Automatic Reference Resolution for Pedestrian Wayfinding Systems

DMYTRO KALPAKCHI
Automatic Reference Resolution for Pedestrian Wayfinding Systems

DMYTRO KALPAKCHI
Abstract

Imagine that you are in the new city and want to explore it. Trying to navigate with maps leads to the unnecessary confusion about street names and prevents you from enjoying a wonderful walk. The dialogue system that could navigate you from by means of a simple conversation using salient landmarks in your immediate vicinity would be much more helpful!

Developing such dialogue system is non-trivial and requires solving a lot of complicated tasks. One of such tasks, tackled in the present thesis, is called reference resolution (RR), i.e. resolving utterances to the underlying geographical entities, referents (if any). The utterances that have referent(s) are called referring expressions (REs).

The RR task is decomposed into two tasks: RE identification and resolution itself. Neural network models for both tasks have been designed and extensively evaluated. The model for RE identification, called RefNet, utilizes recurrent neural networks (RNNs) for handling sequential input, i.e. phrases. For each word in an utterance, RefNet outputs a label indicating whether this word is in the beginning of the RE, inside or outside it. The reference resolution model, called SpaceRefNet, uses the RefNet’s RNN layer to encode REs and the designed feature extractor to represent geographical objects. Both encodings are fed to a simple feed-forward network with a softmax prediction layer, yielding the probability of match between the RE and the geographical object.

Both introduced models have beaten the respective baselines and show promising results in general.
Sammanfattning

Tänk dig att du är i en ny stad och vill känna staden bättre. Du försöker att använda kartor, men blir förvirrad av gatunamn och kan inte njuta av din promenad. Ett dialogsystem, som kan hjälpa dig att navigera med hjälp av talade instruktioner, och som använder sig av framträdande landmärken i din närhet skulle vara mer användbart! Att utveckla ett sådant system är mycket komplicerat och man behöver att lösa ett antal mycket svåra uppgifter. En av dessa uppgifter kallas referenslösning (RR), vilket innebär att associera refererande fraser (RE) i yttranden till de geografiska objekt som avses. RR har brutits ner i två deluppgifter: identifiering av RE i yttranden, och referenslösning av dessa RE.


Båda modellerna fungerade bättre än respektive baslinjemodeller, och visar lovande resultat i allmänhet.
Анотація

Уявіть, що Ви опинилися у місті, яке ніколи не відвідували. Ви хочете побачити все, що місто може Вам запропонувати, але не знаєте нікого, хто може з цим допомогти. Назви вулиць на електронних картах не тільки не допомагають, а ще й заплутують Вас, заважаючи отримувати насолоду від чудової прогулянки. Було б набагато зручніше, якщо Ви могли б говорити з діалоговою системою, як Ви говорите з друзями. Така система допомагала б Вам орієнтуватися, використовуючи помітні орієнтири у Вашому оточенні.

Розробка такої системи включає в себе багато нетривіальних задач, одна з яких називається задача розв’язання географічних посилань (РГП). Словосполучення, вживані з метою вказати на специфічний географічний об’єкт, є досить розповсюдженими у повсякденній мові. Такі словосполучення називаються географічними посиланнями (ГП), а географічні об’єкти, на які вони посилуються — референтами. Задача розв’язання географічних посилань полягає у співставленні їх з відповідними референтами.

У рамках даної дипломної роботи задача РГП була декомпозиційно розподілена на дві частини: ідентифікація географічних посилань (ІГП) та власне розв’язання (ВРГП). Для вирішення обох задач було розроблено, протестовано та оцінено відповідні нейронні мережі. Модель для розв’язання задачі ІГП називається RefNet та використовує рекурентні нейронні мережі, щоб мати змогу обробляти послідовні вхідні дані, як-то фрази. RefNet аналізує висловлене речення дослідно та визначає, для якого слова чи вище знаходиться на початку, всередині чи поза ГП. Модель для розв’язання задачі ВРГП називається SpaceRefNet та використовує рекурентний шар RefNet для представлення поданих на вхід ГП. Географічні об’єкти представляються за допомогою розробленого алгоритму виділення ознак. Обидва представлення подаються на вхід прості нейронні мережі прямого поширення з кінцевим шаром softmax, який обчислює ймовірність того, що подане ГП описує поданий географічний об’єкт.

Обидві мережі показали гарний результат, кращий за відповідні базові моделі. Результати загалом показують, що використання нейронних мереж для вирішення задачі розв’язання географічних посилань є перспективним напрям для майбутніх досліджень.
Acknowledgements

First of all, I want to thank the Swedish Institute (SI) for giving me such an amazing opportunity to study in Sweden and at KTH Royal Institute of Technology in particular. This thesis marks the ending of my two year studies at KTH, thanks to the SI Visby scholarship.

Secondly, I want to express my gratitude to my supervisor, professor Johan Boye, for endless interesting, productive discussions and numerous proofreadings of this whitepaper. Being able to discuss whatever has come to my mind at any time has been extremely useful and made the process of writing this thesis much more exciting and rewarding – thank you for that!

Thirdly, I want to thank my friends for all words of support which helped me to take all arising challenges with a smile in my face and complete all other tasks I had along with this master thesis.

Last, but not least, I would like to say a huge “thank you” to my parents, who has been supporting me during my entire life and have given me much more than I will probably be ever able to realize.
# Contents

1 Introduction  
1.1 Research topic .......................... 2  
1.2 Scope .................................. 3  
1.3 Contributions .......................... 4  
1.4 Sustainability and ethics .................. 4  
1.5 Societal aspects ........................ 4  

2 Background and Related Work  
2.1 Pedestrian wayfinding systems .......... 5  
2.2 Resolving spatial references .............. 8  
  2.2.1 Referring expressions identification .... 8  
  2.2.2 Constructing referents search space .... 13  
  2.2.3 Reference resolution .................. 14  
2.3 Neural networks and their selected NLP applications .. 15  
  2.3.1 Feed-forward neural networks .......... 15  
  2.3.2 Autoencoders ........................ 17  
  2.3.3 Recurrent neural networks ............ 18  
  2.3.4 Loss functions ....................... 20  
  2.3.5 Optimizers .......................... 21  
  2.3.6 Overfitting and underfitting .......... 23  
  2.3.7 Hyperparameter optimization .......... 24  
  2.3.8 Word vectors ....................... 26  
2.4 Evaluation ............................. 28  

3 Methods  
3.1 Gathering data .......................... 29  
  3.1.1 SpaceRef .............................. 29  
  3.1.2 Data labeling strategy ............... 30  
  3.1.3 Collecting new data .................. 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Approaching reference resolution</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3.2.1 Referring expressions identification</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3.2.2 Constructing referents search space</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>3.2.3 Reference resolution</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Baselines</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>3.3.1 Referring expressions identification</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>3.3.2 Reference resolution</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Experimental Results</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>Referring expressions identification</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Reference resolution</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>Test set performance</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>Discussion and Future Work</td>
<td>67</td>
</tr>
<tr>
<td>5.1</td>
<td>Was the amount of data sufficient?</td>
<td>67</td>
</tr>
<tr>
<td>5.2</td>
<td>Were the encodings good enough?</td>
<td>68</td>
</tr>
<tr>
<td>5.3</td>
<td>Was the loss chosen correctly for SpaceRefNet?</td>
<td>68</td>
</tr>
<tr>
<td>5.4</td>
<td>Closing remarks</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>70</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Imagine yourself arriving to a city you have never been to before. You are about to spend a week there participating in an exciting conference, but you would also like to get acquainted with the host city. You want to see some sights, eat delicious food and have a pleasant walk surrounded by wonderful nature. However it turns out to be quite complicated, as you neither are familiar with the surroundings nor speak the local language nor know anybody who can guide you through the city. The only option left is to use a smartphone application with the city map and guide yourself on your own. Unfortunately, you are not that comfortable with using maps and even if you were, you would not prefer it, since you can not enjoy the walk then. What you are looking for is another application, which you can talk to as you would talk to a human guide. This app would suggest some interesting walking routes, guide you to the nearest local food restaurant and answer your questions while you are walking.

Such application requires numerous components to be linked together to make your interaction through a spoken dialogue as natural and flawless as possible. One of these components should solve the problem of guiding you based on the information you have just said. The component should identify where you are, what you can see, where you want to go and how to guide you there. At first glance, the task might sound fairly simple, since Global Positioning System (GPS) can identify where the pedestrian is and how to get to the destination point. This approach works quite well for cars, but would be uncomfortable and somewhat stressful for pedestrians.

Automotive navigation systems (ANS) in cars generally use a for-
mal language of simple actions, distances and street names. For instance, a phrase like “Drive 125 meters straight, then turn right at Odengatan” would be a typical example of ANS instructions. Most of today’s freely available pedestrian wayfinding systems, such as Google Maps or GraphHopper, use the same strategies as ANS (Götze, 2016). Humans, on the contrary to cars, can not determine accurately when they have walked 125 meters. Furthermore, in the complex unknown city environment local street names, e.g. “Odengatan”, are hard to find without confusing with something similar, like “Odenplan”. In real life ANS-like language is never utilized by humans for guiding other people. Instead, the most salient landmarks in the vicinity are used to specify the next point in the route. This approach is way more natural and can often be heard in daily conversations, but it also raises an important and non-trivial problem. When a person mentions a landmark, how would the system match a description of the landmark with the actual geographical object on a map? This question leads us to the problem of reference resolution, which is attacked in the current thesis.

1.1 Research topic

When people navigate other humans or describe their surroundings in a city, they use phrases such as “on the other side of the street”, “towards the building”, “near the church”, etc. These phrases contain references to entities in the city: the street, the building, the church, and so on. Using a geographic database, such as OpenStreetMap, along with information about the pedestrian’s location and direction of movement, it is possible to figure out which entities the person is referring to (which the targets are). This is a non-trivial task, since in general there are more than one “candidate” entity of the right type visible to the pedestrian. Another difficulty is that entities in OpenStreetMap do not always correspond to a person’s perception of entities; e.g. a person might perceive a street as one entity, whereas OpenStreetMap represents it as several entities; a person might perceive many buildings next to each other as separate entities, whereas OpenStreetMap represents the whole block as one entity.

The topic of this master’s thesis is to devise a method that automatically can do the reference resolution outlined above. More precisely, given a sentence, containing one or more referring expressions, along
with information about the pedestrian’s position and direction, the algorithm should:

- find referring expressions in a sentence, and
- detect the “candidate set” of visible entities around the pedestrian, and
- return a correct candidate or a ranked list of the candidates, where the order in the list reflects the probability of being the target of the referring expression.

In general, there might be more than one target, so preferably the algorithm should output the most likely set of targets (where the set can contain 0, 1 or more entities).

Ultimately, the algorithm is deemed successful if it works well (according to a well-defined metric) when used in an interactive wayfinding system, as the one outlined above. However, implementing and evaluating a full interactive wayfinding system is beyond the scope of this thesis. We will therefore evaluate the reference resolution method in isolation, comparing the performance to that of baseline methods. All methods will be evaluated on a test set of referring expressions and associated user positions, using standard evaluation metrics such as precision, recall, and accuracy.

1.2 Scope

This thesis aims at developing further the work on reference resolution presented by Götze and Boye (2017), where logistic regression was used and gave promising, but not satisfactory results. The possibilities of going beyond one-layer feed-forward neural networks, used in logistic regression, are explored in the current work. In particular, recurrent neural networks are exploited, since they have proven to be successful in NLP applications (Mikolov, Karafiát, Burget, Černocký, & Khudanpur, 2010), (Sutskever, Martens, & Hinton, 2011), (Sutskever, Vinyals, & Le, 2014), (Graves, Mohamed, & Hinton, 2013), (Karpathy & Fei-Fei, 2015).
1.3 Contributions

The contributions of the present work are:

- an algorithm for finding referring expressions has been designed and extensively evaluated;
- an algorithm for resolving referring expressions has been designed and extensively evaluated;
- an Android application for collecting experimental data for extending SpaceRef dataset has been developed.

1.4 Sustainability and ethics

The present work focuses on the design and development of novel algorithms for reference resolution. The solution of this problem does not directly contribute to sustainable development. Nonetheless, accurate reference resolution would lead to successful pedestrian wayfinding dialogue system which, in turn, would result in the cities that are smarter, easier to navigate and more attractive for walking.

The thesis does not address or solve ethical issues directly. The data, used for the present work, do not contain any personal data, except the names of subjects, which are not used in any way for training of reference resolution models. All participants of the experiment have signed a written consent form (Götze, 2016, Appendix A).

1.5 Societal aspects

The present work is relevant for the society in multiple aspects. First, pedestrian wayfinding dialogue systems, capable of accurate reference resolution would lead to smarter cities. Secondly, travel companies can benefit from such dialogue systems, for instance, by creating more interactive tourist routes, where the users can ask about any landmark in the vicinity and the system can give a brief information about the landmark. Finally, a wayfinding dialogue system can be adapted to help people with impaired spatial perception and cognition to navigate in the complex city environment.
Chapter 2

Background and Related Work

The chapter begins with a presentation of the concept of pedestrian wayfinding systems in section 2.1, followed by the necessary background and prior works on reference resolution in section 2.2. Then section 2.3 gives a brief description of neural networks and their selected applications in natural language processing, relevant to this thesis. The final section of the chapter describes evaluation methods used within the present work.

2.1 Pedestrian wayfinding systems

Wayfinding, according to Montello (2005), is defined as one of navigation components, along with locomotion. Following the paper, wayfinding is the goal-directed planned efficient movement around the environment. Locomotion, on the other hand, is defined as the physical body movement performed by a pedestrian to coordinate himself in the local or proximal surroundings. The simpler definition of wayfinding is given by Passini (1981), where it is defined in terms of spatial problem solving as the process of reaching the spatial destination in novel or familiar settings.

A system helping humans to wayfind is called a wayfinding system. Over the decades humans have used different kinds of such systems: fingerposts, freestanding panels, monoliths, signage, interactive information points etc. Most of them are located in some fixed places, e.g. railway stations, and contain some map along with a limited amount of information about the closest places of interest. Modern wayfinding systems (and those of interest within the current thesis) are automatic
ones, e.g. smartphone app, which are always with humans and help them to wayfind whenever it is needed. Formally an automatic pedestrian wayfinding system is a wayfinding system that uses route planning and decision-making algorithms.

