Lexical Semantics on Word Embeddings through Deep Metric Learning

Mathias Etcheverry

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República en cumplimiento parcial de los requerimientos para la obtención del título de Doctor en Informática

Supervisora
Dina Wonsever

Tribunal
Luciana Benotti (Revisora)
Horacio Saggion (Revisor)
Héctor Cancela
Javier Couto
Eduardo Mizraji

Uruguay
2024

…

Agradecimientos

Le doy las gracias a mi supervisora Dina Wonsever por su dedicación, orientación y paciencia. Trabajar con Dina siempre ha sido un gusto para mi.

También quiero agradecer al grupo de Procesamiento de Lenguaje Natural del Instituto de Computación, en especial a Aiala Rosá, Diego Garat, Guille Moncecchi, Juanjo Prada, Luis Chiruzzo, Nacho Sastre, Santi Góngora y Santi Castro; y también al resto del Instituto de Computación por el soporte y paciencia, por haber sido una fuente de recursos y conocimientos tanto para esta tesis como en otras actividades.

Me gustaría agradecer también a los colegas que en congresos fueron fuente de motivación y energía para este trabajo. En especial a Enrico Santus, ya que nuestra interacción en portoroz 2016 (LREC) sobre sus resultados en hiperonimia y antonimia con semántica distribucional fueron una de las motivaciones iniciales para el tema de esta tesis. A Edwin Puertas, con quien nos juntamos en Cartagena en 2022 (iberamia), luego de conocernos en 2017 en Murcia (tass, sepln), su motiviación con su tesis de doctorado sobre las mejoras que estaba obteniendo al incluir elementos fonológicos en el análisis de sentimiento me resultó motivante para terminar. Y a Gaurav Gupta por el timepo compartido en Gold Coast 2023 (IJCNN), por insistirme con que me apure en terminar el doctorado para aspirar a una posición postdoctoral.

Le agradezco a la Comisión Académica de Posgrado por el apoyo finaciero al haberme aceptado para la beca de doctorado con la cual realicé este trabajo.

Por último pero no menos importante, le agradezco a mi familia y amigos. ¡Gracias! Esta tesis es para ustedes, claro que no por esto tienen que leerla ☺.

"No podemos ni dispensarnos de conocerlas ni dar un paso sin recurrir a ellas; y sin embargo su deslindamiento es un problema tan delicado, que nos preguntamos si tales unidades existen en realidad."

Ferdinand de Saussure

Abstract

In the last time, remarkable advances in natural language processing, such as translation and dialog systems, have been obtained. The technologies used include neural network models and distributed representations of word semantics, known as word embeddings.

This thesis deepens in the field of word embeddings and the use of deep metric learning approaches for lexical semantics. Three main lexical relations are considered: synonymy, hypernymy, and antonymy. These relations, which are particularly related with paraphrasing and textual entailment, are aligned with three mathematical binary relations: equivalence, partial orders, and antitransitive relations, respectively. Equivalence and partial order relations are well-known in mathematics, differing in the relation symmetry property. Regarding antitransitive relations, they overcome the inadequacy of the two previous ones to model contrariness or opposition, which is present in antonymic terms.

The approach of this thesis is to address the aforementioned semantic relations using techniques inspired by deep metric learning. Particularly, taking into consideration properties like relation symmetry and transitivity, in order to formulate well-suited approaches for each relation. The approach is to reencode a set of pretrained word embeddings on a supervised learning setup to generalize the relation to unseen cases. For synonymy, siamese neural networks are suitable, since equivalence relations fit well to be modeled by terms of distance functions. For hypernymy detection, order embeddings were used to learn ordered representations with successful results. For antonymy, the parasiamese and repelling parasiamese network models were developed, allowing us to distinguish antonyms from synonyms in the embedding space, through weakening the tendency of siamese and triplet networks to learn transitive relations.

The main contributions from this thesis are: (1) the use of order embeddings to encode word embeddings for hypernymy detection, (2) the development of the parasiamese and repelling-parasiamese models to learn antitransitive relations, particularly antonymy, (3) experiments with benchmark and specially tailored datasets obtaining state-of-the-art results, and (4) two datasets in Spanish developed for hypernymy detection and antonymy-synonymy discrimination tasks.

Keywords: lexical semantics, deep metric learning, synonymy, antonymy, hypernymy, word embedding, order embedding, siamese network, parasiamese network.

Resumen

En los últimos tiempos se han obtenido notables avances en el procesamiento del lenguaje natural, como puede verse en los sistemas de traducción automática y agentes conversacionales. Las tecnologías utilizadas incluyen modelos de redes neuronales y representaciones vectoriales de la semántica de las palabras, conocidas en inglés como *word embeddings*.

Esta tesis profundiza en el campo de las representaciones vectoriales de palabras y en enfoques de aprendizaje automático, aplicados a la semántica léxica. Se consideran tres relaciones principales de la semántica léxica: sinonimia, hiperonimia y antonimia. Estas relaciones, que están especialmente relacionadas con la paráfrasis y la vinculación textual, se alinean con tres relaciones matemáticas: equivalencia, órdenes parciales y relaciones antitransitivas, respectivamente. Las relaciones de equivalencia y de orden parcial son bien conocidas en matemáticas, y se diferencian por la propiedad de simetría. En cuanto a las relaciones antitransitivas, son utilizadas para modelar la contradicción u oposición presente en los términos antonímicos.

El enfoque de esta tesis consiste en abordar las relaciones semánticas mencionadas utilizando técnicas inspiradas en el aprendizaje métrico profundo. En particular, teniendo en cuenta el cumplimiento o no de las propiedades de simetría y transitividad para formular modelos de redes neuronales para cada relación. El enfoque general consiste en recodificar un conjunto de vectores de palabras preentrenadas mediante ejemplos de la relación para generalizarla a casos no vistos. Para la sinonimia, son adecuadas las redes neuronales siamesas, ya que las relaciones de equivalencia se adaptan bien a ser modeladas en términos de funciones de distancia. Para la detección de la hiperonimia, se entrenaron morfismos de orden para aprender representaciones ordenadas con resultados positivos. Para la antonimia, se desarrollaron los modelos de redes parasiamesas y redes parasiamesas con rechazo, que permiten distinguir antónimos de sinónimos, mediante una simple modificación que altera la tendencia de las redes siamesas y de tripletas a aprender relaciones transitivas.

Los principales aportes de esta tesis son: (1) el uso de morfismos de orden para recodificar vectores de palabras para la detección de hiperónimos, (2) el desarrollo de los modelos de redes parasiamesas y parasiamesas con rechazo para aprender relaciones antitransitivas, en particular la antonimia, (3) experimentos usando los modelos anteriores con conjuntos de datos de referencia y especialmente adaptados obteniendo resultados del estado del arte, y (4) dos conjuntos de datos en español desarrollados para tareas de detección de hiperónimos y discriminación entre antonimia y sinonimia.

Keywords: semántica léxica, aprendizaje métrico profundo, sinonimia, antonimia, hiperonimia, semántica vectorial, morfismo de órden, red siamesa, red parasiamesa.

List of Figures

List of Tables

Contents

1 Introduction

Natural languages, such as English, Mandarin, Spanish, and Swahili among very many others[1], are what we humans use to communicate. We can roughly consider them as systems based on repetitions that allow the generation of interpretable statements. Languages can be spoken or written, and in both cases, sequences are built. Commonly, written languages are presented as sequences of vocabulary symbols (e.g. Latin alphabet: a, b, c, ...), and speech sounds are sequenced by physics nature, and their pronunciation can be expressed as sequences of phonetic symbols (e.g a, ä, æ, b, ...).

Among the elements of languages, the word is one of the most remarkable. Words are meaningful tokens that are used to build more complex structures, such as phrases and sentences. How words are sequenced to form phrases and sentences is the field of study of syntax theory in linguistics. In addition, words have an internal structure, known as morphology.

Firstly, words may be seen as the names that things have, but if we consider words as a nomenclature, this rapidly becomes problematic [Saussure, 1959]. There are names composed of more than one word (e.g. New York), and the same word may refer to entirely different things, such as homonyms and polysemic words. In addition, the meaning of a word depends on the context it is being used. Multiple words may have similar meanings (e.g. easy, simple, effortless), but there are not two words with the same meaning exactly. Moreover, a distinction in the conception of meaning is between sense and reference [Frege, 1948]. While the reference of a word is the thing, event, or state that it indicates, its sense is proper of its usage and the thoughts it evokes. Different words have different senses, even if they denote the same reference.

Distributional semantics attempt to categorize word meanings through their use, for example using occurrences in a corpus [Lenci, 2008]. The words and phrases that surround text fragments, sometimes referred to as linguistic contexts, are used to build word representations, making use of the distributional hypothesis. The latter relates words' contexts to meaning, stating that the words that occur in the same contexts tend to have similar meanings [Harris et al., 1954]. The distributional view on meaning correlates differences of meanings with differences of distributions [Sahlgren, 2008a].

Endorsed by the distributional hypothesis, word embeddings spring into action. Currently, they are vector representations of words, built using word occurrences in a corpus, where words with related meanings tend to be near in the representation

[1] 7,139 spoken languages are being used today according to Ethnologue [Eberhard et al., 2021], where about 40% are in danger of extinction, with less than 1,000 speakers, and only 23 languages cover the half of world population

space. A variety of methods have been proposed to build word embeddings, such as word2vecs' skip-gram and cbow [Mikolov et al., 2013a], GloVe [Pennington et al., 2014a] and Transformers [Vaswani et al., 2017], among many others. The morphology of the words can be used to enhance the model results giving better word representations, especially to those words that are rarely seen or even do not occur in the training corpus [Joulin et al., 2016].

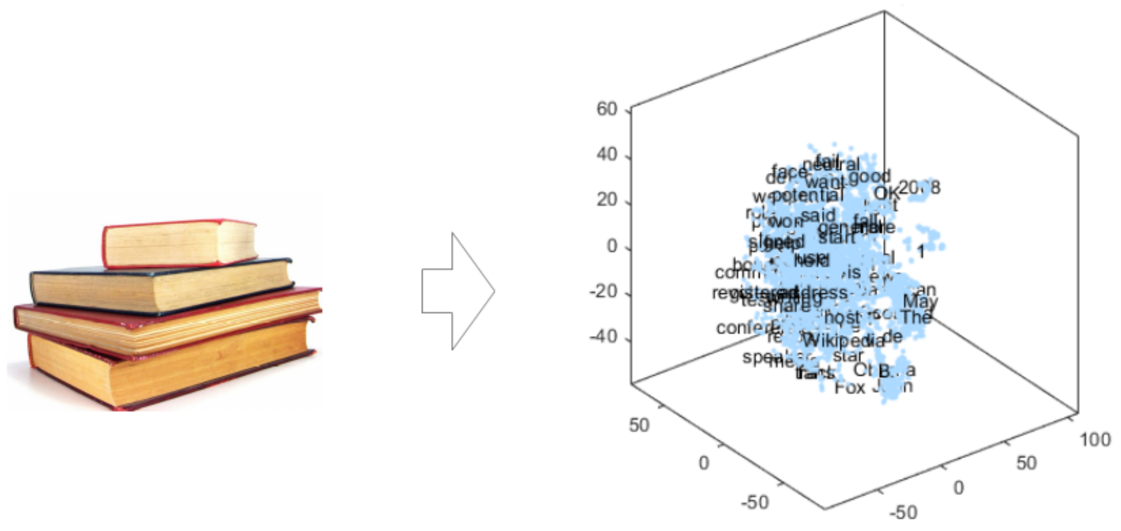


Figure 1 – Diagram of word embedding representation built from text[2]

Word embeddings have become a great tool for natural language processing, that is tightly related to neural networks and neural language models[Bengio et al., 2000, Mikolov et al., 2010]. This is because of the representational power of neural networks. Even, many neural network models that are not specifically designed to obtain word representations obtain them as a side effect. Word embeddings are the most influential idea in modern NLP and state-of-the-art results have been obtained by using them, directly or indirectly, in many, if not all, tasks.

There are a bunch of resources such as models, corpus, and word embeddings publicly available to be used [Akbik et al., 2019]. Word embeddings built using a large corpus can be directly used to face many specific tasks, or they can be specialized [Kiela et al., 2015], or retrofitted [Faruqui et al., 2014, Tissier et al., 2017] to attempt to improve the results in a particular task. In other words, large collections of texts are used to build models of word representations and then these representations are used in a wide variety of tasks. This is the big picture of the most common type of transfer learning used on NLP.

[2] Image composed using images from `https://de.mathworks.com/help/textanalytics/ug/visualize-word-embedding-using-text-scatter-plot.html` and `https://www.alainet.org/es/articulo/179029`

Despite the usefulness of word embeddings, it is not certainly known the semantic features that they encode. Although it seems not necessary to successfully use them to obtain a good performance, deepening their understanding may bring new improvements. Vector distances in word embedding spaces may reflect semantic relatedness (like as in the words *animal*, *feline*, *lion*, *tiger*, *claw* and *sharp*), but that is not enough to distinguish if two words have similar or opposite meanings (e.g *low* and *high*), or if one term is more specific than the other (e.g. *mice* and *rodents*).

Deep metric learning attempts to learn through neural networks data-driven semantic distances. Some of the tasks that have been successfully addressed using deep metric learning include sentence similarity [Mueller and Thyagarajan, 2016], speaker recognition [Wang et al., 2019], person re-identification [Yi et al., 2014], and face verification [Hu et al., 2014], among many others. Additionally, it allows to approach classification scenarios where there are classes that are unknown during training time, and also where just one or a few training examples have been available for each class (i.e. one-shot and few-shot learning) [Peng et al., 2021, Vinyals et al., 2016]. Siamese and triplet neural networks are among the most popular approaches in supervised deep metric learning, which will be detailed later in this work. They correspond to one or more neural networks encoding each input whose weights are fitted through a distance-based loss function in the output space using the training.

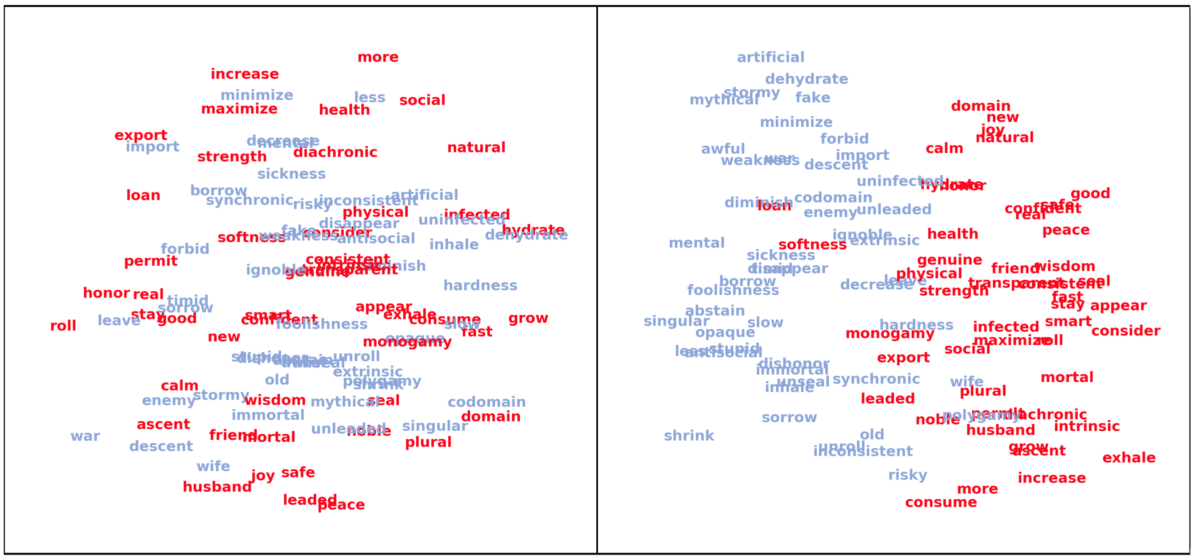


Figure 2 – Word embeddings before and after applying a parasiamese network to distinguish antonymy. Dimensionality reduction for plotting was performed using t-SNE. The words are colored to visualize the separation of the antonyms. The antonyms alternate between red and blue (i.e. if a word is colored blue its antonyms will be colored red), and the color of each word is the same in both images, in the original and in the transformed space.

