Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Department of Automation and Applied Informatics

Semantic parsing with graph transformations

Scientific Student's Assosiactions Report

Author
Kovács Ádám
Gémes Kinga Andrea

Supervisor
Dr. Recski Gábor

October 28, 2018

Kivonat

A szemantikai elemzés célja, hogy természetes nyelvi adathoz készíthessünk szemantikai reprezentációt, így tudjuk modellezni a szoveg jelentését. Ha a nyelvi jelentést fogalmak irányított gráfjaival reprezentáljuk, ezeket pedig a mondat szintaktikai szerkezetét reprezentáló fákból kell előállítanunk, akkor a teljes feladat egyetlen komplex gráf-transzformációként definiálható.

A népszerű szemantikai feladatokra, mint a szemantikai hasonlóság mérése, vagy a gépi szövegértés, ritkán használják a természetes nyelv szemantikájának a reprezentációját, főleg a state-of-the art rendszerekben. Ezek a rendszerek többnyire szó embeddingeket használnak a szavak jelentésének ábrázolására, amik a szavak jelentését legfeljebb néhány száz dimenziós valós vektorként ábrázolják.

Ebben a dolgozatban mi gráf-reprezentációkat és ezek transzformációit használjuk, mint egyszerű, ám hatékony eszközök a következmény viszony felismerésére, valamint leírunk egy módszert a `4lang` szemantikus elemzőrendszer[30] használatára a 2018-as *Semeval Task Machine comprehension using commonsense knowledge*[1] kapcsán. Ez a feladat azt kívánja a résztvevőktől, hogy olyan rendszereket tanítsanak fel, amelyek ki tudják választani a megfelelő választ az egyszerűbb, több válaszlehetőséget kínáló kérdéseknél rövid eseményleíró szövegek elolvasása után. A tanító és teszt adat az `MCSript`[28] adathalmaz részhalmazából lett kinyerve. A két legjobb rendszer, `HFL-RC`[10] és Yuanfudao[39] rendre 84,15% és 83,95% pontosságot ért el a teszt adaton.

Először bemutatunk egy hatékony baseline-t ezen a feladaton csupán a szemantikus gráfok és a köztük lévő hasonlóságok felhasználásával. Ezt követően leírjuk a Yuanfudao state-of-the art rendszert és az ezzel végzett kísérletezéseinket, amelyek során a baseline-unkat extra feature-ként felhasználva javítottunk a rendszer pontosságán. Ennek a kiválasztása magától értetődő volt, mivel a forráskód nyilvánosan elérhető, és már sikeresen alkalmazott tudás alapú reprezentációt a szópárok közötti szemantikai kapcsolatokra, a *ConceptNet*-et. Eredményeink azt mutatják, hogy ezzel a módosítással `0.5` százalékpont növekedés érhető el, és a *ConceptNet* helyettesíthető a mi szemantikus modellünkkel.

[1] `https://competitions.codalab.org/competitions/17184`

Abstract

The main task of semantic parsing is to automatically build semantic representation from the input, so we can model the meaning of raw texts. If we model meaning as directed graphs of concept and we can build them from syntax trees that represent the structure of sentences, then we can define the whole process as one complex graph transformation.

Representations of natural language semantics are rarely used explicitly in state-of-the art systems for popular semantics tasks such as measuring semantic similarity or machine comprehension. These systems mostly use word embeddings as representation of word meaning.

In this paper we use graphical representations and transformations as simple but powerful tools for recognizing entailment and we describe a method using semantic parsing system `4lang` (Recski:2016) and applying it on the 2018 Semeval Task *Machine comprehension using commonsense knowledge*. This task requires participants to train systems that can choose the correct answer to simple multiple choice questions based on short passages describing simple chains of events. Data for both training and testing is extracted from the `MCScript` dataset (Ostermann et al., 2018). The top two systems, `HFL-RC` (Chen et al., 2018) and `Yuanfudao` achieved accuracy scores of 84.15% and 83.95% on the test data, respectively.

First we will present a strong baseline on this task using only semantic graphs and similarities among them, followed by a description of a state-of-the art system `Yuanfudao` (Wang et al., 2018) and our experiments with it where we used our baseline as an extra feature for improving the neural network. The choice of the system was obvious, because the source code is publicly available and it already employs successfully a knowledge base representing semantic relationships among pairs of words, *ConceptNet*. Our results suggest that these features achieve a .5 percentage point improvement, and the *ConceptNet* could be replaced by our semantic model.

Contents

Chapter 1

Introduction

In modern systems distributional models are dominant for a semantic parser. We use graph based methods and apply it to the *2018 Semeval task on Machine Comprehension* (MC). We built a REST-API (available at `http://hlt.bme.hu/4lang`) around the **4lang**[31] (described in Chapter 2) to present a highly automated process constructing semantic models from raw input. Online demo of the service is available at `http://4lang.hlt.bme.hu`. In Chapter 3 we introduce a strong baseline for the task, followed by an enhancement of a state-of-the-art system [39] (Chapter 5). In this chapter we discuss the history of the Natural Language Processing (NLP) applications, and briefly define the structure of this paper.

1.1 Natural Language Processing

While computers can be easily programmed to understand structured data, such as tables and spreadsheets, it can be rather challenging for them to understand human communication. Because there is a vast difference in the magnitude of the unstructured data compared to the structured ones, there is a high demand for tools, that can deal with raw text. That's where NLP comes in. It contains high variety of tools, that we have to use, when we need to deal with natural input.

Every day, we come into contact with human communication, we say a lot of words to other people, and they try to interpret them even when the context of the saying isn't necessarily complete. The listeners can use their common knowledge to fill the needed information. We resolve ambiguities, misunderstanding, and can even understand words we have never heard before just from the context of the communication. Even though these tasks are trivial for us, for a computer it can be really hard.

The interest in NLP research began in the 1950s, the early phase was mainly focused on MT (Machine Translation), because after the World War II, people recognized the importance of the translation from one language to another, and hoped to do it automatically. However MT is still very difficult nowadays, so these researches discovered the main challenges of the syntactic and semantic parsing early. As time passed, researches embraced new areas of NLP as more advanced technology and knowledge became available. Now that

we live in a world where computers and smartphones are widely accessible, collecting data became incredibly easy, as a result, statistical NLP drew attention because these models thrive off big data, but one cannot ignore simple rule based methods which can also be very powerful, especially using them as a hybrid model with statistical methods.

Building NLP applications requires many levels of analysis. The typical pipeline is structured as follows:

- First we need to tokenize our input text, which means breaking up the text into meaningful elements, especially into words
- After we tokenized our text, we need to perform word analysis called `morphology`, which is concerned with the structure of words.
- Part of speech assigns words to syntax behavior in a sentence.
- The main task of syntactic parsing is to analyze the grammatical structure of a sentence. Given a set of words, a parser forms units (subjects, verbs, etc..) according to some grammatical formalism. There are two main types of syntactic parsers:
  - *Constituency parsers* produce trees, that represent the grammatical structure.
  - *Dependency parsers* are the more popular nowadays. They represent the structure of a sentence as a dependency tree, which instead of grammatical relations, tries to model the dependencies between words.

  A parse of the example sentence: *"John has finished the work"* can be seen in Figure 1.1.
- At this point we have various ways to analyze a text, but without modeling its `meaning`. Semantics is the study of meaning, and semantic parsing is a task to find a representation and assign it to the text. This task will be the main topic of our work.

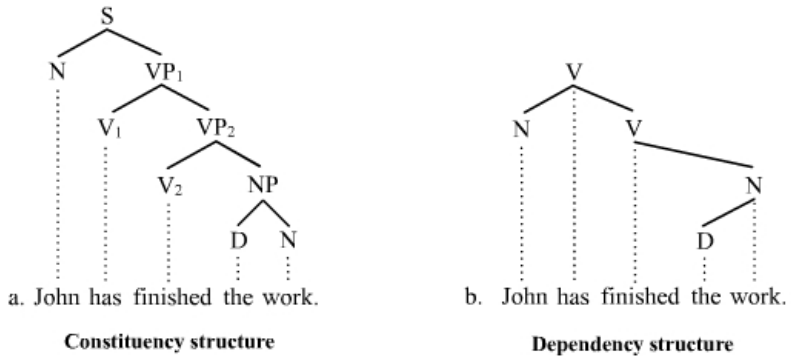

Figure 1.1: *Parses of the sentence "John has finished the work" [2]*

1.2 Objectives

The main focus of this study is computational semantics. Our research includes building explicit representations of natural language semantics, because in today's state-of-the art

systems for popular semantics tasks such as measuring semantic similarity or machine comprehension, they are rarely present. Virtually all systems competing at popular challenges (e.g. [9, 8]) rely on word embeddings as the sole representation of word meaning. Recently [33] has presented a method using graphical representations of natural language text that improved over the state-of-the art on the task of measuring semantic similarity of pairs of English words. In this paper we use similar graphs as simple but powerful tools for measuring textual entailment. Our task includes defining new methods for measuring graph similarities, and building an online available service for building graphs highly automatically, giving us a tool for building strong baseline methods. Our work was based upon measuring our models through state-of-the art systems on the 2018 Semeval Task *Machine comprehension using commonsense knowledge*.

1.3 Results

We present a novel method for recognizing entailment using semantic graphs and apply it to the 2018 Semeval task on Machine Comprehension (MC). Concept graphs are built automatically from MC texts, questions, and answers, using the semantic parsing system `4lang` [30]. First we present a highly automated process of building concept graphs from raw text, then a strong baseline method is presented using only these graphs achieving an accuracy score of 68.3%, followed by an enhancement of a state-of-the art system [39], where we proceeded to use the metric underlying our baseline as an additional feature. Preliminary results suggest that these features achieve a .5 percentage point improvement over the original system.

1.4 References

The code of our system is available on Github[1]. The code was implemented by the authors of this paper based upon the **4lang** system.

1.5 Structure

The structure of the paper is the following:

- **Chapter 1** describes the short history and motivation of the NLP applications, it also gives a short summary about the objectives of the paper, and our results.
- **Chapter 2** gives a short introduction into the field of semantic parsing, and semantic models in general. It briefly explains the semantic parsing system **4lang**, and our process of automating the building of concept graphs
- **Chapter 3** describes the baseline approach to the Machine Comprehension task, which achieved an accuracy score of 68.3%.
- **Chapter 4** gives an introduction into deep learning, focusing on the NLP tasks.

[1] `https://github.com/adaamko/4lang`

- **Chapter 5** presents our experiments with the state-of-the art system **Yuanfudao**. Our preliminary results show a .5 percent improvement over the original system.
- **Chapter 6** summarizes our contributions and describes our ongoing/future work. It briefly discusses our plans for the follow-up, that was beyond the scope of this work.

1.6 Division of labour

This project was a product of the combined work of Kovács Ádám and Gémes Kinga Andrea. Kovács Ádám was responsible for building the service to automate the process of building concept graphs (online demo available at `http://4lang.hlt.bme.hu`), and was mostly working on the baseline methods and metrics. Gémes Kinga Andrea's main work involved applying the baseline to the state-of-the-art system Yuanfudao [39][2] and experiments with IRTGs briefly described in Chapter 6[3].

[2] `https://github.com/GKingA/commonsense-rc`
[3] `https://github.com/GKingA/irtg`

Chapter 2

Semantic models and parsing

In this chapter we first introduce applications, that use semantic models as an essential knowledge for their process. After, the theory and problems of semantic representation are discussed, and we briefly present the upsides and downsides of such representations. After that we go into details about distributional and graph based models, introducing the semantic parsing system **4lang**, which is the main focus of our work. Finally, our micro-services are discussed, built around the parser to highly automate the process of building concept graphs from raw input. An easy example of their usage is shown as well.

When we use the word semantics, we usually refer to the interpretation of a linguistics unit (e.g. words, phrases, sentences, or whole texts) within the boundaries of a certain context. In many scientific fields e.g. philosophy, logic or biology the study of semantics is a highly researched task. While many NLP tasks like syntactic parsing, part-of-speech tagging, or even machine translation can be ignorant of the meaning of units, of what information they hold, there are also many other tasks that rely heavily on semantics.