Automatic pedestrian wayfinding systems are not as popular as car navigation systems, but a number of navigation libraries (see Table 2.1) and mobile applications (see Table 2.2), supporting pedestrian wayfinding, are available. These libraries and applications rely heavily on GPS and employ the same strategy as ANS. The approach is called turn-by-turn navigation, i.e. giving directions as the vehicle, or pedestrian in our case, approaches a navigation point in the form of visual and/or spoken instructions (see example in the figure 2.1).

![Figure 2.1: An example of turn-by-turn navigation feature in Google Maps Android application](image)

As shown in Tables 2.1 and 2.2, every surveyed wayfinding library and app offers the turn-by-turn navigation feature.

<table>
<thead>
<tr>
<th>Library</th>
<th>Turn-by-turn</th>
<th>Landmark-by-landmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapbox Navigation</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GraphHopper</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OSRM</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 2.2: Comparison of mobile pedestrian wayfinding systems

<table>
<thead>
<tr>
<th>Application</th>
<th>Turn-by-turn</th>
<th>Landmarks</th>
<th>Voice input</th>
<th>Dialogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Maps</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MAPS.ME</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sidekix</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Walc</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

However, turn-by-turn navigation does not resemble the way humans give wayfinding instructions to each other. An urban planner K. Lynch conducted an interesting study (Lynch, 1960) on what he called imageability, i.e. the ability of a person to form and reconstruct a solid mental image of the spatial environment. Lynch interviewed the residents of Boston, Jersey City and Los Angeles asking them to sketch a map of some city area from the memory. Given these sketches and verbal interviews, the cities’ imageability was compared in order to find out which physical forms are the most memorable. Interestingly, all city images had five types of spatial elements in common: paths, edges, districts, nodes and landmarks. When people guide each other through the city, they usually use its mental image which relies on the most memorable spatial features. Indeed, people use districts to describe the global area of interest, paths and edges to describe a way leading to a destination, nodes as the points of decision making and landmarks to avoid getting lost. According to Denis (1997), humans give central importance to landmarks located along the route while formulating their route description.

Despite everything mentioned above, only one of the applications in table 2.2 uses landmarks for giving wayfinding instructions. Furthermore, none of them is capable of maintaining a dialogue, facilitating wayfinding, which is the primary way of human-to-human navigation. Hence, interactive wayfinding system would clearly benefit from the incorporated dialogue component. Instead of turn-by-turn navigation, it should employ landmark-by-landmark one, which requires a natural language understanding component, capable of matching an utterance mentioning landmark, with the correct geographical object. This matching task is called the reference resolution problem and is addressed in the next section.
2.2 Resolving spatial references

Using landmarks during wayfinding presupposes that humans simply refer to them while giving instructions to others or talking to a guide. A typical wayfinding instruction, involving landmarks as points of reference, would be like the following:

There you’ll notice an old church on top of the hill. A little bit to the right you’ll see a local grocery store, ICA Nära. Move down that road with ICA until you reach a big brick bridge with some sculptures on top of it.

This short instruction contains a lot of phrases that refer to easily noticeable geographical objects, for instance, “an old church on top of the hill”, “a local grocery store, ICA Nära”, “a big brick bridge with some sculptures on top of it” etc. Such kind of phrases are called referring expressions (RE) and the objects which these phrases refer to are called referents. In our case the referents are geographical objects, hence the respective referring expressions are said to be spatial. The task of resolving referring expressions to a referent is called reference resolution (RR) (Kennington & Schlangen, 2015). Formally, following Kennington, Dia, and Schlangen (2015), this can be defined as the function $f_{RR}$ of the two arguments $U$ (the representation of the RE) and $W$ (the representation of the relevant world aspects), i.e.:

$$I^* = f_{RR}(U, W),$$

where $I^*$ is the identifier of the intended referent. Note that there might be more than one target referent, which complicates the RR automation process.

For humans reference resolution is a daily task, that is quite simple in familiar environments, yet challenging in novel settings. Automating RR in a complex city environment is a non-trivial problem requiring decomposition into three smaller subproblems, which are discussed in the following subsections.

2.2.1 Referring expressions identification

Before commencing the resolution of referring expressions, the system has to identify them first in recognized spoken utterances. To succeed
in this task, the system should know what constitutes a referring expression and how to identify such linguistic units.

Let us have a look at two openings of wayfinding dialogues containing referring expressions.

Dialogue 1
*User*: How can I reach the closest sport center?
*System*: You should go to the subway, which will appear soon if you continue straight. Please take a red line train towards Liljeholmen. It will take 5 minutes from there, I’ll give you further instructions when you arrive.

Dialogue 2
*User*: Where can I have a snack?
*System*: What kind of snack?
*User*: Something like a sandwich.
*System*: You can go to Subway, which is close to your location. Do you want me to guide you there?

Both excerpts contain a referring expression “subway” (capitalized in Dialogue 2, but this information might not be preserved after speech recognition, so we ignore it for the purpose of this example). Despite the fact that RE is the same for both cases, the underlying referent is clearly different – a subway station for Dialogue 1 and a fast food restaurant for Dialogue 2. Another example of a RE is “there” in Dialogue 1 referring to Liljeholmen subway station. The only way to identify and resolve these REs correctly is by considering the context.

Each dialogue is a coherent structured group of sentences, which, in general, is called discourse (Martin & Jurafsky, 2009). The referring expressions, like “there”, that refer to something previously mentioned in the discourse, are called anaphoric. The REs, like “subway”, that has not been mentioned previously and refer to something outside of a linguistic context of the discourse are called exophoric. Spatial REs are usually referring to objects in the geographical context of the discourse and thus are often exophoric.

From a grammatical perspective, Martin and Jurafsky (2009) distinguish five basic types of referring expressions. We describe their taxonomy below and provide both referential and non-referential example usages in the geographical context.
1. **Indefinite Noun Phrases** that introduce new objects into the discourse. Typically such REs start with the determiner *a (an)*, less often with the determiners *this, that* or quantifier *some*, e.g. (a), (b) or (c). Note, however, that in (d) “a subway station” is non-referential, since the person is asking about *any* subway station without introducing any new objects into the discourse (we do not even know if any subway station is present in the vicinity).

(a) You can see a big red building near the church.
(b) I am walking down some stairs now.
(c) The bank is on this bigger street, called Valhallavägen.
(d) Do you know if there is a subway station nearby?

2. **Definite Noun Phrases** that refer to unambiguously identifiable entities, which were either mentioned previously in the discourse, e.g. (a), or are unique in the geographical context, e.g. (b). However, in (c) “the location” does not refer to any specific entity, but serves as an identifier for a potential picnic place.

(a) There is a big brick university building near the subway station. The building is quite old and thus is one of the major sights in the region.
(b) I’d like to visit the city hall during the week.
(c) Do you have any place for a picnic in mind? It would be nice if the location is close to our vicinity.

3. **Pronouns** referring to objects with high degree of salience in the discourse, usually mentioned no further than one-two sentences back. Pronouns can also be part of *cataphora*, i.e. used before the actual RE (see example (b)). Note that *it* in (c) is not an RE, but a *pleonasm*. Other non-referential uses of pronouns include clefts and extraposition.

(a) The event will be hosted at Berwaldhallen. Continue straight on Karlavägen and then turn right to reach it.
(b) Aha, now I see it, a tall glass building near Pressbyrån.
(c) It is raining today.
4. **Demonstratives** referring to previously mentioned objects using demonstrative pronouns, like *this* and *that*, e.g. (a). Note, however, that the demonstrative “this” in (b) is clearly non-referential.

(a) I’ve just seen another embassy nearby. This building is much older than the previous I’ve seen.

(b) I see an old building with colonnades. This architectural style I like the most.

5. **Proper Names** used for referring to some named entities, i.e. specific organizations or locations, e.g. (a). Apparently, in the example (b) “Carl Linnaeus” does not refer to a geographical entity.

(a) Google Stockholm is just 5 minutes from T-Centralen.

(b) The statue in front of the library portrays Carl Linnaeus.

Sometimes additional kinds of referring expressions are of particular interest. For example, Salmon-Alt and Romary (2009) have taken into account three additional grammatical types of REs:

- **one-anaphora** (e.g., “Among three buildings, the left-most one is the tallest”);

- **other-expressions** (e.g., “On the other side of the street there is a bus stop”);

- **ordinals** (“The first sculpture, located in the center, depicts the ancient Scandinavian god Odin”).

The present work focuses on the exophoric spatial REs, because there exists a clear correspondence between these and a fixed set of geographical referents. Anaphoric REs, on the other hand, can not be matched with any geographical object before they are resolved to their antecedents, i.e. preceding exophoric REs that gave meaning to them. In fact, there have been a great interest in anaphoric REs, since they are extremely widespread in written texts, which resulted in a separate field of *anaphora resolution* (AR), i.e. identifying the antecedent that “permitted” usage of the anaphoric RE. Previous works on AR are briefly surveyed in the next few paragraphs.

Over the years many researchers have focused on resolving pronouns to their noun phrase (NP) antecedents, i.e. *pronominal anaphora resolution* (PAR). The baseline method in this field, originally presented
by Hobbs (1978), is the Hobbs naive algorithm, which operates by traversing the syntactic parse tree of the sentences up to and including the one under consideration in the particular order. Each candidate NP, found during traversal, is checked for morphological gender and number. If those agree with the pronoun, NP is proposed as an antecedent. The same paper also presents more complex semantic approach.

Mitkov (1998) presents a knowledge-poor approach to PAR utilizing part-of-speech (POS) tagger on the 2 sentences preceding the anaphor for identifying all NPs using a set of simple rules. As in the Hobbs algorithm, gender and number agreement with the anaphor are checked and then genre-specific antecedent indicators are applied to the survived candidates.

Machine learning (ML) approaches have been applied to another type of anaphora resolution, called coreference resolution, i.e. identifying REs that refer to the same entity (corefer). A set of coreferring expressions is said to be a coreference chain. One ML approach is described in (Soon, Ng, & Lim, 2001), where C5 decision tree coreference classifier, representing each RE as a set of 12 handcrafted features, is trained and used for clustering all REs into coreference chains.

Ng and Cardie (2002) improve results of Soon et al. (2001) by modifying and expanding the RE feature set from 12 to 53 features, changing clustering algorithm and the training set generation approach.

Iida, Inui, and Matsumoto (2005) present another ML approach by reversing the steps of Soon et al. (2001), i.e. first selecting the candidate antecedent and then classifying the pair of anaphor and antecedent as either anaphoric or non-anaphoric. Such approach, introduced by them as selection-then-classification, requires two models: antecedent identification model (tournament model, based on (Iida, Inui, Takamura, & Matsumoto, 2003)) and anaphoricity classification model (support vector machines classifier, introduced in (Cortes & Vapnik, 1995)).

Other approaches to anaphora resolution include, but not limited to, those presented in (Brennan, Friedman, & Pollard, 1987), (Lappin & Leass, 1994) and (Walker, 1998).

In contrast, exophoric REs have received much less attention. Most of the studies on exophoric reference resolution were conducted in an unrealistically small toy domain (to be addressed in the subsection 2.2.2), hence REs can be identified manually, as in (Engonopoulos, Vil-
Götz and Boye (2017) addressed RR in the real city domain, but all REs were annotated manually as well.

Kennington and Schlangen (2015) used Wizard-of-Oz setting, in which the participants were required to refer to specific objects chosen beforehand, thus the RE identification was trivial.

Schutte, Kelleher, and Mac Namee (2010) worked on resolving REs in simple manipulation instructions, e.g., “hit that red button”, and identified REs using a set of simple regular expressions.

Prasov and Chai (2010) used syntactic parsing on word confusion network, constructed out of n-best list of alternative speech recognition hypotheses. All non-pronominal NPs were then detected and said to be a set of exophoric REs.

### 2.2.2 Constructing referents search space

After identifying REs, a set of possible referents, later referred to as referents search space, needs to be constructed. In this subsection we briefly survey the ways, utilized in previous works for constructing referents search space and representation of objects in this space.

As have been mentioned in the previous subsection, most of the research on exophoric reference resolution have been conducted in unrealistic environments. For example, in (Kennington & Schlangen, 2015) the experiments were conducted in the toy Pentomino domain. Participants’ task was to recognize one of 36 Pentomino puzzle pieces placed on the table from the spoken instructions. All puzzle pieces were distinct by shape, size and color. To extract the representation of each object, they segmented images, obtained from the camera recording a video feed of objects, and then extracted a set of features as described in (Kennington et al., 2015).

Matuszek, Bo, Zettlemoyer, and Fox (2014) describe the experiment on human-robot interaction, in which a robot should grasp one object of the small finite set of objects based on the spoken instructions given by a human. Each object is represented by combining features obtained from gesture and vision classifiers.

Funakoshi et al. (2012) explore RR for the task-oriented dialogue aimed at solving the Tangram puzzle (a set of 7 puzzles, different by size and shape, should be combined to obtain the goal shape). The au-
thors also employ the concept of reference domains (RD), originally presented in (Salmon-Alt & Romary, 2000). Funakoshi et al. (2012) defines RDs from the applied perspective as a set of entities (or other RDs) referred to as a whole. In the Tangram experiments RDs are formed by proximity of tangrams and introduced to resolve REs referring to sets of objects. The tangrams themselves are represented by numeric identifiers, since the search space is extremely small.

Engonopoulos et al. (2013) performs RR in a treasure-hunt game, i.e. a more sophisticated virtual 3D environment, within the GIVE challenge (Koller et al., 2010). The objects are represented with color, distance and salience features, which are used to induce a conditional probability distribution over all objects given the uttered phrase and state of the world.

To the best of our knowledge, the only work dealing with exophoric REs in complex real city environment is the one by Götze and Boye (2017). To represent geographical objects they utilize OpenStreetMap (OSM), introduced in (Haklay & Weber, 2008). The GPS position of pedestrian, walking the route, is constantly tracked and a set of possible referents, called a candidate set, includes all objects in pedestrian’s immediate vicinity. Each object is represented by a set of 427 binary features, derived from the OSM annotations following Götze and Boye (2016a), 2 features describing geographical position and 8 features containing context information, i.e. whether an object has been mentioned before, how long ago was the last mention etc. The present thesis follows this approach of Götze and Boye (2017) to a large extent.

