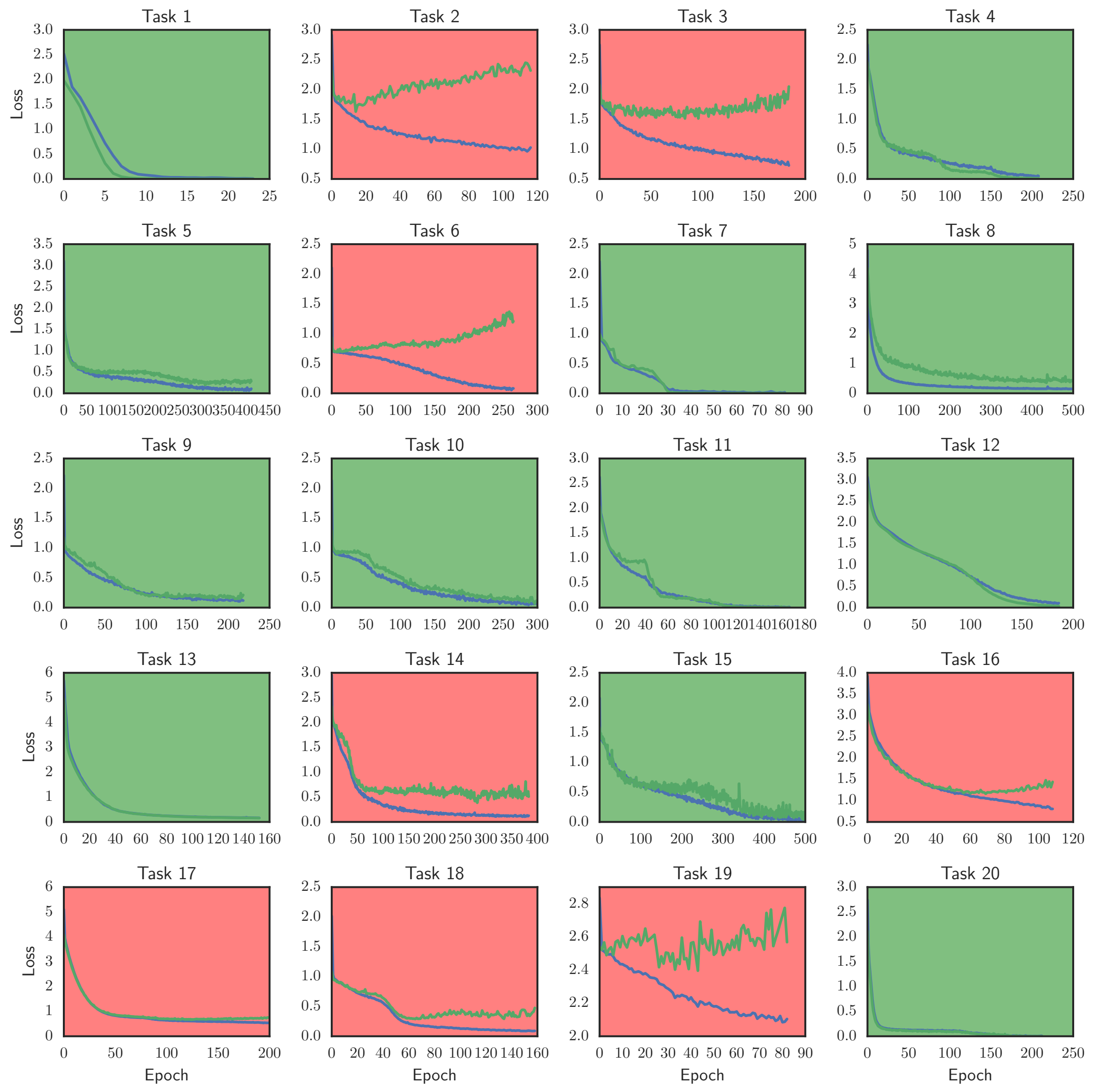


Fig. 5.1 Training and validation loss among epochs for all the 20 bAbI tasks. In the x axes we have the epochs and on the y axes the loss values. The blue line represents training values and the green one the validation. The back-ground of each image represents whether we pass (error less or equal to 5%) the task, green indicate passed, and red failed.

To better analyse the performance of our model (whether it over-fits), we plot training and validation curve of loss function and accuracy. Figure 5.1 and Figure 5.2 show the loss function and the accuracy respectively, of both training and validation set in all the 20 bAbI tasks. As we can notice, in several tasks (number 2, 3, 6, 19 in particular) the training loss decreases (accuracy increases), and the validation loss keep staying very high (typical over-fitting situation). This happens on task which are particularly difficult, like task number 3 which requires three facts among 130 sentences to get the correct answer.