This thesis deepens in the field of word embeddings and the use of deep metric learning approaches for lexical semantics. Three main lexical relations are considered: synonymy, hypernymy, and antonymy. These relations are aligned with three mathematical binary relations: binary equivalence, partial orders, and antitransitive relations, respectively. Equivalence and partial order relations are well-known in mathematics, differing in the relation symmetry property. Regarding antitransitive relations, they overcome the inadequacy of the two previous ones to model contrariness or opposition, which is present in antonymic terms.

The approach of this thesis is to address the aforementioned semantic relations using techniques inspired by deep metric learning. Particularly, taking into consideration properties like relation symmetry and transitivity, in order to formulate well-suited approaches for each relation. The approach is to reencode a set of pretrained word embeddings on a supervised learning setup to generalize the relation to unseen cases. For synonymy, siamese neural networks [Neculoiu et al., 2016] are suitable, since equivalence relations fit well to be modeled by terms of distance functions. For hypernymy detection, order embeddings [Vendrov et al., 2015a, Etcheverry and Wonsever, 2020] were used to learn ordered representations with successful results. For antonymy, the parasiamese [Etcheverry and Wonsever, 2019] and repelling parasiamese neural network models [Etcheverry and Wonsever, 2023] were developed, allowing to distinguish antonyms from synonyms in the embedding space (see figure 2) through weakening the tendency of siamese and triplet networks to learn transitive relations.

The main contributions from this thesis are: (1) the use of order embeddings to encode word embeddings for hypernymy detection, (2) the development of the parasiamese and repelling-parasiamese models to learn antitransitive relations, particularly antonymy, (3) experiments with benchmark and specially tailored datasets obtaining state-of-the-art results, and (4) two datasets in Spanish developed for hypernymy detection and antonymy-synonymy discrimination tasks in the role of supervisor of two engineering projects [Lee et al., 2020, Camacho et al., 2022].

The remainder of this document is organized as follows. Chapter 2 is dedicated to lexical semantics; the relationships of synonymy, antonymy, and hypernymy are detailed there. Chapters 3 and 4 are for word embeddings and deep metric learning, respectively. Chapter 5 focuses on order embeddings for hypernymy detection. Chapters 6 and 7 present the parasiamese and repelling parasiamese networks for synonymy-antonymy discrimination. In chapter 8, the construction of the datasets for hypernymy detection and antonymy-synonymy discrimination for Spanish is detailed. Finally, chapter 9 exposes the conclusion and future work of this thesis.

1.1 Published papers related to this thesis

The three principal papers published for this thesis are the following.

- Etcheverry, M. and Wonsever, D. (2019). Unraveling antonym's word vectors through a siamese-like network. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3297–3307
- Etcheverry, M. and Wonsever, D. (2020). Order embeddings for supervised hypernymy detection. *Computación y Sistemas*, 24(2):565–574
- Etcheverry, M. and Wonsever, D. (2023). Antonymy-synonymy discrimination through repelling parasiamese neural networks. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 01–07. IEEE

In addition, the following papers were published in the role of supervisor of two engineering projects:

- Lee, G. W., Etcheverry, M., Sánchez, D. F., and Wonsever, D. (2020). Supervised hypernymy detection in spanish through order embeddings. In *Proceedings of the 7th Workshop on Linked Data in Linguistics (LDL-2020)*, pages 75–81
- Camacho, J., Camera, J., and Etcheverry, M. (2022). Antonymy-synonymy discrimination in spanish with a parasiamese network. In *17th Ibero-American Conference on Artificial Intelligence, Iberamia*

2 Lexical Semantics

It is evident the relation of many words, either through their meaning or because they share part of how they are written, Consider, for example, the words *bring*, *brings*, and *bringing*; they are not completely different at all, neither in form nor in meaning, but they are not the same either. And for words like *couch* and *sofa*, while they do not present any morphological or phonological similitude, we can see that their meanings are fairly similar.

This chapter gives a brief introduction to some main concepts of lexical semantics. Starting with concepts on word morphology, such as forms and lemmas, continuing with word senses, and concluding with the semantic lexical relations addressed in this thesis: synonymy, hypernymy, and antonymy.

2.1 Forms, Lemmas, and Stems

Take a look again at the words *bring*, *bring**s*** and *bring**ing***; they all evoke the same word *bring*, and suffixes (highlighted with bold) gives additional information (e.g. the -***ing*** suffix for gerunds). These words could be seen as different forms of the same word, or at least, as words that are highly related by their internal structure. Moreover, for what we refer to as *words* til this point, the term used by some authors is **word-forms** [Jurafsky and Martin, 2021]. The term wordform is also found in the literature as word-form, word form, or simply form. In order to go deeper into word and form, the concepts of **lexeme**, **lemma** and **stemming** are introduced.

A **lexeme** is an abstract concept of the meaning shared by a set of wordforms that are related by inflection. Inflection is a morphological transformation that forms words with the same part of speech[1] and different grammatical categories[2], as for example in *bring*, *brings* and *bringing*. On the other hand, a morphological derivation produces a wordform of a different part of speech (e.g *bring* and *bring-er*). It produces words in different lexemes (see figure 3).

The **lemma** of a word is the name given to a particular wordform, that is considered its canonical form. The lemma of a word is commonly its uninflected form. The wordforms that belong to the same lexeme are grouped by their lemma, which in addition becomes

[1] The parts of speech, also named lexical or syntactic categories, are the word classes considered in syntax theory, such as verbs, nouns, adjectives, adverbs, prepositions, etc.

[2] The grammatical categories are properties of language such as tense, mood, aspect, number, person and gender, among others. Each category takes a value from a proper set (e.g *singular* and *plural* for the category *number*).

a suitable name to denote the lexeme. Since it is grounded on inflectional morphology, it can be expressed according to the grammatical categories for each part of speech. For verbs, the lemma is the infinitive form (e.g. the lemma for the word forms *calm*, *calmed*, and *calming* is *calm*). For the rest of the parts of speech, if it allows inflection by number, then the singular form is used (e.g. the lemma of the word *elixirs* is *elixir*), and for those words that make use of gender inflection, the masculine form is used as the lemma (e.g. in Spanish, the lemma for the word *sempiterna* is *sempiterno*).

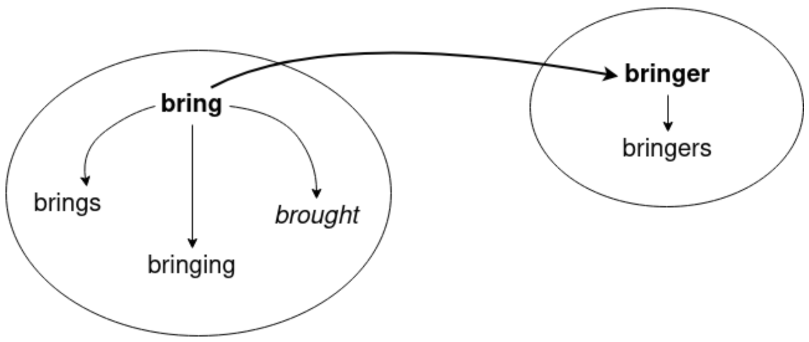


Figure 3 – Lexemes and wordforms. Each circle denotes a different lexeme, bold arrows are derivations, thin arrows correspond to inflections, and wordforms in bold type correspond to the lemma. Morphological exceptions are represented using italics.

2.1.1 Lemmatization and Stemming

Lemmatization is the process of obtaining the lemmas for the words in a sentence. Notice that it is not a deterministic process because of the lexical ambiguity, particularly homonymy (i.e. two different words with the same form, e.g. the word *left* as the opposite of *right* and as the past of *leave*). This makes the use of context an important consideration for the lemmatization task [Bergmanis and Goldwater, 2018]. Consider for example the following two sentences in Spanish:

- Es mejor que sobre y no que falte.
- Sobre gustos no hay nada escrito.

In the first sentence, the word *sobre* is a verb and *sobrar* is its lemma; while in the second example, it is a preposition and its lemma is *sobre*.

Closely related to lemmatization is **stemming**. It refers to chopping off the end of words, removing inflectional and often also derivational affixes. In practical use, both, lemmatization and stemming, takes words to a representative form. A main difference between them is that while lemmatization is context-aware, stemming is often implemented as a context-free heuristic. Considering context in stemming can be useful in some scenarios, like web search, to avoid removing some affixes which removal degrades the precision of search results [Peng et al., 2007], but in general terms, it is driven as context-free.

2.2 Word Senses

Words are a tool to represent meaning. A word may not have one exclusive meaning, and in construction larger than words, such as sentences, words could be substituted by other candidates expressing the same truth values, although expressing a different indirect meaning.

The direct meaning that words have, their literal meaning, is referred to as **denotation** or reference, while **connotation** consists in the indirect thoughts that they evoke. For example, the word *antique* may denote something made a long time ago and its connotation would be positive in many contexts, for example, *antique furniture*, in contrast to *old furniture* that would denote a similar meaning with a negative connotation. The connotation of a word, as well as its denotation, depends on the context it is being used. Notice for example that *old* in *old friends* would have a positive connotation, and the connotation of *antique* in *antique friends* is debatable. Both denotation and connotation, are part of words' meanings.

The different concepts or meanings that a word denotes are called **word senses**. For example, the word *bat*, can be used to refer to the stick used to hit the ball in a baseball game, or the act of hitting something with that same stick, or any specie of the popular winged mammals that move through echolocation in the dark; among others. Word sense disambiguation is performed unconsciously through context when we use language.

2.2.1 Homonymy and Polysemy

A common distinction in lexical ambiguity is between homonymy and polysemy. Two words are **homonyms** if they coincide in form accidentally. They are pronounced (homophones) and written (homographs) exactly the same but they are unrelated, having different etymologies.

On the other hand, **polysemy** is due to the fact that a word can take different related meanings and so, the same form is used to denote them (e.g. *bat* that refers to the stick and also to the act of hitting). In summary, polysemy corresponds to different derived meanings instead of a mere form coincidence as in the case of homonymy.

2.3 Lexical Relations

In semantics, lexical relations are known as a set of binary relations between word meanings. In this thesis, three lexical relations of main importance in semantics are considered: synonymy, antonymy, and hypernymy-hyponymy.

Although relations in lexical semantics are usually defined as relations between word senses rather than word forms, it is useful to consider these relations between forms in the following way: the relation is satisfied between a pair of word forms if it is present in at least one of their senses. This is suitable to consider word representations that condense all word senses and datasets with pairs of words without any particular context or sense identification, becoming particularly beneficial for some computational approaches, such as when (uncontextualized) word embeddings are used.

2.3.1 Synonymy

Two words are said to be synonymous if they have the same or almost the same meaning (e.g *bright* and *shiny*). It is said that there are not perfect synonymous words, since even when they refer exactly to the same object, they are used differently and have different connotations. A good example taken from [Jurafsky and Martin, 2021] is *water* and *H2O*, while both refer to exactly the same element, it is not usual to find *H2O* in, for example, a cooking recipe.

Synonyms are said to have high semantic overlap and a low degree of implicit contrastivity [Cruse, 1986]. For example, the words *lion* and *tiger* have high semantic overlap, being both *felines*, however, they are not synonyms, and many contrasts are present (e.g. the tiger stripes). Synonyms must also have a low degree of contrastivity, if this condition is not satisfied, words would share many similarities but they would not be necessarily synonyms.

Synonymy can be defined in terms of substitution. Two words are synonymous if when substituted in a sentence, then the truth value of the sentence (i.e. its propositional meaning) is unchanged. In addition, it can be seen as an equivalence relationship between word senses (i.e. reflexive, symmetric, and transitive). This allows senses to be grouped in synonym sets, as in the WordNet synsets [Miller, 1995].

Similarity and Relatedness

In many cases, two words are very similar without being synonyms. For example, *horse* and *camel*, both are four-legged animals that are used for transportation, or *bench* and *chair*, which although distinct, are both used for sitting. In lexical semantics, it is common to distinguish between similarity, synonyms, and relatedness.

On the other hand, related words are words that present some relation without being necessarily similar. For example, *tiger* and *claw* do not have similar meanings but they are related.

2.3.2 Antonymy

Antonyms are words with opposite meanings. They correspond to pairs of words such as *big* and *small*, *long* and *short*, *high* and *low*, *tidy* and *untidy*, and so on.

Antonyms are usually classified into different categories depending on the nature of the opposition. The three most widely accepted and generally used are **gradable**, **complementary**, and **relational**.

Gradable (or scalar) antonyms are characterized by being used in comparisons (e.g. *slower-faster*, *thicker-thinner*), involving pairs of words that lie on a continuous spectrum (e.g. *high-low*, *hot-cold*, *rich-poor*).

Complementary (or non-gradable) antonyms, unlike gradable antonyms, cannot be used in comparisons and their meanings do not allow intermediate points (e.g *on-off*, *odd-even*, *live-dead*).

Relational (or converse) antonyms are reciprocal to each other in a situation that involves two roles. Each word in the pair denotes each role of the situation (e.g. *buy-sell*, *give-receive*, *teacher-student*).

Antonyms have high semantic overlap, sharing most dimensions of meaning and contrasting on only a few. This is known as the paradox of simultaneous similitude and difference. This implies that antonyms could be confused with synonyms or other related (and not opposed) terms in NLP, taking to relevant errors in final applications. Distinguishing antonyms is quite an important task in NLP [Jurafsky and Martin, 2009].

Linguistic studies propose that some antonyms are more representative than others, called canonical antonyms [Paradis et al., 2009]. If human candidates are asked about an antonym of a candidate word, for example *hot*, most would give *cold* as an answer, instead of *chilly* or *cool*; suggesting that *hot-cold* presents a higher degree of canonicity than *hot-chilly* or *hot-cool.*

2.3.3 Hyponymy and hypernymy

Hypernymy is a hierarchical relationship that captures the concept *is a type of* or *is a kind of.* It indicates that one concept is more general than the other. It can be thought that the meaning of the more general term is included in the meaning of the more specific one.

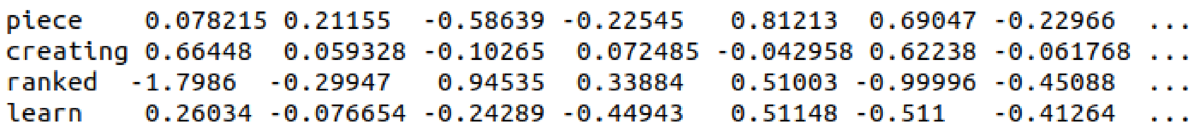
The more general concept is called a **hyperonym** and the specific one a **hyponym**.

For example, *bat-mammal* and *whale-mammal* are cases of hypernymy, where *mammal* is the hyperonym and *bat* and *whale* are hyponyms. When two concepts have a common hyperonym, such as *bat* and *whale*, they are called **cohyponyms**.

Hypernymy is an asymmetric and transitive relation. From a mathematical point of view, it can be understood as a partial order relation.

3 Word Embeddings

Word embeddings, word vectors, and vector semantic spaces are some of the names given to the corpus-based vector representations that are behind the state-of-the-art of most tasks in natural language processing. These representations instantiate the **distributional hypothesis** which says that words that occur in the same contexts tend to have similar meanings. This hypothesis and the abundant text available on the internet make it possible to represent words into a high-dimensional vector space, condensing on each word representation the distribution of the contexts where it occurs. These vectors can be used in multiple ways, for example to compute corpus-based semantic distances between words, to leverage a wide range of NLP tasks transferring learning from a big general text corpus to a much smaller annotated corpus, or in the internals of generative models such as large language models.



```
piece    0.078215 0.21155  -0.58639 -0.22545   0.81213  0.69047 -0.22966  ...
creating 0.66448  0.059328 -0.10265  0.072485 -0.042958 0.62238 -0.061768 ...
ranked  -1.7986  -0.29947   0.94535  0.33884   0.51003 -0.99996 -0.45088  ...
learn    0.26034 -0.076654 -0.24289 -0.44943   0.51148 -0.511   -0.41264  ...
```

Figure 4 – Word vectors examples from [Pennington et al., 2014a].

In practice, a word vector is a vector on a high-dimensionality space (e.g. 300) and is usually dense (i.e. most of their values are not zero). In figure 4 it can be seen examples of the firsts components of some word vectors. Each word vector depends on the contexts where that word occurs. So, words occurring in similar contexts may have similar vectors and therefore similar meanings according to the distributional hypothesis. Figure 5 shows vectors whose dimension have been reduced to 2 using the dimensionality reduction technique t-SNE (t-distributed Stochastic Neighbor Embedding) [Maaten and Hinton, 2008], where it can be observed that words with related meanings tend to be closer in the representation space.