### 2.2.3 Reference resolution

As discussed in the previous subsection, reference resolution (RR) is typically performed in an unrealistic environments where objects are quite distinct and described by relatively small number of features. In such scenarios, RR is typically solved by taking the stochastic approach and using a generative probabilistic model to estimate the distribution over a set of candidate objects and then find the most probable intended referent as:

$$I^* = \arg \max_I P(I|U, W)$$  \hspace{1cm} (2.2)

Such stochastic approach is pursued, for instance, in (Kennington & Schlangen, 2015), (Engonopoulos et al., 2013), (Matuszek et al., 2014)
Pedestrian wayfinding systems require RR to be performed in a complex real city environment with a bigger spatial area to consider, greater variety of objects and stronger similarity between them. To the best of our knowledge RR of such scale and with such properties of referents is addressed only by Götze and Boye (2017) where they utilized a probabilistic approach as well and attacked the problem using logistic regression. However their approach to RR has a weakness: the treatment of multiple targets is ad-hoc.

2.3 Neural networks and their selected NLP applications

The present work explores the possibility of extending the work of Götze and Boye (2017) beyond logistic regression, i.e. single-layer feed-forward network. More sophisticated types of neural networks (NN) which have proven to work quite well for natural language processing (NLP) are of particular interest. In the following subsections we briefly describe the NN types used within the current thesis, followed by a short survey of relevant neural NLP applications.

In general, neural networks try to find the mapping $f$ from a set of data points $x = \{x_1, ..., x_N\}$, each of which is represented as $D$-dimensional feature vector, to their respective labels $y = \{y_1, ..., y_N\}$. This is the default problem statement for this section, unless specified otherwise.

2.3.1 Feed-forward neural networks

A neural network (NN) consists of simple computational units, called neurons, and directed weighted connections between them. Kriesel (2007) formally defines neural network as follows.

**Definition 2.3.1.** A neural network is a sorted triple $\langle N, V, w \rangle$, where $N$ is the set of neurons, $V = \{(i, j) | (i, j) \in N\}$ is the set of connections and $w : V \rightarrow \mathbb{R}$ is a weight function.

Each neuron processes the data by applying the following three functions in chain:
• **Propagation function** transforms outputs of other neurons to an input of the current neuron. Often, this function is a simple weighted sum.

• **Activation function** transforms the neuron input (and for some topologies previous activation) to new activation. Common activation functions are logistic sigmoid, hyperbolic tangent and rectified linear unit (ReLU).

• **Output function** calculates the values that are fed to other neurons. Typically, this function is the identity.

The neurons are grouped together in *layers*. Various ways of within-layer and inter-layer connections result in different *topologies*. The simplest topology is *feed-forward*. The first such network was perceptron, originally presented in (Rosenblatt, 1957). The class of NNs with such topology inherited its name and is referred to as *feed-forward neural networks* (FNN). Typically FNNs consist of at least three layers: *input layer* that gets the network input, *hidden layers* (at least one is required) that process the obtained input using *hidden neurons* or *hidden units* and *output layer* that produces the final result. In FNNs neurons within the layer are disconnected and can be connected only to the next layer, i.e. jumps to further layers are prohibited. Formally, feed-forward neural network can be defined as:

\[
x^{(l)} = a((w^{(l)})^T x^{(l-1)}), \forall l \in \{1, 2, \ldots, L\},
\]

where \( L \) is the total number of layers, except the input layer (denoted with index 0), \( a \) is the activation function, \( x^{(0)} \) is the network input.

The output layer \( x^{(L)} \) aims at approximating the scalar value \( y_i \) for the data point \( x_i \). Hence, the final layer contains only one neuron and the learned approximation for data point \( x_i \) is denoted as \( \hat{y}_i = x_i^{(L)} \).

For assessing how good the approximation is and to improve the network’s performance, the loss function, penalizing the differences between \( \hat{y}_i \) and \( y_i \) for each data point, is defined. Feed-forward networks are usually trained using a mean squared error loss function (to be addressed in subsection 2.3.4). Training means optimizing (minimizing) the loss function using some, usually gradient-based, optimization techniques (to be discussed in subsection 2.3.5).
2.3.2 Autoencoders

Autoencoder is a special case of feed-forward networks, which is trained to reproduce its input as an output with the aim of either compressing or denoising the data (Goodfellow, Bengio, Courville, & Bengio, 2016, Chapter 14). An autoencoder architecture typically consists of input and output layers, which have the same number of neurons $n_x$, as well as the hidden layer with a different number of neurons $n_h, n_h \neq n_x$ (see figure 2.2).

![Figure 2.2: The typical autoencoder architecture](image)

Another way is to view an autoencoder as a composition of two functions: the encoder $f$, governed by the weights of connections from the input to the hidden layer (blue area in the figure 2.2) and the decoder $g$, governed by the weights of connections from the hidden to the output layer (green area in the figure 2.2).

The type of autoencoders, that is of particular interest in the present work, is called undercomplete autoencoders (Bourlard & Kamp, 1988). Such networks follow the architecture, presented in the figure 2.2, but the hidden layer have a smaller number of neurons than the input layer, i.e. $n_h < n_x$. Hence, the rationale behind using undercomplete autoencoders is compressing the data.

Autoencoders are trained as regular feed-forward networks using gradient-based optimization methods (to be discussed in the subsection 2.3.5) and optimizing a mean squared error loss function (to be addressed in the subsection 2.3.4).
2.3.3 Recurrent neural networks

Feed-forward neural networks assume that the input data points are independent of each other. However, in natural language processing, we often have to deal with sequential input, i.e. sentences, where the data points, i.e. words, are dependent on the previous data. Such case is handled by using a network with recurrent topology, which is referred to as simply recurrent neural network (RNN).

RNNs introduce the notion of time, meaning that at each time step one element of a sequence is processed. In classical RNNs there is one hidden layer, whose output is denoted as $h_t \in \mathbb{R}^{n_h}$. Notably it depends not only on the current input $x_t \in \mathbb{R}^{n_x}$, but also on the hidden layer output at the previous time step $h_{t-1}$, hence the name recurrent. Formally $h_t$ is computed as follows:

$$h_t = g(W_h h_{t-1} + W_x x_t) = g(W[[h_{t-1}, x_t]])$$

(2.4)

where the matrix $W = [W_h, W_x]$ denotes horizontally stacked matrices $n_h \times n_h$-dimensional $W_h$ and $n_h \times n_x$-dimensional $W_x$; $[[h_{t-1}, x_t]]$ denotes vertically stacked vectors $h_{t-1}$ and $x_t$. These definitions are introduced to simplify the notations in future subsections.

The network’s output at a time step $t$ is then computed as:

$$\hat{y}_t = g'(W' h_t)$$

(2.5)

where $g'$ is the activation function for the output layer and $W'$ are the propagation weights for the output layer.

Note that the weights $W_h$ and $W_x$ are shared between all time steps which enables RNN to use the knowledge from previous time steps while predicting a label for the current one.

However, as shown by Bengio, Simard, and Frasconi (1994), it is difficult to learn long-term dependencies with plain recurrent neural networks using gradient-based optimization, because gradients tend either to vanish or explode (more rare). This means that the further in time we are from the current, the (exponentially) smaller or bigger gradients it has, respectively. One approach to solving this problem is to introduce activation functions, more complex, than those in the equation 2.4. Two such activation functions, also called recurrent units, which are popular and well-performing in practice, are called long short-term memory units (LSTM) and gated recurrent units (GRU).
Long short-term memory

LSTM was originally presented in (Hochreiter & Schmidhuber, 1997) to enable handling of long-term dependencies in a sequential input. The key idea lies in introducing two cells at each time step $t$:

- the memory cell $C_t$, that keeps all information from the previous time steps;
- the new memory $\tilde{C}_t$, that keeps the information learned from only the incoming input element $x_t$.

Computationally the new memory depends on the previous hidden state $h_{t-1}$ as follows:

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t])$$

(2.6)

The new memory is then incorporated into the memory cell by forgetting something of the past memories using the forget gate $f_t$ and keeping only relevant information from the new memory using the input gate $i_t$. Formally, the computations proceed as follows:

$$f_t = \sigma(W_f[h_{t-1}, x_t]),$$

(2.7)

$$i_t = \sigma(W_i[h_{t-1}, x_t]),$$

(2.8)

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t,$$

(2.9)

where $\sigma$ is the logistic sigmoid function, $\circ$ means Hadamard product, i.e. element-wise multiplication.

The output value of a hidden state $h_t$ is a partially exposed memory cell, where the amount of exposure is controlled by the output gate $o_t$.

$$o_t = \sigma(W_o[h_{t-1}, x_t])$$

(2.10)

$$h_t = o_t \tanh(C_t)$$

(2.11)

Gated recurrent units

GRU was introduced in the work of Cho et al. (2014). It is motivated by LSTM to large extent, but is computationally cheaper and easier to implement. Instead of 4 gates, GRU units have only 2 gates and the memory cell, used in LSTMs is absent, i.e. $C_t = h_t$. The new information is encoded in the candidate hidden state $\tilde{h}_t$, which computationally depends on previous hidden state $h_{t-1}$ and current input $x_t$. Any
irrelevant information in the previous hidden state can be dropped using the reset gate \( r_t \). Computations proceed as follows:

\[
\begin{align*}
  r_t &= \sigma(W_r[[h_{t-1}, x_t]]) \\
  \tilde{h}_t &= \tanh(W_h[[r_t \circ h_{t-1}, x_t]])
\end{align*}
\]

The new information is then incorporated into the hidden state in LSTM-like fashion, but with only one update gate controlling the amount of new information to be kept. Formally, it can be put as follows:

\[
\begin{align*}
  u_t &= \sigma(W_u[[h_{t-1}, x_t]]) \\
  h_t &= (1 - u_t) \circ h_{t-1} + u_t \circ \tilde{h}_t
\end{align*}
\]

Note that the hidden state is computed as a convex combination of previous hidden state and a candidate hidden state, meaning that the more new information is included, the less old information will be kept.

**Bidirectional recurrent neural networks**

In some cases not only preceding words are of interest, but also the succeeding ones. For instance, consider the following two sentences:

- Major improvement in the air quality was detected over the past 3 months.
- Major Jackson was the brave soldier.

Let us say you want to identify the expressions mentioning people in these sentences. It is not clear whether “major” is a part of such expression in these sentences, until one looks at the next word at each case. In order to take into account the succeeding words in the sequences, two RNNs are used: one RNN for processing the words from beginning to end, and another RNN, processing the words in the opposite direction from last to first. Such architecture is called a bidirectional recurrent neural network (BiRNN) and is trained using back-propagation through time (as standard RNNs). Interested readers can refer to the work of Schuster and Paliwal (1997).

### 2.3.4 Loss functions

Let us say we have a dataset \( D = \{ (x_i, y_i) \}_{i \in [1; N]} \). The task is then to find such a model \( f_\theta \) that \( f_\theta(x_i) \) approximates \( y_i \) as well as possible.
order to measure the quality of such approximation we define a point-wise loss function $J_i(\theta)$, that provides us with a quantitative way of calculating the “cost” of approximation for each data point. A total loss function, sometimes called a cost function, is then a sum of point-wise losses, i.e.

$$J = J(\theta) = \sum_{i=1}^{N} J_i(\theta)$$  \hspace{1cm} (2.16)

Below we present the types of loss functions that will be further used in the current thesis.

**Mean squared error**

$$J = \frac{1}{N} \sum_{i=1}^{N} (y_i - p_i)^2,$$  \hspace{1cm} (2.17)

where $y_i$ is a true label for the data point $x_i$ and $p_i$ is the predicted value for the same data point.

**Cross-entropy loss**

$$J = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p_i),$$  \hspace{1cm} (2.18)

where $y_i$ is a one-hot encoded true label and $p_i$ is the predicted probability distribution over labels for the data point $x_i$.

### 2.3.5 Optimizers

All optimizers surveyed in this subsection are gradient-based, meaning that they are successors of the iterative optimization method called gradient descent (GD). To put it formally, let us say our neural network has the vector of parameters $\theta$, then at each iteration $i$, the new vector of parameters is learnt using the following update rule:

$$\theta^{(i)} = \theta^{(i-1)} - \eta \frac{\partial J(\theta)}{\partial \theta},$$

where $\eta$ is called learning rate, $J(\theta)$ is a loss function.

Depending on the length of optimization iteration, there exist different variations of GD. If one parameter update is done by looping over all training examples, then the networks is said to be trained with
a batch gradient descent (BGD). If a parameter update is performed after looping over a randomly sampled subset of training examples instead, then a mini-batch gradient descent (MGD) is used. If every data point leads to updating the parameters, then such method is called a stochastic gradient descent (SGD).

Back propagation

Back propagation (BackProp) is an efficient algorithm for computing gradients, invented in 1970s, but popularized for neural networks due to the work of Rumelhart, Hinton, and Williams (1986). BackProp enables fast gradient descent optimization by relying on the fact that a neural network can be represented as a composition of functions. Hence, the gradient computations are simplified using the chain rule of derivatives. The proper derivation of the algorithm is out of scope of this thesis. If a reader wants to dive into more details, the good starting point might be (Rojas, 2013, Chapter 7).

Back propagation through time

Back propagation through time (BPTT) is the version of BackProp for training recurrent neural networks. BPTT starts by unfolding the RNN in time and then applies the regular BackProp taking into account that the weights for hidden layer are shared between all time steps. For more details refer to the work of Werbos (1990).

AdaGrad

AdaGrad was originally presented in (Duchi, Hazan, & Singer, 2011). The authors took inspiration from the famous information retrieval technique TF-IDF, i.e. the idea that rare features are more informative. During training process AdaGrad adapts learning rate in a way that frequently occurring features are updated with low learning rates, while infrequent features “need to be noted” and thus are given high learning rates. This approach works especially well with sparse data where a small set of parameters carries useful information. Formally, for each parameter, the algorithm calculates the outer product gradient matrix $G$ such that

$$ G = \sum_{i=1}^{N} g_i g_i^T, $$
where $g_i$ is gradient in iteration $i$ and $N$ is number of all past iterations. Then instead of using a global learning rate $\eta$ we use

$$\eta^* = \eta \cdot \text{diag}(G)^{-\frac{1}{2}},$$

where $\text{diag}(G)$ is the diagonal matrix obtained from $G$ matrix.

One problem with AdaGrad is that the $G$ will just increase over time, meaning that the learning rate will decrease, until it becomes infinitesimally small, i.e. the learning stops. Other optimizers were designed to overcome this limitation and Adam is one of them.