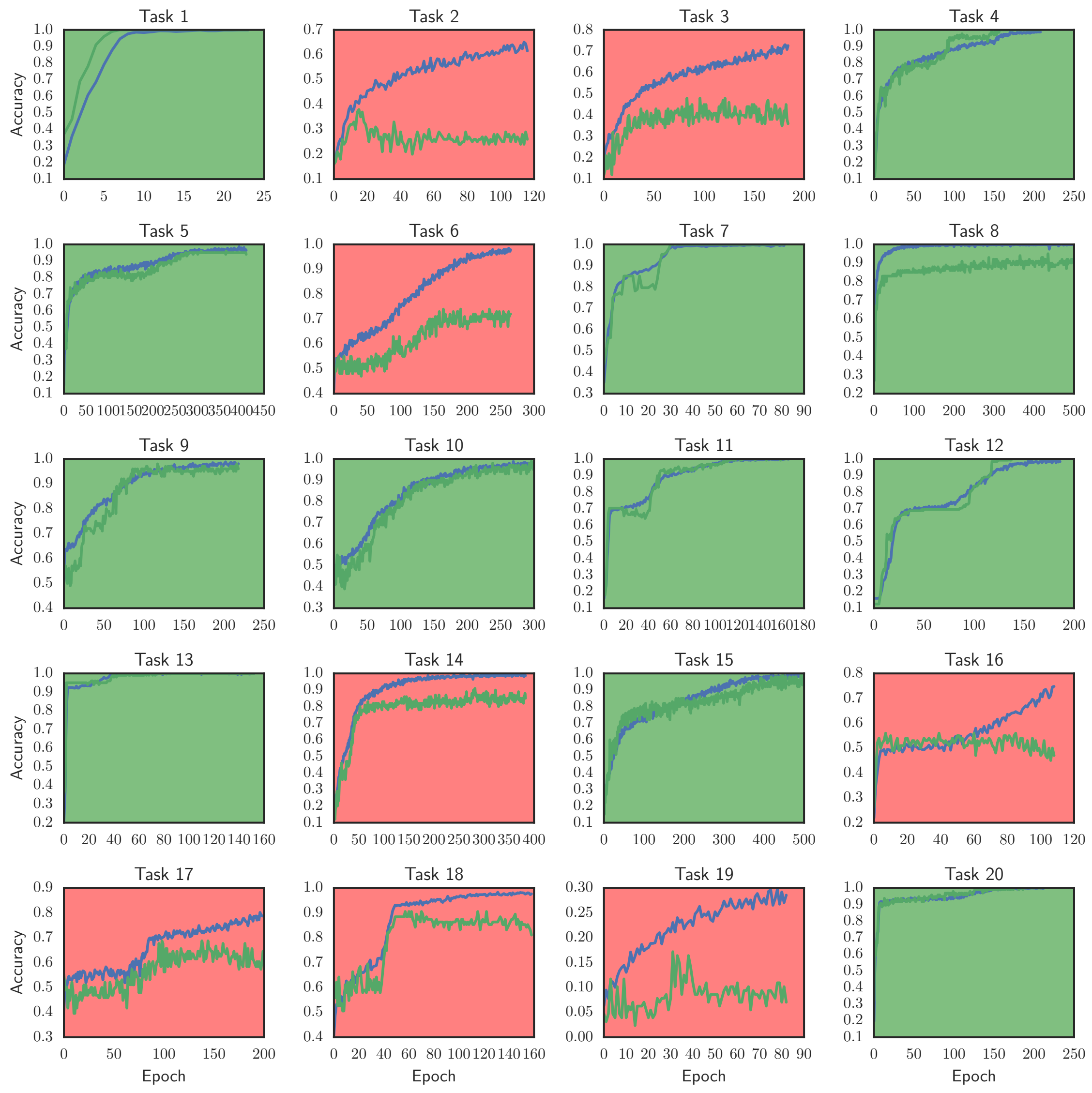


Fig. 5.2 Training and validation accuracy among epochs for all the 20 bAbI tasks. In the x axes we have the epochs and on the y axes the accuracy values. The blue line represents training values e and the green one the validation. The back-ground of each image represents whether we pass (error less or equal to 5%) the task, green indicate passed, and red failed.

We compared our results with four models: n-gram model, LSTM, original REN and End To End Memory Network (MemN2N) [75], which is currently the State-Of-The-Art in this setting. To the best of our knowledge we achieved the lowest number of failed tasks, failing just 8 tasks compared with the previous State-Of-The-Art which was 11. Comparing our QDREN with the original Recurrent Entity Network (REN) we achieved, on average, an improvement of 11% in the error rate and we passed 7 tasks more.

Table 5.7 Comparisons between n-gram, LSTM, QDREN, REN and End To End Memory Network (MemN2N). All the results have been take from the original articles where they were firstly presented. In bold we highlight the task in which we greatly outperform the other 2 models.

Task	**n-gram**	**LSTM**	**MemN2N**	**REN**	**QDREN**
1	64.0	50.0	0.0	0.7	0.0
2	98.0	80.0	8.3	56.4	67.6
3	93.0	80.0	40.3	69.7	60.8
4	50.0	39.0	2.8	1.4	0.0
5	80.0	30.0	13.1	4.6	**2.0**
6	51.0	52.0	7.6	30.0	29.0
7	48.0	51.0	17.3	22.3	**0.7**
8	60.0	55.0	10.0	19.2	**2.5**
9	38.0	36.0	13.2	31.5	**4.8**
10	55.0	56.0	15.1	15.6	**3.8**
11	71.0	28.0	0.9	8.0	0.6
12	91.0	26.0	0.2	0.8	0.0
13	74.0	6.0	0.4	9.0	**0.0**
14	81.0	73.0	1.7	62.9	15.8
15	80.0	79.0	0.0	57.8	**0.3**
16	57.0	77.0	1.3	53.2	52.0
17	54.0	49.0	51.0	46.4	37.4
18	48.0	48.0	11.1	8.8	10.1
19	10.0	92.0	82.8	90.4	85.0
20	24.0	9.0	0.0	2.6	0.2
Failed Tasks (>5%):	20	20	11	15	8
Mean Error:	65.9	50.8	13.9	29.6	18.6

Table 5.7 shows the error rate obtained using each compared model. We improve the mean error compared to the original REN, however we still do know reach the error rate achieved by the End To End Memory Network (even if we passed more tasks). It is worth to notice the following two facts: Firstly, in task 14 and 18 the error is very close to the threshold for passing the task (5%); Secondly, in task 2, we achieved a slightly worst result (10% error more) compare to the original REN.

In conclusion, using QDREN we improved the accuracy in many bAbI tasks achieving a new State-Of-The-Art in this setting. However, some tasks are still suffering of over-fitting mostly due to the usage of very few sample in training. In the next section, we show results obtained using QDREN in a Reading Comprehension task.

5.3 Reading Comprehension: CNN dataset

In this section, we employ the Question Dependent Recurrent Entity Network (QDREN) for solving a Reading Comprehension Task. As we described in the State-Of-The-Art Chapter, this task is very challenging and many different models have been proposed. To test the performance of our model, we selected the CNN news article dataset [33] since it is considered as a standard benchmark. This dataset hides entities from the input paragraph, which makes the task even more challenging. To have a feeling of how difficult this task becomes after the entity anonymization, we show a sample of the dataset in Table 5.8. The

Table 5.8 Sample from the CNN news article dataset. We show both the raw article and the anonymized one, including the question in Cloze form and the expected answer.