There exist a variety of methods to build word embeddings based on different ideas and mathematical frameworks. Neural language models and methods based on dimensionality reduction of word co-occurrence matrices (known as count-based methods) are among the most popular approaches at the present time. The word vector quality increases as larger corpora and better models are used. The vast amount of text available makes it possible to have a considerably big corpora, spanning multiple domains and languages. Although there are many different methods to build word embeddings, the results obtained are somehow similar: the words are embedded into a vector space, presenting meaningful structures concerning their meanings. Commonly, word vectors don't need to have interpretable components to be used, and in fact, popular methods do not give

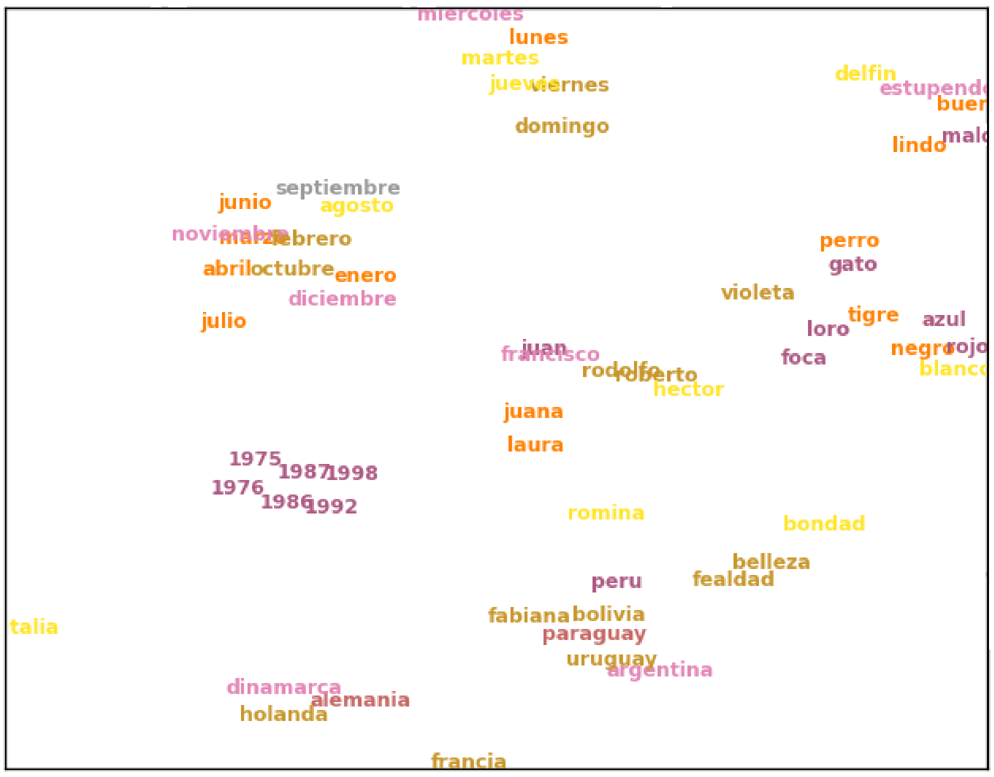


Figure 5 – 2D dimensionality reduction of word vectors using t-SNE showing that related words tend to be clustered. Image from [Etcheverry and Wonsever, 2016] .

an interpretation to any component of the vector. It is enough that they can be used on machine learning models to accomplish tasks.

Concerning the multiple senses that words may have, such as occurs in polysemy and homonymy cases, two families of word embeddings have been developed. On one hand, **static word embeddings** implies that all the senses of a word are represented in a single vector. For example, the vector of the word *right* will be near to the vector of the word *left* and also to the vector of the word *correct*. On the other hand, **dynamic (or contextualized) word embeddings** represents each word occurrence in the text as a vector that corresponds to the meaning of the word in that particular context. Contextualized word embedding models at present time use as input some type of decontextualized word or subword vector. In this thesis, I focus on decontextualized word relationships through the use of static word embeddings.

In the remainder of this section, a review of the methods used in this thesis for word embeddings is presented. The methods used and the approaches considered for lexical semantics tasks are based on neural networks, so first of all a brief overview of neural networks is included. There is a fervently active research area that offers a variety of methods and studies regarding word embeddings and neural network models. This document left out of the scope very many of them. Books such as [Jurafsky and Martin, 2021] and surveys such as [Almeida and Xexéo, 2019] and [Ruder et al., 2019] for static, and [Liu et al., 2020] for dynamic word embeddings can considered for further reading, among many others.

3.1 Artificial Neural Networks Overview

Artificial neural networks is the name given to an on-research family of models of main importance in machine learning. In a few words, they are parameterized functions, whose structure was originally inspired by biological neural networks, that are defined by the composition of more simple functions (often called neurons or units). These parameters are fit during a training process to attempt to solve a given task using input-output samples. Artificial neural networks are suitable for processing multimodal data such as text, images, sounds, videos, etc. to give an output (e.g. give the sentiment orientation of written reviews) as an end-to-end function. There are several techniques and ideas to define and train neural network models. Commonly, neural networks are composed of layers and neurons, taking from the input to the output through successive transformations. The field of neural networks is often referred to as deep learning. At present time, deep learning models are the state of the art in most tasks in fields like natural language processing and computer vision, to name some popular ones, although they are used to process and make inferences on multi-domain data.

A remarkable aspect of neural networks is their capability to build rich task-oriented representations of input data. For example, it can be obtained representation of images where close vectors tend to correspond to images sharing features of concern for the task, for example, images of animals of the same species. Neural network models often undergo successive transformations of the input to obtain an output, with a model that was previously trained through examples of the task in concern.

An important example in NLP is the task of predicting the probability distribution of the next word given a prefix sequence. This is known in NLP as a language model and can be used for text generation (such as chatGPT). When a language model is a neural network it is known as a neural language model. Different neural models have been proposed for language models, and although they treat the sequence differently, in general, they all share the same task format: predict the next token using text previous ones. Language models can be used to generate text through successive applications of the model, using as input the text generated until that moment. As recently mentioned, neural models are usually composed by a sequence of transformations to take from the input to the output. On these transformations internal representations are obtained known as hidden representations. This hidden representations can be seen as a side effect of neural networks, which is deeply related to its essence, that allows obtaining representations of words and other input elements.

Neural network models are a powerful tool to solve tasks and obtain rich representations. In the following, the most important neural network models for the purpose of this thesis are presented.

3.1.1 Feed-forward Networks

The first neural network model is attributed to the artificial neuron of McCulloch-Pitts [McCulloch and Pitts, 1943]. It introduced a model that allows to implement formulas of the propositional calculus through formal neurons. Their work does not present a learning algorithm. A learning algorithm was introduced by the **perceptron** [Rosenblatt, 1958]. The learning algorithm consisted of fitting the internal weights of the model to perform binary classification. These are the predecessors of the artificial neurons that are being used nowadays. From the early stages, precisely since the work of [Minsky and Papert, 1969], it was noticed the importance of using more than one layer of neurons to perform some tasks. This work presented a brilliant analysis of these models, and also their demolition by showing the impossibility of implementing some functions using a single neuron (e.g. the xor function). It showed that **multilayered** models can learn functions that a single layer of neurons cannot, because its inputs are not linearly separable and the models perform a linear combination. In this multilayered scheme, non-linear activation functions become necessary, otherwise, the network would always be a linear function. The models composed by multiple layers have different independent origins, but they were consolidated with the popularization of the **backpropagation** algorithm [Rumelhart et al., 1986]

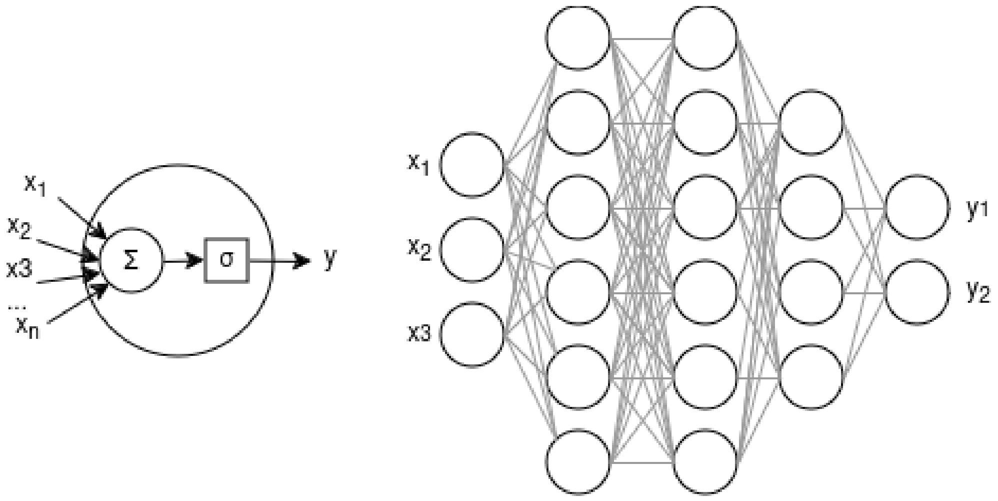


Figure 6 – An artificial neuron (left) and a layered fully-connected neural network diagram (right).

The fully connected feed-forward network, sometimes called **multilayer perceptron** (MLP) for historical reasons, is probably the most basic type of neural network and is often used as a building block in more sophisticated models. This network consists of multiple stacked layers of artificial neurons, where the output of a layer is the input of each neuron of the following layer. The first layer receives the input (e.g a vector to be classified), and the last layer returns the output (e.g. the predicted class through the softmax function). Each neuron is a function with internal weights that receives a vector as input and returns a scalar as output. The neuron computation is performed as the dot product of the input with the neuron weights and the obtained result is passed through a non-linear function, referred to as **activation function**, such as ReLU or sigmoid. The artificial neuron equation can be written as

$$h = \sigma(\vec{x}.\vec{w} + b) \tag{3.1}$$

where h is the neuron output, $\vec{x}$ its input, $\vec{w}$ and b its weights, and σ is the non-linear activation function.

So, in summary, an MLP is a function defined as a composition of neurons, organized in a layered structure, as shown in figure 6.

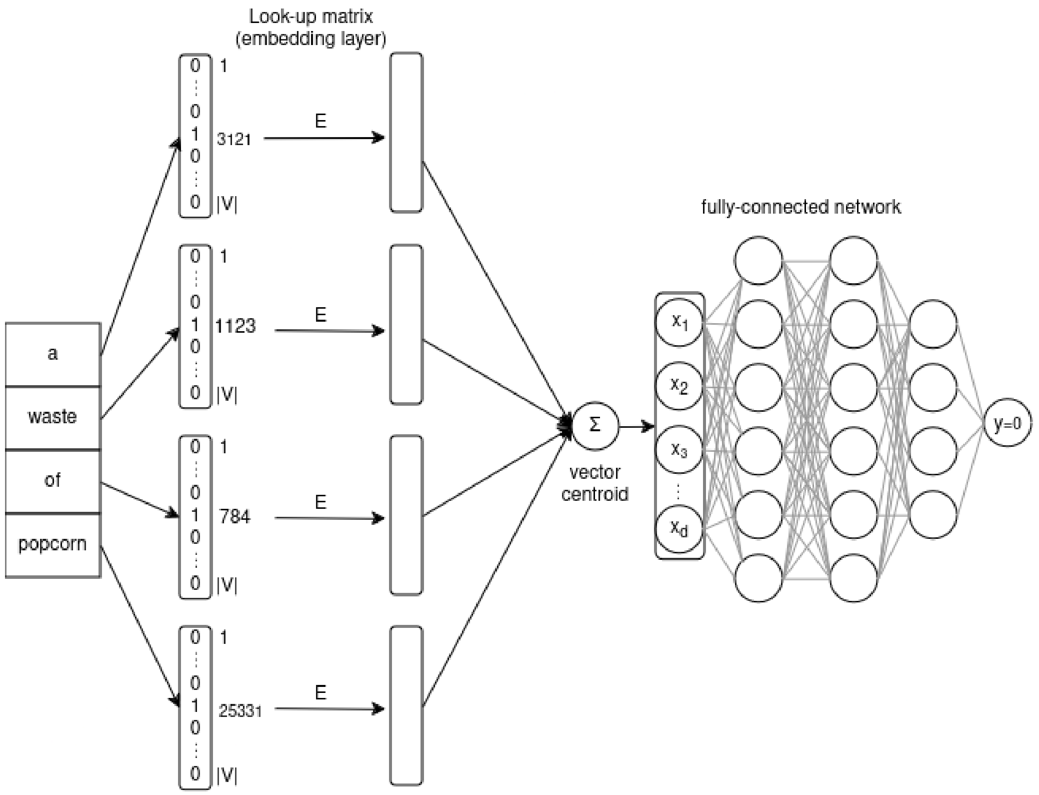


Figure 7 – Example of a sentence as input of a multilayer perceptron for sentiment analysis. The centroid of the word embeddings (that were previously obtained through the look-up matrix) is used as the network input.

Two fundamental processes of neural networks are forward and backward passes, where the latter is used in the learning process of the network. The network **forward pass** is to obtain the computation of a given input. It is the whole network function invocation for a given input. Every neuron in the network is activated, from the first layer to the output. The **backward pass**, performed for network training, consists of the numerical partial derivatives of the loss function with respect to the network weights. The gradients are calculated in the backward direction of the network, i.e. from the output to the input, using the chain rule for derivatives between layers since each layer is in fact a function composition. This is the essence of **backpropagation**.

A main design decision for network training is the **loss function**. It gives a measure of how well the network performs, comparing the predicted output to an expected one based on a dataset. There are many common loss functions, such as, mean squared error (MSE), mean absolute error (MAE), and cross-entropy, among others.

In practice, to train the network is common to use a mini-batched stochastic gradient descent alternative. This is to split the training set in batches and train the network performing passes, batch by batch, over the entire training set. In addition, instead of direct gradient descent, some alternatives, based on the network gradient, have been developed, such as Adam or RMSprop, among others, throwing better results in some situations. All these design decisions in the network definition and training process are treated as model hyperparameters, and finding a hyperparameter configuration as better as possible often involves time and compute-consuming searching processes.

As example, figure 7 shows a possible use of an MLP for sentiment analysis. Word vectors obtained from the look-up matrix are summed and divided by the input length to obtain the vector centroid, which is the input of an MLP to compute the sentiment value (positive or negative) of the input. Is interesting to note that when taking the centroid of vectors of the input, the order of the sentence is lost due to the commutativity of the vector sum. While this is a problem when dealing with texts because meaning can change by altering the order, using the centroid, despite not considering the order, often gives fairly good results, depending on the task.

3.1.2 Convolutional, Recurrent and Transformer Networks

Other architectures of neural networks have been designed. The layered structure of the network remains in all architectures. In fact, the different architectures often can be modeled as different types of layers: such as recurrent and convolutional layers.

Convolutional Neural Networks (CNNs) were first designed for images and then ported to language and other modalities of inputs. A convolutional network is a network that contains at least one convolutional layer, this is a layer that applies a predefined number of learnable convolution filters over the input, instead of matrix multiplication. For example, a 2d convolutional layer receives a matrix (e.g. an image) and applies K filters over the input, each of these filters is made of learnable weights.