**Adam**

Adaptive moment estimation method, named *Adam*, was introduced in (Kingma & Ba, 2014) as an improvement of AdaGrad. The algorithm keeps exponentially decaying gradients and their variance similar to the momentum trick for GD. Then it uses decayed gradients and variances to adjust learning rate for each parameter. Formally, given parameters $\theta$ and the loss function $J(\theta)$, the update rule at iteration $t$ looks as follows:

$$g_t = \nabla \theta J(\theta_{t-1}),$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t,$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2,$$

$$M_t = \frac{m_t}{1 - \beta_1^t}, \quad V_t = \frac{v_t}{1 - \beta_2^t}, \quad \theta_t = \theta_{t-1} - \eta \frac{M_t}{\sqrt{V_t} + \epsilon},$$

where $m_t, v_t$ are biased first and second moment estimates, respectively; $M_t, V_t$ are bias-corrected first and second moment estimates, respectively; $\eta$ – learning rate; $\beta_1, \beta_2, \epsilon$ – Adam’s hyper-parameters.

### 2.3.6 Overfitting and underfitting

Machine learning (ML) methods learn from data, i.e. they are looking at the provided annotated data points and try to learn the correct mapping from inputs to outputs. The power of ML models is assessed by its ability to *generalize*, i.e. produce correct outputs on previously unseen inputs. In practice, poor generalization is usually caused by one of two common phenomena. If the model fits training data too well (i.e. for fitting 10 points drawn from noisy square function it chooses
The generalization is assessed on the data, unseen during training, called test set, which can not be used to tweak model further. In order to prevent overfitting (underfitting), the data, given for training an ML model, is often divided into training set and validation set (sometimes also called development set). Training set is then used to learn model parameters and validation set for assessing its generalization performance. Test set is never available in practice, so a common approach is to adapt 80/10/10 split of training data, i.e. 80% of the data comprises training set, 10% used for validation and final 10% is left as a test set.

2.3.7 Hyperparameter optimization

Each machine learning (ML) model has a set of variables to be learnt, which are called model parameters. However these are not the only variables, associated with the model; others, called hyper-parameters, are associated with the model’s learning algorithm, e.g. number of hidden neurons, batch size and learning rate for neural networks. A search over the hyper-parameter space in order to find those leading to the best possible generalization performance is called hyper-parameter optimization or model selection.

Two widespread model selection methods are manual search and grid search. The latter involves searching through the specified grid of hyper-parameters values, taking all possible combinations of those into account and assessing model’s performance on the validation set. Manual search usually consists of two stages: coarse-grained search, involving searching over range of potentially feasible hyper-parameters, and fine-grained search, searching in the smaller ranges around promising hyper-parameters, found during the coarse search.

Both manual and grid searches are computationally exhausting, since the search space grows exponentially as the number of hyper-parameters in consideration increases (known as curse of dimensionality first described in (Bellman, 2015)). Bergstra and Bengio (2012) introduced the random search over hyper-parameters space, which results in statistically equal and sometimes superior performance. In this work, they defined a hyper-parameter response function $\Psi(\lambda)$ over the hyper-
parameters $\lambda \in \Lambda$ as:

$$\Psi(\lambda) = \text{mean}_{x \in X^{(valid)}} L(x; A_\lambda(X^{(train)})�),$$  \hspace{1cm} (2.19)

where $L(x; A_\lambda(X^{(train)})उ$ denotes the loss function on the data point $x$ for the model trained using the training algorithm $A$, governed by a set of hyper-parameters $\lambda$, on the training set $X^{(train)}$; $X^{(valid)}$ denotes a validation set.

Bergstra and Bengio (2012) show that random search, i.e. drawing independent samples from the uniform distribution over the hyper-parameters within specified ranges of values, is more efficient than grid search. This is especially the case for those functions $\Psi$ that have a low effective dimensionality, i.e. most of hyper-parameters do not affect $\Psi$. The influence of each hyper-parameter is then assessed using Automatic Relevance Determination (ARD), i.e. fitting Bayesian regression into the collected experimental dataset, where each data point represents a sampled set of hyper-parameters and the corresponding label is a performance metric. The detailed ARD description is out of scope of this thesis, for diving into more details, refer to (Neal, 2012) or (Rasmussen, 2004).

In order to decide a number of random trials to perform during such search, a simple probabilistic derivation is needed. Assume that the target hyper-parameter combinations occupy the volume $\nu$ relative to the unit hypercube, e.g. $\nu = 0.01$ means that only 1% of the whole search space contains the desired hyper-parameter combinations. Then the probability of not finding a desired combination is $(1 - \nu)$. The probability to miss a target combination in $N$ trials amounts to $(1 - \nu)^N$. Then the probability of finding at least one target combination in $N$ trials is $1 - (1 - \nu)^N$. Assuming that at least one target combination is desired to be found with a probability of at least $p$, the number of necessary random trials is then found as follows:

$$1 - (1 - \nu)^N \geq p$$

$$(1 - \nu)^N < 1 - p$$

$$N \log(1 - \nu) < \log(1 - p)$$

$$N \geq \frac{\log(1 - p)}{\log(1 - \nu)}$$  \hspace{1cm} (2.20)
2.3.8 Word vectors

Traditionally in NLP, words were represented by themselves, i.e. raw strings. However, neural networks require words to be encoded numerically. The simplest option is called one-hot encoding, i.e. getting a vector with a size of the predefined fixed vocabulary with all dimensions being 0 except for the one denoting the word of interest. Such vectors are sparse and take neither the word order in the sentence nor semantical relation between words into account.

A way to represent words with dense vectors of lower dimensionality, compared to one-hot encoded ones, was introduced by Bengio, Ducharme, Vincent, and Jauvin (2003). The method is based on a distributional hypothesis stating that if two words $w_1$ and $w_2$ co-occur with the same words, then $w_1$ and $w_2$ have a similar meaning. The idea was then developed further by Mikolov, Sutskever, Chen, Corrado, and Dean (2013) presenting a neural network architecture for learning word vectors, called word2vec and by Pennington, Socher, and Manning (2014), where they made use of a global word-word co-occurrence information and constructed a new weighted regression model for learning Global Vectors, or GloVe. The latter distributional word embeddings, GloVe, are used within this thesis and, hence, are described in a more detailed way.

GloVe

Let a sequence of $T$ words be denoted by $w_{1:T} = \{w_1, w_2, ..., w_T\}$. A local context window of size $k$ for the word $w_i$, $C_k(w_i)$, is a set of words containing $k$ words to the left of $w_i$ and $k$ words to the right of it, i.e. $C_k(w_i) = \{w_{i-k}, ..., w_{i-1}, w_{i+1}, ..., w_{i+k}\}$. A word $w_j$ is said to occur in the context of word $w_i$ if $w_j \in C_k(w_i)$, where the choice of $k$ is arbitrary. The matrix $X$ of word-word co-occurrence counts for any sequence of words is then a square matrix with each element $X_{ij}$ tabulating the number of times word $w_j$ occurs in the context of word $w_i$. Finally, the probability that word $w_j$ appears in the context of word $w_i$ can then be calculated as $P_{ij} = \frac{X_{ij}}{X_i}.$

The key observation for constructing GloVe vectors is that the relationship between words $w_i$ and $w_j$ can be studied using their co-occurrence probabilities with respect to various probe words $w_k$. This observation and few other enforced demands (for instance, the ability
to encode word information using vector differences) lead to a word embedding model of the following form:

\[ F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \tag{2.21} \]

where \( \tilde{w} \) are separate context vectors, which are equivalent to \( w \) in our case and might be different only as a result of random initialization. An elaborate discussion on the role of separate context vectors can be found in the original paper (Pennington et al., 2014).

Given the equivalence of \( w \) and \( \tilde{w} \) we want our model to be invariant under the exchange \( w \leftrightarrow \tilde{w} \). This can be achieved by requiring \( F \) to be a homomorphism between the algebraic groups \((\mathbb{R}, +)\) and \((\mathbb{R}_{>0}, \times)\), i.e.,

\[ F((w_i + w_j)^T \tilde{w}_k) = F(w_i^T \tilde{w}_k)F(w_j^T \tilde{w}_k). \tag{2.22} \]

Since we are interested in vector differences, the summation and multiplication are substituted with their respective inverse operations,

\[ F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}. \tag{2.23} \]

Combining Equations (2.21) and (2.23), the following simplification is obtained,

\[ F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}. \tag{2.24} \]

A natural choice of a function satisfying the homomorphism defined in Equation (2.23), is \( F = e^x \), hence Equation (2.24) transforms as follows,

\[ e^{w_i^T \tilde{w}_k} = \frac{X_{ik}}{X_i}, \tag{2.25} \]

\[ w_i^T \tilde{w}_k = \log X_{ik} - \log X_i, \tag{2.26} \]

Absorbing the term \( \log X_i \) into bias \( b_i \), since it is independent of \( k \), and adding an additional bias \( \tilde{b}_k \) provides the required invariance,

\[ \log X_{ik} = w_i^T \tilde{w}_k + b_i + \tilde{b}_k. \tag{2.27} \]

Finally a weighted least squares regression model is formulated for the vocabulary of the size \( V \) and a weighting function \( f \),

\[ J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{2.28} \]

The solution of this least squares problem with respect to \( w \) gives us the desired GloVe word embeddings.
2.4 Evaluation

For evaluating our results we use standard information retrieval metrics, precision and recall. In order to simplify their definition let us first introduce a confusion matrix, which looks as follows:

<table>
<thead>
<tr>
<th>True class</th>
<th>Predicted class</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
</tr>
<tr>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

Figure 2.3: Confusion matrix

Precision (PR) and recall (REC) are then computed as follows:

\[
PR = \frac{TP}{TP + FP} \quad REC = \frac{TP}{TP + FN}
\] (2.29)

The metric that combines precision and recall into one scalar value is called F1-score and is calculated as the harmonic mean of precision and recall, i.e.:

\[
F1 = \frac{2 \times PR \times REC}{PR + REC}
\] (2.30)
Chapter 3

Methods

3.1 Gathering data

This section describes the dataset, used in the current thesis, presents the challenges in annotating the data and describes the utilized ways of collecting new data.

3.1.1 SpaceRef

The dataset, used within the current thesis, is called SpaceRef and is described in (Götze & Boye, 2016b). This dataset contains descriptions of immediate geographical environment given by pedestrians following predefined routes. The spoken utterances were automatically recorded and recognized, while referring expressions (RE) were manually annotated. Along with linguistic information, the physical context is available in the form of pedestrian’s location (GPS coordinates) and the Open Street Map (OSM) representation of objects in the proximal surroundings, called a candidate set (to be described in the subsection 3.2.2).

During the initial examination of the dataset, we found that the manual RE annotation in SpaceRef is sometimes inconsistent. In the following examples underlined words denote correctly annotated REs, whereas under-waved words denote REs with missing annotations. For instance, pronominal anaphora is sometimes annotated as RE and sometimes not, e.g. “I just took this street coming out from the square and I’m walking it straight” and “the fountain in front of you – cross it and arrive to the fountain”, although in both cases “it” is referen-
Other inconsistencies include some geographical objects not being labeled and some REs, similar by structure, being annotated inconsistently in different utterances, e.g. “I’m supposed to now walk downstairs I guess” and “I’m downstairs now”. In order to resolve the ambiguities, we created a set of RE annotation rules (described in the next subsection), which is non-exhaustive and tackles only the ambiguous cases met during re-labeling the SpaceRef.

### 3.1.2 Data labeling strategy

As mentioned in section 2.2, reference resolution (RR) implies 3 stages: REs identification, delineation of a search space and the actual resolution. We see identification task as a classification problem, where each word is to be assigned one of the three labels (inspired by the BIO labeling strategy for named entity recognition (NER)):

- **O** if the word is outside of an RE;
- **B-REF** (or simply **B**) if the word is starting a new RE;
- **I-REF** (or simply **I**) if the word is in the middle of an RE.

We found annotating the REs being a non-trivial and ambiguous task, requiring resolution on the semantic level and leading to different annotations depending on the linguistic context and the annotator. Hence, in order to be consistent, we developed the following RE annotation guidelines (annotations in bold are the preferred ones among the given options).

- **RE with interjections** should be annotated by assigning the label **O** to interjections.
  
  \[
  \text{near } \text{eh the red eh brick building}
  \]
  \[
  O \quad O \quad B-REF \quad I-REF \quad O \quad I-REF \quad I-REF
  \]

- **Specificational RE** is the one including a reference along with referent’s properties that can not be interpreted separately (should be annotated as one whole).
  
  \[
  \text{there is a fountain to my left}
  \]
  \[
  O \quad O \quad B-REF \quad I-REF \quad I-REF \quad I-REF \quad I-REF
  \]
• **Nested locational RE** is the one containing another RE, specifying a referent’s location. The internal RE should be annotated as a separate one, since it is referring to a distinct geographical object.

```
a fountain in the middle of the park
B-REF I O B-REF I I I I
B-REF I O O O O B-REF I
B-REF I I I I I I I I
```

• **Nested specificational RE** is the one containing another RE, specifying referent’s properties that can not be interpreted without the outer (inner) RE. Both REs should be annotated as one whole.

```
an entrance to the tunnelbana station
B-REF I-REF I-REF I-REF I-REF I-REF
B-REF I-REF O B-REF I-REF I-REF I-REF
```

• **Definitional RE** is the one defining (clarifying) the previous RE with the help of a verb phrase (VP), e.g. “is called”. The REs divided by VP should be annotated as separate.

```
the first street is called valhallavägen
B-REF I-REF I-REF O O B-REF
B-REF I-REF I-REF I-REF I-REF I-REF
```

• **Clarificational indefinite RE** is the one referring to a broad set of geographical entities, but without an intention to specify one clear referent. This typically occurs in clarificational questions.

```
is there an embassy in front of you
O O O O O O O O O
O O B-REF I-REF O O O O O
```

• **Clarificational definite RE** is the one containing multiple geographical entities, of which one entity is perceived as the main one and the others are giving additional information that might be useful during the resolution step. All such entities should be annotated as one continuous RE.

```
at uggelviksgatan and valhallavägen in the corner
O B I I I I I I
O B O B O B I I I
```
These guidelines are reflecting the design decisions taken in the present work. The ambiguity and necessity of understanding semantics, arising during annotation, suggest machine learning methods to be worth exploring for identifying the references.

### 3.1.3 Collecting new data

Modern machine learning (ML) methods, especially neural networks, benefit from large amounts of data. Therefore, the new data have been collected to enable better training and evaluation of ML models. Instead of Wizard-of-Oz setting, used for collecting a SpaceRef dataset, a simple Android app WalkTalk (see Figure 3.1) has been developed for recording the descriptions of pedestrian’s immediate vicinity. When the app is started, a pedestrian has to wait until the GPS connection is established and then can start a walk (see Figure 3.1a). Each walk is a collection of audio files, containing the descriptions of the environment (see Figure 3.1b), and XML files with geographical data obtained from GPS. A pedestrian is not supposed to describe the environment continuously, but only specific points of interest, such as landmarks (see Figure 3.1c).