Some examples, where semantic analysis is unavoidable:

- **Question answering** is the process of generating meaningful answers to the user's question, using some kind of knowledge. It might be considered one of the oldest tasks in NLP or in AI in general with machine translation. With the recent rise of products like Siri, Alexa, or Watson, it is still one of the most researched area.
- **Recognizing entailment** is whether a statement implies another or not, it is closely connected to **machine comprehension**, which is the main focus of our work.
- **Chatbots** are systems, that somewhat can simulate the conversation of humans.
- **Sentiment analysis** is the task of understanding the opinion about a subject. Usually can be considered as a classification problem, where in the simplest case two class, "*negative*" and "*positive*" is given. More complex version of the task is also present, where the detection of the target of the opinion is also the part of the problem. There are solutions for the problem with hand-crafted rules and machine learning algorithms as well.

For modeling the meaning of linguistic units, choosing an appropriate representation of

our model is needed. While for syntactic analysis widely accepted concepts and ideas (e.g. dependency tree, phrase structure trees) of such representation are already in use, for a semantic model it still remained a challenging task. Even answering the question *"What is a semantic representation?"* is not well defined, and mostly decided by the nature of our task. The units of a semantic model are also not globally decided (word, phrase or a sentence?).

Finding an absolute representation of semantics knowledge is among the most difficult task in Artificial Intelligence (AI). While we are yet to find a representation that has no downside, various experimental models are present, that are applicable for certain domains, but are problematic in other scenarios. So before choosing one, we need to take into account the limitations of each choice.

Representation of semantic knowledge with logical expressions (e.g. zero-order logic, prepositional logic) exists, and although many companies still rely on hand-crafted rules as a knowledge representation, they are very unpractical and have serious problems with scalability and automation. Distributional models have risen in popularity in recent years, where meaning is represented with multi-dimensional vectors. While vector based models can give us a uniform representation, they mostly lack interpretability in a way that we never really know what happens inside these models. Another issue is the choice of the dimension. Many algorithms exist for reducing the dimension of such vectors to a fixed number, but determining the correct length can be difficult. Rare words are a big issue as well, if a word in the training data is rarely present, distributional models will fail to handle them correctly.

On the other hand graph based solutions have high level of interpretability, and handling rare words is one of their biggest advantage. But automating the process of building graphs is a challenging task, and using them as a sole solution for a semantic parsing task in most cases would come short, but in hybrid systems they make up for the weaknesses other models have.

In the next sections we will briefly discuss graph based models and distributional models, and because both of them have their merits, it is necessary to know the strengths and weaknesses of each one. Our work focuses around graph based solutions.

2.1 Distributional model

In the field of natural language processing one way of encoding semantic meaning is to use distributional models. They model semantic meaning as real-valued vectors. These vectors need to be constructed from a training data, and we can calculate similarity as cosine distance between the vectors.

Let's have a look at these two sentences *"The cat is walking in the bedroom"* and *"A dog was running in a room"*. Words *"dog"* and *"cat"* have a similar semantic meaning, so if they are represented by vectors, and their cosine distance from each other is small, then we can vary the sentences *"The dog is walking in the bedroom"* and *"A cat was running in a room"* [6]. These models take surrounding words into account and their goal is to obtain the

meaning of the target word from their surroundings[16], and because *"dog"* and *"cat"* are close in vector space, they most likely will appear in the same context. This intuition helps us generalize sentences. These meanings are represented by vectors, called embeddings. One of the first models built around these intuitions were introduced by Bengio [6].

These word embeddings are used in basically all state-of-the art systems related to natural language processing applications. Mikolov [26] showed that word embeddings can be applied for vector operations, like addition or subtraction, and these operations often result in meaningful representation. If we have example words "*King*", "*Man*", "*Woman*", then the vector("King") - vector("Man") + vector("Woman") 2.1 will most likely result in vector that is close to vector("Queen") in the embedding space.

While the usage of word embeddings brought an important breakthrough in modeling word meaning, applying them for bigger linguistics units like phrases, sentences or even whole text remained a difficult challenge even for nowadays. One of the biggest reasons for this is the additive aspect of these models: if we model A B C expression with vector v(A) + v(B) + v(C), then we represent "John killed Bill" and "Bill killed John" sentences in the same way.

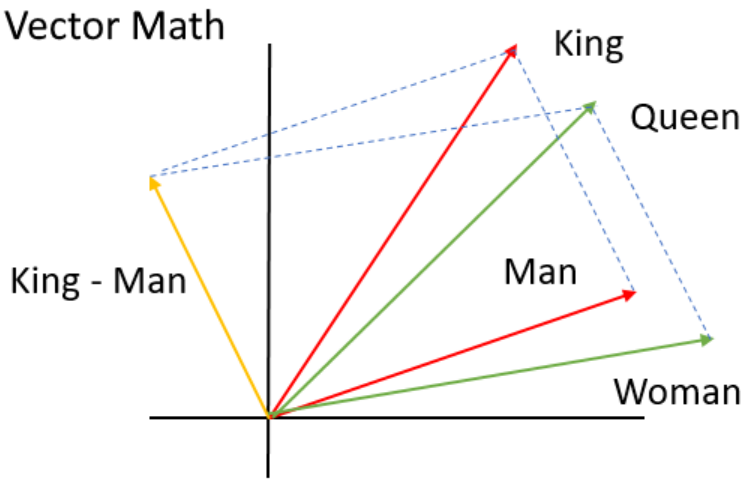

Figure 2.1: *Vector addition example [1]*

The other issue is that we know very little about the structure of a multi-dimensional real-value vectors (for embeddings it can be from 300 up to 1000 dimension), so this makes it very hard to understand their structure, and exactly in what scenario they work, and the reason when they don't. So while most state-of-the art systems use word embeddings as a sole representation of meaning, and while it can be useful to encode meaning as vectors, so it can create connection from language specific and non-language specific data, we cannot deny the importance of having other semantics representations, such as graph-based ones.

In the majority of our work, we researched graph-based solutions, where we model the meaning of linguistics unit with graphs, and the whole process can be defined with graph transformations. Next I will briefly introduce graph based formalism starting with Abstract Meaning Representations (AMR) followed by the introduction of the **4lang** formalism, which will be the focus of the work, and I will go into details in the next section.

2.2 Abstract Meaning Representations

Abstract Meaning Representation (AMR) was introduced by Banarescu[5] for representing the meaning of linguistic structures. They represent meaning as directed acyclic graphs (DAGs), that can be used to capture the meaning of whole sentences. In the past few years, AMR related works have appeared e.g. parsing applications, or annotated corpuses [5, 27, 11].

Nodes of AMR graphs can be represented various ways. Each node in the graph represents a semantic concept [36], that can be either an English word, or frameset from PropBank [29], essentially used for abstraction. The framesets are English verbs. The AMR introduced these variables for entities, events, properties, and states. An AMR can be converted to multiple formats:

- Logic format
- AMR format
- Graph format

  These formats can be seen on Figure 2.2 for sentence *"the boy wants to go"*, and the corresponding 4lang representation on Figure 2.3.

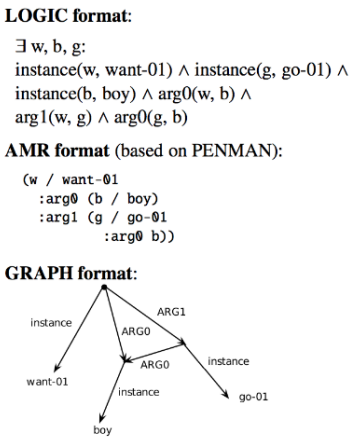

Figure 2.2: *Example sentence and representations [29]*

In this paper we use the semantic parser **4lang** [32], and unlike **4lang**, AMR handles wider range of phenomenas, mostly typical of English. AMR's usage is mostly biased towards English [29], while 4lang can be configured to handle multiple languages.

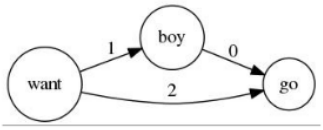

Figure 2.3: *Example sentence and representations in 4lang*

In the next section, we will go into details about the **4lang** formalism, and the parser

itself. After that we will describe our method of measuring similarities between semantic graphs, and its usage on semantic related tasks.

2.3 4lang

The **4lang** system is in the main focus of our work, in this section we will discuss the formalism and possible applications. **4lang** also means the manually built dictionary of mapping more than 2000 words to graphs, this is described in [21]. After discussing the main formalism of **4lang**, we will demonstrate our highly automated process of building concept graphs, that was achieved by wrapping **4lang** functionality in micro-services building a REST-API, followed by our baseline for the machine comprehension task.

2.3.1 The formalism

The `4lang` system of semantic representation [20] represents the meaning of linguistic units (both words and phrases) as directed graphs of syntax-independent concepts. Every node of a **4lang** graph is a concept, which means that they are not taken as words, and they don't have any grammatical functions, like part-of-speech, voice, tense, etc.[31]. Since these concepts have no grammatical attributes and no event structure, e.g. the phrases *water freezes* and *frozen water* would both be represented as *water* $\xrightarrow{0}$ *freeze*. This also means that 4lang defines a many-to-one relation between the words and concepts.

4lang formalism defines three types of edges:

- **The 0-edge** represent represent attribution (`dog` $\xrightarrow{0}$ `large`), hypernymy (`dog` $\xrightarrow{0}$ `mammal`) and unary predication (`dog` $\xrightarrow{0}$ `bark`)
- **1- and 2-edges** those representing binary relations are connected to their arguments via edges labeled `1` and `2`, e.g `cat` $\xleftarrow{1}$ `catch` $\xrightarrow{2}$ `mouse`. Binaries that are shown with uppercase are binaries that must have two outgoing edges as shown in Figure 2.4. If we look at the sentence *"Kinga broke Adam's bike"*, and the corresponding graph shown in Figure 2.4, if the 0-connection wouldn't be present between *Kinga* $\xrightarrow{0}$ break, that would mean we consider that the relationship depend on whether the object of breaking is established or not. So in **4lang** the connection of 0-edge is present between a subject and a predicate regardless of the other arguments.

The example in Figure 2.10 shows the `4lang` definition of the concept `bird`. This definition was built manually, as part of the `4lang` dictionary [21], but similar definitions have been created automatically from definitions of monolingual dictionaries such as Longman, using the `dict_to_4lang` tool [30].

The open-source **4lang** pipeline[1] contains tools for generating directed graphs from raw text by mapping dependency edges in the output of the Stanford parser [13] to `4lang` subgraphs over concepts corresponding to each word of the original sentence. The Stanford parser builds a dependency tree from the raw text that captures the syntactical relations

[1] `https://github.com/kornai/4lang`

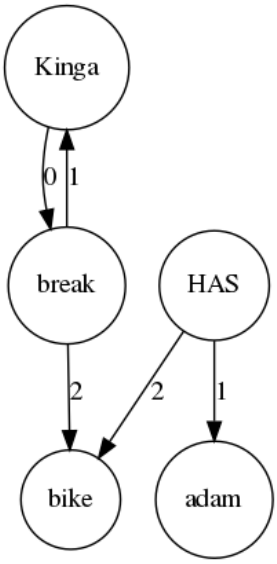

Figure 2.4: *4lang with binaries*

between the linguistics units. **4lang** graph construction involves mapping from these relations to **4lang** semantics graphs, assigning the dependencies to **4lang** subgraphs. The mapping is presented in Table 2.1, and an example is shown for sentence *"I like swimming"* in Figure 2.6, where we can see the dependency tree coming out of the Stanford parser, and the corresponding 4lang graph is present in Figure 2.5, where the mapping from the dependency tree to **4lang** graph is done.