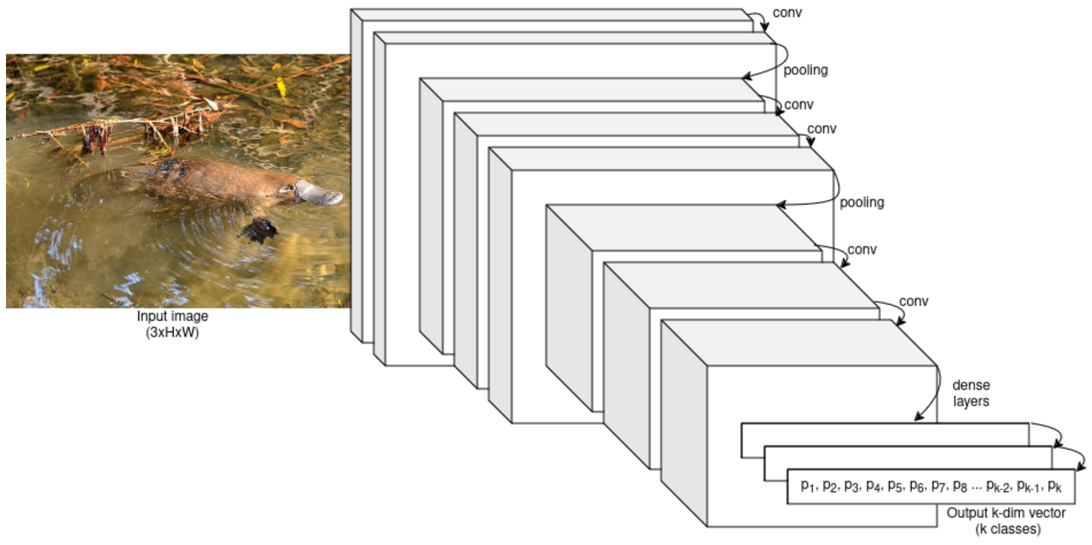

conv
pooling
conv
conv
pooling
conv
conv
dense layers
Input image (3xHxW)
p1, p2, p3, p4, p5, p6, p7, p8 ... pk-2, pk-1, pk
Output k-dim vector (k classes)

Figure 8 – Convolutional neural network indicating the convolutional, pooling, and dense layers[1].

Convolutional layers process input elements by breaking them down into context-aware features. After a convolutional layer, many samples of the inputs are obtained, one per each filter; and it is common to use a pooling layer to reduce the output dimension after one or more convolutional layers (see figure 8), improving model translation invariance and computational performance. The final layers in the networks are commonly fully connected layers, especially in image classification problems.

Recurrent Neural Networks (RNNs) have been designed to process inputs sequentially. A recurrent layer works in the following way. The network is applied on each element of the sequence, using the output of the recurrent layer (known as recurrent hidden state) as part of the input (along with the next element of the sequence) in the next invocation. This exhibits a sort of temporal memory when processing the sequence. Figure 9 shows a diagram of Elman Recurrent Layer, also referred to as vanilla recurrent neural network. It shows how the inputs of the sequence and the hidden states are used as input to obtain the next state. Figure 10 diagrams the unfolding of a recurrent layer, showing how the sequence is processed and the hidden state is passed through the invocations.

On a language input, if a RNN consumes token by token in order, in the hidden state of each token it is expected to have a sort of memory of the previous invocations (i.e. the previous or left context). This is of main importance on most language tasks, including language modelling as it is shown in the next section.

Recurrent layers, as well as fully connected layers and other types of layers, are stacked and combined with other types of layers, using layers where the output and input of the stacked layers are compatible. A network with more than one recurrent layer is

[1] Platypus image was obtained from `https://commons.wikimedia.org/w/index.php?curid=32551315`

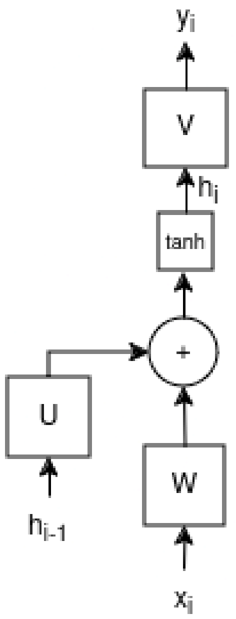


Figure 9 – Elman recurrent neural network diagram with *tanh* non-linearity to be applied to an input sequence. A recurrent hidden state h_i is obtained on each invocation and used as input on next invocation, starting from an initial h_0 state. Each h_i state if multiplied by the matrix V to obtain the output y_i for each step. The outputs y_i are potentially passed by other non-linearities to be used on a variety of tasks such as part-of-speech tagging or language modelling through next token prediction.

known as mulitlayered or stacked RNN.

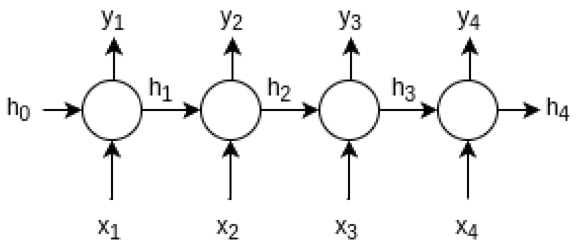


Figure 10 – Unfolding of an elman recurrent neural network. Each token x_i is processed consecutively. The (recurrent) hidden state h_i obtained in the current step i is used as input in the next step $i+1$, starting from a initial vector h_0 to process the first element x_1.

In addition, since RNNs initially consider only one side of the context on each token invocation (for example left if the network is invoked from left to right), **bidirectional RNNs** have been developed to consider both contexts. A bidirectional recurrent layer consists of two recurrent layers, one left-to-right and the other one right-to-left, with their outputs concatenated at each step. Thus one of the states gives information from the left context and the other from the right one. This state composed of the concatenation of the two recurrent layers is the output of the bidirectional recurrent layer. These bidirectional layers can also be stacked, taking into consideration that the layers used for each direction should be applied in tandem. In other words, a sub-network left-to-right application, then sub-network right-to-left application, and then the concatenation of the outputs of each direction on the bidirectional layers.

A major weakness of RNNs is that context memory tends to decline rapidly, performing poorly when the considered task needs to take into account tokens that are far away of the one that is being processed (i.e. distant contexts). This occurs because of the **gradient vanishing problem**. For example, if the final RNNs state is used to classify a whole sequence, it may perform poorly on inputs whose classification depend on tokens that are not near the end of the input, or in general with long term dependencies. Some RNN variants have been designed to work better with long term dependencies, the most popular ones are the Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] and the Gated Recurrent Unit (GRU) [Cho et al., 2014].

Other techniques to improve the processing of long term dependencies consist of using more than one of the previous states, instead of only the previous one consecutively. A popular one is to consider a weighted sum of the previous states, whose weights, that are learned through the training process, are obtained for each step, attempting to take into account which states are more relevant on each case. This is known as an **attention mechanism** [Bahdanau et al., 2014], and the weights on each step are commonly called attention weights.

Finally, the **Transformer** model was introduced in [Vaswani et al., 2017]. It is presented as a seq2seq model using purely an attention mechanism named multi-head self-attention, without using any recurrent or convolutional layer, enhancing the language capabilities currently seen in the most sophisticated models. Figure 11 diagrams the essence of the self-attention mechanism. Each word vector is transformed into 3 vectors that are intermediate steps to compute the attention matrix and layer output.

The Transformer has shown better results than recurrent and convolutional neural networks. In addition, Transformers, more precisely self-attention, allows the processing of all the tokens of the input concurrently, in contrast to recurrent networks, which need to process them sequentially.

3.1.3 Neural Language models

A language model is a probability distribution over a sequence of words, computed based on a text corpus. Language models have two main uses in natural language processing tasks: (1) to choose between different text candidates (e.g. choose between speech recognition output candidates), and (2) to predict the following word in a sequence of words, and thus can be used for text generation.

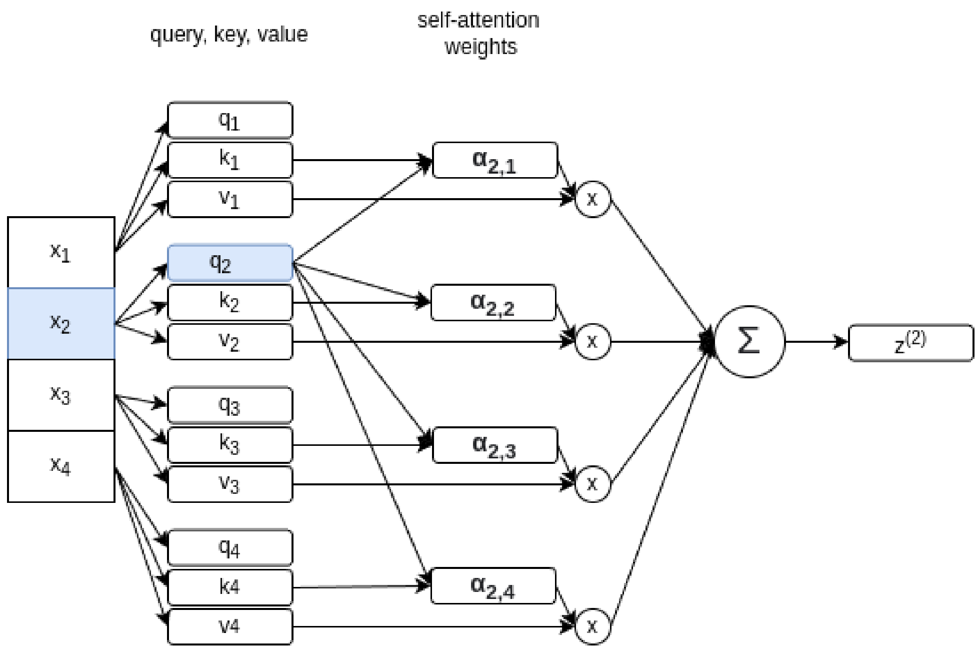


Figure 11 – Self-attention mechanism, the core module of the Transformer model.

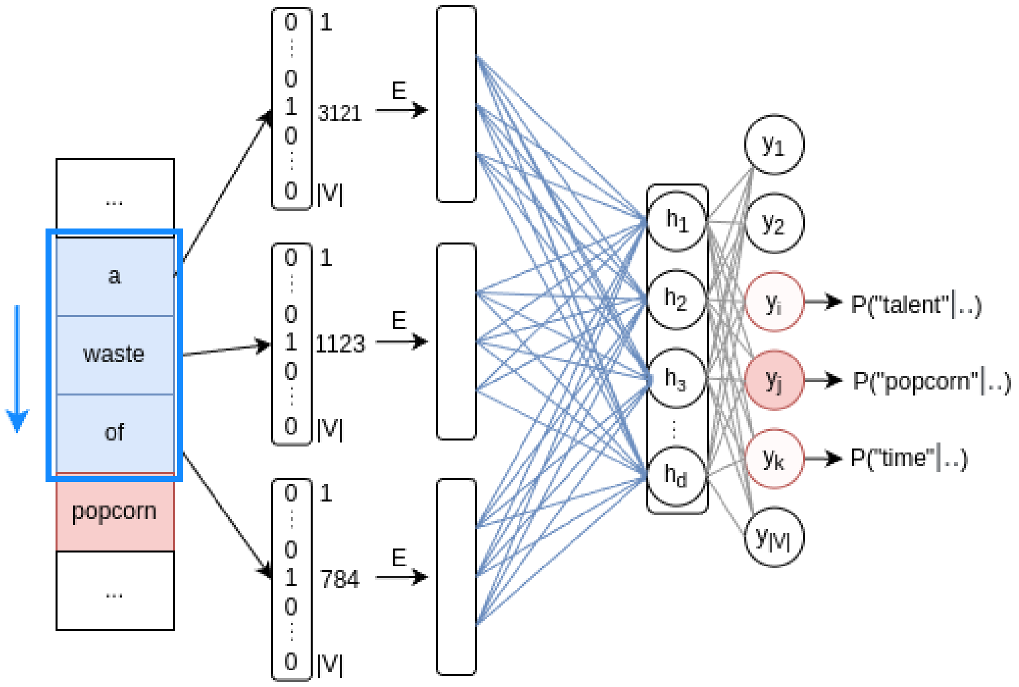


Figure 12 – Neural language model with three words of context, using a fully connected network and an embedding layer for words. This diagram is based on [Jurafsky and Martin, 2021].

Prior implementations of language models used n-grams (i.e. sequences of n words in a text), allowing to compute probabilities under the assumption that the probability of a word in a sequence depends only on the previous n-1 words (i.e. $P(w_i|w_1...w_{i-1}) = P(w_i|w_{i-n+1}...w_{i-1})$, and so $P(w_1w_2...w_l) = \prod P(w_i|w_{i-n+1}...w_{i-1})$). Adjusting n to a proper value and including some smoothing techniques to alleviate nonexistent word sequences interesting results can be obtained from n-gram language models with a fairly low computational cost.

At present time, language models have been being implemented using neural networks models structured to process sequential inputs. These models commonly uses a look-up table, often called embedding layer, for the vocabulary of words (or tokens). This table maps each token to a vector whose weights are fit during the training process (see figure 12). This gives as result a decontextualized vector representation of the words.

The usage of neural networks for language models, often referred as neural language models, has been showing interesting results since a couple of decades. [Bengio et al., 2000] used fully connected neural networks with a few words of context in large-medium size corpora. This network computes a distribution probability of the next word, over a vocabulary, given the input context; and each word vector is used as input through an embedding layer. Later, unified networks for multi-task learning, over a set of NLP tasks (where language model is included), have also been used, showing the usefulness of the learned representations in several tasks jointly [Collobert et al., 2011]. Recurrent networks were also used as a neural language model [Mikolov et al., 2010], performing the context consideration through the recurrent state. More recently the Transformer model has been introduced [Vaswani et al., 2017]. This model allows to compute the entire input simultaneously, unlike recurrent networks that need to do it sequentially; and throws much better results, performing better context treatment. The Transformer model, along with improvements in computational power and lots of available data, made possible the creation of the Large Language Models (LLMs), presenting an impressive jump in the field, in word and text representation, and in generative capabilities.

3.2 Skip-gram with negative sampling

Word embeddings are pretended to be built using corpora as big as possible, so efficiency is key in the model design. Skip-gram and CBOW (Continuous Bag of Words) from word2vec [Mikolov et al., 2013a] keep this in mind. In particular, skip-gram with negative sampling [Mikolov et al., 2013b] formulates a binary classification to discriminate context words from noise.

Skip-gram with negative sampling is defined using two embedding matrices W and C, for target and context words, respectively. The probability of a context word c to be in the context of a target word w is defined as

$$p(+|w,c) = \sigma(w.c) = \frac{1}{1 + exp(-c.w)}$$

So, the probability that a word c is not a possible context of a target word w (i.e. a negative example) is given by:

$$p(-|w,c) = 1 - p(+|w,c) = \sigma(-w.c) = \frac{1}{1 + exp(c.w)}$$

The positive examples, i.e. pairs (w,c) where c is in the context of w, are obtained by sliding a window over the training corpora, and negative examples are obtained through sampling random words.

For a target word w with a context word w_c and k noise words w_{n1}, ... ,w_{nk} (k is a hyperparameter), the loss function of skip-gram with negative sampling is defined by

$$\begin{aligned} L_{SGNS} &= -log(p(+|w,w_c) \prod_{i=1}^{k} p(-|w,w_{ni})) \\ &= -log(\sigma(w.c)) - \sum_{i=1}^{k} log(\sigma(-w.c)) \end{aligned}$$

Minimizing this loss means minimizing the dot product between target and context words and maximizing it for k randomly sampled non-context words. This loss is minimized using stochastic gradient descent for every pair (word, context) in the corpus, according to the chosen window.

In skip-gram with negative sampling, as in other methods that slide a window over the training corpus to obtain context words, the size of the window is a hyperparameter of the method and different sizes take to different results. Shorter windows (e.g. 2 or 3) produce "more syntactic" vectors, in the sense that give good results in syntactic word analogies, such as detecting words' inflections or derivations, but give worse results in semantic ones such as detecting capital city or the currency of a country [Mikolov et al., 2013c].

3.3 Subword Information: fastText

Word composition or morphology, often mentioned as subword information, is used by some approaches to improve the word vectors obtained from text. It becomes useful in the process of building word vectors, thus taking advantage of word derivations and inflections and also that some suffixes and prefixes may refer to some parts of the meaning of the word (e.g. -ing for gerunds or in- for negation).

Considering subword information is mainly useful for low-frequency and out-of-vocabulary words. Notice that, unlike the methods that do not use subword information, a representation for out-of-vocabulary words, can be obtained from its subword vectors.

fastText [Bojanowski et al., 2017] proposes to enhance word2vec (skip-gram and CBOW) by considering vectors for n-grams of characters and formulating the representations of the words as the sum of their n-grams vectors. For example, the word *<does>* (<

and > are used to delimit the beginning and the end of the word) and considering 3-grams will be represented as the sum of the vectors for <*do*, *doe*, *oes*, and *es*>, in addition to the vector of the whole sequence <*does*> (see figure 13).