![Figure 3.1: WalkTalk application UI](image)

(a) Establishing GPS connection  
(b) Main view featuring walks  
(c) Recording a description

The data have then been transcribed using Google’s speech recognition system with some manual corrections. The REs have been annotated manually following the guidelines, described in the subsection 3.1.2.
3.2 Approaching reference resolution

In this section we describe and justify the methods used in this thesis for resolving spatial references. Since the aim of the present work is to explore neural architectures beyond logistic regression, most of the presented machine learning (ML) methods rely on neural networks.

3.2.1 Referring expressions identification

The BIO data labeling strategy, described in the subsection 3.1.2, is inspired by the one used for annotating named entity recognition (NER) datasets. The inspiration comes from the structural similarity of the problems, i.e. both tasks aim at identifying a specific type of expressions in the sentence (recall from subsection 2.2.1 that named entities can be REs). The methods for RE identification (REI) might then also draw inspiration from NER methods, as solving both problems requires taking into account the semantic context, i.e. a label of each word depends on preceding and succeeding words. For example, consider the following 4 passages (NEs are in bold, REs are underlined).

1. “Teddy bear is my daughter’s favourite toy. I want to buy one as a present, they are sold near the **Odenplan** subway station.”

2. “Do you know if there is a subway station nearby? I want to visit the **Teddy Roosevelt** monument.”

3. “You can see a train station near **KTH**, it is for commuter trains and is called **Stockholm’s Eastern**.”

4. “You can see a train departing from the second track. It goes to **Nacka**, one of the Stockholm’s eastern parts.”

In the passages 1 and 2, the word “Teddy” can be resolved correctly only by considering the next word, i.e. “Teddy Roosevelt” is a NE, whereas “Teddy bear” is not. The same applies to the word “train”, which can be identified as RE correctly only using the next word, i.e. “train station” is RE, whereas “train departing” is not.

The preceding word is as important as the next one. For instance, in passage 1 “subway station” is clearly a part of RE, referring to “Odenplan station”, mentioned previously. In the passage 2, a person asks if there is any “subway station” nearby without referring to something
specific, which means it is non-referential. Similar logic applies to “Stockholm’s Eastern” in passages 3 and 4, where a decision on being a part of the named entity also depends on the previous words.

Given the demonstrated structural similarity between NER and REI, we design our REI neural architecture, called RefNet, taking inspiration from the work of Ma and Hovy (2016), whose network produced state-of-the-art results on NER. The RefNet architecture diagram is presented in Figure 3.2.

Figure 3.2: RefNet architecture diagram. The purple block specifies the pre-trained layer; the blocks with a dashed border denote optional layers; thick arrows emphasize that 2D tensors of dimensionality specified to the left of arrows are passed; the blue block denotes RefNet encoder. (Best viewed in color)
Let us now describe the way \textit{RefNet} operates. We start by padding (with a special word \textless pad\textgreater ) or trimming every utterance to some fixed sentence length $L_s$. Each utterance is then fed into \textit{RefNet} word by word, as a part of the training batch. Each word is encoded using $D_w$-dimensional distributional word representations, also called word embeddings or word vectors (GloVe vectors are used in this thesis, for more information refer to Subsection 2.3.8), which are pre-trained, and $D_c$-dimensional character embeddings which are to be trained.

The motivation behind taking into account character level embeddings is the presence of named entities as referring expressions (RE), that will inevitably lack word vectors, e.g. “Valhallavägen”. This happens because such words were most probably absent in the corpora used for training word vectors. In such cases the corresponding word embedding is assigned to be a zero vector leaving character level embedding as the only source of information. Character level embeddings should to be particularly useful when the names of geographical objects are in the different language, e.g. Swedish street names, while a language of a pedestrian is English.

Character level embeddings are trained as a part of \textit{RefNet}. First the word under consideration is split into characters, which are mapped (using a projection layer) to $D_c$-dimensional embeddings, initialized randomly using Xavier initialization (Glorot & Bengio, 2010). Each projected character vector is then fed to the BiRNN with GRUs having $H_c$-dimensional hidden states. The character level word encoding is then a concatenation of hidden states of forward and backward GRUs, i.e. $\tilde{D}_c = 2H_c$. The final word encoding is then a concatenation of the word vector and the character level word encoding, i.e. the dimension is $\tilde{D}_w = D_w + \tilde{D}_c$. These $\tilde{D}_w$-dimensional word encodings are then collected into a sentence encoding, represented by an $L_s \times \tilde{D}_w$ matrix.

The sentence encoding is then fed into another BiRNN (with forward and backward GRUs having $H_s$ hidden units) to encode the sentence level information, which is stored in $L_s \times \tilde{D}_s$ matrix, where $\tilde{D}_s = 2H_s$. This encoding is considered to be the final sentence representation, used as the \textit{RE encoder}, also called the \textit{RefNet encoder}, (the blue block in Figure 3.2), during the reference resolution (RR) step.

After obtaining sentence features, as described above, different linear combinations of those features are explored using the fully connected layer(s) (FCL). We explored two alternatives: the simpler one, when sentence features are fed directly into the final fully connected
softmax prediction layer, an the more complex one, when an extra FCL with \( H_f \) hidden units having ReLU activations followed by a dropout layer is inserted before the softmax FCL (these extra layers are depicted as dashed blocks in Figure 3.2). The final softmax layer produces a matrix with each row representing a probability distribution over 3 labels, i.e. \( O, \ \text{B-REF}, \ \text{I-REF} \), given the word of the utterance.

RefNet is trained by minimizing the cross-entropy loss (see equation (2.18)) using gradient-based optimization methods.

### 3.2.2 Constructing referents search space

As mentioned in the subsection 2.2.2, we need to narrow down the search space of all geographical referents to a smaller set of objects, called the *candidate set*. In the present work we follow the approach, described in (Götze, 2016, Subsection 4.3.2) and (Götze & Boye, 2017), to a large extent. In this subsection we briefly present their method of candidate set construction and highlight the minor introduced changes.

Götze (2016) relies on OpenStreetMap (OSM) for encoding the information about the geographical context. As described in (Götze, 2016, Subsection 2.4.2), the entities in OSM are divided into three types:

- **nodes**, which represent point-like structures, e.g. entrances;
- **ways**, which represent a sequence of nodes, e.g. streets or buildings;
- **relations**, which represent a set of nodes and ways and are used to define logical or geographical relationships between its members, e.g. a bus route, which consists of a set of bus stops (nodes) and roads (ways).

Following Götze (2016), we consider only the first two types of objects as possible referents. To fill in the candidate set we first compute lines of sight around the pedestrian location in 1 degree steps using the “visibility engine”, described in (Boye, Fredriksson, Götze, Gustafson, & Königsmann, 2014). The closest nodes and ways, intersecting with these lines of sight, are included into the candidate set as OSM identifiers. Note, that in contrast to the work of Götze (2016), where lines of sight were computed in every direction between -100 and 100 degrees with respect to the pedestrian’s current direction, we compute lines of sight for all possible directions, i.e. 360 degrees.
Each candidate object is then encoded using the following features:

- **427 binary type features**, automatically extracted from OSM, later referred to as *OSM representation* (see (Götze, 2016, Subsection 4.3.2) for a description of the extraction procedure and (Götze, 2016, Appendix B) for the full list of extracted features);

- **the distance feature**, representing the distance between the pedestrian’s and landmark’s locations, calculated as a minimum of distances to each of the OSM nodes that comprise the landmark in OSM;

- **the angle feature**, which is calculated as the bearing of the landmark, unlike (Götze, 2016), where the angle of the landmark with respect to the pedestrian walking direction was used;

- **the duplicates feature**, which denotes how many other objects from the same candidate set have the same values for all type features.

The last three features will be referred to as *extra geofeatures*. In total 430 features are extracted from OSM, considering the pedestrian’s current location, and represented as a numeric vector, which will be referred to as *geoencoding*.

### 3.2.3 Reference resolution

The resolution step implies matching a textual referring expression with a candidate geographical object. For performing such reference resolution (RR) we employ another neural network architecture, which we refer to as *SpaceRefNet* (see Figure 3.3).

SpaceRefNet takes as input a referring expression (RE) and a candidate geographical object, or simply the candidate. The RE is fed as a padded vector of words, and then encoded using the pre-trained RefNet feature extractor (see subsection 3.2.1), resulting in $L_s \times \tilde{D}_w$ matrix. This sentence encoding matrix is then either reshaped into a $L_s \cdot \tilde{D}_w$-dimensional row-vector by stacking its rows horizontally or averaged over the rows, resulting in $\tilde{D}_w$-dimensional vector. The input candidate is fed either as an OSM representation, optionally compressed using the pre-trained OSM encoder (details of the architecture and the training procedure are given later in this subsection), or in the geoencoded form. Vectors, obtained after encoding both RE and candidate are then concatenated and passed to the fully connected layer.
with $N_h$ hidden units having ReLU activation functions, followed by the final fully connected softmax prediction layer, which produces the probability of a match between the RE and the candidate.

![SpaceRefNet architecture diagram](image)

Figure 3.3: SpaceRefNet architecture diagram. The blocks in purple specify pre-trained layers; the block with dashed borders denotes an optional layer, dashed arrows – optional connections

SpaceRefNet is trained by optimizing the weighted cross-entropy loss using gradient-based optimization methods. Weights for the loss function are introduced, because the SpaceRef dataset has high class imbalance – it has much more negatives (when a candidate and an RE mismatch) than positives (when a candidate and an RE match). To counteract this, a contribution of each data point (a candidate and an RE) to the global loss is adjusted using the class-dependent multiplication factors (negatives receive lower weights than positives), allowing us to penalize the network more for the mistakes made on positive data points. Formally, the cross-entropy loss, introduced in the equation (2.18), changes its form as follows:

$$ J = -\frac{1}{N} \sum_{i=1}^{N} w_i y_i^T \log(p_i), \quad (3.1) $$

where $y_i$ is the true label for the data point $x_i$, $y_i$ is a one-hot encoded
true label and $p_i$ is the predicted probability distribution over labels for the same data point.

As mentioned above, the encoded candidate might be optionally compressed via the undercomplete autoencoder, which we call OSM autoencoder (see Figure 3.4). Input and output layers have 430 neurons, corresponding to the number of features in geoencoding (see subsection 3.2.2), whereas the hidden layer has $D_{enc} < 430$ neurons. The neurons of both hidden and output layers have sigmoid activation functions. The training is performed by minimizing a mean squared error loss, introduced in the equation (2.17). When the training is finished, the output layer is disregarded and the remaining network is used as the OSM encoder in the SpaceRefNet.

![Figure 3.4: OSM autoencoder architecture](image)

\[ 427 \text{ type features} \]

\[ D_{enc} \]

\[ 427 \text{ type features} \]

3.3 Baselines

In order to evaluate the performance of RefNet and SpaceRefNet, the respective baselines for REI and RR tasks were implemented. In this section a brief description of both baselines is given.

3.3.1 Referring expressions identification

The REs are always represented by the noun phrases (NP), so the natural baseline is just returning every found NP as a candidate RE. The
baseline was implemented as follows:

- a part-of-speech (POS) tag was defined for each word in an utterance using the Stanford POS tagger for English (Toutanova, Klein, Manning, & Singer, 2003), to be more specific, the wsj-0-18-bidirectional-distsim version was used;

- the POS-tagged utterance is then parsed using NLTK RegexpParser (Bird, Klein, & Loper, 2009), supplied with the following grammar:

  \[
  NP: \{ (<DT>\{? <RB.*>* <JJ>* <JJ>* <NN.*>)* <IN.*> + <IN.*> \} + \}
  \]

- all found NPs are returned as REs.

### 3.3.2 Reference resolution

The natural RR baseline is just querying OSM database and checking for geographical objects with an OSM property containing at least one word from the utterance (except stop words) either in a property key or value. For example, consider two utterances, “a very nice big park” and “a huge green area”, are being matched with the geographical object “Humlegården” (see Figure 3.5). The utterances are first split by space and then all the stop words are removed. The result would be as follows:

1. {very, nice, big, park}
2. {huge, green, area}

Figure 3.5: “Humlegården” geographical object representation in OSM
Each word is then checked against all properties of the OSM object (both keys and values are checked). The first utterance will then be matched with “Humlegården”, because the “leisure” tag has value “park”, which is part of the utterance. The second utterance will not be matched, since none of the words matches any of the property keys or values.
Chapter 4

Experimental Results

In this section the experimental results for the referring expressions identification (REI) and reference resolution (RR) models, described in chapter 3, are presented. In order to make distinguishing between model variations easier, specific identifiers are introduced for each variation in Table 4.1.

<table>
<thead>
<tr>
<th>Model ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefNet</td>
<td>the architecture shown in Figure 3.2</td>
</tr>
<tr>
<td>RedRefNet</td>
<td>RefNet without ReLU FCL (shown as dashed)</td>
</tr>
<tr>
<td>SpaceRefNet</td>
<td>the architecture shown in Figure 3.3 without OSM encoder (shown as dashed)</td>
</tr>
<tr>
<td>SpaceRefNet+</td>
<td>the architecture shown in Figure 3.3 with OSM encoder</td>
</tr>
</tbody>
</table>

Another set of identifiers is introduced for those hyper-parameters of RefNet and SpaceRefNet that were not mentioned in the subsections 3.2.1 and 3.2.3. This set includes three identifiers: \( \eta \) for learning rate, \( b \) for the batch size and \( ||g||_{max} \) for the maximum gradient norm used for clipping the gradients.

The dataset was split using 80/10/10 strategy for all experiments, i.e. 80% of the data comprised a training set, 10% were used as a validation set and 10% were left out for a test set.

For the purpose of repeatability all experiments were conducted with a random number generator seeded with 29 for both TensorFlow (version 1.5) and NumPy (version 1.14). The underlying inter-
preter was Python 3.5. Note, however, that TensorFlow implementation is parallel and some operations are internally implemented as non-deterministic for the sake of speed (e.g. “reduce_sum”). Therefore, obtaining exactly the same results might be impossible. However, due to the presence of efficiency curves (the concept is introduced later in this chapter), the variance of results can be compared.

4.1 Referring expressions identification

In this section the experimental results on the REI task are described. The performance of the REI baseline, described in the subsection 3.3.1, on the validation set, containing 119 REs, is presented in Table 4.2. Sentence accuracy shows the proportion of sentences in which every word has been assigned the correct label.