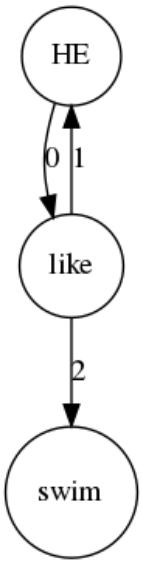

Figure 2.5: *4lang example of a sentence*

2.3.2 Expansion

Optionally, the `4lang` system allows us to *expand* graphs, a process which unifies the graph with the definition graphs of each concept. The implementation is written in the **dict_to_4lang** module, that extends the functionality of the discussed **text_to_4lang**

Dependency	Edge
amod advmod npadvmod acomp dep num prt	$w_1 \xrightarrow{0} w_2$
nsubj csubj xsubj agent	$w_1 \underset{0}{\overset{1}{\rightleftharpoons}} w_2$
dobj pobj nsubjpass csubjpass pcomp xcomp	$w_1 \xrightarrow{2} w_2$
appos	$w_1 \underset{0}{\overset{0}{\rightleftharpoons}} w_2$
poss prep_of	$w_2 \xleftarrow{1} \texttt{HAS} \xrightarrow{2} w_1$
tmod	$w_1 \xleftarrow{1} \texttt{AT} \xrightarrow{2} w_2$
prep_with	$w_1 \xleftarrow{1} \texttt{INSTRUMENT} \xrightarrow{2} w_2$
prep_without	$w_1 \xleftarrow{1} \texttt{LACK} \xrightarrow{2} w_2$
prep_P	$w_1 \xleftarrow{1} \texttt{P} \xrightarrow{2} w_2$

Table 2.1: *Mapping from Stanford dependency relations to **4lang** subgraphs [32, p. 12.].*

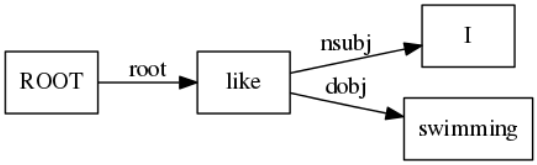

Figure 2.6: *Stanford example of a sentence*

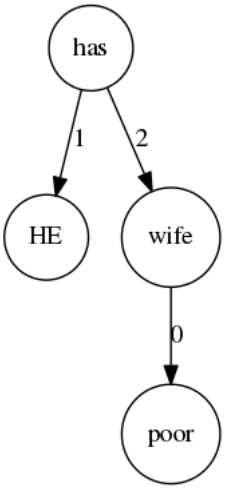

Figure 2.7: *4lang definition of sentence "My poor wife".*

pipeline with dictionaries. **4lang** takes advantage of this, and implements the expansion step, which is essentially joining the definitions graphs to the main graph. This allows us to build a larger graph, that contains more information, and allows us to model the text better by simply adding the definition of words.

Let us look at the example sentence *"My poor wife"*, that results the graph shown in Figure 2.7. Looking at the definition of the word *poor*: *having very little money and not many possessions*, we can build a definition graph and essentially join the two graphs together. This can result us a better model if we are ready to take word definitions into account, and with this method we can have higher similarities between graphs whose sentences are also similar [2]. Doing this for every word in the sentence resulting in a merged graph Figure 2.8. If we look at the graph, it is clear that the expanded graph gives us much more accurate context and definition. Our work was built around the expanded graph, and we will see that how better it actually performs on a real task. This will be the main topic of the next chapter.

2.3.3 The service

At the beginning of our research we put a high emphasis on generating graphs from raw text with a highly automated method, so besides being an open-source software library, we made `4lang` accessible via a public REST API at `http://hlt.bme.hu/4lang`. We used

[2] Of course not every interpretation of *"poor"* is related to money. It requires a higher level mechanism to handle these kind of occurrences, see more in [19].

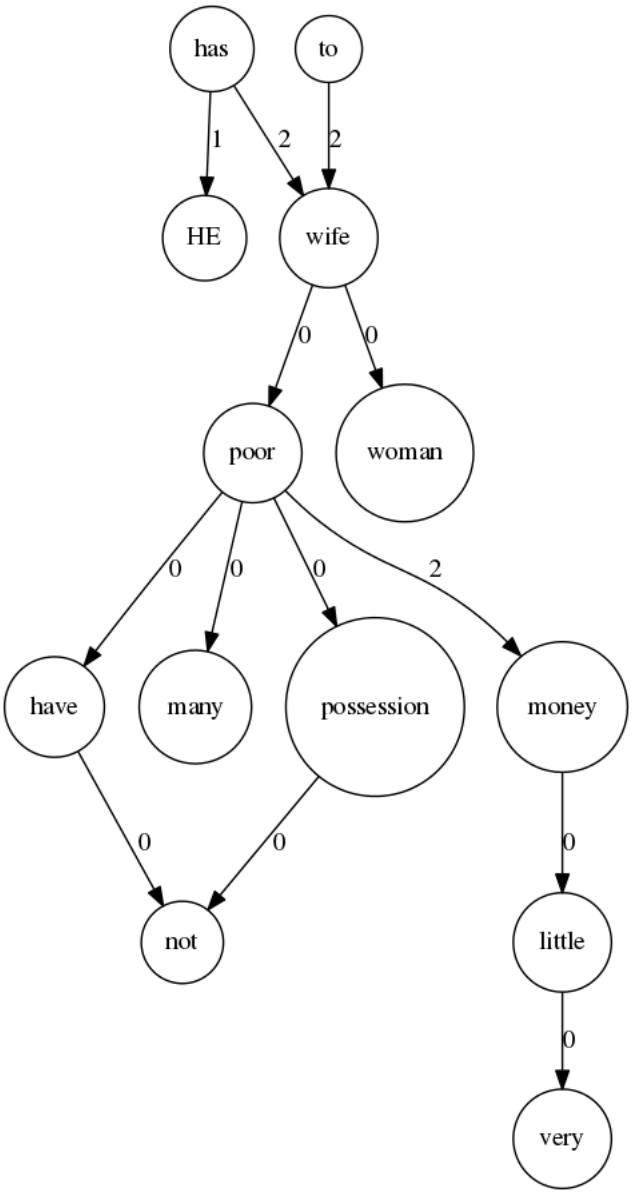

Figure 2.8: *4lang definition of expanded sentence "My poor wife".*

the python language for implementing the service, and for the framework we used the flask[3] package. The service generates input for the `text_to_4lang` module from the raw text input, then after processing it, returns a graph in **networkx multigraph**[4] format to make it easy to visualize it on essentially any client side. Online demo of the service is available at `http://4lang.hlt.bme.hu`. The process is presented in Example 2.1.

```
sentence = "I like micro-services."
data = {'sentence':   sentence}
data_json = json.dumps(data)
payload = {'json_payload': data_json}
headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
r = requests.post("http://hlt.bme.hu/4lang/sendef",
data=data_json, headers=headers)
s_machines = r.json()['sentence']
```

Example 2.1: *Demonstration of the service in python language.*

The generated graph can be seen in Figure 2.9. The code of the service is publicly available on Github[5]. We can generate graphs with raw text by calling the service. Currently our service has multiple endpoints, with each of them representing different methods. If you are only interested in processing a single sentence, the following endpoints are available:

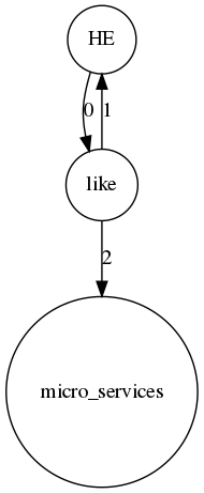

Figure 2.9: *4lang definition of sentence "I like micro-services".*

- **/sendef** - Returns the graphs built from the sentence.
- **/senexp** - Returns the graphs, where the word's definition has been added to the graph.
- **/senabs** - Calling this function, we defined some rules, where we can build a more abstract graph using the definitions, this is more of a future work, we will briefly mention the algorithms and ideas behind it in the last chapter.

[3] http://flask.pocoo.org/

[4] `https://networkx.github.io/documentation/networkx-1.9.1/reference/classes.multigraph.html`

[5] `https://github.com/adaamko/4lang`

You can get a word's definition by calling the defined endpoint:

- **/definition** - Returns the graphs built from the word's definition.

For the machine comprehension task we defined a dedicated endpoint:

- **/rally** - Returns a merged graph, where we merge a question sentence with an answer sentence. The main goal of this endpoint is that we can get a graph that can be explicitly compared with graph built from the passage text (it will be explained in the next chapter in details).

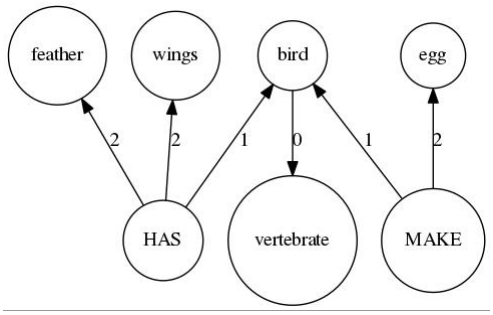

Figure 2.10: *4lang definition of* `bird`.

Graphs generated by the `4lang` parser have previously been used successfully in measuring semantic similarity. The current state of the art system on the `SimLex-999` benchmark [15] outperforms previous top systems by utilizing a simple similarity metric between `4lang` definitions of pairs of English words [33], this was the main idea of trying it in a different task with a different state-of-the-art system.

In the next chapter, we will briefly discuss the machine comprehension challenge, and we will introduce our baseline method for solving it using our micro-service for automatically building concept graphs. After that we will present how it is applicable to an already working system.

Chapter 3

Machine Comprehension

In this chapter a brief introduction to the machine comprehension task is given, followed by the definition and application of our metric, that was defined for measuring similarity between concept graphs. After our baseline for the task presented.

The 2018 Semeval Task *Machine comprehension using commonsense knowledge*[1] requires participants to train systems that can choose the correct answer to simple multiple choice questions based on short passages describing simple chains of events. Data for both training and testing is extracted from the `MCScript` dataset [28]. Some questions can only be answered using commonsense knowledge, and are explicitly labeled as such. For example, one passage might describe a story of a gardener planting a tree, and one of the questions would subsequently ask whether the gardener used his hands or a shovel to dig a hole for the tree, even though the answer to this question is not present in the passage. The top two systems, `HFL-RC` [10] and `Yuanfudao` [39] achieved accuracy scores of 84.15% and 83.95% on the test data, respectively. In our experiments we used semantic graphs to augment the `Yuanfudao` system, since its source code is publicly available[2] and since it already employs successfully a knowledge base representing semantic relationships among pairs of words.

3.0.1 Comprehension, entailment, and knowledge bases

In the next section we shall present a simple method for measuring *support*, the continuous counterpart of *entailment*, based on graphical representations of meaning, and use this metric in a baseline for machine comprehension and to improve a state of the art MC system. Although explicit representations of semantics are rarely used for this purpose, in recent years there have been several attempts at leveraging lexical ontologies in machine comprehension, and the approach of using textual entailment as an intermediary task is also not new. [37] achieves competitive results on the `MCTest` dataset [34] by generating answer candidates and ranking them using a separate RTE system, which is trained on the Stanford Natural Language Inference (SNLI) dataset [7] but also relies on an explicit measure of lexical overlap between sentence pairs. Other recent systems are various extensions of a baseline proposed by [34] that measures a weighted overlap between pairs of bag-of-words

[1] `https://competitions.codalab.org/competitions/17184`
[2] `https://github.com/intfloat/commonsense-rc`

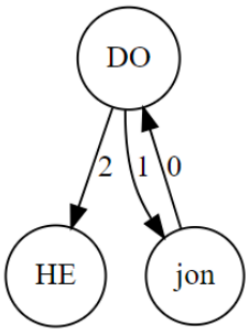

Figure 3.1: *Merged graph of answer candidate "Jon" for the question Who did it?*

representations, e.g. [38] applies the frame semantic parser of [12] and includes features representing overlap between bag-of-frame and bag-of-argument representations. Finally, the `Yuanfudao` system presented in this section is the most recent example of enhancing the performance of an MC system using a lexical knowledge base: ablation studies show that their top-ranking accuracy score of 83.84% drops to 82.78% if `ConceptNet`-based features are removed.

3.1 Method

We define a simple metric between pairs of `4lang` graphs that we intend to use for measuring entailment between a paragraph and a sentence. We shall define the degree to which some graph G_1 *supports* another graph G_2 as the ratio of edges in G_2 that are also present in G_1:

$$S(G_1, G_2) = \frac{|E(G_1) \cap E(G_2)|}{|E(G_2)|}$$

Two (directed) edges are identical if their source and target nodes and their labels are all identical. Based on early findings we used only *expanded* `4lang` graphs (see Section 2.3) for measuring support. To create a simple baseline solution for the Machine Comprehension task, we compare answer candidates to each question by comparing the degree of support for each in the passage, based on the `4lang` representations of each piece of text. For wh-questions we can create representations of each answer by merging the question graph's wh-node with the graph of each answer graph (see Figure 3.1), for this we use the **/rally** endpoint of our service defined in the previous chapter. Our baseline method will simply pick the answer candidate with the higher support score.