3.4 Word Embeddings and Lexical Semantics

Since word embeddings become popular and efficient methods such as skip-gram with negative sampling have been developed, a lot of research has been done in the intersection of lexical relations and word embeddings [Vylomova et al., 2016, Ustalov et al., 2017, Vulić and Mrkšić, 2017, Nguyen, 2018, Jadhav et al., 2020].

Lexical relations such as antonymy, synonymy, or hypernymy are not directly expressed in word embeddings (although there are some relations such as gender alternation captured in the space structure without supervision [Mikolov et al., 2013c]). However, they are an excellent tool for lexical relation detection tasks. Antonymy, synonymy, and hypernymy detection will be taken up further in this document. In section 5, hypernymy detection will be detailed, and section 6 is dedicated to synonymy and antonymy detection.

In addition to lexical relation detection using word embeddings, some work has been done in the direction of considering lexical relations as constraints to enhance the representations or take some semantic relation into consideration in the embedding space (e.g. push antonyms and attract synonyms). Some examples can be to retrofit them using a semantic lexicon [Faruqui et al., 2014], specialize them for similarity or relatedness [Kiela et al., 2015], or integrating synonymy and antonymy information such as in dLCE embeddings [Nguyen et al., 2016a].

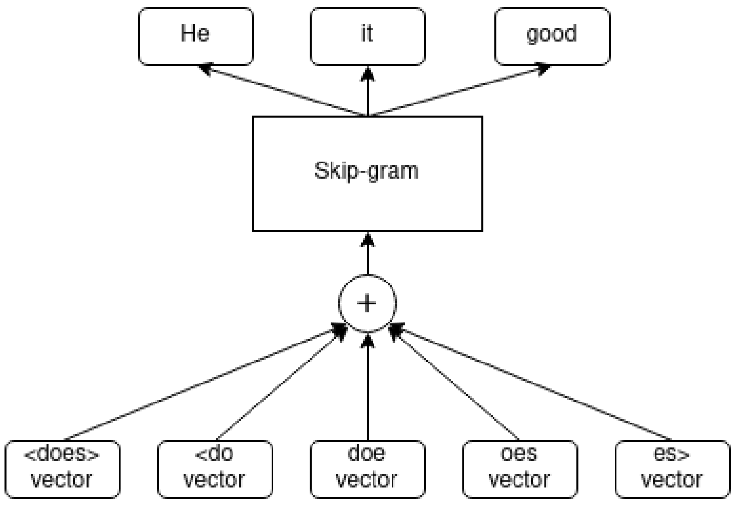


Figure 13 – FastText diagram using 3-gram characters for subwords with skip-gram and a context window size of two words. The example illustrates the target word *does* in the example *He does it good.*

4 Deep Metric Learning

Deep Metric Learning (DML) is the name given to a family of approaches that uses neural networks to learn distance or similarity functions between elements of one or more domains. Usually, a DML model consists of one or more neural networks to transform the inputs into vector of a space where a metric has some semantic meaning, driven through a loss function. DML models are trained using data annotated as similar and dissimilar, attracting similar pairs and pushing away dissimilar ones in the transformed space. To name some examples, DML can be used in speaker identification, to get same-speaker utterances nearer than others, or in paraphrase detection, to cluster same meaning sentences. The notion of similarity to be learned is given through the data and the loss function. For example, in a DML approach to face recognition, same-person images would be considered as similar pairs, while if the task is gender identification, similar pairs would correspond to images of the same gender people.

DML models attempt to learn a similarity or distance function between elements in a given domain. In the computer vision field, a deep metric learning approach would be useful to reflect high-level features on images, such as pictures of animals of the same species, instead of direct pixel comparisons. In natural language processing, DML can be used to detect synonymy or paraphrases, taking as input word vectors, that may present related words as near (e.g. *lion* and *mane*), making it difficult to distinguish synonyms from other related words. The DML model may attempt to learn through examples how to encode the word vectors to make synonyms closer than other related words. In a DML approach for paraphrase detection, a neural network model that allows to process a input sequence (such as a transformer or recurrent neural network) may be used to encode each sentence into a vector. This model is trained so that the resulting vectors are closer in case of paraphrasing than otherwise. This is trained usually using a dataset consisting of pairs of sentences indicating if they are paraphrases or not.

DML has been used in a broad range of applications on multiple domains and modalities of data. They can be used in a supervised task where positive and negative pairs are available, sometimes called weakly supervised classification, or they can be used in a classification task with a fixed number of predefined classes and a dataset of elements indicating to which class they belong. This sometimes is called strong supervision or simply supervised classification. This thesis uses DML for relationship learning, particularly semantic relations between words vectors, so, they correspond to the first case: models trained on positive and negative pairs, that is, related and unrelated words, respectively.

This thesis focuses mostly on two of the most popular deep metric learning models:

siamese and triplet networks. This section presents the siamese and triplet networks, followed by some important aspects of deep metric learning concerning tuple mining and few-shot learning.

4.1 Deep Metric Models

Deep metric learning attempts to encode a domain of elements into a metric space to reflect a semantic feature of the data. For example, in a speaker recognition dataset, a successful DML approach would tend to cluster the speech segments according to its owners. In general terms, it may bring positive (or similar) pairs closer and put the negative (or dissimilar) pairs further away. Notice that, if the DML proposed approach fits well on the data used, this can lead to form tight clusters notably separated between them.

Siamese and triplet networks are two highly used DML models. They share the common goal of using neural networks to encode inputs into a metric space, defining their loss functions based on the distances between similar and dissimilar pairs of inputs. They differ in how they handle the inputs, defining different constraints on the distances between the elements being encoded. The siamese network processes pairs of elements and is trained using the contrastive loss function. On the other hand, the triplet network processes tuples of three elements (known as triplets) and is trained using the triplet loss function, a loss function that relates the distances between these three elements. In any case, both approaches get similar pairs closer and dissimilar pairs farther apart (see figure 14).

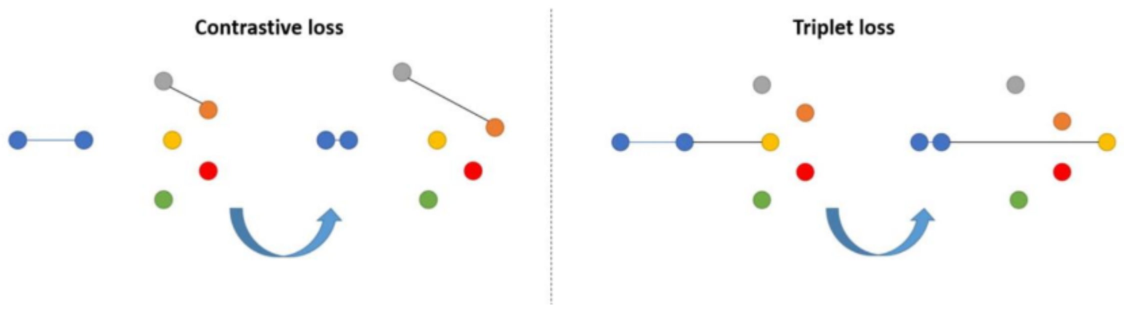


Figure 14 – Siamese and triplet networks behaviors. The color of the points indicates if they are similar or dissimilar. Image from [Medela and Picón, 2019].

In what follows in this section, each of these two models is described. Their general idea and a possible formulation for each case are discussed, although there may be similar formulations or variations.

4.1.1 Siamese Network

A siamese network is a model that receives two inputs, a neural network is applied to each input, and the distance between both outputs is measured and returned as the output of the model (see figure 15). It can be interpreted as a trainable distance function that can be used on a range of tasks on different domains and modalities of data.

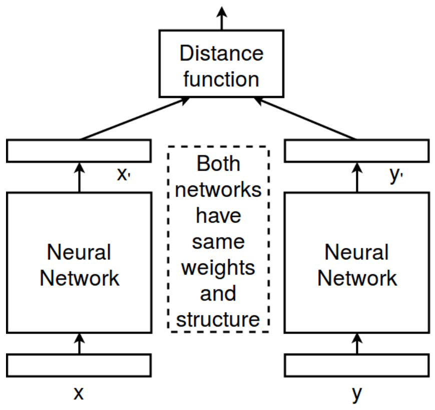


Figure 15 – A diagram of a siamese network model.

The siamese network is trained through the contrastive loss function, relying on similar and dissimilar pairs, decreasing the distance of similar pairs and increasing the distance of dissimilar ones.

The contrastive loss function can be written as

$$L = \sum_{(x,y)\in P} max\{0, D(x,y) - \alpha\} + \sum_{(x',y')\in N} max\{0, \beta - D(x',y')\},$$

where $D(x,y) = d(F_\theta(x),F_\theta(y))$, the encoder $F_\theta : D \rightarrow \mathbb{R}^n$ is a neural network of parameters θ, applied on both inputs in tandem (that is why it is called siamese network), with $d : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^+$ a vector distance function, such as $d = ||x - y||$. P and N are similar and dissimilar pairs of the training data. The hyperparametrs α and β are the thresholds for similar (P) and dissimilar (N) examples, respectively, pretending to make the encoder network $F_\theta(x)$ to locate in the same cluster elements that belong to a pair in P. It is a common practice to use D^2 instead of D because of its differentiation benefits, so, a common formulation for the loss function of a siamese network is

$$\begin{aligned} L_{contrastive} = &\sum_{(x,y)\in P} max\{0, ||F_\theta(x) - F_\theta(y)||^2 - \alpha\} + \\ &\sum_{(x',y')\in N} max\{0, \beta - ||F_\theta(x') - F_\theta(y')||^2\} \end{aligned}$$

Siamese networks have been used in a variety of tasks and fields, such as sentence similarity [Chi and Zhang, 2018], speaker identification [Wang et al., 2019], palmprint recognition [Zhong et al., 2018], and object tracking [Bertinetto et al., 2016], among very many others.

4.1.2 Triplet Network

The triplet network [Hoffer and Ailon, 2015] is a deep metric learning model that as the siamese network is based on similar and dissimilar inputs, but in this case the elements are arranged in triplets. It considers triplets (a,p,n) where the element a is similar to p and dissimilar to n. Each element of the triplet is encoded using the same neural network whose outputs are combined using the triplet loss function (see figure 14).

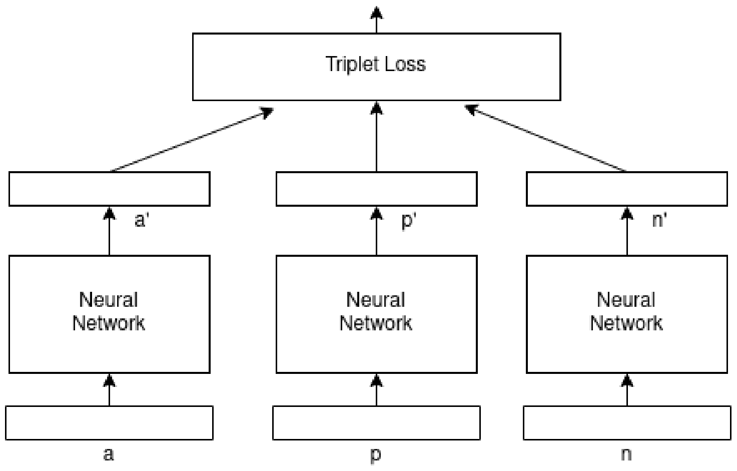


Figure 16 – Diagram of the triplet network model. The same neural network is applied three times before the triplet loss function is applied.

The triplet loss function attempts to make the distance between a and p smaller than the distance between a and n. It can be written as:

$$L_{tripl} = \sum_{(a,p,n)\in T} max\{0, D(a,p) - D(a,n) + \mu\},$$

where $D(x,y) = d(F_\theta(x),F_\theta(y))$ as in the siamese network, with d a distance function, μ is the separation margin between positive and negative pairs, and T is the training set conformed by triplets.

Triplet network is claimed as presenting the advantage of making better use of relative distances in comparison to the siamese network, guessed by the margins considerations in both models. However, triplet networks need triplet mining of data, instead of the pair mining needed by the siamese network. Triplet mining may be more challenging because of a larger input space, however, it may also lead to a more beneficial situations during training, because of having on each training input two pairs with a common element to rely on.

4.2 DML for Classification

Deep metric learning can be used on any classification task. In a classification scenario, similar pairs correspond to same-class elements and dissimilar pairs are elements of different classes.

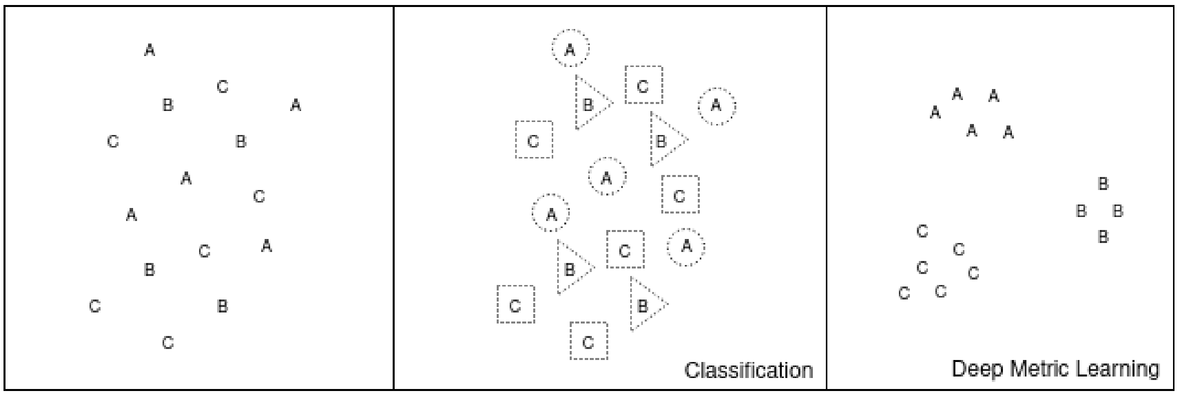


Figure 17 – Deep metric learning and classification. In the classification part, the elements enclosed in a circle, triangle, and square denote that are classified as A, B, and C, respectively.

The DML model will attempt cluster elements by classes in the output space (see figure 17). In contrast to other techniques such as using softmax for classification or multiple binary classifiers, deep metric learning can be used without predefined number of classes, where classes that have not been seen during training can be added for testing. Authorship attribution and other recognition tasks such as face, voice, and fingerprint recognition [Neculoiu et al., 2016] are possible applications.

It is a desirable property on a classification task addressed by clustering to form tight clusters fairly separated between them. The distance between elements of the same or different classes is often named intra-class and inter-class distances, respectively. Notice that the loss functions in deep metric learning models such as siamese and triplet networks, attempt to decrease intra-class and increase inter-class distances, tending to form tight and separated clusters.

It is interesting to observe that the softmax function, commonly used as the output activation function of neural network in classification tasks, can be described from a deep metric learning perspective. Geometrically, the last layer of the network just before the softmax function, is a set of vectors that can be interpreted as class representatives: one vector is assigned to each class [Deng et al., 2019]. During training, through each input with its corresponding class and the loss function, the softmax function fits the class representatives and the layer before it, to decrease the distance of the encoded input to the correct class representative and increase it to the others.

4.3 Tuple Mining

In deep metric learning, it is necessary to build tuples (pairs or triplets) to train and evaluate the models. The number of possible tuples can be overwhelming. The policy of what tuples are considered can accelerate the training process, and improve the final performance of the model. The process of building and selecting which tuples use to train the model is called tuple mining.

Tuple mining becomes completely necessary when tackling a classification problem with deep metric learning. For example, if we consider a dataset with 3 classes with N_1, N_2 and N_3 elements, from a classification perspective using softmax, there are available $N_1+N_2+N_3$ elements, each annotated with one of the 3 classes. If this dataset is pretended to be used on a deep metric model,for example to train a siamese network, it is necessary to build positive and negative pairs, i.e. pairs of elements that belong to the same class and pairs of elements that belong to different classes. The initial 3-class classification problem becomes a binary classification over pairs of elements (same-class vs no-same-class). The initial dataset of $N_1 + N_2 + N_3$ elements becomes $N_1^2 + N_2^2 + N_3^2$ positive pairs (counting reflexive pairs), and $2 \times (N_1 \times N_2 + N_1 \times N_3 + N_2 \times N_3)$ negative pairs available to be used. This number can be considerably big, depending on the number of classes and elements in each class. Therefore, the success of the approach may depend in the choice of an appropriate tuple mining policy.