Raw Article	**Anonymized Article**
(CNN)Robert Downey Jr. may be Iron Man in the popular Marvel superhero films, but he recently dealt in some advanced bionic technology himself. Downey recently presented a robotic arm to young Alex Pring, a Central Florida boy who is missing his right arm from just above his elbow. The arm was made by Limbitless Solutions, a ...	(@entity1) @entity0 may be @entity2 in the popular @entity4 superhero films , but he recently dealt in some advanced bionic technology himself. @entity0 recently presented a robotic arm to young @entity7 , a @entity8 boy who is missing his right arm from just above his elbow. the arm was made by @entity12 , a ...
Raw Highlights	**Question**
"Iron Man" star Robert Downey Jr. presents a young child with a bionic arm	"@placeholder" star @entity0 presents a young child with a bionic arm
	Answer: @entity2

CNN dataset has been already tokenized and cleaned, therefore we did not apply any text pre-processing. As it has been done in other models, the set of possible answers has been reduced to the set of hidden entities in the text, that are much less, around 500, compared to all the words (120K) in the vocabulary. Moreover, training, validation and test set were already split, thus we keep this division in order to be able to make comparisons with existing

Table 5.9 Summary of training, validation and test sets of the CNN news article dataset. In the Info columns we show several relevant statistics.

Sample		**Info**	
Train	380298	Vocabulary size	118497
Validation	3924	Avg story tokens	761.8
Test	3198	Avg. # entities	26.2

Table 5.10 Model selection's Hyper-parameters used in the CNN dataset.

Parameter	**Values**
Learning Rate	0.1,0.01,0.001,0.0001
Window	2,3,4,5
Number of Blocks	10,20,50,70,90
λ	0.0,0.001,0.0001,0.00001
Optimizer	Adam,RMSProp
Batch Size	128,64,32
Dropout	0.2,0.5,0.7,0.9

results. Since the proposed validation and test sets are very small compared to the Training set, as shown in Table5.9 together with several other statistics, we plot the distribution of the expected answers (single entity) for all the sets in Figure A.1 of Appendix A. The plot shows that the label's class are well sampled, since they have qualitatively the same distribution.

5.3.1 Experiments

The experiments have been conducted using the same implementation explained in the previous chapter, except for the activation function of the output layer (sigmoid instead of parametric ReLU[13]). Differently from the bAbI task, the input was not split into sentences, thus we divided the text into sentences using the dot token ("."). Notice that in many cases this splitting task is challenging, but in this case the input has been already cleaned and normalised. Thus, the slitting worked pretty well.

Since the sentence can be very long, we have also implemented a window-based approach. The same method has been used in the End To End Memory Network [75] as a way to encode the input sentence. This method takes each entity marker ($@entity_i$) and it creates a window of b words around it. Formally, $\{w_{i-\frac{(b-1)}{2}},\dots,w_i,\dots,w_{i+\frac{(b-1)}{2}}\}$, where w_i represent the entity of interest. For the question, a single window is created around the placeholder marker (the word to predict). Moreover, we add $2(b-1)$ tokens for the entities at the beginning and at the end of the text.

To check whether our QDREN could improve the existent REN and whether the window-based approach makes any difference in comparison with plain sentences, we separately trained four different models:

- **REN + SENT**: original model with sentences as input

[13]After several experiments we notice that such activation was hurting the model's performance.

- **REN + WIND**: original model using the window-based input
- **QDREN + SENT**: our proposed model with sentences as input
- **QDREN + WIND**: our proposed model using window-based input

For each of this model, we conduct a separated model selection using a various number of hyper-parameters, shown with their values in Table 5.10. The latter includes: Learning Rate, Window Size, Number of Blocks, L2 regularization (λ), Optimizer (i.e. Adam [44] and RMSProp [81]), Dropout value, and the Batch Size. We also bound the maximum number of sentences to the first 50, since, on average, each training sample has 30 sentences. As for the bAbI task, we used early stopping, thus we stopped the training once the validation accuracy does not improve for 20 epochs. Notice that, we considered the best validation accuracy and not the one at the last epoch. This was possible by saving the model's weights in the epoch in which we achieved the best accuracy in validation.

Since each training required a large amount of time[14], we opted for a random search technique [6], and we used just a sub-sample of the training set, i.e. 10K sample, for the model selection[15]. Obviously, this is not an optimal parameter tuning, since the model is selected on just 10K samples. Indeed, we noticed that the selected model, which is trained using all the samples (380K), tends to under-fit. However, it was the only way to try different parameters in a reasonable amount of time. In Table B.1 of Appendix B, we report the best and the worst accuracy values achieved in the validation set using the 10k training set.

Moreover, we also limited the vocabulary size to the most common 50K words, and we initialize the embedding matrix using Glove [65] pre-trained word embedding of size 100. Differently from SemEval competition, we keep updating the weights of the embedding matrix during the training phase. Indeed, the number of learnable parameters in our model is dominated by the Word Embedding matrix[16]. In a holistic view, the proposed model is learning the most suitable vector representation for each word.

As before, we selected the models that achieved the highest accuracy in the validation set, and then we estimate its generalisation error using the provided test set. The selected models, with their hyper-parameters, are shown in Table 5.11. The learning curve of training and validation set are reported in Appendix A as Figure A.2 and A.3.

[14]Using a batch size of 64 an epoch takes around 7 hours
[15]We still keep the validation set as it was.
[16]5 million parameters

Table 5.11 Comparison between the four model **REN+SENT**, **QDREN+SENT**, **REN+WIND** and **QDREN+WIND**. We show the best hyper-parameters picked by the model selection. In the last rows, we report the loss and accuracy values of each set.