3.1.1 Experiments

Demonstrating the algorithm behind our baseline, let's look at the example passage:

"I went into my bedroom and flipped the light switch. Oh, I see that the ceiling lamp is not turning on. It must be that the light bulb needs replacement. I go through my closet

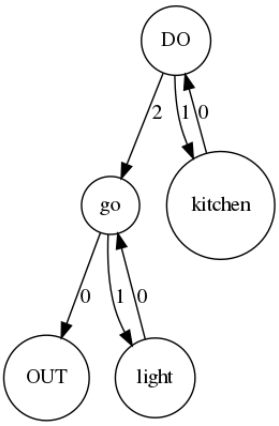

Figure 3.2: *Merged graph of answer candidate "Kitchen."*

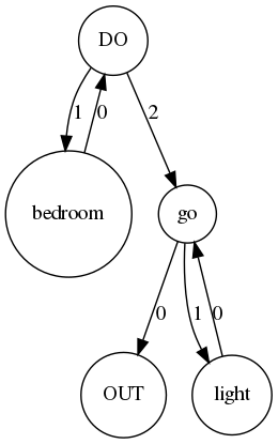

Figure 3.3: *Merged graph of answer candidate "Bedroom."*

and find a new light bulb that will fit this lamp and place it in my pocket. I also get my stepladder and place it under the lamp. I make sure the light switch is in the off position. I climb up the ladder and unscrew the old light bulb. I place the old bulb in my pocket and take out the new one. I then screw in the new bulb. I climb down the stepladder and place it back into the closet. I then throw out the old bulb into the recycling bin. I go back to my bedroom and turn on the light switch. I am happy to see that there is again light in my room."

And a question related to the text: *Which room did the light go out in?* and the answers:

- *"Kitchen."*
- *"Bedroom."*

First we build the expanded graph from the text. After we build the merged graphs (for the demonstration, we now only build the graphs without expansion) seen in Figure 3.2 and Figure 3.3. The graph without merging can be seen in Figure 3.4. After the merging, we compare both of the graphs to the passage graph applying our defined metric.

We tested the baseline method described in the previous section on a subset of all questions in the train section of the MC dataset: wh-questions that were not categorized

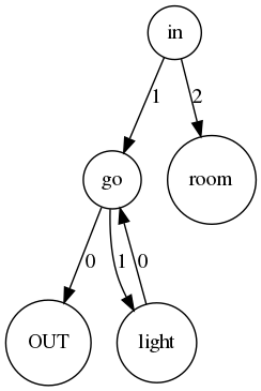

Figure 3.4: *Graph built from the question*

as "common-sense". Of this subset of 5,375 questions (of a total of 9,731), our method correctly answers 3,671, achieving an accuracy score of 68.3%. We then proceeded to use the metric underlying our baseline as an additional feature in the `Yuanfudao` system.

In the next chapter we will briefly give an introduction into the field of deep learning in general, and particularly in NLP applications, and then we will demonstrate the Yuanfudao system.

Chapter 4

Deep learning neural networks

In this chapter we will introduce the basic deep learning related concepts necessary to understand the model described in the next chapter.

4.1 Context

Deep learning neural networks gained popularity again in recent years, after the AI winter ended. Since then neural networks became part of the everyday life, we hear about artificial intelligence being used in smart phones to enhance images quality or recognize certain settings and take a picture accordingly. AI is also being used in cancer research and other fields of bioinformatics, and personal assistants became extremely popular as well.

But even with the seemingly endless capabilities of artificial neural networks we are far from creating anything that could have the cognitive capacities of humans. This problem is considered by many to be AI-complete or AI-hard, meaning that creating a network that could keep up a human-like conversation would require data scientist and researchers to construct a universal artificial intelligence. A small but important step to achieve this is to create a system capable of doing simple reading comprehension tasks that also require some common sense knowledge.

The most common structures of these experimental neural networks include recurrent neural network layers and attention layers. Before we explain the function of these building-blocks we need to lay down a foundation.

In this chapter we will introduce the deep learning mechanisms that are the essential building blocks of the Yuanfudao system described in Chapter 5.

4.2 Basics

Some arbitrary definitions we will use in this chapter:

- **Batch**: The chunks of training data that we use for training in one iteration. Its size can be a crucial parameter to set.
- **Epoch**: One training iteration is called an epoch. The number of epochs determines how long we want to train our network.
- **Learning rate**: η the velocity of the learning process. Setting it too low could result in slow learning and getting stuck in a local minimum, but setting it too high could result in jumping over the optimum and bouncing.
- **Accuracy**: The ratio of the correct predictions from all of the predictions. Our goal is to achieve a high accuracy with our neural network on the test set.
- **Test set**: The part of the dataset that we don't use during the training of our neural network. This determines the actual accuracy of the network.
- **Training set**: The data we could use for training our network is usually split into two parts in 80:20 ratio. We use the bigger set of data to train our network and we call this the training set.
- **Development set or Validation set**: The smaller portion of the data. We use this section to validate the network during its training. We do not use this section of the data to train the network.

People usually use supervised learning in the field of natural language processing to achieve the desired structure, but we might face multiple challenges while training. That's the reason why we have to use optimization and regularization methods.

4.2.1 Optimization

The goal of optimization is to overcome the following problems:

- Local minimum
- Setting the learning rate
- Setting the batch size

We have touched on the first two briefly before, but setting the batch size is also very important. We give the training data to the network in batches, and we iterate through them multiple times (depending on the epoch).

Methods used for optimizing:

- **Stochastic gradient descent**
- **Momentum**

- **Adaptive learning rate**
- **Adam** (Adaptive learning rate and momentum)
- **AdaMax** (Adam variant)

Stochastic gradient descent

$$g \leftarrow \frac{1}{batchSize}\Delta_w \sum_{i=1}^{batchSize} Loss(f_w(x_i), y_i)$$

Where f is the network itself.

$$w \leftarrow w - \eta g$$

This is a classical method for optimizing. It does not account for dynamic parameter settings. The system described in Chapter 5 can use this function. We will consider this as our base, and only highlight the differences between this and other optimization methods.

Momentum

$$g \leftarrow \alpha g + \frac{1}{batchSize}\Delta_w \sum_{i=1}^{batchSize} Loss(f_w(x_i), y_i)$$

Where α is a parameter we can set. This method has an added parameter, that helps to take into account the previous iterations, giving a momentum to the learning towards the optimum.

Adaptive learning rate

$$r \leftarrow r + g^2$$

Where r is a parameter thats initial value can be set.

$$w \leftarrow w - \frac{\eta}{\sqrt{r}} g$$

As the name suggests the adaptive learning rate uses a parametrization to slowly decrease the learning rate through time, depending on the previously calculated gradient.

Adam

$$r \leftarrow \varphi_r r + (1 - \varphi_r) g^2$$

$$s \leftarrow \varphi_s s + (1 - \varphi_s) g$$

Where s is a parameter thats initial value can be set and φ is a parameter of the decay rate.

$$w \leftarrow w - \frac{\eta s}{\sqrt{r}} g$$

Adam is one of the most popular optimizer function nowadays. It combines the adaptive learning rate and the momentum hence the name: Adam.

AdaMax

$$r \leftarrow \max(\varphi_r r, |g|)$$

$$s \leftarrow \varphi_s s + (1 - \varphi_s) g$$

$$w \leftarrow w - \frac{\eta}{1 - \varphi_s} \frac{s}{r}$$

A variant of the Adam optimizer. It was first described in the *Adam: A method for stochastic optimization* article [17]. It differs from Adam in that it uses a max operation and the infinity norm. The system described in Chapter 5 can use this optimization function too.

4.2.2 Regularization

The goal of regularization is to avoid over-fitting. Over-fitting happens when our neural network produces good results on the training set's dedicated subset, the development or validation set, but it can't predict the expected outputs properly on a new dataset for example the test set. It happens very often, so machine learning experts developed a couple functions to avoid this phenomenon.

- **Weight decay**: Also known as L2 regularization. It uses a parameter to make sure that the previously learned weight won't influence the new one too much.

$$w \leftarrow (1 - \eta\alpha) w - \eta \Delta_w Loss$$

- **L1 regularization**: This is a normalization method that modifies the cost function similarly to the L2 regularization. The difference is, that in this case the weights might get reduced to zero.
- **Dropout**: It randomly disables weights for an epoch, so they won't be used in that iteration.
- **Early stopping**: It stops the learning if the results on the development set haven't shown any progress in the last couple of epochs.
- **Noise injection**: It injects noise into the training set.

The system later described in Chapter 5 uses dropouts while learning. This might cause the system's performance to fluctuate a little bit from epoch to epoch but it is a powerful tool for regularization and as we'll see later it manages to achieve consistently good results on the development set.

4.3 Natural Language Processing with Deep learning

As mentioned above the kind of layers used in Natural Language Processing are mostly recurrent neural network layers, attention layers and sometimes embedding layers.

4.3.1 Embedding layers

The function of embedding layers is to turn integer values to fixed length vectors, for example word embedding vectors discussed in Chapter 2. They are used mostly in natural language processing to work as word2vec translation.

When using embedding layers we want to find vectors for each word so that it can model the word's meaning. We achieve this by looking at the context, the word usually appears in. If two words like *apple* and *orange* usually appear in the same context, than the vectors assigned to these words should have low cosine distance between them. You can read about this idea in detail in Chapter 2.

If you are building an NLP model, the embedding layer should be in the first layer, since its purpose is to make the transition from word to vector, and the word in this case is the input.

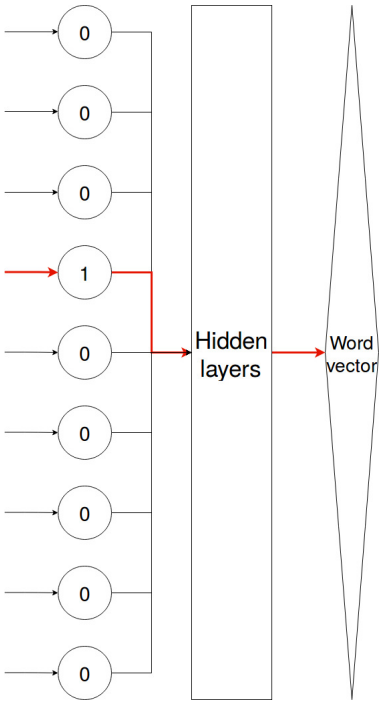

Figure 4.1: *An embedding layer.*

The input dimension of this layer is the size of the vocabulary and the output dimension is the size of the dense vector. Usually the vocabulary size greatly exceeds the embedding dimension since the output vectors size is fixed and can range from 300 to 1000, and the vocabulary - depending on the dataset - can be way higher than that. See at Figure 4.1.

They work mostly like a lookup table that can be trained. A lot of the times we use pre-trained models, like the GloVe embeddings that have been trained on enormous datasets.

We can also train them on our specific problem, or use the pretrained and our own embeddings simultaneously.

4.3.2 Recurrent Neural Networks

In a simple feed forward neural network, the information only moves in one direction: from the input layer to the output layer. On the other hand recurrent neural networks take into account their immediate past, the output of the network with the previous timestamp. This internal "memory" like functionality allows the network to remember what it had calculated before. This is illustrated at Figure 4.2.

At every timestamp the network gets two sets of inputs: the actual input at the timestamp and the hidden state of the network for the previous input. In one iteration it calculates its output using the calculated hidden state in the timestamp. It all could be imagined like the same feed forward network being repeated after one other.

The hidden state mentioned above is the "memory" of the network that is calculated with the previous hidden state and the input.

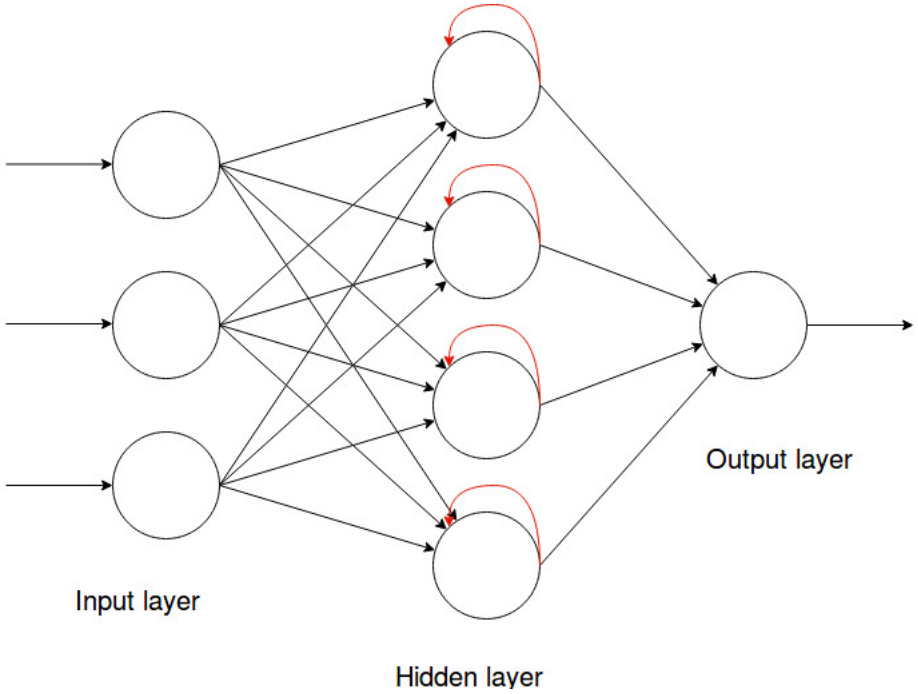

Figure 4.2: *A recurrent neural network.*

The backpropagation is also slightly different in this case, it's called backpropagation through time, you need to "unroll" the network (see at Figure 4.3), and use the backpropagation starting from the right timestamps. Each timestamp's backpropagation could be understood as backpropagation on a separate feed forward neural network.