<table>
<thead>
<tr>
<th>Performance metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.3125</td>
</tr>
<tr>
<td>Recall</td>
<td>0.4622</td>
</tr>
<tr>
<td>Sentence accuracy</td>
<td>0.2742</td>
</tr>
<tr>
<td># of returned REs (# of true REs)</td>
<td>176 (119)</td>
</tr>
</tbody>
</table>

In order to compare RefNet with a baseline, the former needs to be trained. First, RefNet’s word embedding projection layer was fixed to use 50-dimensional GloVe.6B\(^1\) word vectors, pre-trained by the Stanford NLP research group. Next a number of hyper-parameters needs to be chosen beforehand. The standard procedure is performing either a manual or grid search or a combination of them. However, for RefNet with its 8 hyper-parameters, these methods are either computationally infeasible (a grid search) or lack efficiency (a manual search). Instead, exploration of the hyper-parameter space was performed using a random search, described in the subsection 2.3.7. Target hyper-parameter combinations were assumed to occupy 2.5% of the search space (because of computational reasons) and the probability of finding target hyper-parameters was desired to be at least 0.95. Hence,

\(^1\)Pre-trained GloVe.6B word vectors can be downloaded at their official website https://nlp.stanford.edu/projects/glove/
according to the equation (2.20), a number of random trails $N$ to be performed is,

$$N \geq \frac{\log(0.05)}{\log(0.975)} \approx 118,$$

which was rounded to 120 random trials.

The hyper-parameter search space was also narrowed down to 7-D space by fixing the number of training epochs to 10 for speeding up the computations. The hyper-parameters for each random trial were sampled in the alphabetical order. Experiments on the number of training epochs were performed after the random search.

Results of the hyper-parameter random search (HPS) for RefNet are partially shown in Table 4.3 by presenting sampled hyper-parameter configurations that resulted in the 5 best and 5 worst performances on the validation set in terms of F1 score. Following Bergstra and Bengio (2012), the RefNet HPS results are then summarized using the efficiency curve, presented in Figure 4.1a, and relevance boxplots, shown in Figure 4.1b. To let a reader understand subsequent comments on results without consulting with (Bergstra & Bengio, 2012), the concepts of efficiency curves and relevance boxplots are briefly presented in the next paragraphs with the highlighted deviations from the original source.

A random experiment efficiency curve is constructed using the fact that random trials, constituting the experiment, are independent identically distributed (i.i.d.). Hence, $N$ i.i.d. trials can also be interpreted as $K$ independent experiments of $n$ trials each, if $Kn \leq N$. On the x-axis, a number of trials $n$ is varied, on the y-axis a performance metric boxplot is presented. For each experiment the maximum value of a performance metric among all trials is used as a data point, contributing to a boxplot.

To perform automatic relevance determination (ARD), the HPS data were used as an input and the respective performance metrics as labels. The relevance of each hyper-parameter was estimated using ARD method in the spirit of Bergstra and Bengio (2012). However, instead of fitting a Gaussian process regression, a variation of Bayesian ridge regression was fit with a non-isotropic covariance, i.e. different hyper-parameters have different precisions. The relevance boxplots were obtained by re-sampling 80% of HPS data 50 times and fitting the Bayesian ridge regression on each of the samples. The ARDRegression implementation, available in scikit-learn, a framework introduced in (Pedregosa et al., 2011), was utilized for performing ARD.
Table 4.3: TOP-5 and BOTTOM-5 results (by F1 score) of RefNet hyper-parameter random search experiments

<table>
<thead>
<tr>
<th>ID</th>
<th>Sampled configuration</th>
<th>Prec.</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#83</td>
<td>$H_c = 40; H_s = 76; H_f = 38; D_c = 91;$ $\eta \approx 0.0081;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#47</td>
<td>$H_c = 36; H_s = 84; H_f = 46; D_c = 18;$ $\eta \approx 0.0035;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#72</td>
<td>$H_c = 39; H_s = 69; H_f = 34; D_c = 33;$ $\eta \approx 0.0069;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#26</td>
<td>$H_c = 21; H_s = 94; H_f = 34; D_c = 74;$ $\eta \approx 0.0061;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#27</td>
<td>$H_c = 27; H_s = 72; H_f = 13; D_c = 73;$ $\eta \approx 0.0071;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>#115</td>
<td>$H_c = 35; H_s = 53; H_f = 12; D_c = 27;$ $\eta \approx 0.0003;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#39</td>
<td>$H_c = 12; H_s = 14; H_f = 17; D_c = 59;$ $\eta \approx 0.0003;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>$H_c = 22; H_s = 33; H_f = 23; D_c = 56;$ $\eta \approx 0.0001;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#103</td>
<td>$H_c = 42; H_s = 94; H_f = 34; D_c = 90;$ $\eta \approx 0.0001;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#6</td>
<td>$H_c = 28; H_s = 66; H_f = 14; D_c = 69;$ $\eta \approx 0.00008;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.1: Hyper-parameter random search results for RefNet
Analyzing the results of hyper-parameters optimization for RefNet, multiple observations are of particular interest. Firstly, the most relevant hyper-parameters are a learning rate $\eta$ and a batch size $b$, while a gradient clipping norm $||g||_{max}$ is irrelevant. Secondly, the most irrelevant among all the hidden layers is a fully connected layer, later referred to as FCBP layer (shown as a dashed block in Figure 3.2). Finally, the performance for random trials ranges from 0.4 to 0.8, as can be seen from the first boxplot of the efficiency curve in Figure 4.1a.

Taking into account mentioned observations, the RefNet might be simplified by removing the FCBP layer and the subsequent dropout layer. This relaxed version of RefNet, which is later referred to as RedRefNet, has a potential of a better (or at least not worse) generalization according to not only the ARD results, but also the general principle of Occam’s razor (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987).

Results of hyper-parameter search (HPS) for RedRefNet are partially presented in Table 4.4 where those sampled hyper-parameter configurations that resulted in the 5 best and 5 worst performances on the validation set in terms of F1 score are shown. The efficiency curve and relevance boxplots for RedRefNet are shown in Figure 4.2.

The HPS results revealed that RedRefNet performance is similar to RefNet ranging from roughly 0.4 to 0.8 in F1 score. The TOP-5 best results of RedRefNet were slightly better than those of RefNet with the difference in F1 score of 0.03 between the respective 5th places and the difference of 0.0053 between the winners. A learning rate and a batch size remain the most relevant hyper-parameters, along with a character embedding dimensionality that has become relevant for RedRefNet as well. Sizes of hidden layers along with a gradient clipping norm are found to be totally irrelevant.

Taking into account the results, presented above, and the Occam’s razor principle, the RedRefNet was chosen as the preferred model for REI task, on which the subsequent hyper-parameter search was performed. Note, that in order to speed up the computation process for an RR task, the best RedRefNet model (with id 82) was chosen to serve as RE encoder during SpaceRefNet random hyper-parameter search, results of which are presented in the section 4.2. Note that further reductions in the REI network’s architecture might be possible, since the size of the hidden layers is not found to be of a particular importance. However, such experiments were not performed within the present work because of restrictions in time and computational power.
### Table 4.4: TOP-5 and BOTTOM-5 results (by F1 score) of RedRefNet hyper-parameter optimization experiments

<table>
<thead>
<tr>
<th>ID</th>
<th>Sampled configuration</th>
<th>Prec.</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#82</td>
<td>$H_c = 22; H_s = 65; D_c = 14$; $\eta \approx 0.0056;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#70</td>
<td>$H_c = 23; H_s = 90; D_c = 38$; $\eta \approx 0.0084;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#108</td>
<td>$H_c = 23; H_s = 87; D_c = 75$; $\eta \approx 0.0079;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#51</td>
<td>$H_c = 10; H_s = 50; D_c = 88$; $\eta \approx 0.0056;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#55</td>
<td>$H_c = 29; H_s = 100; D_c = 31$; $\eta \approx 0.0055;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>#86</td>
<td>$H_c = 49; H_s = 83; D_c = 66$; $\eta \approx 0.0081;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#78</td>
<td>$H_c = 25; H_s = 57; D_c = 79$; $\eta \approx 0.0003;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#7</td>
<td>$H_c = 28; H_s = 14; D_c = 88$; $\eta \approx 0.0008;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#84</td>
<td>$H_c = 34; H_s = 38; D_c = 72$; $\eta \approx 0.0003;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>#44</td>
<td>$H_c = 12; H_s = 64; D_c = 49$; $\eta \approx 0.0003;</td>
<td></td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>

![Efficiency curve and Relevance of hyper-parameters](image.png)

**Figure 4.2:** Hyper-parameter random search results for RedRefNet
Next, experiments on the required number of training epochs were performed in order to prevent RedRefNet model from overfitting. For this purpose, RedRefNet models were trained for 20 epochs, instead of 10 used for hyper-parameter optimization. Hyper-parameters from the best quarter of hyper-parameter configurations in terms of F1 score (later referred to as simply top quarter), sampled during the random search, were used. The overfitting criterion was defined as the increase of validation loss for three consecutive epochs. The tipping point, i.e. the epoch after which the overfitting criterion was satisfied, is referred to as an overfitting epoch. The histogram of overfitting epochs is shown in Figure 4.3. Surprisingly, over a quarter of all configurations in TOP-30 started to overfit after the 10th epoch, furthermore, every trial overfitted within 20 epochs. Overfitting during the first 10 epochs was detected for 7 configurations in the TOP-10. Considering the results, presented in Figure 4.3a, number of epochs was fixed to 15 with the possibility of breaking the training procedure when reaching the overfitting criterion, defined above.

![Histogram of overfitting epochs](image)

(a) TOP-30 (quarter)  (b) TOP-10

Figure 4.3: The histogram of overfitting epochs for sampled configurations of RedRefNet hyper-parameters

Indeed, checking the training procedure summary for the best performing model clearly indicates early overfitting, as shown in Figure 4.4. The overfitting was detected at 570th iteration, 10th epoch, (indicated in blue), i.e. the validation loss was increasing for three consecutive epochs. Notably, the training loss was steadily decreasing, whereas validation set accuracy, precision and recall dropped, which indicates a classic overfitting situation.
Next, a small grid search over the relevant hyper-parameters was performed. Two of the irrelevant hyper-parameters were set to those of the best performing configuration, found during random search, i.e. $H_s = 65, H_c = 22$, whereas a gradient clipping was removed. Relevant hyper-parameters were searched in the ranges of the top quarter mean +/- 1 standard deviation. First, a coarse-grained search over 27 combinations of relevant hyper-parameters was performed searching over $b \in \{32, 55, 78\}$, $\eta \in \{0.0032, 0.0054, 0.0076\}$ and $D_c \in \{25, 52, 79\}$. Results of this grid search are visualized in Figure 4.5.

Figure 4.5: Results of the coarse-grained grid search over the relevant hyper-parameters of RedRefNet. Green triangles represent configurations passing the threshold; red circles – those failing to pass.
F1 scores for all experiments were thresholded at the values of 0.7 and 0.75 and those, passing the threshold (marked as green triangles) were considered as successful. Performance of such successful configurations is shown in Table 4.5.

Table 4.5: RedRefNet coarse-grained grid search results (configurations in the first two rows passed a threshold of 0.75)

<table>
<thead>
<tr>
<th>Checked configuration</th>
<th>Prec.</th>
<th>Recall</th>
<th>F1</th>
<th>Overfit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b = 32; \eta \approx 0.0054; D_c = 25 )</td>
<td>0.7479</td>
<td>0.7542</td>
<td>0.7510</td>
<td>6th epoch</td>
</tr>
<tr>
<td>( b = 32; \eta \approx 0.0076; D_c = 79 )</td>
<td>0.7479</td>
<td>0.7542</td>
<td>0.7510</td>
<td>8th epoch</td>
</tr>
<tr>
<td>( b = 32; \eta \approx 0.0032; D_c = 25 )</td>
<td>0.7522</td>
<td>0.7203</td>
<td>0.7359</td>
<td>8th epoch</td>
</tr>
<tr>
<td>( b = 32; \eta \approx 0.0076; D_c = 25 )</td>
<td>0.6772</td>
<td>0.7288</td>
<td>0.7021</td>
<td>8th epoch</td>
</tr>
<tr>
<td>( b = 55; \eta \approx 0.0054; D_c = 25 )</td>
<td>0.7227</td>
<td>0.7288</td>
<td>0.7257</td>
<td>5th epoch</td>
</tr>
<tr>
<td>( b = 55; \eta \approx 0.0076; D_c = 25 )</td>
<td>0.7143</td>
<td>0.7627</td>
<td>0.7377</td>
<td>6th epoch</td>
</tr>
<tr>
<td>( b = 78; \eta \approx 0.0076; D_c = 25 )</td>
<td>0.8734</td>
<td>0.5847</td>
<td>0.7005</td>
<td>No</td>
</tr>
</tbody>
</table>

An interesting observation is that 7 out of 8 successful configurations were detected to overfit and all of them – before the 10th epoch. The found successful configurations were considered as a base for the second phase of the grid search – a fine-grained search. This phase was performed for the following relevant hyper-parameter values:

- \( D_c \) was fixed to a value of 25 (the vast majority of coarse search results support this value);
- \( b \in \{30, 31, 33, 34, 35\} \) (values in the vicinity of 32);
- \( \eta \in \{0.0056, 0.006, 0.0064, 0.0068, 0.0072\} \) (values between 0.0054 and 0.0076).

Three best-performing hyper-parameter configurations are shown in Table 4.6.

Table 4.6: RedRefNet fine-grained grid search results

<table>
<thead>
<tr>
<th>ID</th>
<th>Checked configuration</th>
<th>Prec.</th>
<th>Recall</th>
<th>F1</th>
<th>Overfit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( b = 34; \eta \approx 0.0064 )</td>
<td>0.8515</td>
<td>0.7288</td>
<td>0.7854</td>
<td>10th epoch</td>
</tr>
<tr>
<td>2</td>
<td>( b = 35; \eta \approx 0.0064 )</td>
<td>0.8019</td>
<td>0.7203</td>
<td>0.7589</td>
<td>10th epoch</td>
</tr>
<tr>
<td>3</td>
<td>( b = 30; \eta \approx 0.0072 )</td>
<td>0.75</td>
<td>0.7627</td>
<td>0.7563</td>
<td>8th epoch</td>
</tr>
</tbody>
</table>
Results of the fine-grained search are better in precision, but worse in recall. In an ideal case, REI should favor recall more, because false positives should simply not be resolved during RR step (if RR system works perfectly). In the real world situations, however, the RR system also makes mistakes, hence, the precision should be prioritized, i.e. given two models with a comparable recall (say, within a distance of 0.05), the one with a higher precision is of interest. The best model, found during fine-grained search has much lower recall than the best model found during RedRefNet hyper-parameter random search, i.e. RedRefNet variation #82. Thereafter, the most promising REI model is RedRefNet #82 that achieves the precision of 0.7917 and the recall of 0.8051 on the validation set.