	REN+SENT	**QDREN+SENT**	**REN+WIND**	**QDREN+WIND**
Number of Blocks	20	10	50	20
Window	-	-	5	4
Learning Rate	0.001	0.001	0.0001	0.01
Optimizer	Adam	Adam	RMSProp	RMSProp
Dropout	0.7	0.2	0.5	0.5
Batch Size	128	64	64	64
λ	0.0001	0.001	0.0001	0.0001
Loss Training	2.235	2.682	2.598	2.216
Loss Validation	2.204	2.481	2.427	1.885
Loss Test	2.135	2.417	2.319	1.724
Accuracy Training	0.418	0.349	0.348	0.499
Accuracy Validation	0.420	0.399	0.380	0.591
Accuracy Test	0.420	0.397	0.401	**0.628**

The best accuracy is achieved by **QDREN+WIND** with a value of 0.627, and all the other models could not achieve an accuracy greater than 0.42. The windows based without the question supervision could not achieve an accuracy higher than 0.401. Indeed, saving only facts relative to the question seems to be the key to achieving a good score in this task. We also noticed that using plain sentences, even with QDREN, we cannot achieve a high accuracy. This may be a problem of the sentence encoder, which just using the multiplicative masks do not have enough expressive power for getting key features of the sentence.

Moreover, we notice that the accuracy achieved in the training set is always lower than validation and test set. The same phenomenon is present also in other models presented in the State-Of-The-Art Chapter. In our particular case can be due to the strong regularisation term selected by our models, which also explain why the training accuracy could never reach 100%.

Our model achieves an accuracy comparable to the Attentive and Impatient Reader [33], but it is still not reaching State-Of-The-Art models (i.e. Attention over Attention). However, it is worth to notice that our model is way simpler and it is going through the paragraph just one time. Indeed, even the End To End Memory Network, which is the simplest among other State-Of-The-Art models, iterates many time[17] over the paragraph and the question.

5.4 Analysis

To better understand how our proposed model (i.e. QDREN) works and how it improves the accuracy of the existing REN, we studied the gating function behaviour. Indeed, the output of this function decides how much and what we store in each memory cell, and it is also where our model differs from the original one. Moreover, we trained the QDREN and the original REN using the bAbI task number 1 (using 20 memory blocks).

We pick up this task since both models pass it, and it is one of the simplest, which also allows to better understand and visualise the result. Indeed, we have tried to visualise other tasks but the result was difficult to understand since there were too many sentences in input and it was difficult to understand how the gate opened. Ones we trained the model, we predict the last sample of the validation set (bAbI task 1), and we recorded the gating function of each memory block. Notice that the gating function returns a scalar value in the interval $[0,1]$[18] for each block. This allows visualising the activation of each gate while we input the sentences.

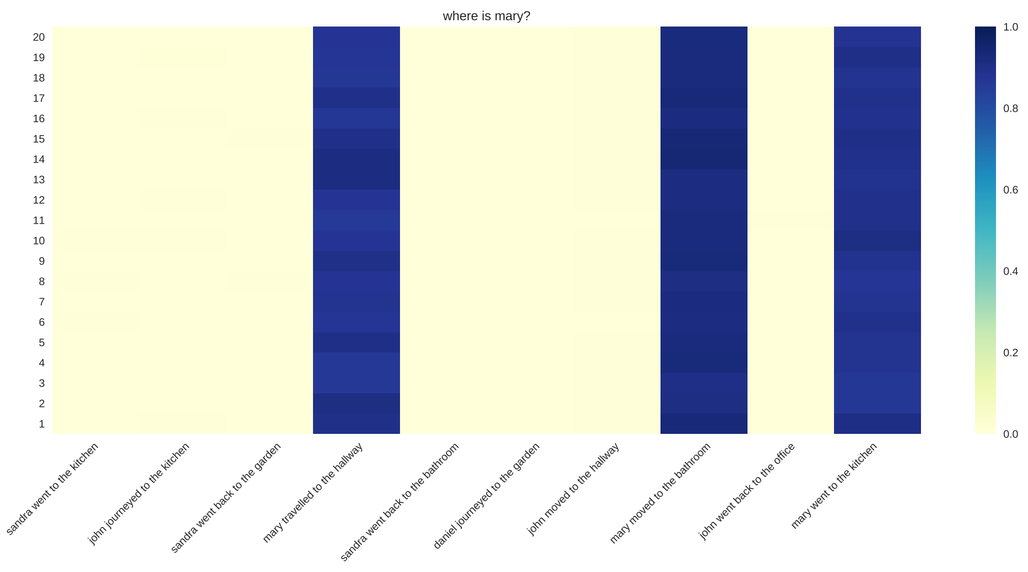
The visualisation result is shown in Figure 5.3, where we plotted the activation matrix for both models. In these plots, we can notice how the two models learn which information to store. Through this, we can make several considerations about the achieved results:

- the gaiting activation of the original REN is sparser than the QDREN, which is very good if we would like to learn all the relevant facts in the text. Moreover, the model effectively assigns a block to each entity and it opens the gates just ones such entity appears in the input sentences. For example, in Figure 5.3 (b) the block cell number 13 supposedly represents the entity Sandra, since each sentence in which appears this name the gate function of the block fully opens (value almost 1). Furthermore, we can notice the same phenomenon with the entity John (cell 10), Daniel (cell 4), and Mary (cell 14). Other entities (e.g., kitchen, bathroom, etc.) are more difficult to recognise in the plot since their activation is less strong and probably the model distribute this information among blocks.

[17]called hops in their paper

[18]Since it is the output of a sigmoid function.