The gradient vanishing or explosion can be a problem with this RNN.

There is a multitude of solutions for the exploding gradients, one of which is called gradient clipping. This is one of the method the state-of-the art system, Yuanfudao uses.

This technique is a very simple yet powerful way of dealing with exploding gradients. All

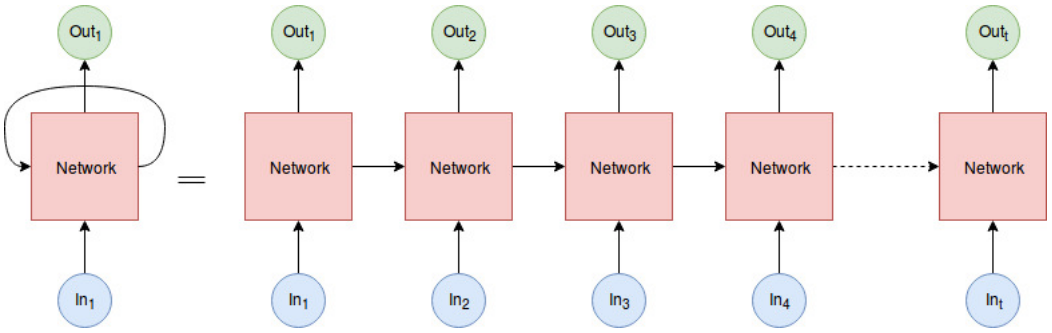

Figure 4.3: *Unrolled recurrent neural network.*

it does is that it limits the size of the gradient, if its norm is higher than a set threshold.

RNNs can be used in supervised and also unsupervised learning. They are used when the data is sequential, like text, audio, etc.

4.3.2.1 Long-Short Term Memory

The system we worked with uses LSTM layers as its default RNN type. Long-short term memory networks are the extension of the previously discussed recurrent neural network. The main difference is that it also has an internal long term memory. Usually these type of networks are more reluctant to have the exploding gradient problem.

Like the simple RNN, LSTMs also have hidden states, that are calculated slightly differently.

The LSTMs hidden states are calculated using three gates:

- **input gate**: determines whether to let new input in
- **forget gate**: determines whether to forget an input because it's not relevant anymore
- **output gate**: determines whether to let the input impact the output with the current timestamp

These gates are analog and their values ranges from 0 to 1 with the sigmoid function. A simplified depiction can be seen at Figure 4.4[1].

[1] `https://en.wikipedia.org/wiki/Long_short-term_memory`

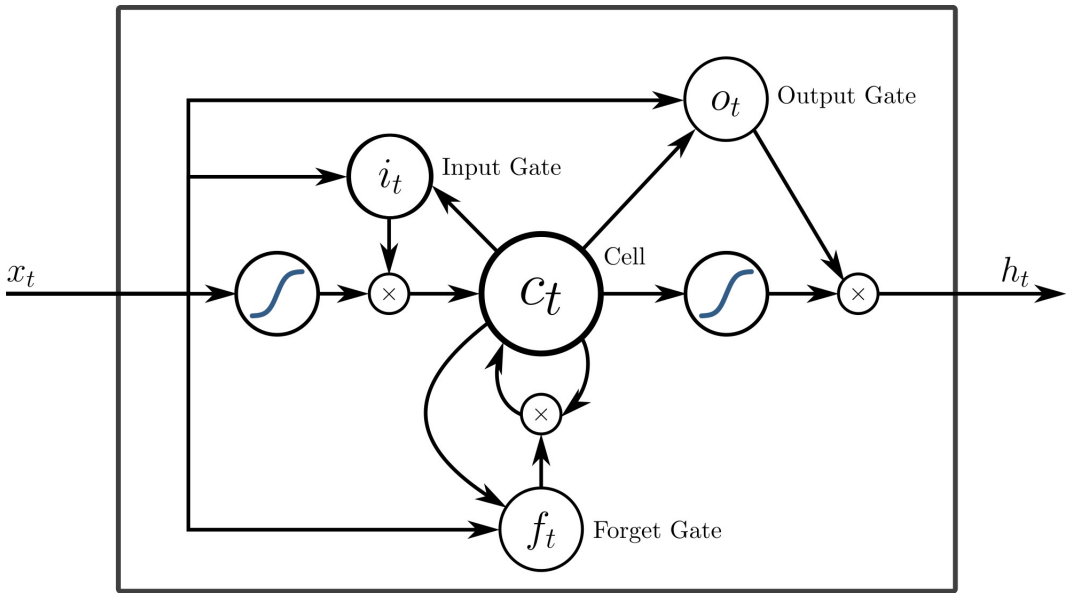

Figure 4.4: *A long-short term memory network's gates. Image from Wikipedia*

The sigmoid function allows this structure to be able to learn, meaning that we can use the backpropagation through time method described above.

Long-short term memory networks are used in natural language processing, but also in generative networks, like video or image description generation, text generation and so on.

Bidirectional long-short term memory also known as BiLSTM
BiLSTM networks are a variant of long-short term memory that read the input from the beginning to the end then also from the end to the beginning. The main idea behind this is that we may need not just the previous output, but also the next one too. These networks usually outperform simple LSTM systems in their predictions and the learning velocity too.

4.3.2.2 Gated recurrent unit

The system we worked with can use GRU layers as its RNN type, but it's not its default setting and it did not perform as good. Gated recurrent units are also a type of RNN and have a similar structure (Figure 4.5[2]) to the long-short term memory network, but has been shown to exhibit better performance on smaller datasets, than the LSTM.

[2] `https://en.wikipedia.org/wiki/Gated_recurrent_unit`

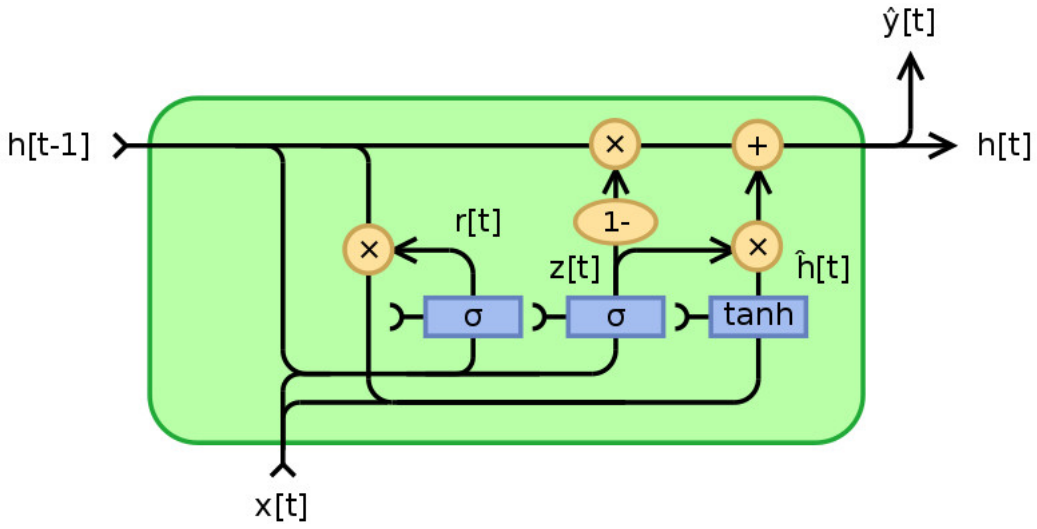

Figure 4.5: *A gated recurrent unit's gates.*
Image from Wikipedia

It has four main building blocks:

- **update gate**: the gate gets the x[t] and the h[t-1] as its input and z[t] is the output on the image
- **reset gate**: the gate gets the x[t] and the h[t-1] as its input and r[t] is the output on the image
- **current memory content**: the gate gets the x[t], the h[t-1] and the r[t] as its input and $\hat{h}$[t] is the output on the image
- **final memory at current time step**: the h[t] on the image

Gated recurrent units are mostly used on the field of speech recognition and music modeling while the LSTM is more relevant on the field of natural language processing.

4.3.3 Attention

The Attention mechanism was first described in [4] and was used for machine translation. Since then it became a widely used tool in natural language processing. The idea behind this mechanism is that when the neural network predicts the output, it only uses parts of the given input instead of the full input. That is where the most relevant information is concentrated and this mechanism only pays *attention* to these parts and the network has to learn what to pay attention to.

Usually in the "sequence-to-sequence" tasks like MT there are two main parts of the model an encoder and a decoder. The encoder and the decoder are usually some type of RNN, mostly LSTM. The encoder is responsible for creating a so called context-vector from the input sequence. This context-vector has a fixed length and it serves as the representation of the sequence inside the model. The decoder then decodes this context-vector to a sequence again, in the case of the machine translation this sequence is in a different language. A depiction can be seen at Figure 4.6.

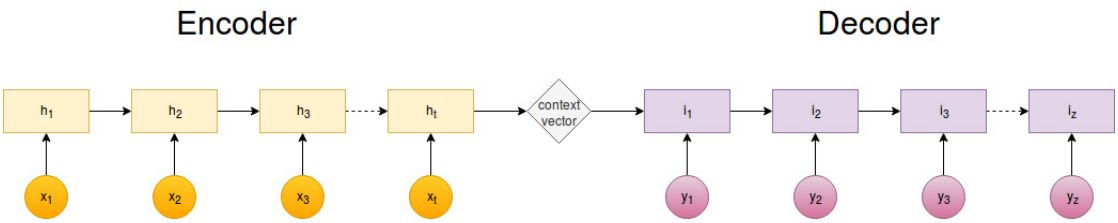

Figure 4.6: *A sequence-to-sequence model with encoder and decoder.*

The attention mechanism described in [4] was used in the decoder part of this model, the encoder functions the same way. The paper explicitly stated that this attention mechanism relieves the encoder from having to encode every sequence to a fixed length context vector. In this case we have a context vector for every word of the expected output. These context-vectors are the weighted sums of the encoder's states (*annotations*).

$$c_i = \sum_{j=1}^{t} \alpha_i j h_j$$

Where α parameter is calculated like the following:

$$\alpha_i j = \frac{exp(e_{ij})}{\sum_{k=1}^{t} e_{ik}}$$

and e_{ij} is its energy

$$e_{ij} = a(s_{i-i}, h_j)$$

This is an *alignment model* that scores how well the input around j and the output around i match. This *alignment model* is a feed forward neural network that is trained simultaneously with the other components of the system. The decoder uses the previous state's output and its assigned context-vector when calculating its own target.

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

The attention based model is at Figure 4.7.