4.2 Reference resolution

The performance of the RR baseline, described in the subsection 3.3.2, on the validation set is presented in Table 4.7.

<table>
<thead>
<tr>
<th>Class</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positives</td>
<td>0.1036</td>
<td>0.2149</td>
<td>0.1398</td>
</tr>
<tr>
<td>Negatives</td>
<td>0.9864</td>
<td>0.9684</td>
<td>0.9773</td>
</tr>
</tbody>
</table>

The SpaceRefNet hyper-parameter space is 7-D (too large for both manual and grid search), therefore a random search was performed in the spirit of the one for RefNet. Following the same calculations as in the equation (4.1), the number of random trials was set to 120. The search space is narrowed down to 6-D by fixing the number of training epochs to 5, which is less than for the case of RefNet, as the SpaceRefNet training takes more time because of the necessity to encode an RE with a RefNet encoder. It should also be noted that gradient clipping was not performed, because of the network’s simplicity. Extra geofeatures were scaled to avoid their numerical domination over an OSM representation. The chosen method aimed at rough scaling, i.e. the distance was divided by 100, the angle – divided by 360, the duplicates feature – divided by 10. Experiments on the number of epochs were performed afterwards.
A set of additional identifiers for hyper-parameters, specific to SpaceRefNet, was introduced, consisting of three identifiers: \(d\) for a distance feature, \(a\) for an angle feature and \(dup\) for a duplicates feature.

Results of the SpaceRefNet hyper-parameter random search (HPS) are partially shown in Table 4.8 by presenting those sampled hyper-parameter configurations that resulted in the 5 best and 5 worst performances on the validation set (the results were sorted by F1 score for positives).

Table 4.8: TOP-5 and BOTTOM-5 results (by F1 score for positives) of SpaceRefNet hyper-parameter optimization experiments

<table>
<thead>
<tr>
<th>ID</th>
<th>Sampled configuration</th>
<th>Positives Prec.</th>
<th>Positives Recall</th>
<th>Negatives Prec.</th>
<th>Negatives Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>#36</td>
<td>(N_h = 45; \eta = 0.0027; b = 365) (d = 0; a = 1; dup = 0;)</td>
<td>0.1</td>
<td>0.6364</td>
<td>0.9932</td>
<td>0.9026</td>
</tr>
<tr>
<td>#95</td>
<td>(N_h = 35; \eta = 0.0031; b = 293) (d = 1; a = 1; dup = 1;)</td>
<td>0.0878</td>
<td>0.5868</td>
<td>0.9922</td>
<td>0.8963</td>
</tr>
<tr>
<td>#65</td>
<td>(N_h = 54; \eta = 0.0094; b = 432) (d = 1; a = 1; dup = 1;)</td>
<td>0.0851</td>
<td>0.6942</td>
<td>0.9941</td>
<td>0.8731</td>
</tr>
<tr>
<td>#57</td>
<td>(N_h = 100; \eta = 0.0014; b = 256) (d = 1; a = 1; dup = 1;)</td>
<td>0.0819</td>
<td>0.6942</td>
<td>0.9940</td>
<td>0.8676</td>
</tr>
<tr>
<td>#82</td>
<td>(N_h = 89; \eta = 0.0023; b = 422) (d = 0; a = 1; dup = 0;)</td>
<td>0.0822</td>
<td>0.6198</td>
<td>0.9927</td>
<td>0.8824</td>
</tr>
</tbody>
</table>

All SpaceRefNet HPS results are then summarized for positives
and negatives separately to find out the impact of hyper-parameters on the class-wise performance. The efficiency curve and relevance boxplots are presented in Figures 4.6a and 4.6b, respectively, for positives, and in Figures 4.6c and 4.6d for negatives. For a brief introduction to the concepts of an efficiency curve and relevance boxplots, refer to the section 4.1.

![Figure 4.6: Hyper-parameter random search results for SpaceRefNet](image)

In the SpaceRefNet HPS results multiple observations are of particular interest. Firstly, the learning rate and batch size are relevant hyper-parameters for both positives and negatives, while the number of hidden units is seen as relatively relevant. Secondly, the most relevant among extra geofeatures are the distance feature for positives and the duplicates feature for negatives. Finally, the performance during random trials ranges from 0.03 to 0.18 in F1 score for positives and from 0.45 to 0.95 in F1 score for negatives (ignoring outliers).

The next experiment was conducted in order to find out whether
the SpaceRefNet can be simplified further (which according to the fa-
mous Occam’s razor should lead to the same or better performance if
the model does not become too simple). The simplification is made by
reducing the size of the vector coming out of the concat layer. This is
done with the help of the OSM encoder network, introduced in Figure
3.4, pre-trained for 20 epochs resulting in the mean squared error of
the order of $10^{-4}$. This variation of SpaceRefNet is later referred to as
SpaceRefNet+. The results of hyper-parameter search for SpaceRefNet+
are presented in Table 4.9 (sampled hyper-parameter configurations,
resulted in 5 best and worst performances on validation set) and in
Figure 4.7 (efficiency curve and relevance boxplots).

Table 4.9: TOP-5 and BOTTOM-5 results (by F1 score for positives) of
SpaceRefNet+ hyper-parameter optimization experiments

<table>
<thead>
<tr>
<th>ID</th>
<th>Sampled configuration</th>
<th>Positives</th>
<th>Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Prec.</td>
<td>Recall</td>
</tr>
</tbody>
</table>
| #70  | $N_h = 99; \eta = 0.0043; b = 461$
      | $d = 0; a = 0; dup = 1$ | 0.0848     | 0.7355     | 0.9948     | 0.865     |
| #95  | $N_h = 35; \eta = 0.0031; b = 293$
      | $d = 1; a = 1; dup = 1$ | 0.0817     | 0.7355     | 0.9947     | 0.8595    |
| #82  | $N_h = 89; \eta = 0.0023; b = 422$
      | $d = 0; a = 1; dup = 0$ | 0.0815     | 0.7025     | 0.9942     | 0.8655    |
| #26  | $N_h = 25; \eta = 0.0076; b = 345$
      | $d = 0; a = 0; dup = 1$ | 0.0790     | 0.7107     | 0.9943     | 0.8592    |
| #110 | $N_h = 55; \eta = 0.0096; b = 57$
      | $d = 0; a = 0; dup = 0$ | 0.0782     | 0.6033     | 0.9924     | 0.8792    |
|      | ...                   | ...        | ...        | ...        | ...        |
| #67  | $N_h = 50; \eta = 0.0087; b = 95$
      | $d = 1; a = 0; dup = 0$ | 0.0317     | 0.8678     | 0.9959     | 0.5492    |
| #91  | $N_h = 38; \eta = 0.0079; b = 232$
      | $d = 0; a = 0; dup = 1$ | 0.0299     | 0.9339     | 0.9977     | 0.4841    |
| #4   | $N_h = 56; \eta = 0.0065; b = 61$
      | $d = 1; a = 0; dup = 1$ | 0.0260     | 0.9504     | 0.9979     | 0.3958    |
| #63  | $N_h = 89; \eta = 0.0073; b = 58$
      | $d = 1; a = 1; dup = 1$ | 0.0254     | 0.9587     | 0.9981     | 0.3745    |
| #66  | $N_h = 15; \eta = 0.0080; b = 104$
      | $d = 0; a = 0; dup = 0$ | 0.0173     | 0.9835     | 0.9944     | 0.0503    |
Interestingly, the performance of SpaceRefNet+ is worse for positives compared to SpaceRefNet, but almost the same for negatives, which can be explained by previously mentioned high class imbalance in the dataset (far more negatives). Another interesting observation is that the distance feature becomes even more relevant (relative to other features) for positives, whereas duplicates feature loses its relevance for negatives. However, both batch size and learning rate remain relevant, which is an extra indicator that ARD works correctly, since these two hyper-parameters are known to be relevant from general theory on neural networks, briefly presented in the section 2.3. The final observation is that SpaceRefNet+ results in higher recall for positives, i.e. more true positives, and lower recall for negatives, i.e. less true negatives, compared to SpaceRefNet. Nonetheless, the precision for positives is somewhat lower in the case of SpaceRefNet+, meaning that it tends to return more false positives.

In the case of reference resolution, precision is of relatively higher importance, since false positives might lead a pedestrian to a totally
different place. Note, that the recall is also of interest, but between two models with a comparable recall (within 0.1), the one with a higher precision should be chosen. Hence, the chosen neural reference resolution model is SpaceRefNet.

Next, experiments on the required number of training epochs were performed in order to prevent SpaceRefNet model from overfitting. For this purpose, SpaceRefNet model variations were trained for 20 epochs, instead of 5, used for hyper-parameter optimization. The best quarter (by F1 score for positives) of hyper-parameter configurations, sampled during a random search (later referred to as simply top quarter) were used to experiment on the number of epochs. The same overfitting criterion was utilized as for RefNet (see section 4.1). The histogram of overfitting epochs is shown in Figure 4.8.

![Graph showing overfitting epochs](image_url)

Figure 4.8: The histogram of overfitting epochs for the sampled configurations of SpaceRefNet hyper-parameters

As can be seen, more than a half of experiments overfitted during the first 3 epochs, which is surprising given that the SpaceRefNet has only one hidden layer. The reason for such an unexpected behaviour might be that the defined overfitting criterion is irrelevant in the context of SpaceRefNet. In order to investigate this phenomenon further, the training procedure summary for three best performing models, according to the random search, was examined (see Figure 4.9).

Multiple observations are of interest. First, the training procedure is highly unstable – a training loss fluctuates too much. This leads to an unstable performance on the validation set and the inability to reason properly about the network’s generalization capabilities. Taking into account that such chaotic training procedure is observed for
the three best models, the respective performance might have been achieved simply by chance.

(a) Model variation #36, overfitted at the 4th epoch

(b) Model variation #95, overfitted at the 12th epoch

(c) Model variation #65, overfitted at the 3rd epoch

Figure 4.9: Training procedure summaries for SpaceRefNet variations. A number of training iterations is plotted on the x-axes, network’s performance metrics are plotted on the y-axes. The period of training when the overfitting was detected, i.e. the overfitting epoch and two next ones, is shown in blue (best viewed in color)
Secondly, in all three considered cases the increase of the validation loss does not lead to performance decrease. This is not only true for the presented cases, but also is a general trend. On average, early stopping (when overfitting first appears) prevented a validation loss from increasing by $1.6 \times 10^{-10}$, while leading to the following average performance losses (i.e. the network would perform better without early stopping):

- the loss of 0.0892 in the accuracy, whereas the average accuracy is 0.7298;
- the loss of 0.0486 in the F1 score for positives (0.0062 in the precision, whereas the average precision for positives is 0.0618);
- the loss of 0.1090 in the F1 score for negatives (0.0001 in the precision and 0.09 in the recall, whereas the averages are 0.9939 and 0.7299 respectively).

The only metric omitted in the list above is the recall for positives, because this is the only performance gain observed. On average, early stopping prevented a decrease of 0.0151 in the recall for positives, whereas the average recall for positives is 0.7217.

A general trend in the overfitting cases is that a recall for positives drops (in 60.71% of the cases), whereas a precision for positives and a recall for negatives increase (in 78.57% and 67.86% of the cases, respectively). The precision for negatives drops and increases roughly the same number of times (46.43% and 53.57% of the cases, respectively). The accuracy is highly dominated by the recall for negatives (also increases in 67.86% of the cases), because the dataset suffers from a high class imbalance. The loss value increases, because positives are given more weight, meaning that decrease in a recall for positives has high chances of causing the increase in loss. The recall for negatives decreases less frequently, but can also influence the loss significantly, because of the high class imbalance, mentioned previously. Considering all the facts, presented above, one can conclude that the overfitting criterion does not reflect the performance of SpaceRefNet.

The next attempt on determining the necessary number of training epochs was searching a general pattern in the network’s behaviour. For this purpose the best SpaceRefNet model variation #36, found during a random search, was trained for 200 epochs. The training procedure summary is shown in Figure 4.10.
Clearly, validation loss steadily increases over time, along with the precision for positives and the recall for negatives, whereas the precision for negatives and the recall for positives decrease. No clear performance tipping point is observed. The training procedure is unstable and does not reach a plateau within 200 epochs. The observed performance hints that the SpaceRefNet can not be trained to reach a stable performance. For instance, the validation precision for positives can decrease from near 0.2 to near 0.12 in one epoch, which warns about possible poor generalization properties.

The only part of the network that remained stable over all previous experiments is the RE encoder. Recall that for each RE the respective encoding is obtained by concatenating rows of \( L_s \times \widetilde{D}_s \) matrix, resulting in the \( L_s \widetilde{D}_s \)-dimensional vector and, hence, the same number of weights from the input to the hidden layer of SpaceRefNet to be learned. It was previously observed that the OSM encoder does not improve SpaceRefNet performance (and arguably worsens it). Let us now experiment on the RE encoder to observe whether any performance effects are present.

One of the ways to improve the RE encoder is to reduce the dimensionality of encoding (should improve performance by Occam’s razor). Recall that each row of the \( L_s \times \widetilde{D}_s \) RE encoding matrix represents the encoding of one word. The easiest way to reduce the dimensionality of the sentence encoding is by averaging over all rows instead of concatenating. Such variation of SpaceRefNet will be referred to as...
SpaceRefNetAv. Before performing a hyper-parameter random search for SpaceRefNetAv, let us run a small experiment and train such network for 10 epochs using the following hyper-parameters:

- the learning rate $\eta = 0.0046$ and the number of hidden units $N_h = 60$ (top quarter means of the configuration, sampled during SpaceRefNet HPS random search);
- batch size $b = 300$ (top quarter mean rounded to the nearest integer divisible by 100)
- all extra geofeatures are enabled, i.e. $d = 1$, $a = 1$, $dup = 1$.