(a)



(b)

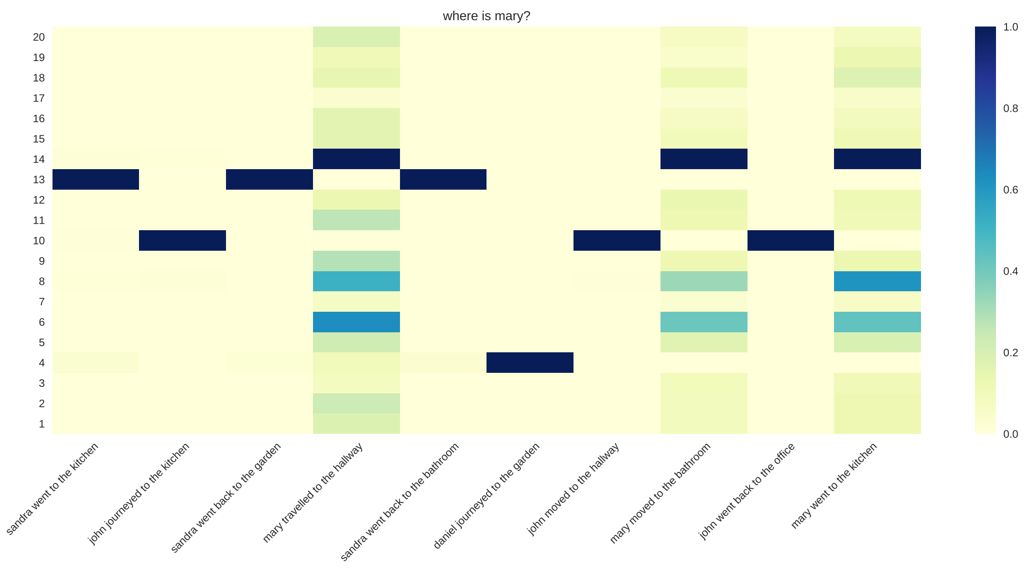


Fig. 5.3 Heatmap representing the gating function result for each memory block. In the y-axes represents the memory block number (20 in this example), in the x-axes, there are the sentences in the input divided into time steps, and at the top, there is the question to be answered. Darker colour means a gate more open (values close to 1) and lighter colour means the gate less open. (a) The figure on the top shows the QDREN model and the figure in the bottom (b) represents the original REN model.

- the activation of the QDREN model is very focused in just a few relevant sentences. In Figure 5.3 (a), we notice that the model is opening the gates just when in the sentence appears the entity named Mary. This because such entity is also present in the question (i.e., "where is Mary?"). Even though the model is focusing on the right entity, its gates are opening all at the same times. In fact, we guess that a sparser activation would be better since it may have modelled which other entities are relevant for the final answer. Finally, we can consider our model as a specialisation of the original REN which is tailor made for a question answer task.

In conclusion, we qualitatively identified the behaviour of the gating function in both REN and QDREN models. Moreover, to identify a similar pattern in more complicated task it is still challenging since the model acts in a very distributed way.

Chapter 6

Conclusions

In this Chapter, we provide a short summary of the topic presented in this thesis. After that, we discuss further extensions of our models.

6.1 Summary

In Chapter 1, we introduced the general topic of the thesis, and its motivations and contributions.

In Chapter 2, we introduced basic terminology and a short review of existing Natural Language Processing tasks. Then, we described different methodologies to handle such tasks, in particular, we introduced Artificial Neural Networks and its Recurrent variations (RNN, LSTM and GRU). Moreover, we presented the Word Embedding concept with different methods for build it. At the end of the Chapter, we described several well-known applications of the listed techniques, which also inspired our proposed models.

In Chapter 3, we presented several State-Of-The-Art solutions. In particular, we introduced well-known models, frameworks, and benchmark datasets used for Question Answering tasks. For instance, we presented three types of tasks that are used in our thesis, which are: *Community Question Answering*, *Reasoning Question Answering* and *Reading Comprehension*. Finally, we summarised the performance of the existing models in the existing benchmark datasets (bAbI, CNN, etc.). This allowed us to have a mean of comparison for our models.

In Chapter 4, we presented our two models: *ThRee Embedding Recurrent Neural Network*, and *Question Dependent Recurrent Entity Network*. For both models, we discussed their design with reference to the task they need to accomplish.

In Chapter 5, we evaluated the performance of the presented models. In particular, we introduced the datasets used for each task, the implementation details of each model, and the results achieved in the evaluation phase. Finally, we quantitatively analysed the behaviour of our model through a visualisation. These gave us an insight of how our model can achieve a better performance compared to its original version.

6.2 Discussion and Future Works

In this section, we discuss several possible extension and future works.

- in the ThRee Embedding Recurrent Neural Network, which we presented for the Community Question Answering task, we tried to combine semantic and syntactic information into a single vector space. We would like to further investigate this combination through the use of syntactic relations holding between content words, rather than exploiting the whole set of dependency relations (e.g. different tag-sets, partial or shallow parsing of sentences etc.). Our experiments have explored different possibilities regarding the choice of the word embedding system and all of them proved in the end to achieve similar results. However, it may be worth to try to build an ad-hoc embedding space, which mixes parsing and lexical information, aiming to improve the performances of our model.

  Future works may include: an improvement to the RNN in order to better represent longer sentences (e.g. using a proper dynamic unrolling), or the use of a Recursive Neural Network which directly uses the tree structure given by the dependency parsing (instead of creating triples). In general, we would like to find a task-independent way to properly combine word vectors. To do so, it would be interesting to employ a set of sparse autoencoder, one for each dependency relations, which fold the tree into a single vector. In this way, we may hopefully find a general way to fuse sentences in a single vector.

- in the Question Dependent Recurrent Entity Network, used for reasoning and reading comprehension tasks, we exploit a particular RNN cell in order to store just relevant information about the given question. In this way, in combination with the original Recurrent Entity Network (keys and memory), we achieved promising results in both tasks. However, we believe that the optimal behaviour for the proposed cell is still an open problem. Indeed, the cell has not enough expressivity power to make a selective activation among different memory blocks (notice in Figure 5.3 (a) the gates open for all the memories). This does not seem to be a big problem since we actually outperform other models, but it could be the key to finally pass all the bAbI tasks in the 1K setting.

Future works may include an improvement to the memory cell structure and a different infrastructure for the output module. The former can be achieved by making a more selective gating function (adding further parameters). On the other hand, changing the output module could extend this model to other tasks. Indeed, we could plug an RNN that generates sentences from the memory blocks, in a similar way as the Seq-To-Seq model [76]. For example, we could create a task that requires retrieving all the information about a given entity. For instance, it could generate a short sentence that describes all the information of such entity. Many other tasks can be proposed, and it would be very interesting, at the same time challenging, to check the effectiveness of our models.