Hard example mining refers to give to the model during training examples that is failing to predict correctly (e.g. a positive pair whose embeddings are farther apart than the margin). This is to give to the model false positives (hard negative mining) or false negatives (hard positive mining) since it may benefit from learning to discern them correctly. It is important to notice that even with good metric models and architectures, the learning ability of the network is limited to the samples presented during training [Kaya and Bilge, 2019].

Not all examples are equally significant. It is possible to give to the model the examples that are being the harder to classify than others (e.g. false negatives whose elements are further away). Finding out which tuples are the most significant (i.e. the hardest) is a current research area [Vasudeva et al., 2021]. It is needed to take into account, in addition to the effectiveness of the tuple mining policy, the computational cost to implement it. In this direction, some approaches aim to have hard examples that are not necessarily the hardest, this is known as semi-hard tuple mining.

Negative tuple mining is particularly important in many tasks. In general it is a considerably larger space than positive tuples. Having good negative tuple mining is crucial in the proper performance of the deep metric learning model in many tasks.

4.4 Zero-shot Learning

Deep metric learning models have been used in zero, one and few-shot learning scenarios (i.e. learn using zero, one or few examples of a class). One-shot and few-shot learning can be easily performed by humans (e.g. we can recognize an animal species with only one example) but this is challenging for a machine learning algorithm. Regarding zero-shot learning, in a classification carried out with softmax, the classes for which no elements are available during training will not be even represented in the network structure. Deep metric learning models do not leave classes fixed by the structural design of the network as in softmax. This crucial for zero-shot learning, allowing the model to generalize to new classes without the need for additional labeled data.

Zero-shot, as well as one and few-shot, are important in the field of machine learning. Commonly, in a classification problem it is expensive to have annotated data for all classes, especially in tasks with a very large number of classes. Considering a model with good zero-shot and few-shot learning capabilities contributes to better data utilization and generalization [Xu et al., 2019]. Having zero-shot capabilities allows to a new range of applications where other conditions of data are almost impracticable (e.g. open-world image classification, brain-computer interfaces, etc.) [McCartney et al., 2022].

5 Hypernymy Detection[1]

Hypernymy, as commented in section 2.3.3, refers to the general-specific relationship between two words. This is the case of biology taxonomies (e.g vertebrate-mammal, mammal-pangolin), tools and (e.g. tool-hammer) colors (e.g. color-green), professions (e.g. composer-Lennon)[2], among very many others.

In NLP, hypernymy detection can be useful for tasks such as question answering [Clark et al., 2007], textual entailment [Chen et al., 2017], among others, and contributes to the understanding of distributional semantics techniques and capabilities. Hypernymy detection can be treated as a supervised or unsupervised task. Although most of attention in NLP research has focused on supervised approaches, it is worth mentioning that there are unsupervised approaches with interesting results. While supervised approaches attempt to generalize the relationship using example pairs, in addition to corpus based information such as textual patterns or word embeddings, unsupervised approaches do not use any example of the relationship, relying purely on corpus statistics such as the distributional inclusion hypothesis [Zhitomirsky-Geffet and Dagan, 2005] or entropy-based measures [Santus et al., 2014].

In this thesis, supervised hypernymy detection is addressed using order embeddings to detect hypernymy through word embeddings [Etcheverry and Wonsever, 2020]. In addition to experiments performed for English, the task is also considered in Spanish [Lee et al., 2020]. This has been performed by making a dataset using WordNet[3] and corpus statistics.

5.1 Pattern-based and Distributional Approaches

Hypernymy detection has been addressed using two types of information from a corpus: (1) paths and (2) context distributions. Path-based (or pattern-based) approaches detect hypernyms using the paths of words that connect two words in the same sentence. For example, the path “is a type of” would match cases like “tuna is a type of fish” allowing us to detect that “tuna” is a hyponym of “fish”. [Hearst, 1992] presented the first path-based approach, where hand-crafted patterns were used for hypernymy extraction. Also, paths of joint occurrences in syntactic dependency trees result useful for hypernymy

[1] The results of this chapter were presented in [Etcheverry and Wonsever, 2020] and [Lee et al., 2020].

[2] In strictly linguistic terms, *Lennon* is not a hyponym for *composer*, it is an instance. In this thesis a broader notion for the hypernymy relation is used, including instances of a class as its hyponyms.

[3] WordNet [Miller, 1995] is a hand-tailored well-known resource. It is a large database with lexical relations, including hypernymy among them. It was originally created for the English language and later other languages have been included through scheme transfer and translations.

detection [Snow et al., 2004]. Patterns can be improved generalizing them using part-of-speech tags and ontology types by [Nakashole et al., 2012]. Another kind of pattern-based approach is proposed in the work of [Navigli and Velardi, 2010], they consider word lattices to extract definitional sentences in texts and then extract hypernymy related pairs from them. This work is later extended to learning lexical taxonomies [Navigli et al., 2011]. The main disadvantage of path-based approaches is that can suffer of low recall since, to be detected as related, both candidates must occur simultaneously in a same context in the corpus.

Distributional approaches for supervised hypernymy detection rely on the contexts of each word independently, instead of paths connecting joint occurrences. Some works propose a supervised classification after applying a binary vector operation to the pair of word embeddings, such as vector concatenation [Baroni et al., 2012] and difference [Roller et al., 2014, Fu et al., 2014, Weeds et al., 2014]. A wider set of lexical relations was studied by [Vylomova et al., 2016] using vector difference, they remarked on the importance of negative training data to improve the results. [Ustalov et al., 2017] performed hypernyms extraction based on projection learning. Instead of classifying the pair of representations, they learned a mapping to project hyponyms embeddings to their respective hypernyms, remarking also the importance of negative sampling. A related approach is presented by [Dash et al., 2020], where a neural network architecture is designed to enforce asymmetry and transitivity through non-linearities and residual connection. These last two approaches present some overlap with the work of [Vendrov et al., 2015b], whose order embedding approach is the one considered in this work.

[Shwartz et al., 2016a] combined path-based and distributional information in supervised hypernymy detection, concatenating the embedding of each term independently with a distributional representation of all paths between the terms in a dependency-parsed corpus. The representation was built with the average of the LSTM resulting representation of each path. Additionally, they introduced a dataset for lexical entailment where they tested their model. LEAR (Lexical Entailment Attract-Repel) [Vulic and Mrksic, 2017] obtains great performance on hypernymy detection specializing the word embeddings using WordNet constraints, where the direction of the hypernymic relation is encoded in the resulting vector norms and the cosine distance enforces the synonym's semantic similarity. The resulting vectors were specialized simultaneously for lexical relatedness and entailment.

5.2 Order Embeddings for Hypernymy Detection

An order embedding is a function $f : X \to Y$ that preserves and reflects the orders between two partially ordered sets $(X, \preceq_X)$ and $(Y, \preceq_Y)$.

In other words, the function $f : (X, \preceq_X) \rightarrow (Y, \preceq_Y)$ is an order embedding if fulfills that $x_1 \preceq_X x_2$ if and only if $f(x_1) \preceq_Y f(x_2)$. In machine learning, [Vendrov et al., 2015a] introduces a method to attempt to learn an order embedding into $\Re^m_{\geq 0}$, through an order relation defined as the *reversed product order*. The reversed product order defines an order relation in $\Re^m_{\geq 0} \times \Re^m_{\geq 0}$ as:

$$x \preceq y \iff \bigwedge_{i=1}^{m} x_i \geq y_i, \tag{5.1}$$

where x,$y \in \Re^m_{\geq 0}$ and x_i and y_i correspond to the i-th component of x and y, respectively. It can be easily proved that $\preceq$ defined by reversed product order is reflexive, anti-symmetric and transitive, being $\vec{0}$ the top of the hierarchy. The proofs for reflexivity and transitivity come up directly form reflexivity and transitivity of $\geq$. Regarding the anti-symmetry proof, if x and y are such that $x \preceq y$ and $y \preceq x$, then necessarily $x = y$, since $x_i \geq y_i$ and $y_i \geq x_i$, and so $x_i = y_i$, for $i = 1...m$.

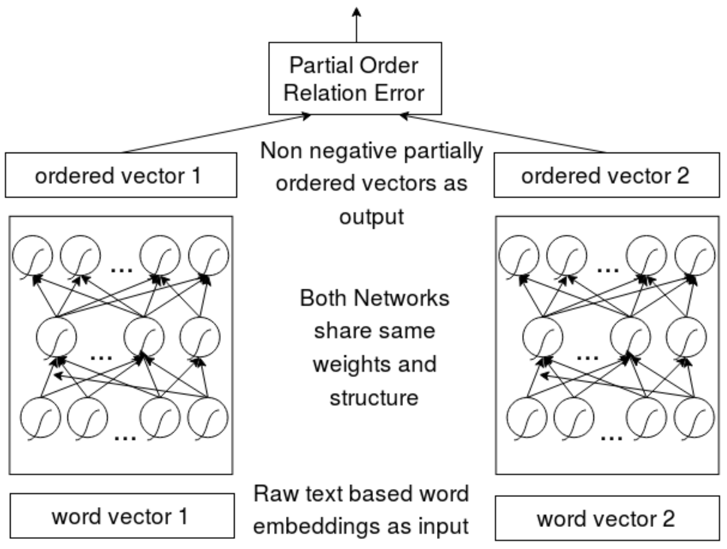

Partial Order Relation Error
ordered vector 1
Non negative partially ordered vectors as output
ordered vector 2
Both Networks share same weights and structure
word vector 1
Raw text based word embeddings as input
word vector 2

Figure 18 – Order embedding diagram.

The relation $(\preceq, \Re^m_{\geq 0})$ defined above can be expressed by

$$E_p(\vec{x},\vec{y}) = ||max(\vec{0}, \vec{y} - \vec{x})||^2, \tag{5.2}$$

where $\vec{x}$,$\vec{y} \in \Re^m_+$ and $max(..)$ is the element-wise maximum function between vectors. $E_p(x,y)$ can be interpreted as the degree to which x and y are not related, being $E_p(x,y) = 0$ if and only if $\vec{x} \preceq \vec{y}$.

Regarding unrelated pairs, E_p can be intended to be higher than a given threshold α for them through the hinge loss as follows:

$$E_n(\vec{x},\vec{y}) = max\{0, \alpha - E_p(\vec{x}, \vec{y})\}. \tag{5.3}$$

So, $E_n(\vec{x'},\vec{y'})$ is 0 when $E_p(\vec{x'},\vec{y'}) \geq \alpha$ and therefor $\vec{x'} \not\preceq \vec{y'}$.

Finally, the order embedding loss function consists on summing (5.2) and (5.3) for positive and negative examples, respectively. Minimizing the resulting loss minimizes E_p and E_n jointly and can be seen as an instance of the contrastive loss. It stands as follows:

$$L = \sum_{(x,y)\in P} E_p(\vec{x},\vec{y}) + \sum_{(x',y')\in N} E_n(\vec{x'},\vec{y'}), \tag{5.4}$$

where P and N are sets of positive and negative examples, respectively. Note that L is differentiable allowing to fit a neural network as an order embedding through gradient descent-based techniques, as diagrammed in figure 18. The order embedding consist on fitting a neural network to encode the inputs through the recently introduced loss function and positive and negative examples.

5.2.1 Experiments

In this section, the conducted experiments for English are detailed. These experiments were conducted on the dataset introduced in [Shwartz et al., 2016a], consisting of related and unrelated pairs of words, confeccionated from knowledge resources such as WordNet [Miller, 1995] and DBPedia [Auer et al., 2007]. This dataset provides two variants: lexical and random split, which are explained below.

5.2.1.1 Random and Lexical Dataset Split

As usual in supervised training, the dataset (positive and negative pairs) is split into the train, validation, and test partitions. [Shwartz et al., 2016a] considered two ways of splitting the dataset: random and lexical split. While the random split is performed randomly, without any consideration of the pairs contained on each subset, the lexical split avoids words in common between the subsets.

The concept of **lexical split** was introduced in [Levy et al., 2015], argued by the fact that hypernymy detection may suffer from lexical memorization phenomena. This occurs when instead of learning the hypernymy relationship between pairs, the model learns a specific word as a strong indicator independently of the other. For example, given positive pairs such as: (*cat*, *animal*), (*dog*, *animal*), (*horse*, *animal*), the model tends to learn the word *animal* as an indicator and given any pair (_, *animal*) classifies it as a positive pair. To improve evaluation, and take into account the phenomenon of lexical memorization, a lexical variant of the dataset is included, i.e. the training, validation, and test sets are split without words in common between any of the three subsets. So, if a pair occurs in one subset, its words will not occur in any pair of the other two subsets. This is done by selecting the pairs for each dataset depending on whether its words already exist in any of the three subsets or not. The pairs that have each of its word in different subsets are discarded.

In the other hand, the **random split** variant of the dataset is performed randomly without any considerations between its parts. This splitting process has the advantage of considering all the available pairs of the dataset without discarding anything, leading to a larger dataset.

The dataset size information is detailed in table 1.

	Pos	Neg	Total
Train Rand.	9,942	39,533	49,475
Valid Rand.	681	2,853	3,534
Test Rand.	3,512	14,158	17,670
Train Lex.	4,067	16,268	20,335
Valid Lex.	270	10,80	1,350
Test Lex.	1,322	5,288	6,610

Table 1 – Shwartz's dataset size for random (Rand.) and lexical (Lex.) splits.

5.2.1.2 Word Embeddings and Model Hyperparamters

The word embeddings used for these experiments was the publicly available FastText vectors trained on English Wikipedia [Joulin et al., 2016][4].

Regarding the model hyperparametrs, several structural and training configurations were tried, in a manual search approach. As activation functions it were considered ReLU, tanh, sigmoid, and SELU. Dropout regularization was used for ReLU, tanh and sigmoid; and alpha dropout for SELU, with a 0.2 drop probability and 0.1 for alpha dropout. SELUs were initialized using LeCun normal initialization and ReLU, tanha and sigmoid with Glorot uniform initializer [Glorot and Bengio, 2010].

The models were trained using Adam [Kingma and Ba, 2014] with a learning rate of 5×10^{-4} over mini-batches of 64 examples. Early stopping was used and model checkpoint to get the best model according the validation set. Positive instances were those where E_p is lesser than .02, using a margin for negative examples of 1.0. For this implementation Keras [Chollet et al., 2015] was used.

[4] `http://mattmahoney.net/dc/enwik9.zip`

5.2.1.3 Results

The models are evaluated using precision, recall, and F measures. The results are presented in table 2. The reported results correspond to three layered networks of 600, 400, and 200 units on each layer from input to output; and are the best obtained of three runs for each model against the validation set. For comparison, the results of the best distributional model reported by [Shwartz et al., 2016a] and HypeNET combined have been included. HypeNET combined consists of the distributional and path-based combined approach that achieves the best results in their work.

	P_{rand}	R_{rand}	F_{rand}	P_{lex}	R_{lex}	F_{lex}
Best Dist. [Shwartz et al., 2016b]	0.901	0.637	0.746	0.754	0.551	0.637
HypeNET Int [Shwartz et al., 2016b]	0.913	**0.890**	0.901	0.809	0.617	0.700
OrdEmb ReLU	0.936	0.876	**0.905**	**0.958**	0.615	0.749
OrdEmb SELU-ReLU	0.932	0.845	0.887	0.740	**0.872**	**0.801**
OrdEmb tanh-sigm	**0.967**	0.836	0.897	0.788	0.756	0.771

Table 2 – Order embedding results with different activation functions on Shwartz's English dataset, HypeNET Integrated, and the best distributional results reported by Shwartz are included for comparison.

Figure 19 shows the SELU-ReLU model accuracy evolution in lexical split for training and validation sets along the training process. It can be seen the joint progress of accuracy on train and validation sets, showing the performance of the model to generalize the hypernymy relation from training data to pairs that have not been seen during training.

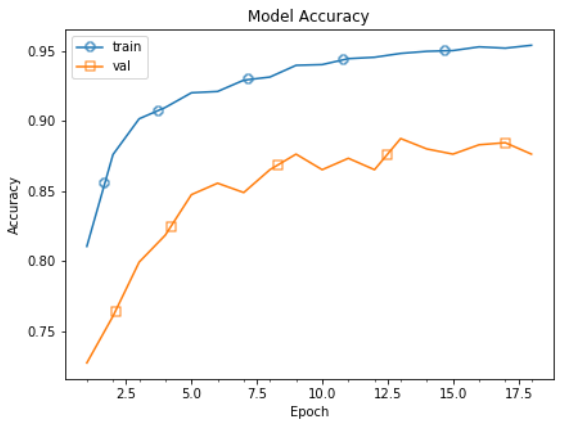


Figure 19 – Accuracy evolution on each epoch in the SELU-ReLU model.