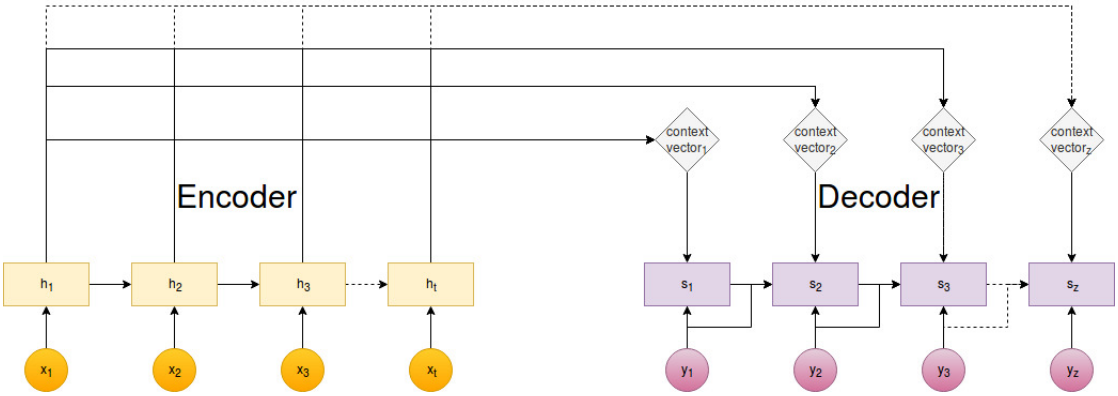

Figure 4.7: *A sequence-to-sequence model with encoder-decoder and attention.*

A big advantage of using this mechanism is the ability to interpret our model. These days it's more important than ever to be able to tell why does the network predict what it predicts, and thanks to the attention mechanism we are able to say that (in a purely attention-based network). One example is shown at Figure 4.8.

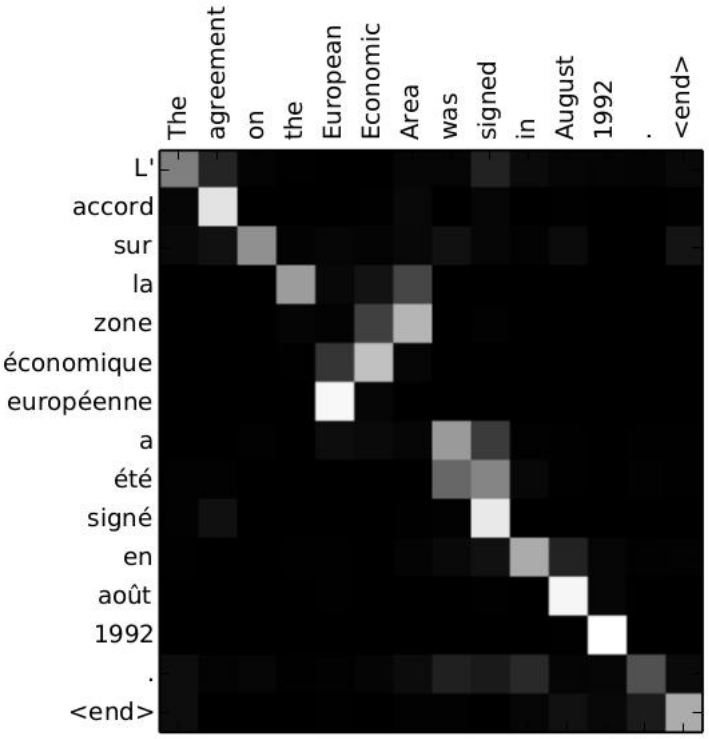

Figure 4.8: *An interpretation of a french - english sequence to sequence translation.*
Image from [4].

Since its first description in [4] the attention mechanism has been used for:

- **Image caption/description**: a convolutional neural network translates the image to the context vectors and the decoder creates a description for it. A recent system using attentions for image captioning is described in the *Image Caption with Global-Local Attention* paper [24].
- **Grammar representation**: in this case the decoder builds a grammatical representation for the input. On a related field there was a study regarding linguistic representations and attentions [3].
- **Advanced machine translation**: Since it was first introduced it has revolutionized the field of machine translation. A study on this field is *Effective Approaches to Attention-based Neural Machine Translation* [25].
- **Machine comprehension tasks**: question answering based on a previously read text. The system described in Chapter 5 is like this [39].

Chapter 5

Yuanfudao system

On the 2018 Semeval Task *Machine comprehension using commonsense knowledge* competition the `Yuanfudao` [39] system reached second place with 83.95% accuracy on the test data.

5.1 The original system

The `Yuanfudao` system implements a Three-way Attentive Network (TriAN), an ensemble of three LSTMs augmented with various attention mechanisms, to model for each question interactions between question, possible answers, and the passage that may or may not contain the correct answer to the question.

The system was implemented using python programming language and the `pytorch` package for the implementation of the neural network. The source code is available on Github[1].

5.1.1 Preprocessing

This system processes the input data as follows:

1. Using the `spacy` package's tokenizer function it generates the part of speech (pos) tag, named entity recognition (ner) tag and the lemma for each word in the passage, and the pos tags of the questions.

2. It assigns a number representation and an offset for each word in the passage, questions and answers.

3. It also saves the ids of the passages, questions and answers and whether the answer was correct.

4. The preprocessor finds the words and lemmas in the questions and answers, that also occurred in the passage's word and lemma list.

5. It stores each word's frequency using the `wikiwords` library.

[1] `https://github.com/intfloat/commonsense-rc`

6. It establishes the *ConceptNet* relation between the words of the passage and question and also between the words of the passage and answer.

7. The preprocessor saves all this data to their respective json files.

Conceptnet [35]

ConceptNet plays a major part of the `Yuanfudao` system, as it was shown in the original paper [39]. This is a metric used to show the possible relationship between two words. These relations could be "RelatedTo", "IsA", "Synonym", "PartOf" etc.

The *ConceptNet* itself is a large graph of general knowledge that has shown to be affective at determining word relations.

The preprocessor compares the words in the passage with the words in the "query" (question or answer) using *ConceptNet* and stores only one of the matches per word, if there were any.

5.1.2 System description

An overview of the original system is reproduced in Figure 5.1.

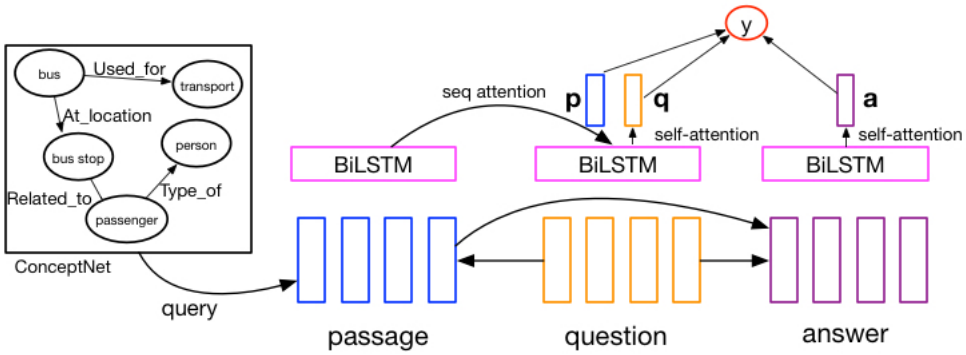

Figure 5.1: *Structure of the original network [39]*

This system is a deep learning neural network consisting of embeddings, recurrent neural networks and attention mechanisms.

First the inputs generated in the preprocessing phase go through three embedding layers, each corresponding to the passage, question and answer respectively. There are also pos-embedding, ner-embedding and rel-embedding layers. The pos-embedding gets the passage's and the question's pos tags as its input, the ner-embedding layer gets the passage's ner-tags and the relation-embedding gets the relationship vectors generated using the *ConceptNet*. This input embedding layer is shown in Figure 5.2.

The word embeddings' outputs are paired up (passage-question, answer-question, answer-passage) and go through a so called *sequence attention matching layer*. The sequence attention matching layer at its core uses the bmm function in `pytorch` which performs a batch matrix-matrix product of the input matrices. This way it "matches" the two inputs together. This is shown in Figure 5.3.

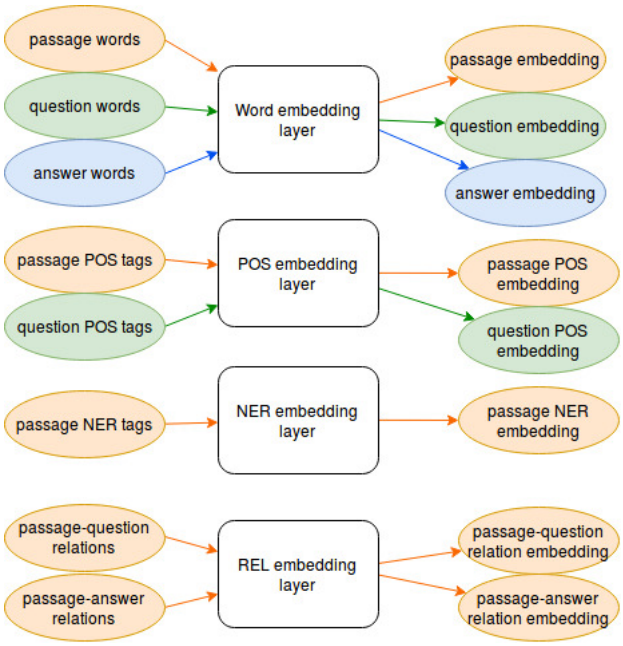

Figure 5.2: *Structure of the input embedding layers.*

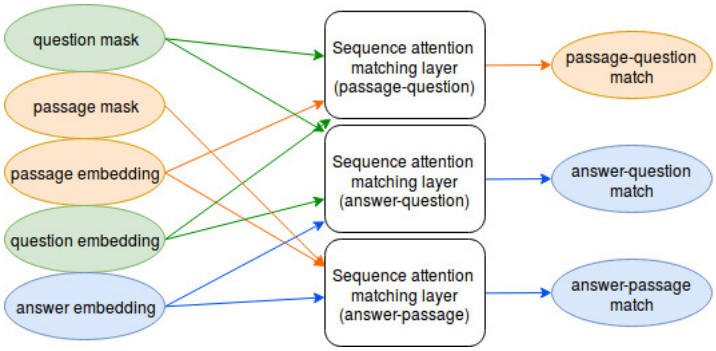

Figure 5.3: *Structure of the sequence attention matching layers.*

The system uses dropouts after the embedding and sequence attention matching layer layers to avoid over-fitting.

These layers are followed by three *stacked bidirectional RNN layer*, each corresponding to the passage, question and answer respectively. It differs from the standard bidirectional RNN layer in one aspect: it can concatenate the hidden states of the RNN. By default the type of the RNN is LSTM, but it can also be GRU. Their inputs are sort of self explanatory. The passage's stacked bidirectional RNN layer gets the passage's word embedding layer, the output of the sequence attention matching layer for the passage-question input pair, the passage's pos- and ner-embedding layers, the word frequency tensor created with the `wikiword` library, and the two relation-embedding layer's output. The question's stacked bidirectional RNN layer expects the question's word and pos-embedding outputs on its input. The answer's stacked bidirectional RNN layer's inputs are the answer's word embedding output and the output of the sequence attention matching layer for the answer-question and the answer-question input pairs. These RNN layers are shown in the Figure 5.4. This layer implicitly uses a dropout rate for regularization.

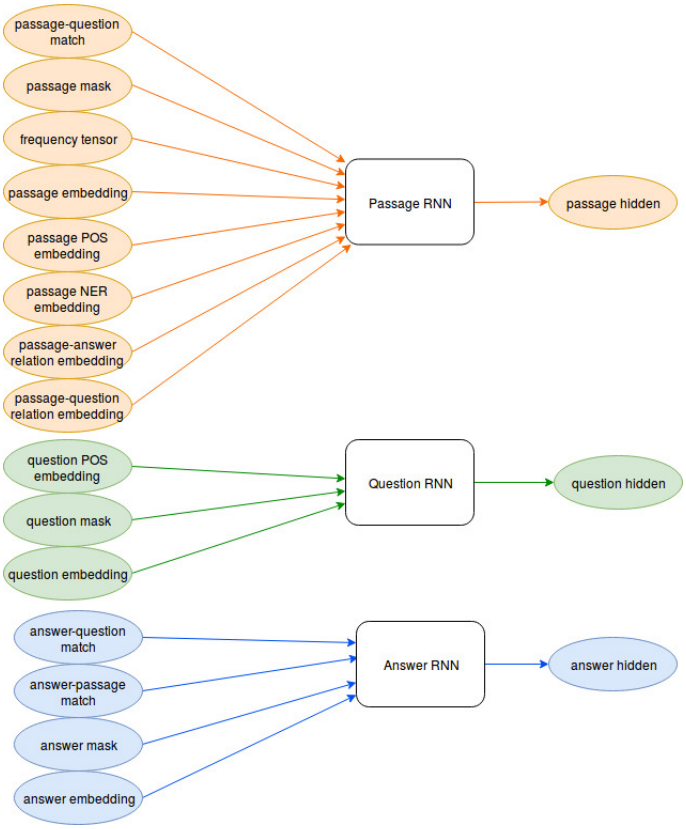

Figure 5.4: *Structure of the stacked bidirectional RNN layers.*

The question's and the answer's stacked bidirectional RNN layer's outputs are used in two *linear sequence attention layers*, or better known as *self-attention layers over a sequence* for the question and the answer respectively. This layer is basically a linear layer slightly modified, so the infinite outputs are masked and it uses a softmax function at its output.