References

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

[2] Attardi, G., Carta, A., Errica, F., Madotto, A., and Pannitto, L. (2017). Fa3l at semeval-2017 task 3: A three embeddings recurrent neural network for question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 290–295, Vancouver, Canada. Association for Computational Linguistics.

[3] Attardi, G., Dell'Orletta, F., Simi, M., Chanev, A., and Ciaramita, M. (2007). Multilingual dependency parsing and domain adaptation using desr. In *EMNLP-CoNLL*, pages 1112–1118.

[4] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[5] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[6] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

[7] Binsted, K., Nijholt, A., Stock, O., Strapparava, C., Ritchie, G., Manurung, R., Pain, H., Waller, A., and O'Mara, D. (2006). Computational humor. *IEEE Intelligent Systems*, 21(2):59–69.

[8] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

[9] Buchholz, S. and Marsi, E. (2006). Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics.

[10] Chen, D., Bolton, J., and Manning, C. D. (2016). A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*.

[11] Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

[12] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM.

[13] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

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

[15] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[16] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011a). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

[17] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

[18] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011b). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

[19] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[20] Cui, Y., Chen, Z., Wei, S., Wang, S., Liu, T., and Hu, G. (2016). Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423*.

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

[22] De Marneffe, M.-C. and Manning, C. D. (2008). Stanford typed dependencies manual. Technical report, Technical report, Stanford University.

[23] Deriu, J. M. and Cieliebak, M. (2017). Swissalps at semeval-2017 task 3: Attention-based convolutional neural network for community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 325–329, Vancouver, Canada. Association for Computational Linguistics.

[24] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[25] Feng, W., Wu, Y., Wu, W., Li, Z., and Zhou, M. (2017). Beihang-msra at semeval-2017 task 3: A ranking system with neural matching features for community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 271–277, Vancouver, Canada. Association for Computational Linguistics.

[26] Filice, S., Da San Martino, G., and Moschitti, A. (2017). Kelp at semeval-2017 task 3: Learning pairwise patterns in community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 317–324, Vancouver, Canada. Association for Computational Linguistics.

[27] Goller, C. and Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE.

[28] Golub, G. H. and Reinsch, C. (1970). Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420.

[29] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[30] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

[31] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[32] Henaff, M., Weston, J., Szlam, A., Bordes, A., and LeCun, Y. (2016). Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*.

[33] Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

[34] Hill, F., Bordes, A., Chopra, S., and Weston, J. (2015). The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*.

[35] Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5):5947.

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

[37] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[38] Hsu, W.-N., Zhang, Y., and Glass, J. (2016). Recurrent neural network encoder with attention for community question answering. *arXiv preprint arXiv:1603.07044*.

[39] Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80.

[40] Jones, K. S. (1994). Natural language processing: a historical review. In *Current issues in computational linguistics: in honour of Don Walker*, pages 3–16. Springer.

[41] Jordan, M. I. and Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260.

[42] Juang, B.-H. and Rabiner, L. R. (2005). Automatic speech recognition–a brief history of the technology development. *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara*, 1:67.

[43] Kadlec, R., Schmid, M., Bajgar, O., and Kleindienst, J. (2016). Text understanding with the attention sum reader network. *arXiv preprint arXiv:1603.01547*.

[44] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[45] Kobayashi, S., Tian, R., Okazaki, N., and Inui, K. (2016). Dynamic entity representation with max-pooling improves machine reading. In *Proceedings of NAACL-HLT*, pages 850–855.

[46] Kong, L., Alberti, C., Andor, D., Bogatyy, I., and Weiss, D. (2017). Dragnn: A transition-based framework for dynamically connected neural networks. *arXiv preprint arXiv:1703.04474*.

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

[48] Lebret, R. and Collobert, R. (2013). Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*.

[49] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[50] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[51] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[52] Mihaylov, T. and Nakov, P. (2016). Semanticz at semeval-2016 task 3: Ranking relevant answers in community question answering using semantic similarity based on fine-tuned word embeddings. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, pages 804 – 811, San Diego, California. Association for Computational Linguistics.

[53] Mihaylova, T., Gencheva, P., Boyanov, M., Yovcheva, I., Mihaylov, T., Hardalov, M., Kiprov, Y., Balchev, D., Koychev, I., Nakov, P., Nikolova, I., and Angelova, G. (2016). Super team at semeval-2016 task 3: Building a feature-rich system for community question answering. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 836–843, San Diego, California. Association for Computational Linguistics.

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

[55] Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Hlt-naacl*, volume 13, pages 746–751.

[56] Mohtarami, M., Belinkov, Y., Hsu, W.-N., Zhang, Y., Lei, T., Bar, K., Cyphers, S., and Glass, J. (2016). Sls at semeval-2016 task 3: Neural-based approaches for ranking in community question answering. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 828–835, San Diego, California. Association for Computational Linguistics.

[57] Moschitti, A. (2006). Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning*, pages 318–329. Springer.

[58] Mozer, M. C. (1995). A focused backpropagation algorithm for temporal. *Backpropagation: Theory, architectures, and applications*, page 137.

[59] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[60] Nakov, P., Hoogeveen, D., Màrquez, L., Moschitti, A., Mubarak, H., Baldwin, T., and Verspoor, K. (2017). Semeval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 27–48, Vancouver, Canada. Association for Computational Linguistics.

[61] Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376.

[62] Onishi, T., Wang, H., Bansal, M., Gimpel, K., and McAllester, D. (2016). Who did what: A large-scale person-centered cloze dataset. *arXiv preprint arXiv:1608.05457*.

[63] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318.

[64] Peng, B., Lu, Z., Li, H., and Wong, K.-F. (2015). Towards neural network-based reasoning. *arXiv preprint arXiv:1508.05508*.

[65] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[66] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151.

[67] Richardson, M., Burges, C. J., and Renshaw, E. (2013). Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*, volume 3, page 4.

[68] Robinson, A. and Fallside, F. (1987). *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering.

[69] Rumelhart, D. E., McClelland, J. L., Group, P. R., et al. (1988). *Parallel distributed processing*, volume 1. IEEE.

[70] Shultz, D. (2015). Which movies get artificial intelligence right?