Results on reduced training data

The performance of the order embedding model restricting the available training data is studied. Figure 20 shows the F measures obtained using a SELU-ReLU model trained on gradually increased sizes of training data. Interestingly, the results rapidly improves before the first 10% of the training data is considered, and very little progress is obtained after the first 30% of the data is used. This occurs in both, random and lexical splits, suggesting that the results are not due to particular characteristics of the way the dataset was increased. In both cases, lexical and random split, the dataset was increased randomly.

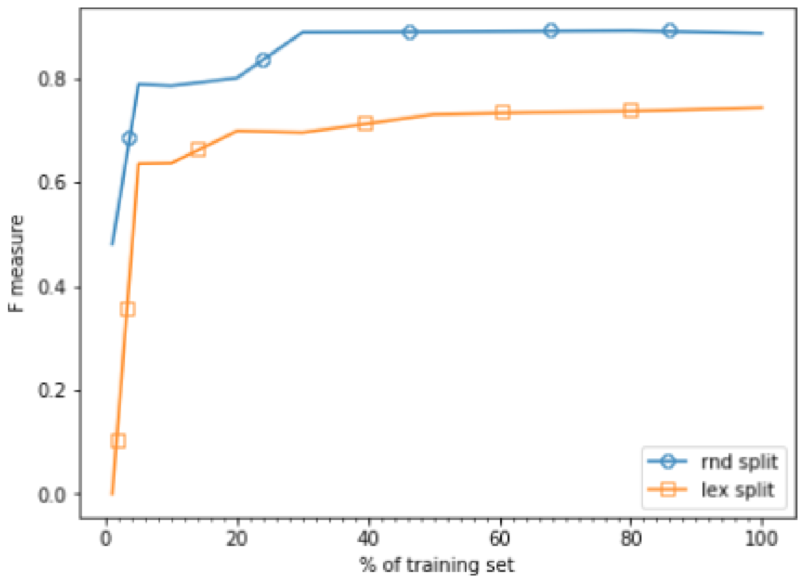


Figure 20 – F-score on test dataset partition of SELU-ReLU model on different sized random partitions of the training set. Each dot reports the results of an entire training loop.

The results detailing precision and recall are included in table 3.

	P_{rand}	R_{rand}	P_{lex}	R_{lex}
1%	0.507	0.458	0.000	0.000
5%	0.703	0.899	0.495	0.893
10%	0.694	0.904	0.495	0.895
20%	0.712	0.916	0.574	0.896
30%	0.909	0.869	0.583	0.862
50%	0.916	0.866	0.611	0.912
80%	0.925	0.862	0.630	0.889

Table 3 – SELU-ReLU model precision and recall on different training sizes.

Note that the model first achieves high coverage and then seems to refine its results. It can be seen in the precision improvement without a remarkable decrease in the recall.

5.2.1.4 Results Analysis

This section presents a brief analysis of the results obtained with the SELU-ReLU model on the lexical split dataset, one of the most interesting experiments previously performed according to the result obtained. The confusion matrix for this experiment can be seen in table 4. There were 169 hypernyms, from a total of 1322, that the model fail to detect according to its positive margin. Of these 169, although they were classified as false since they presented a value greater than 0.02 (i.e. the positive margin), 101 presented a value less than 1.0, i.e. the negative margin. This could be interpreted to mean that only 68 were reliably classified as false.

	False	True
False	4883	405
True	169	1153

Table 4 – Confusion matrix for the SELU-ReLU model in the lexical split.

Concerning the false negatives that the experiment throw, i.e. pairs labelled in the dataset as hypernyms that the model fails to predict correctly, it was noticed that many cases involve ambiguous terms. The table 5 presents a random sampling of the false negatives. In the samples presented, note for example that *stubbs* may refer to the surname of *William Stubbs* since it is presented as hyponym of *historian*, but it is ambiguous for example with the artist *George Stubbs.* The same stands for **sting**, which can be confused with the singer, and it is worth mentioning the case of the word *cosmos* which may be confused with the most known use of cosmos as the universe as a whole, instead of the cosmos flowering plant in the sunflower family.

Hyponym	Hypernym
building	structure
contentment	happiness
diver	swimmer
cosmos	flower
moment	present
stubbs	historian
sting	pain

Table 5 – False negatives random sampling. Each row correspond to a pair that the model predict as non-hypernym but are labelled as hypernym in the dataset.

The table 6 presents a random sampling of false positive, i.e. pairs that have been incorrectly predicted as related. These cases looks like errors made by the model, since it seems correctly labelled. It is interesting to note that excepting the case of *wjre-country*, the sampled cases present some relationship between the words, for example part-whole (i.e. meronymy), as in the case of *rice* and *sake.* It is interesting to note that in the case of *stump* as a hyponym of *tree*, although it is not a tree, it is the base of a tree that has fallen or that has been cut down.

Hyponym	Hypernym
stump	tree
rice	sake
tofee	sugar
nucleus	brain
wjre	country
edge	limb

Table 6 – False positives random sampling. Each row correspond to a pair that the model predict as hypernym but are labelled as non-hypernym in the dataset.

The table 7 presents a sampling of pairs that the model predicts as hypernyms, but although they are labelled as non-hyperonyms in the dataset, it seems that the label is not correct, or at least debatable. These examples principally correspond to occupations like writers or actors and creations like novels. However, there are other pairs that also seem to be incorrectly labeled. For example, terms like *abode* as hyponym of *place* and *beretta* as hyponym of *weapon* are predicted as related while in the data appear as negative examples. These seem to be wrong or debatable examples in the dataset, possibly as result of some error or automatic process, rather than mistakes made by the model. It is worth mentioning that the term *novel* is involved in 129 of the 405 false positives instances, some of these terms are incorrectly labeled in the dataset. For example, terms such as *pollyanna*, *aelita*, and *jaws*, are presented as non-hyponyms of *novel* when indeed they are hyponyms of *novel*, taking into account the criteria that the dataset has had for other instances.

Hyponym	Hypernym
voltaire	writer
bill mantlo	writer
fledgling	novel
summerland	novel
sathyaraj	actor
ferdinand de saussure	linguist
menecrates	sculptor
nicomedes	mathematician
kofax	software
encarta	encyclopedia
abode	place
beretta	weapon

Table 7 – Cases considered as false positives with a debatable label in the dataset.

6 Antonymy-Synonymy and Parasiamese Networks[1]

In this section, the antonymy-synonymy discrimination (ASD) task is introduced and it is addressed using the parasiemse network model. ASD consists of automatically detecting synonyms and antonyms from a list of pairs of words. This can be challenging for a machine because both cases refer to very related words, contrasting in only one dimension of its meanings on antonyms.

Many approaches can be found for ASD in the NLP literature. In the following many of them are commented. After that, some algebraic notion of antonymy and synonymy is introduced as background for the parasiamese network. Then, the parasiamese network is described, and experiments are presented comparing its results with one of the best performing models known for ASD.

6.1 Antonymy-Synonymy Discrimination

Detecting antonyms and synonyms automatically is a challenging NLP task that can benefit many others. For example, textual entailment [Haghighi et al., 2005, Snow et al., 2006], machine translation [Bar and Dershowitz, 2010] and abstractive summarization [Khatri et al., 2018].

Hand-crafted lexical databases, such as WordNet [Miller, 1995], have been built and maintained to be used in NLP and other fields containing antonyms, synonyms, and other lexical-semantic relations. However, its construction and maintenance takes considerable human effort and it is difficult to achieve a broad coverage. Detecting antonyms automatically, relying on existent resources such as text, dictionaries, and lexical databases is an active NLP research area.

In the last decade, the use and research concerning word vectors have increased rapidly. Word vectors rely on word co-occurrence information in a large corpus. The key idea behind word vectors is the distributional hypothesis that can be expressed as "the words that are similar in meaning tend to occur in similar contexts" [Sahlgren, 2008b, Rubenstein and Goodenough, 1965]. A variety of methods have been developed to train word vectors, such as skip-gram [Mikolov et al., 2013a], GloVe [Pennington et al., 2014b], FastText [Joulin et al., 2016] and ElMo [Peters et al., 2018]. Word vectors are used widely in NLP, for example, a well-known use is in supervised learning, taking advantage of the expansion through word relatedness of the training data.

[1] Some results of this chapter were presented in [Etcheverry and Wonsever, 2019].

As already mentioned, main problem to discriminate antonymy automatically in a distributional unsupervised setting is that the oppositeness is not easily distinguishable in terms of the context distributions. In fact, pairs of antonyms are very similar in meaning. Antonyms are usable in the same contexts but lead to opposite meanings. Antonymy is said to have the paradox of simultaneous similarity and difference [Cruse, 1986] because antonyms are similar in almost every dimension of meaning except the one where they are opposite.

The paradox of simultaneous similarity and difference is notorious in word space models. The contexts of a word and its antonyms contexts usually are similar and therefore they have close vector representations[2].

Due to this paradox, word space models seem not suitable for antonymy detection. Then, a commonly used resource is the path of words connecting the joint occurrence of two candidate words [Nguyen et al., 2017]. Path-based approaches take profit of the fact that antonyms co-occur in the same context more than expected by chance [Scheible et al., 2013, Miller and Charles, 1991], so it is possible to obtain a significant amount of patterns.

In this thesis, we claim that vector space models, despite giving close representations for synonyms and antonyms, contain subtle differences that allow us to discriminate antonyms. In order to stick out those differences we propose a method based on a neural network model that takes account of algebraic properties of synonymy and antonymy. The model formulation is based on the transitivity of synonymy and the antitransitivity of antonymy, on the symmetry of both relations, and on the reflexivity and irreflexivity of synonymy and antonymy, respectively. Moreover, the model exploits the property that two antonyms of the same word tend to be synonyms [Edmundson, 1967] (figure 21). We use these properties to define a model based on siamese networks and a training strategy through antonyms and synonyms.

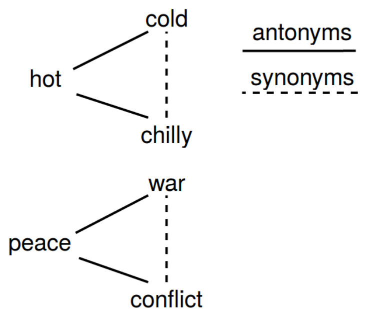


Figure 21 – The antonyms of a same word tend to be synonyms.

We show that the presented approach gives promising results, even in comparison

[2] In fact, word space models may give similar representations to a broader range of related words, such as synonyms and hyponyms. Note the difference between the terms word similarity and word relatedness. While word similarity refers to similar words (synonyms), the concept of word relatedness includes other semantic fields, like antonyms, hypernyms, co-hyponyms, and specific relations (e.g. dog-bone).

to models that use external information, such as input dependency parsing graphs, part-of-speech tags, or path patterns from a corpus. The introduced model is a way to learn any kind of antitransitive relations between distributed vectors. Antitransitivity may be useful, for instance, to represent the relation of being adversary [Bonato et al., 2017]. A different application of the presented approach could be in social networks in order to find out possible unknown enemies relying on a given set of known enmity and friendship links.

6.1.1 Distributional Approaches

Antonymy detection, and antonymy and synonymy discrimination, have been treated principally by distributional approaches. It refers to the use of word vectors or word corpus distributional information to face the task. At first glance, word vectors seem not suitable to discriminate antonymy from synonymy because pairs of antonyms correspond to similar vectors. Many research studies and experiments have focused on the construction of vector representations that deems antonymy.

[Scheible et al., 2013] showed that the context distributions of adjectives allow discriminating antonyms and synonyms if only words from certain classes are considered context words in the vector space mode. [Hill et al., 2014] found that word vectors from machine translation models outperform those learned from monolingual models in word similarity. They suggested that vectors from machine translation models should be used on tasks that require word similarity information, while vectors from monolingual models are more suitable for word relatedness. [Santus et al., 2014] proposed APAnt, an unsupervised method based on average precision of contexts intersections of two words, to discriminate antonymy from synonymy.

Symmetric patterns in the corpus (e.g. X and Y) were used by [Schwartz et al., 2015] to build word vectors and they showed that the patterns can be chosen so that the resulting vectors consider antonyms as dissimilar. [Ono et al., 2015] proposed an approach to train word vectors to detect antonymy using antonymy and synonymy information from a thesaurus as supervised data. A main difference between their approach and ours is that they did not rely on pretrained vectors. They used distributional information jointly with the supervised information to train vectors through a model based on skip-gram. Also, [Nguyen et al., 2016b] integrated synonymy and antonymy information into the skip-gram model to predict word similarity and distinguish synonyms and antonyms.

More recently, [Nguyen et al., 2017] distinguish antonyms and synonyms using lexico-syntactic patterns jointly with the supervised word vectors from [Nguyen et al., 2016b]. To finish, [Vulić, 2018] obtain great performance injecting lexical contrast into word embeddings by terms of their ATTRACT-REPEL strategy.

6.2 Parasiamese Network for Antonymy-Synonymy Discrimination

In this section, we describe the parasiamese network to discriminate antonymy and synonymy [Etcheverry and Wonsever, 2019]. It consists of an approach inspired by siamese networks to magnify the subtle differences in antonyms that distinguish them from synonyms.

6.2.1 Algebra of Synonymy and Antonymy

In order to define and substantiate the approach we introduce an axiomatic characterization of antonymy and synonymy based on the work done by [Edmundson, 1967]. Precisely, synonymy and antonymy are modeled as relations, and a set of axioms is proposed. These axioms, as we are going to show, are essential to formulate our approach.

At first glance, synonymy and antonymy can be seen as binary relations between words. However, based on empirical results Edmundson defined synonymy and antonymy as ternary relations in order to consider the multiple senses of the words, as follows:

$$xS_iy \equiv x \text{ synonym of } y \text{ according to sense } i$$
$$xA_iy \equiv x \text{ antonym of } y \text{ according to sense } i$$

Note that the senses of the words are represented in the relationship rather than in the words themselves. Each i (and therefore S_i and A_i) reflects a particular configuration of the senses of the words in the vocabulary.

Firstly, synonymy is considered a reflexive, symmetric, and transitive relationship. This is expressed by the following axioms:

$$\forall i \forall x (xS_ix) \tag{6.1}$$
$$\forall i \forall x \forall y (xS_iy \implies yS_ix) \tag{6.2}$$
$$\forall i \forall x \forall y \forall z (xS_iy \wedge yS_iz \implies xS_iz) \tag{6.3}$$

S_i is an equivalence relation for each fixed i and therefore it splits the set of words into equivalence classes. In the next section, we show that this is suitable for siamese networks.

Antonymy is also a symmetric relation but it is irreflexive and antitransitive:

$$\forall i \forall x \neg(x A_i x) \tag{6.4}$$

$$\forall i \forall x \forall y (x A_i y \implies y A_i x) \tag{6.5}$$

$$\forall i \forall x \forall y \forall z (x A_i y \wedge y A_i z \implies \neg x A_i z) \tag{6.6}$$

So far, synonymy and antonymy are described separately. The following two axioms involve both relationships:

$$\forall i \forall x \forall y \forall z (x A_i y \wedge y A_i z \implies x S_i z) \tag{6.7}$$

$$\forall i \forall x \forall y \forall z (x A_i y \wedge y S_i z \implies x A_i z) \tag{6.8}$$

Axiom 6.7 is a refined version of the antitransitive property (axiom 6.6)[3]. Assuming that two words cannot be synonyms and antonyms simultaneously, it is direct to prove that axiom 6.7 implies axiom 6.6. We include axiom 6.6 for clarification purposes.

The right-identity, axiom 6.8, says that synonyms of an antonym of a word are also antonyms of this word. Consequently, antonymy relation can be extended to operate between synonymy equivalence classes.

To introduce our model and the considered task setting, we simplify this definition by enforcing a binary relation. We consider:

$$xRy \iff \exists i (x R_i y),$$

where R and R_i are S or A and S_i or A_i, respectively. This simplification encapsulates the multiple senses of the words and therefore it is suitable for word embeddings. However, the presented axioms may not be completely fulfilled under this simplification.