The passage's stacked bidirectional RNN layer's output is used differently. The system passes it and the question's *stacked bidirectional RNN layer's* output to a *bilinear sequence attention layer*, which is similarly to the sequence attention matching layer uses the bmm function as its core function.

The two linear sequence attention layer's and the bilinear sequence attention layer's output is passed through a weighted averaging function with their respective stacked bidirectional RNN layer's output. This part of the network is shown in Figure 5.5.

The averaged passage output is passed through a *linear feed forward layer* then multiplied by the answer's averaged output. The question's averaged output is passed through an other linear feed forward layer than multiplied by the answer's averaged output. At the end its all summed and sigmoid function used at its output. The output in this case is whether the answer was correct to the given question or not. This last section is at Figure 5.6.

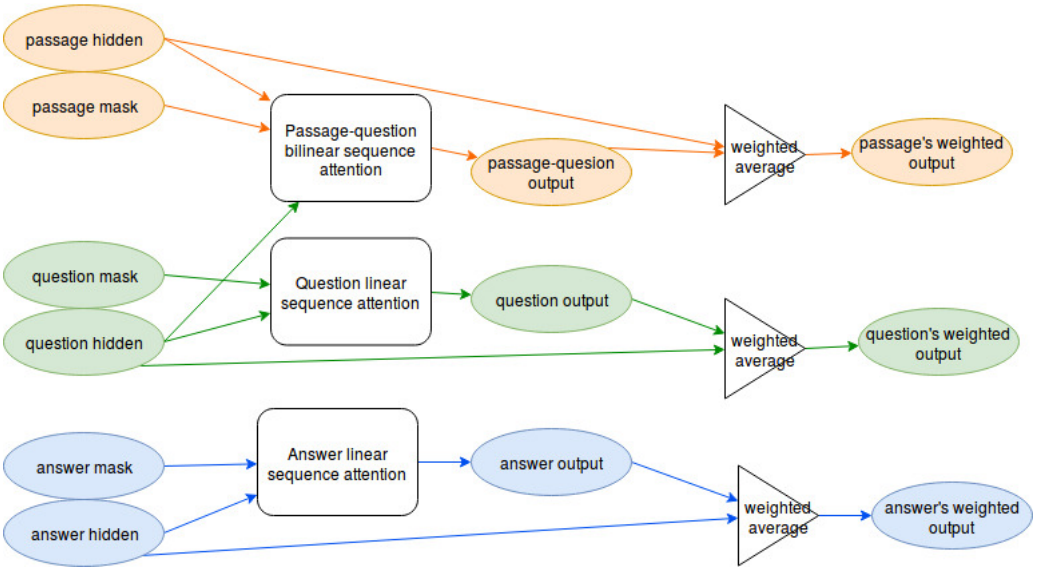

Figure 5.5: *Structure of the sequence attention layer and the following weighted average function.*

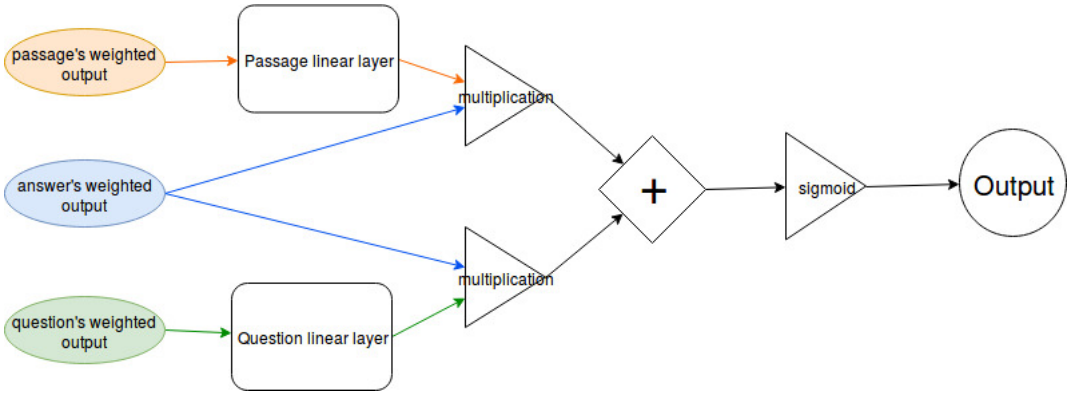

Figure 5.6: *Structure of the output of the network.*

5.1.3 Parameters

The `Yuanfudao` system has these following command line arguments:

- **GPU**: the training of the system can be done on GPU which is much faster than training it on CPU
- **using cuda**: `pytorch` can support CUDA for parallelization. The system uses CUDA by default.
- **optimizer**: the optimizer function can be adamax (default) or SGD
- **RNN type**: the RNN used by the system can be LSTM or GRU
- **dropout rate**: there are separate dropout rates for embeddings and RNNs
- **embedding dimension**: each embedding dimension in the system can be manually set
- **gradient clipping**: the gradient clipping threshold can be set
- **epoch**
- **learning rate**
- **batch size**
- **random seed**
- other parameters related to input handling, RNN settings and testing

You can read about the deep learning related arguments and their functions in Chapter 4.

5.1.4 Learning curve

Without the recommended pretraining (Figure 5.7):

- Max dev accuracy: 82.7% reached in the 26th epoch
- Train accuracy: 97.7% reached in the 26th epoch
- Max train accuracy: 99.8% reached in the 50th epoch
- Last dev accuracy: 81.9%
- Average dev accuracy after ten epochs: 81.9%

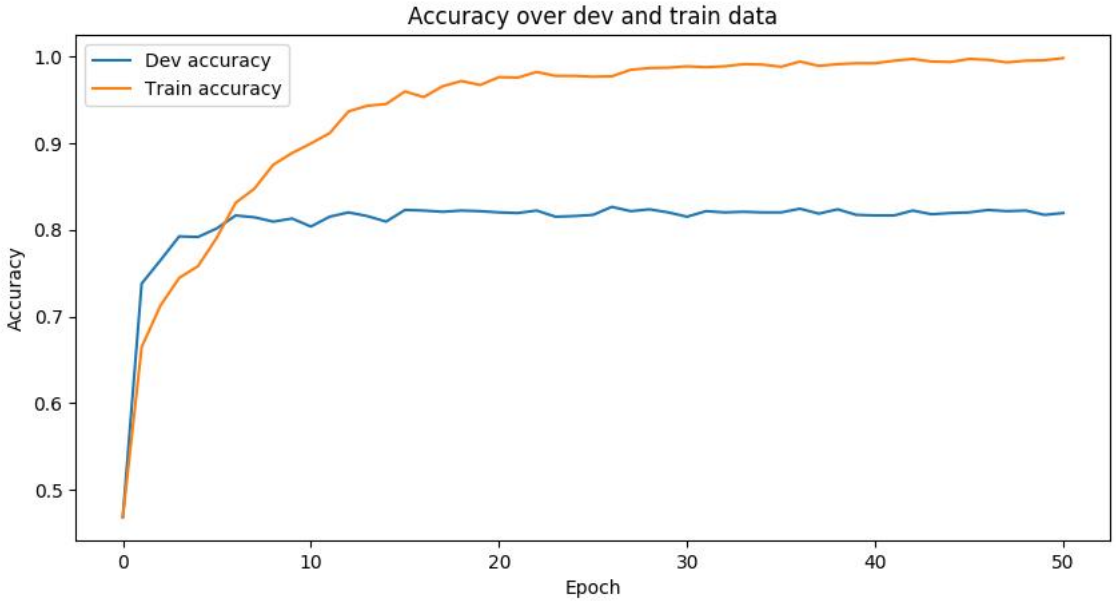

Figure 5.7: *Learning curve without pretraining.*

With the recommended pretraining(Figure 5.8):

- Max dev accuracy: 82.5% reached in the 38th epoch
- Train accuracy: 99% reached in the 38th epoch
- Max train accuracy: 99.7% reached in the 50th epoch
- Last dev accuracy: 82.2%
- Average dev accuracy after ten epochs: 81.9%

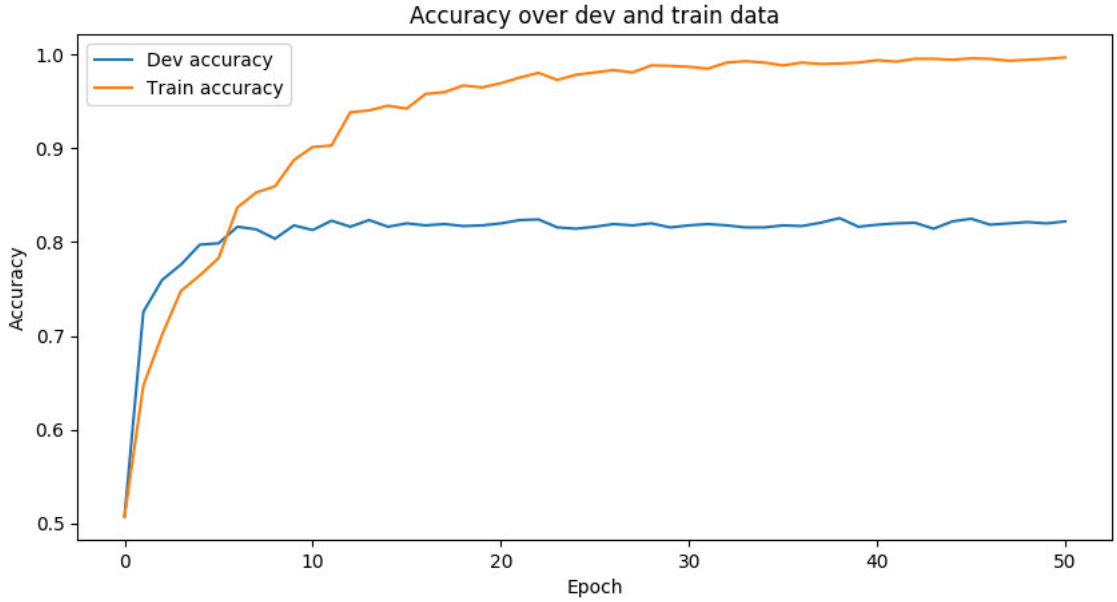

Figure 5.8: *Learning curve with pretraining.*

As you can see there is no significant difference between the learning curve with or without pretraining.

5.2 Modifications

Our modifications are available on Github[2].

We modified the preprocessing part of the system to incorporate the similarity calculating method from Chapter 3. The most straightforward way of incorporating our metric into the system is by creating vectors similar to those representing *ConceptNet* relations between words of a passage and words in each answer candidate. Since these vectors represent word-to-word relationships, we measure the support between pairs of `4lang` definition graphs, and for each word in the passage we take the maximum support score over all words of the answer candidate. Elements of a vector for a passage P and a possible answer A are hence defined as:

$$S_i^{(P,A)} = \max_{A_j \in A} S(P_i, A_j)$$

Elements of a vector for a passage P and a question Q are defined as:

$$S_i^{(P,Q)} = \max_{Q_j \in Q} S(P_i, Q_j)$$

Elements of a vector for a question Q and an answer A are defined as:

$$S_i^{(Q,A)} = \max_{A_j \in A} S(Q_i, A_j)$$

We used these new input vectors as the input of a new `4lang` embedding layer that functions similarly to the other embedding layers. It is shown at Figure 5.9. The input of this layer is 101 dimensional, since the similarities are on a scale to 0 to 100.

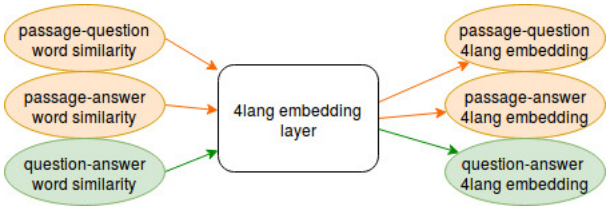

Figure 5.9: *`4lang` embedding layer.*

The outputs of this layer are passed to the RNN layers. This is depicted at Figure 5.10.

Since we also wanted to see how the system changes if we replace *ConceptNet* relations with our metric, we also trained systems without *ConceptNet* rel-embeddings.