[71] Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

[72] Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., Potts, C., et al. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer.

[73] Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735.

[74] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[75] Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

[76] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[77] Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

[78] Taylor, W. L. (1953). “cloze procedure”: a new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.

[79] Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646. MORGAN KAUFMANN PUBLISHERS.

[80] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[81] Tieleman, T. and Hinton, G. (2012). Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report, Technical report.

[82] Trischler, A., Ye, Z., Yuan, X., and Suleman, K. (2016). Natural language comprehension with the epireader. *arXiv preprint arXiv:1606.02270*.

[83] Vapnik, V. N. and Chervonenkis, A. Y. (2015). On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. Springer.

[84] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

[85] Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., and Mikolov, T. (2015). Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

[86] Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

[87] Williams, D. and Hinton, G. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–538.

[88] Xiong, C., Merity, S., and Socher, R. (2016). Dynamic memory networks for visual and textual question answering. *arXiv*, 1603.

[89] Yu, W., Zeng, G., Luo, P., Zhuang, F., He, Q., and Shi, Z. (2013). Embedding with autoencoder regularization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 208–223. Springer.

[90] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Appendix A

Table A.1 Best hyper-parameters selected by the model selection of each bAbI tasks.

Task	**Number of Blocks**	λ	**Learning Rate**	**Dropout**
1	20	0	0.001	0.5
2	30	0	0.001	0.5
3	40	0	0.001	0.5
4	20	0	0.001	0.5
5	50	0	0.001	0.2
6	30	0	0.001	0.5
7	30	0	0.001	0.5
8	20	0.001	0.001	0.7
9	40	0.0001	0.001	0.5
10	20	0	0.001	0.5
11	20	0	0.001	0.5
12	20	0	0.0001	0.5
13	40	0.001	0.001	0.7
14	30	0.0001	0.001	0.5
15	20	0	0.001	0.5
16	20	0.001	0.001	0.5
17	40	0.001	0.001	0.5
18	30	0.0001	0.001	0.5
19	20	0	0.001	0.5
20	20	0	0.001	0.5

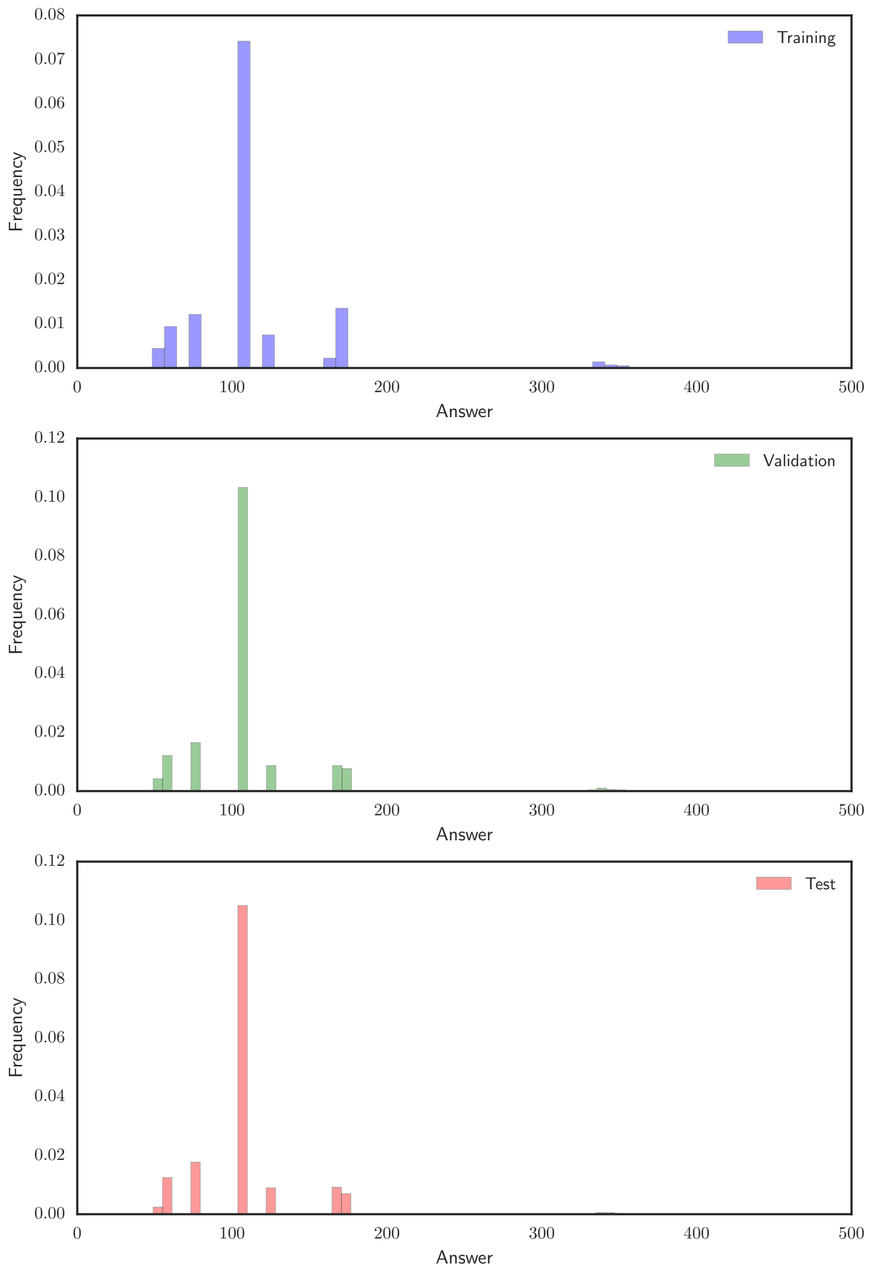
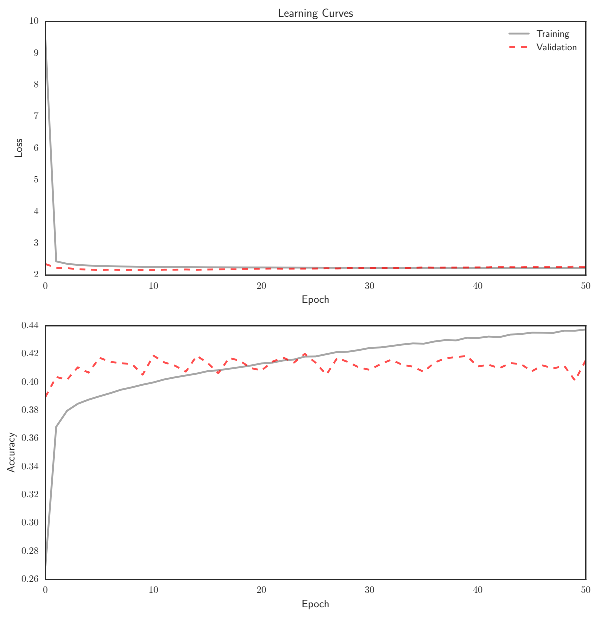
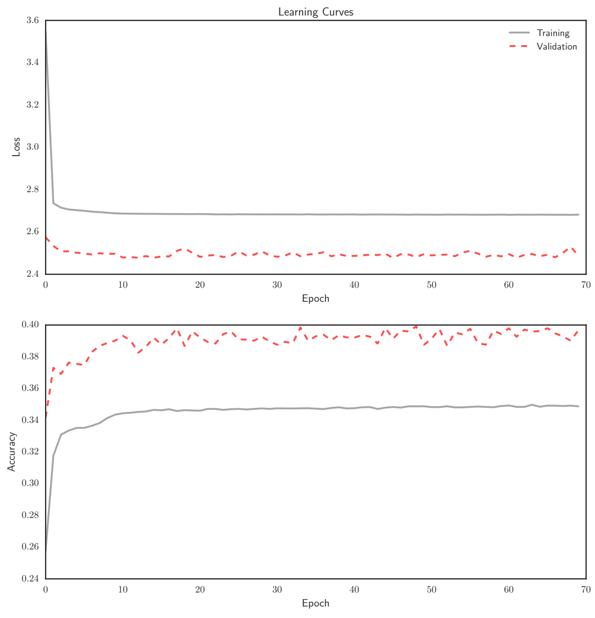