The training procedure summary for the model described above, later referred to as *spacerefnet_average_mtr1*, is presented in Figure 4.11. It can be observed that the training procedure is far more stable for this model than for both SpaceRefNet and SpaceRefNet+. Furthermore, the performance of this model, trained for 10 epochs, is better than the average performance of the SpaceRefNet models, trained for 20 epochs! Thereafter, SpaceRefNetAv is a promising model for performing a random search on. However, because of time limitations, the random search assumptions were relaxed – target hyper-parameter combinations were assumed to occupy 5% of the search space. Hence, according to the equation (2.20), the number of random trials $N$ to be performed is,

$$N \geq \frac{\log(0.05)}{\log(0.95)} \approx 58,$$

which was rounded to 60.

Figure 4.11: Training procedure summary for the SpaceRefNet model variation *spacerefnet_average_mtr1*. A number of training iterations is plotted on the x-axes, performance metrics are plotted on the y-axes.
The SpaceRefNetAv hyper-parameter random search (HPS) results are partially shown in Table 4.10 by presenting those sampled hyper-parameter configurations that resulted in the 5 best and 5 worst performances on the validation set (the results were sorted by F1 score for positives). All SpaceRefNetAv HPS results are then summarized for positives and negatives separately to find out the impact of hyper-parameters on the class-wise performance. The efficiency curve and relevance boxplots are presented in Figures 4.12a and 4.12b, respectively, for positives, and in Figures 4.12c and 4.12d for negatives. A brief introduction to the concepts of an efficiency curve and relevance boxplots can be found in the section 4.1.

Table 4.10: TOP-5 and BOTTOM-5 results (by F1 score for positives) of SpaceRefNetAv hyper-parameter optimization experiments

<table>
<thead>
<tr>
<th>ID</th>
<th>Sampled configuration</th>
<th>Positives</th>
<th>Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Prec.</td>
<td>Recall</td>
</tr>
<tr>
<td>#4</td>
<td>$N_h = 56; \eta = 0.0065; b = 111$; $d = 1; a = 0; dup = 1$;</td>
<td>0.1359</td>
<td>0.8347</td>
</tr>
<tr>
<td>#54</td>
<td>$N_h = 94; \eta = 0.0072; b = 245$; $d = 1; a = 1; dup = 1$;</td>
<td>0.1346</td>
<td>0.8512</td>
</tr>
<tr>
<td>#3</td>
<td>$N_h = 77; \eta = 0.0029; b = 202$; $d = 1; a = 1; dup = 0$;</td>
<td>0.1292</td>
<td>0.8595</td>
</tr>
<tr>
<td>#25</td>
<td>$N_h = 79; \eta = 0.0047; b = 168$; $d = 1; a = 0; dup = 0$;</td>
<td>0.1301</td>
<td>0.8182</td>
</tr>
<tr>
<td>#50</td>
<td>$N_h = 89; \eta = 0.0088; b = 304$; $d = 1; a = 1; dup = 0$;</td>
<td>0.1256</td>
<td>0.8430</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>#59</td>
<td>$N_h = 64; \eta = 0.0004; b = 358$; $d = 1; a = 0; dup = 1$;</td>
<td>0.0780</td>
<td>0.8099</td>
</tr>
<tr>
<td>#29</td>
<td>$N_h = 36; \eta = 0.0004; b = 299$; $d = 0; a = 0; dup = 1$;</td>
<td>0.0718</td>
<td>0.7769</td>
</tr>
<tr>
<td>#20</td>
<td>$N_h = 62; \eta = 0.0001; b = 293$; $d = 1; a = 0; dup = 0$;</td>
<td>0.0699</td>
<td>0.7769</td>
</tr>
<tr>
<td>#48</td>
<td>$N_h = 27; \eta = 0.0010; b = 437$; $d = 0; a = 0; dup = 0$;</td>
<td>0.0699</td>
<td>0.7355</td>
</tr>
<tr>
<td>#56</td>
<td>$N_h = 33; \eta = 0.0002; b = 370$; $d = 0; a = 1; dup = 1$;</td>
<td>0.0649</td>
<td>0.7686</td>
</tr>
</tbody>
</table>
As can be observed, the performance of SpaceRefNetAv is superior to SpaceRefNet, which indicates that averaging word encodings results in a sentence encoding that is easier to use for RR than one obtained with a concatenation. The performance superiority can also be seen from the efficiency curves, where the F1 score ranges from 0.13 to 0.23 for positives and from 0.9 to 0.95 for negatives. Note that a substantially shorter range in F1 score for negatives also indicates a more stable training procedure.

One interesting change in hyper-parameter relevance is that the distance feature became relevant for both positives and negatives along with stably relevant hyper-parameters, i.e. the learning rate, the number of hidden units and the batch size.

Next experiment involved modifying the SpaceRefNetAv architecture by including the OSM encoder with the aim of further reducing the number of parameters to be learned. Such model variation is later referred to as SpaceRefNetAv+. Another motivation for this experiment
is to check whether OSM encoder worsens the performance in general (as in the case with SpaceRefNet).

The SpaceRefNetAv+ hyper-parameter random search (HPS) results are partially shown in Table 4.11 by presenting those sampled hyper-parameter configurations that resulted in the 5 best and 5 worst performances on the validation set (the results were sorted by F1 score for positives). All SpaceRefNetAv+ HPS results are then summarized for positives and negatives separately to find out the impact of hyper-parameters on the class-wise performance. The efficiency curve and relevance boxplots are presented in Figures 4.13a and 4.13b, respectively, for positives, and in Figures 4.13c and 4.13d for negatives.

Table 4.11: TOP-5 and BOTTOM-5 results (by F1 score for positives) of SpaceRefNetAv+ hyper-parameter optimization experiments

| ID | Sampled configuration | Positives | Negatives | | | |
|----|----------------------|------------|-----------|----------------|----------------|
|    |                      | Prec.      | Recall    | Prec.         | Recall         |
| #17| $N_h = 28; \eta = 0.0060; b = 118$; $d = 1; a = 0; dup = 0;$ | 0.1190 | 0.7520 | 0.9954 | 0.9053 |
| #25| $N_h = 28; \eta = 0.0047; b = 168$; $d = 1; a = 0; dup = 0;$ | 0.1169 | 0.8099 | 0.9964 | 0.8960 |
| #19| $N_h = 67; \eta = 0.0064; b = 117$; $d = 1; a = 1; dup = 0;$ | 0.0963 | 0.8182 | 0.9964 | 0.8695 |
| #22| $N_h = 28; \eta = 0.0044; b = 169$; $d = 1; a = 1; dup = 0;$ | 0.0929 | 0.8182 | 0.9964 | 0.8641 |
| #4 | $N_h = 56; \eta = 0.0065; b = 111$; $d = 1; a = 0; dup = 1;$ | 0.0926 | 0.7769 | 0.9957 | 0.8706 |
| ...| ... | ... | ... | ... | ... |
| #5 | $N_h = 33; \eta = 0.0096; b = 382$; $d = 0; a = 1; dup = 1;$ | 0.0599 | 0.8430 | 0.9966 | 0.7750 |
| #23| $N_h = 16; \eta = 0.0061; b = 375$; $d = 1; a = 1; dup = 1;$ | 0.0585 | 0.9008 | 0.9978 | 0.7537 |
| #45| $N_h = 30; \eta = 0.0084; b = 327$; $d = 0; a = 1; dup = 1;$ | 0.0579 | 0.8099 | 0.9958 | 0.7757 |
| #56| $N_h = 33; \eta = 0.0002; b = 370$; $d = 1; a = 0; dup = 1;$ | 0.0576 | 0.7851 | 0.9953 | 0.7815 |
| #10| $N_h = 14; \eta = 0.0092; b = 403$; $d = 0; a = 1; dup = 1;$ | 0.0438 | 0.8512 | 0.9963 | 0.6844 |
Interestingly, the performance of SpaceRefNetAv+ is worse than the one of SpaceRefNetAv. According to the efficiency curves the F1 score ranges from 0.11 to 0.17 for positives, excluding outliers, (compared to 0.13 and 0.23 for SpaceRefNetAv) and from 0.87 to 0.93 for negatives, excluding outliers, (compared to 0.9 and 0.95 for SpaceRefNetAv). Surprisingly the learning rate has become irrelevant for both positives and negatives, while the batch size and the number of hidden units are still relevant. The distance and duplicates features are relevant for positives, whereas negatives favor only duplicates feature.

The clear RR model winner is SpaceRefNetAv. The validation set performance after training for 100 epochs is presented in Table 4.12 and the training procedure summary is shown in Figure 4.14.
Table 4.12: SpaceRefNetAv model variation #4 performance on the validation set after training for 100 epochs (maximum is taken by F1 score for positives)

<table>
<thead>
<tr>
<th>Result</th>
<th>Positives Prec.</th>
<th>Positives Recall</th>
<th>Negatives Prec.</th>
<th>Negatives Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final (100th epoch)</td>
<td>0.2160</td>
<td>0.6694</td>
<td>0.9942</td>
<td>0.9587</td>
</tr>
<tr>
<td>Maximum (93rd epoch)</td>
<td>0.2547</td>
<td>0.6694</td>
<td>0.9942</td>
<td>0.9667</td>
</tr>
</tbody>
</table>

Figure 4.14: Training procedure summary for the SpaceRefNet model variation #4. A number of training iterations is plotted on the x-axes, network’s performance metrics are plotted on the y-axes.

As can be seen, the maximum performance is not reached at the final epoch, which hints on the usage of early stopping. However, the training procedure is definitely more stable than for the SpaceRefNet, which makes the SpaceRefNetAv a very promising RR model.

### 4.3 Test set performance

In this section the performances of the best found models (winners) and the baselines for referring expressions identification (REI) and reference resolution tasks (RR) are presented.

The REI winner is RedRefNet model variation #82, whose performance on the test set is presented in Table 4.13 along with the respective baseline.
Table 4.13: REI winner model (RedRefNet variation #82) and baseline performances on the test set

<table>
<thead>
<tr>
<th>Performance metric</th>
<th>Baseline</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.4505</td>
<td>0.8256</td>
</tr>
<tr>
<td>Recall</td>
<td>0.5495</td>
<td>0.7802</td>
</tr>
<tr>
<td>Sentence accuracy</td>
<td>0.3820</td>
<td>0.7303</td>
</tr>
<tr>
<td># of returned REs</td>
<td>111 (91)</td>
<td>86 (91)</td>
</tr>
</tbody>
</table>

As can be seen, the winner model confidently beats the baseline, resulting in the far superior precision and sentence accuracy.

The RR winner model is SpaceRefNetAv variation #4, whose test set performance is shown in Table 4.14 along with the RR baseline.

Table 4.14: RR winner model (SpaceRefNetAv variation #4) and baseline performances on the test set

<table>
<thead>
<tr>
<th>Model</th>
<th>Class</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Positives</td>
<td>0.1981</td>
<td>0.2143</td>
<td>0.2059</td>
</tr>
<tr>
<td></td>
<td>Negatives</td>
<td>0.9846</td>
<td>0.9831</td>
<td>0.9838</td>
</tr>
<tr>
<td>Winner</td>
<td>Positives</td>
<td>0.3607</td>
<td>0.8980</td>
<td>0.5146</td>
</tr>
<tr>
<td></td>
<td>Negatives</td>
<td>0.9979</td>
<td>0.9689</td>
<td>0.9832</td>
</tr>
</tbody>
</table>
Chapter 5
Discussion and Future Work

In this chapter a brief discussion of the obtained results is presented and directions for future work are suggested. Each section in the chapter addresses one specific question, specified in the title, by presenting the found answers and/or clues along with direction for future investigations.

5.1 Was the amount of data sufficient?

The SpaceRef is a medium-sized dataset, containing 1463 referring expressions (REs) and the candidate set of 61 geographical object per RE on average. Training recurrent neural networks (RNNs) on such a small dataset is somewhat risky, as the generalization properties might be poor. However, both RefNet and SpaceRefNet demonstrated promising results on the test data, meaning that the size and quality of the dataset were sufficient to detect at least some regularities in the data.

Nonetheless, remembering the famous quote “Data overwhelms the prior”, new data were collected using the developed Android application. The data consists of the GPS logs and the recorded utterances at some points of time in the regions of Stockholm, different from the region, utilized for collecting SpaceRef. Data annotation and cleaning turned out to be more time-consuming than expected, therefore, all the data were not processed and, hence, not used for training the model.

To get further insights about this question the new data should be fully annotated and the performance of the existing models should be
tested on this new data. As the new data was collected in the different geographical environments, generalization capabilities of the trained models will be evaluated better in future.

Another viable approach might be to create data by simulation, for instance using Generative Adversarial Networks (GANs), introduced in (Goodfellow et al., 2014). The generator network could generate the GPS traces and utterances and discriminator network would learn to distinguish between real and generated utterances, thus improving the quality of generated data points (hopefully).

5.2 Were the encodings good enough?

The results, presented in Chapter 4, have revealed that the OSM encoder worsens the performance (or slows down the training procedure), hence the OSM encoding should not be obtained using the OSM autoencoder, introduced in Figure 3.4.

The RE encoding, taken from RefNet, has shown promising results as an underlying representation of REs for SpaceRefNet. As have been observed, different ways of combining word encodings into a sentence encoding lead to different performances. Recall from Subsection 3.2.1, that RE encoding is a matrix, where each row contains a word encoding, which is a concatenation of BiRNN hidden states (for forward and backward cells). The concatenation of rows in the matrix into one large vector turned out to result in worse performance than averaging over the rows. This is somewhat natural, because the number of parameters to be learned by SpaceRefNet has been dramatically reduced.

Nevertheless, averaging over the hidden states can be improved further. One possible improvement would be the concatenation of forward cell’s last state and backward cell’s first state, as those state are expected to capture the information about the whole sequence (being fed in the forward or backward direction, respectively).

5.3 Was the loss chosen correctly for SpaceRefNet?

As discussed in Section 4.2, the weighted cross-entropy loss does not reflect the network’s performance due to the high class imbalance in
the dataset. For instance, the increase in validation set loss does not lead to the decrease in precision for positives. On the other hand, the recall for positives decreases, which is the primary reason of the validation loss increase (a positive example increases the loss 120 times more than a negative). Hence, detecting overfitting using a weighted cross-entropy loss is problematic. Nonetheless, the way of choosing a loss that would increase only when both precision and recall decrease is not clear.

5.4 Closing remarks

To summarize, this thesis presents two neural network models for the tasks of referring expressions identification and reference resolution. Both models were extensively evaluated and have been proven to beat the respective baselines. Both models have given promising results in general leading to conclusion that using neural network based models for dealing with REs is the promising direction for the future research.
Bibliography


Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton project para*. Cornell Aeronautical Laboratory.