[2] `https://github.com/GKingA/commonsense-rc`

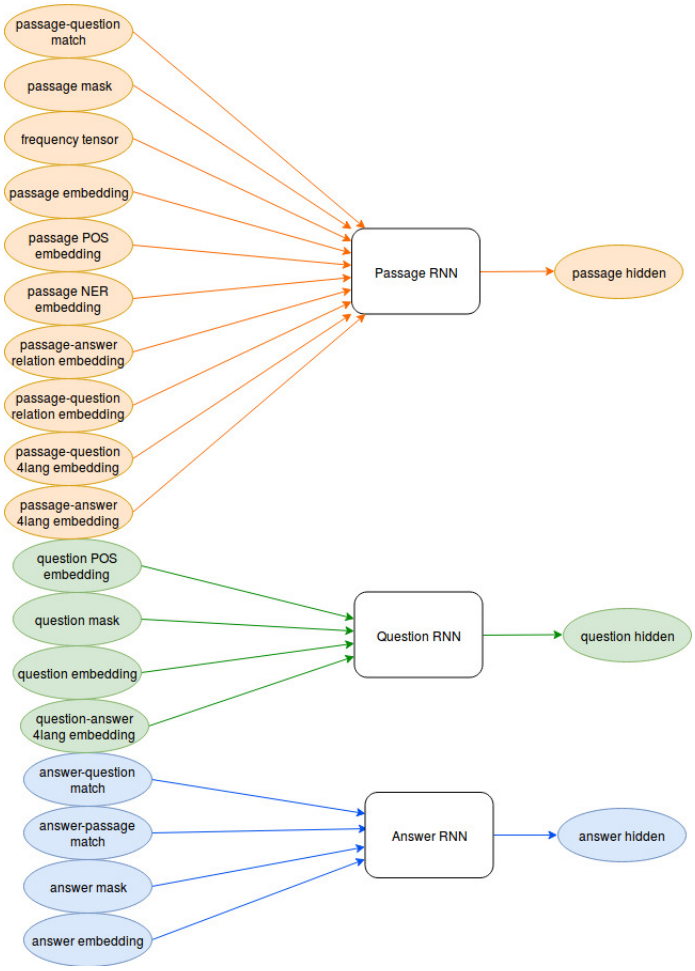

Figure 5.10: *Structure of the modified stacked bidirectional RNN layers.*

5.3 The results

The original `Yuanfudao` [39] publication said its system was able to reach 83.95% accuracy on the test data. We were only able to reproduce a 80.3% accuracy on the test set and 82.5% on the development set with the recommended pretraining on the `RACE` [23] dataset. We will take these results as our bases of the comparison.

We tested our model by turning on and off the usage of *ConceptNet* and `4lang`. There were 4 combinations: using neither, just *ConceptNet*, just `4lang` and both.

model	dev	test
pretrained TriAN, no ConceptNet	83.7%	81.9%
pretrained TriAN, with ConceptNet	82.5%	80.3%
pretrained TriAN, with 4lang	84.2%	81.5%
pretrained TriAN, with both	**83.4%**	**82.9%**
TriAN, no ConceptNet	82.8%	80.2%
TriAN, with ConceptNet	82.7%	80.5%
TriAN, with 4lang	83.2%	80.9%
TriAN, with both	83.1%	80.8%

Table 5.1: *Effect of `4lang` and `ConceptNet` on results*

It is evident that without pretraining the Yuanfudao system performs best if we use the relation scores calculated from `4lang` graphs instead of the *ConceptNet* relationships. After pretraining the network on the `RACE` dataset the results show that using both of the relation metric is the most beneficial.

Chapter 6

Conclusion and future work

6.1 Summary

This thesis proposes a novel method for recognizing entailment using semantic graphs and apply it to the 2018 Semeval task on Machine Comprehension (MC). First, a brief overview of the field of natural language processing is given focusing on real life applications, that are in need of the NLP technologies. Then the topic of computational semantics was discussed in details, focusing on one-two major tasks, like question-answering or information retrieval. After that the formalism of 4lang was presented with examples, and the method of expansion was discussed. After simple yet a strong baseline method was presented for measuring textual entailment and its application to the comprehension task, followed by an introduction to the field of deep learning. Finally the last chapter reports the results of applying the baseline method to the MC task and also of using it as an extra feature in the neural network based `Yuanfudao` system.

6.2 Future work

Our results are quite promising, but further experiments are required to explore whether our enhancements can improve the top-ranking system that also employs pretraining and an ensemble of multiple models. We also plan to incorporate sentence-level support into the system as a more direct application of our baseline.

We also have some experiments with defining new additional rules added to the **4lang** parser, that could potentially be giving us a more abstract and simpler definitions than the *expansion* method. Let us look back the sentence "My poor wife" and the expanded graph shown in Figure 2.8. In this example if we look at the edge **wife** $\xrightarrow{0}$ **woman** we can make an assumptions, that native speakers can easily make using simple inference rules [22]. In our example, within the boundaries of the sentence we can use the concept **woman** instead of the concept **wife**. Using this simple rule we can reduce our graph to a simpler definition shown in Figure 6.1.

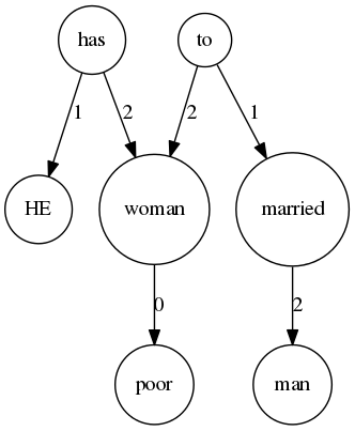

Figure 6.1: *Example of the "abstract" method*

6.2.1 Interpreted Regular Tree Grammar

We recently started experimenting with Interpreted Regular Tree Grammars [18] (IRTG) that we could also use to construct **4lang** graphs, since they implement graph transformations, so graph grammars can be used. And by modifying the rules of the grammar, we can also accomplish the *expand* functionalities and we also can define inference rules. These experiments are not yet perfect, but this approach shows great potential. See the phrase *"Ordinary email"* represented in Figure 6.2.

The base of this approach is to define grammar files where we describe the rules using multiple graph, tree or string algebras. This allows us to use it as a graph rewriting grammar file which we can use to transcribe for example a universal dependency graph to a `4lang` graph.

We used an already functioning grammar that defined the relationships between universal dependencies and `4lang` graphs and modified it to incorporate the definition of the words also [14].

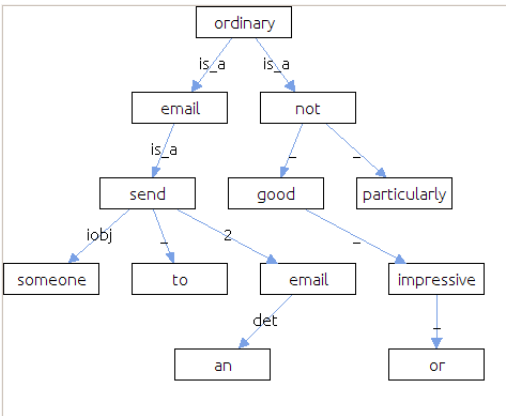

Figure 6.2: *Example of the "expand" method using IRTG*

Chapter 7

Acknowledgement

We would like to thank Dr. Recski Gábor and Dr. Kornai András for their constant support and help with our work. Without their guidance we would not have been able write this paper.

List of Figures

List of Tables

Bibliography

[1] Embedding additions. `https://blogs.mathworks.com`.

[2] Parser differences. `https://upload.wikimedia.org/wikipedia/commons/e/e7/Johnhasfinishedthework-1.jpg`.

[3] Afra Alishahi Ákos Kádár, Grzegorz Chrupala. Representation of linguistic form and function in recurrent neural networks. *arXiv preprint arXiv:1602.08952*, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR 2015)*, 2015.

[5] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, 2013. Association for Computational Linguistics.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[7] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, 2015. Association for Computational Linguistics.

[8] Jose Camacho-Collados, Mohammad Taher Pilehvar, Nigel Collier, and Roberto Navigli. Semeval-2017 task 2: Multilingual and cross-lingual semantic word similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 15–26, Vancouver, Canada, 2017. Association for Computational Linguistics.

[9] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada, 2017. Association for Computational Linguistics.

[10] Zhipeng Chen, Yiming Cui, Wentao Ma, Shijin Wang, Ting Liu, and Guoping Hu. Hfl-rc system at semeval-2018 task 11: Hybrid multi-aspects model for commonsense reading comprehension. *arXiv preprint arXiv:1803.05655*, 2018.

[11] Vu; Le Minh Nguyen; Satoh Ken Dac Viet, Lai; Trong Sinh. Convamr: Abstract meaning representation parsing for legal document. In *ConvAMR*, 2017.

[12] Dipanjan Das, Nathan Schneider, Desai Chen, and Noah A Smith. Probabilistic frame-semantic parsing. In *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*, pages 948–956. Association for Computational Linguistics, 2010.

[13] Marie-Catherine DeMarneffe, William MacCartney, and Christopher Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, volume 6, pages 449–454, Genoa, Italy, 2006.

[14] Ács Evelin and Gábor Recski. Semantic parsing using interpreted regular tree grammar. In *AACS*, 2018.

[15] Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2014.

[16] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. 3rd edition edition.

[17] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, pages 8–9, 2015.

[18] Alexander Koller and Marco Kuhlmann. A generalized view on parsing and translation. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*, Dublin, 2011.

[19] András Kornai. *Semantics*. Springer Verlag, 2018.

[20] András Kornai, Judit Ács, Márton Makrai, Dávid Márk Nemeskey, Katalin Pajkossy, and Gábor Recski. Competence in lexical semantics. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM 2015)*, pages 165–175, Denver, Colorado, 2015. Association for Computational Linguistics.

[21] András Kornai and Márton Makrai. A 4lang fogalmi szótár. In Attila Tanács and Veronika Vincze, editors, *IX. Magyar Számitógépes Nyelvészeti Konferencia*, pages 62–70, 2013.

[22] Ádám Kovács and Gábor Recski. Knowledge base population using natural language inference. In *AACS*, 2018.

[23] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[24] Lixi Deng Yongdong Zhang Qi Tian Linghui Li, Sheng Tang. Image caption with global-local attention. 2017.

[25] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics, 2015.

[26] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. Xiv preprint arXiv:1309.4168, 2013.

[27] Tim O'Gorman, Michael Regan, Kira Griffitt, Ulf Hermjakob, Kevin Knight, and Martha Palmer. Amr beyond the sentence: the multi-sentence amr corpus. In *COLING*, 2018.

[28] Simon Ostermann, Ashutosh Modi, Michael Roth, Stefan Thater, and Manfred Pinkal. MCScript: A Novel Dataset for Assessing Machine Comprehension Using Script Knowledge. In *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, 2018.

[29] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The Proposition Bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106, March 2005.

[30] Gábor Recski. Building concept graphs from monolingual dictionary entries. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia, 2016. European Language Resources Association (ELRA).

[31] Gábor Recski. *Computational methods in semantics*. PhD thesis, Eötvös Loránd University, Budapest, 2016.

[32] Gábor Recski. Building concept definitions from explanatory dictionaries. *International Journal of Lexicography*, 31:274–311, 2018.

[33] Gábor Recski, Eszter Iklódi, Katalin Pajkossy, and Andras Kornai. Measuring semantic similarity of words using concept networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 193–200, Berlin, Germany, 2016. Association for Computational Linguistics.

[34] Matthew Richardson, Christopher JC Burges, and Erin Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the*

2013 Conference on Empirical Methods in Natural Language Processing, pages 193–203, 2013.

[35] Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 4444–4451, 2017.

[36] Melanie Tosik. Abstract meaning representation. `http://www.melanietosik.com/files/amr.pdf`.

[37] Bingning Wang, Shangmin Guo, Kang Liu, Shizhu He, and Jun Zhao. Employing external rich knowledge for machine comprehension. In *IJCAI*, pages 2929–2925, 2016.

[38] Hai Wang, Mohit Bansal, Kevin Gimpel, and David McAllester. Machine comprehension with syntax, frames, and semantics. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 700–706, 2015.

[39] Liang Wang, Meng Sun, Wei Zhao, Kewei Shen, and Jingming Liu. Yuanfudao at semeval-2018 task 11: Three-way attention and relational knowledge for commonsense machine comprehension. *arXiv preprint arXiv:1803.00191*, 2018.