Fig. A.1 Label distribution of the Training, Validation, and Test sets. We can notice how the three sets follows a similar distribution.

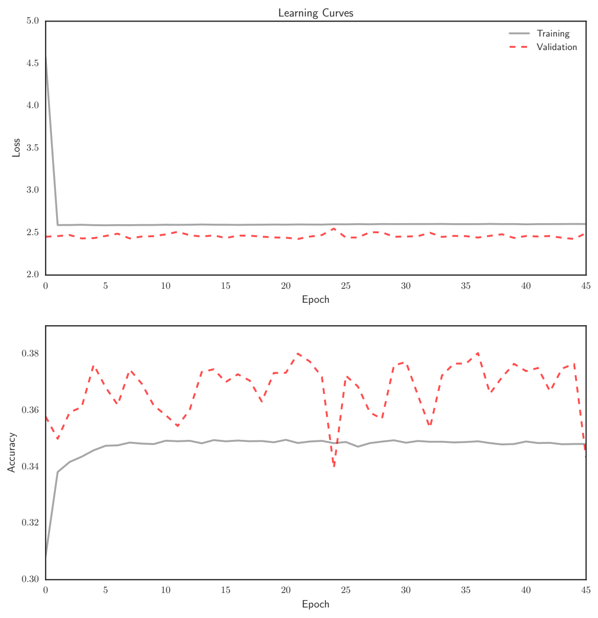


(a)

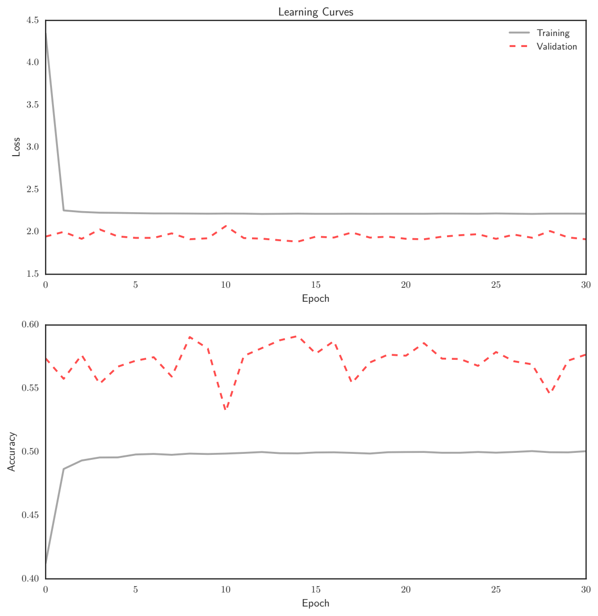


(b)

Fig. A.2 Learning curve of the **REN+SENT** (a) and **QDREN+SENT** (b) models.



(a)



(b)

Fig. A.3 Learning curve of the **REN+WIND** (a) and **QDREN+WIND** (b) models.

Appendix B

Table B.1 Best and worst value achieved in the Validation Set. We report the results of all the models and dataset used. In the SemEval dataset we report MAP instead of accuracy since we used such measure to select our best model.

bAbI					
Task	Worst	Best	Task	Worst	Best
1	0.191	1.000	11	0.751	0.997
2	0.183	0.400	12	0.774	1.000
3	0.194	0.419	13	0.170	1.000
4	0.489	1.000	14	0.210	0.861
5	0.313	0.987	15	0.218	0.998
6	0.496	0.718	16	0.233	0.528
7	0.781	0.994	17	0.513	0.643
8	0.328	0.987	18	0.546	0.926
9	0.641	0.960	19	0.090	0.165
10	0.469	0.972	20	0.720	1.000
CNN		**REN + SENT**	**QDREN + SENT**	**REN + WIND**	**QDREN + WIND**
	Best	0.341	0.321	0.331	0.478
	Worst	0.251	0.286	0.218	0.248
SemEval					
MAP		Best	0.705		
		Worst	0.435		