6.2.2 Synonymy and Siamese Networks

Consider a vocabulary V of words where we want to discriminate synonyms and a given word vector set of dimension n for that vocabulary. Then consider a neural network $F_\theta : \mathbb{R}^n \to \mathbb{R}^n$ with weights θ to be used as a siamese network over P, a set of pairs of synonyms, and N, a set of pair of words that are not synonyms. This model can be trained using a backpropagation-based technique and the encoded vectors that are closer than a given threshold are classified as synonyms.

It can be proved that the relation induced by a siamese network (figure 22) is necessarily reflexive and symmetric. Transitivity is a little more tricky. It is satisfied when

[3] In fact, the term antitransitivity is used in Edmundson's article to refer to axiom 6.7 instead of axiom 6.6

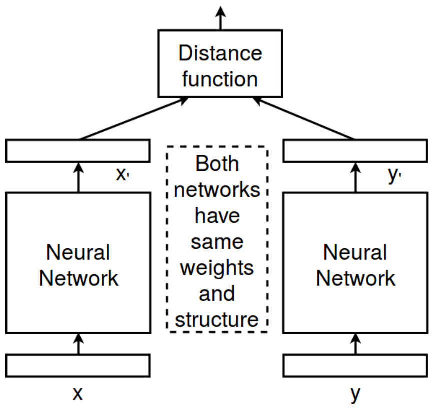


Figure 22 – A diagram of a siamese network model.

the sum of the distances of the antecedent pairs is below the threshold due to the triangular inequity. Therefore, a siamese network is a reasonable approach for supervised synonymy detection.

6.2.3 Antonym's Antitransitivity and Siamese Networks

While a siamese network seems a reasonable choice for supervised synonym detection, antonymy presents a different scenario. Consider $F_{\theta*}$ as the base neural network in a siamese scheme and suppose that it is trained and working perfectly to discriminate pairs of antonyms. Consider also three words w_1,w_2,w_3 such that w_1 is antonym of w_2 and w_2 is antonym of w_3, then

$$\begin{aligned} F_{\theta*}(w_1) &= F_{\theta*}(w_2), \\ F_{\theta*}(w_2) &= F_{\theta*}(w_3) \end{aligned}$$

hence, $F_{\theta*}(w_1) = F_{\theta*}(w_3)$ an therefore, w_1 and w_3 would be recognized as antonyms, violating axiom 6.6.

A siamese network tend to induce a transitive relationship but antonymy is actually antitransitive. To model an antitransitive relation, we propose the following variation of the siamese network that we refer to as the parasiamese network.

6.2.4 The Parasiamese Network Model

Let's consider F_θ and the model diagrammed in figure 23. It consists of a model that consumes two vectors with the same dimension and applies a base neural network once to one input and twice to the other. The idea behind this scheme is that if two words are antonyms then the base network applied once in one of the word vectors and twice in the other will return close vectors. It can be interpreted as one application of the base network taking to a representation of the equivalence class of the synonymy relation and the second application to a representation of its opposite class in terms of antonymy.

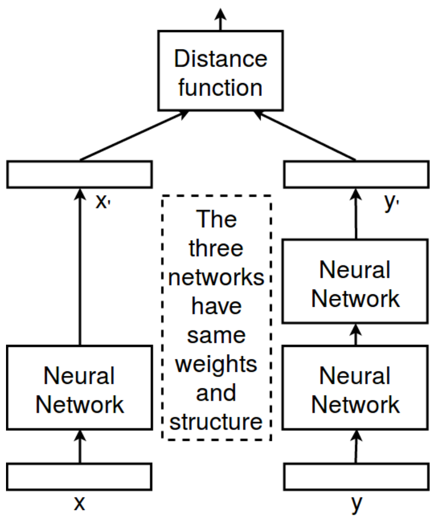


Figure 23 – The parasiamese network model.

Assume that $F_{\theta*}$ is trained and behaves perfectly on data according to the following loss function:

$$L_{ant} = \sum_{(x,y)\in P} d(F_\theta(x), F_\theta(F_\theta(y))) + \sum_{(x',y')\in N} max\{0, \alpha - d(F_\theta(x'), F_\theta(F_\theta(y')))\},$$

where P and N are positive and negative example pairs, respectively; α is the threshold for the negative examples, and d a distance function as in siamese network. Then, it can be seen that the relation induced fulfills the antitransitivity property if $F_{\theta*}(w) \neq F_{\theta*}(F_{\theta*}(w))$, which is expected since antonymy is an antireflexive relation.

Symmetry is not forced by definition but can be included in the loss function or by data, adding the reversed version of each pair in the dataset. The latter is the alternative chosen in this work.

Relaxed Loss Function

In order to classify a pair of words we rely on a threshold ρ. If the candidate pair obtains a distance (between its transformed vectors) below ρ, then it is classified as positive, otherwise as negative. So, it is not necessary to minimize the distance to 0 to classify it correctly. The positive part of the contrastive loss function is modified as

$$\sum_{(x,y)\in P} max(d(F_\theta(x),F_\theta(F_\theta(y))) - \rho\nu, 0)$$

where ν is a factor in [0,1] that states the importance given to ρ, the rest of the terms remain the same as in the previous section. If $\nu = 0$ then the original loss function is recovered. We consider $\nu = 1/2$ and we experimentally observe consistently improved results when this relaxed loss function is used.

Pretraining using synonyms

Consider $F_{\theta*}$ trained and perfectly working to detect pairs of antonyms using the parasiamese scheme presented in the previous section. Consider the word vectors w_1,w_2 and w_3 such that w_1 is an antonym of w_2 and w_2 is an antonym of w_3. According to the parasiamese loss function, we have that

$$F_{\theta*}(w_1) = F_{\theta*}(F_{\theta*}(w_2)),$$
$$F_{\theta*}(F_{\theta*}(w_2)) = F_{\theta*}(w_3).$$

This implies that $F_{\theta*}(w_1) = F_{\theta*}(w_3)$, suggesting to F the role of a siamese network. On the other hand, using axiom 6.7 we have that w_1 and w_3 tend to be synonyms, which, as we previously comment, fits fine for siamese networks.

Using this result, we propose to pretrain F_θ, minimizing a siamese network on synonymy data as in Section 6.2.2, and then perform the parasiamese training to detect antonyms as described in Section 6.2.3. We use the same antonymy/synonymy dataset to pretrain and train the parasiamese network and we experimentally observe that this pretraining phase improves the performance of the parasiamese model.

6.2.5 Experiments

In this section, we describe the experiments performed to evaluate the parasiamese network in antonymy-synonymy discrimination. Here we give the complete information to reproduce the experiments performed. We describe the dataset, the word vectors, and the random search strategy used for the hyper-parameter configuration.

Antonymy-Synonymy Discrimination Dataset

To perform our experiments we use the dataset created by [Nguyen et al., 2017]. This dataset contains a large number of pairs of antonyms and synonyms grouped according to their word class (noun, adjective, and verb). This dataset was built using pairs extracted

	Adjective			Verb			Noun		
	P	R	F_1	P	R	F_1	P	R	F_1
Baseline (concat)	0.691	0.664	0.677	0.756	0.641	0.694	0.764	0.716	0.739
AntSynNet	0.763	0.807	0.784	0.743	0.815	0.777	0.816	**0.898**	**0.855**
Psiam (no relax)	0.735	0.804	0.768	0.815	0.894	0.853	0.786	0.857	0.820
Psiam (no pre)	0.764	0.848	0.804	0.825	0.892	0.857	0.787	0.849	0.817
Psiam ElMo	0.838	0.844	0.841	0.830	0.910	0.869	0.802	0.855	0.827
Psiam FastText	**0.855**	**0.857**	**0.856**	**0.864**	**0.921**	**0.891**	**0.837**	0.859	0.848

Table 8 – The table shows the performance of the parasiamese network (abbreviated as *psiam*). The first row presents the results of a feed-forward network on word vectors concatenation, as reference, and the second row exposes the best results of the AntSynNet model [Nguyen et al., 2017]. The third row refers to the parasiamese model without including the relaxed loss, and the fourth includes relaxed loss but it was skipped the pretraining stage; both of them using FastText vectors, that it were the best performing word vectors. Finally, the fifth and sixth rows show the results of the model using pretraining and the relaxed loss function on ElMo and FastText vectors, respectively.

by [Nguyen et al., 2016b] from WordNet and Wordnik[4] to induce patterns through a corpus. Then, the induced patterns were used to extract new pairs, filtering those that match less than five patterns. Finally, the dataset was split into train, validation, and test partitions; and balanced to contain the same number of antonyms and synonyms. The number of pairs contained on each partition of each word class is shown in table 9.

	Train	Val	Test
Adjective	5562	398	1986
Verb	2534	182	908
Noun	2836	206	1020

Table 9 – [Nguyen et al., 2017] number of word pairs of each partition in the dataset.

Pretrained word vectors

For the experimental setting, we consider pretrained general-purpose word vectors. We avoid out-of-vocabulary terms using character-based approaches. The following publicly available resources were considered:

- FastText [Joulin et al., 2016] vectors trained on English Wikipedia dump [5]. We use default hyper-parameters and the vector dimension is 300.

[4] `http://www.wordnik.com`
[5] `http://mattmahoney.net/dc/enwik9.zip`

- ElMo [Peters et al., 2018] vectors for English from [Che et al., 2018] [6]. We use the first layer of ElMo that gives representations for decontextualized words.

In the case of FastText, we compute 300 dimensional vectors for each word in the dataset. In the case of ElMo embeddings, the pretrained model was already defined to generate representations of 1024 dimensions.

Base Network Structure

The base network transforms each word vector into a representative form of synonymy and antonymy. Any differentiable function that inputs and outputs vectors of the same dimension of the word embeddings space can be used as a base (or encoder) network. In this work, we consider feed-forward fully connected networks with ReLU activation functions.

The presented model involves dozens of hyper-parameters and some of them with many options. We use random search to find a good hyper-parameter configuration, since it may lead to a better and more efficient solution in comparison to grid or manual search [Bergstra and Bengio, 2012]. This improvement is given by the fact that some hyper-parameters do not really matter and grid or manual search would consume time exploring each combination of them (for each combination of the rest), while random search does not exhaustively explore irrelevant parts of the hyper-parameters space.

We perform random search sampling models according to the following considerations:

- 2,3,4 and 5 layers uniformly chosen
- for each hidden layer (if any) we sample its size from a Gaussian distribution with $\mu = d/2$ and $\sigma = d/5$, where d is the dimension of the word vectors.[7]
- dropout with 1/2 of probability to be activated or not and dropout probability is given by a Gaussian distribution ($\mu = 0.25$ and $\sigma = 0.1$).
- prediction (or positive) threshold and contrastive loss threshold are uniformly chosen between {2.5,2,1.5,1,0.5,0.2} and {3,5,10}, respectively.
- batch size uniformly chosen from {32,64,128}
- we choose between SGD and Adam with equal probability with a learning rate chosen from {0.01,0.001,0.0001}
- the patience for the early stopping was sampled uniformly from {3,4,7,9}

[6] `http://vectors.nlpl.eu/repository/11/144.zip`
[7] The dimension of input and output layers is d by model definition.

The Glorot uniform function [Glorot and Bengio, 2010] is used for weights initialization. We stop the training using early stopping and we check out the best model in the whole run against the validation set. For the implementation, we use Keras [Chollet et al., 2015].

After analyzing the results of 200 sampled hyperparameter configurations using the FastText vectors we found that an adequate hyper-parameters setting is a four-layered network of input dimensions $[300, 227, 109, 300]$ on its layer from input to output, without dropout, and ReLu activation function for every neuron. For training, a batch size is 64, an acceptance threshold of 2.0, and of 3.0 for the negative part of the contrastive loss. The optimizer method is SGD with a learning rate of 0.01 and a patience of 5 for the early stopping. This training setup was used in both phases: pretraining and training. For the experiments with ElMo embeddings, we uniquely adjust hidden layers sizes, probably ElMo results may improve by a dedicated hyperparameter search.

Results

In this section, we discuss the results obtained with the presented approach and we analyze the model behavior through the outputs of the base network in siamese and parasiamese schemes. We include two baselines with different motivations for comparison purposes. We analyze the model outputs for related and unrelated pairs (i.e. pairs that are not synonyms or antonyms). At the end of this section, we analyze the output of the base network.

We consider two baselines to compare our experiments. The first baseline is a feed-forward network classifier that consumes the concatenation of the embeddings of each word in the candidate pair. This baseline compares the performance boost of the proposed model against a conventional supervised classification scheme using neural networks. For this baseline we consider the FastText vectors to feed a four-layered network with layer dimensions of [600,400,200,1] from input to output and ReLu activation functions. This model was trained through binary cross-entropy loss and SGD with a learning rate of 0,01.

The second baseline we consider for comparison is AntSynNet [Nguyen et al., 2017], a pattern-based approach that encodes the paths connecting the joint occurrences of each candidate pair using an LSTM. It relies on additional information, such as part-of-speech, lemma, syntax, and dependency trees.

The obtained results are reported in table 8. The first baseline is included to compare the performance of our model with a word vector concatenation classification. We also report results with and without pretraining to show the performance gain that pretraining contributes. Notice that, in contrast to AntSynNet, no path-based information is considered in our approach.

Siamese and parasiamese outputs

In this section, we show the outputs of the siamese and parasiamese networks on word pairs chosen from the validation set (see table 10).

Word_1	Word_2	Cos	Siam	Psiam
cold	warm	0.327	3.051	1.01
cold	hot	0.367	3.576	0.801
raw	hot	0.467	5.398	1.424
peace	war	0.448	6.167	0.367
stupid	clever	0.406	7.635	1.192
stretch	contract	0.526	4.351	0.354
love	hate	0.378	2.771	0.153
reject	take	0.528	2.341	0.66
close	harsh	0.544	4.11	0.682
small	large	0.178	4.076	0.302
auntie	uncle	0.351	1.721	0.561
day	night	0.366	1.098	0.803
sloping	vertical	0.331	0.254	2.687
hot	cool	0.253	1.753	2.239
invisible	visible	0.137	1.816	2.11
change	mutate	0.593	0.853	4.879
ample	large	0.398	0.115	4.05
flee	depart	0.499	0.682	3.958
cure	heal	0.332	0.646	5.769
elegant	classy	0.48	0.964	5.917
herald	hail	0.507	0.752	5.826
live	exist	0.527	0.126	4.737
agitate	disturb	0.282	0.627	4.683
chop	divide	0.657	1.56	4.085
sturdy	hardy	0.333	1.894	3.493
stout	robust	0.541	1.903	4.361
scatter	dot	0.477	2.05	1.402
fizzle	fail	0.437	3.706	3.464
see	discern	0.501	3.878	3.724

Table 10 – A word sampling and their vector cosine distances, siamese and parasiamese (Psiam) outputs. The upper and lower parts correspond to pairs in the validation data as antonyms and synonyms, respectively. The threshold for acceptance is 2.0.

It can be observed in the obtained results, in general, a suitable behavior of the model. We also include the cosine distance to compare and show that it is unable to distinguish between antonyms and synonyms. It is interesting to notice, for instance, in the upper part of the table, that corresponds to antonyms, the difference in outputs between the pairs *cold-warm* and *cold-hot*. It may be interpreted as that *cold-hot* are *more* *antonyms* than *cold-warm*, which seems adequate. Below the dashed line of each part we include some failure cases.

Unrelated pairs

The task setting considered for this work only uses synonyms and antonyms for training. It is interesting to notice that in this work the behavior of the model with unrelated pairs is learned from related pairs and word embeddings, without considering any unrelated pairs during training. Table 11 shows the outputs given by siamese and parasiamese networks on unrelated pairs.

Word_1	Word_2	Cos	Siam	Psiam
see	disturb	0.579	3.803	3.866
flee	cure	0.534	3.189	3.764
man	wolf	0.507	4.017	2.649
ascend	speak	0.63	1.927	2.776
safe	adverse	0.475	4.57	0.625
change	mature	0.511	1.118	3.602
cold	night	0.48	1.134	1.98
cold	day	0.574	3.727	0.483
warm	night	0.462	0.773	0.658
warm	day	0.52	0.733	1.601

Table 11 – Unrelated pairs and its word vectors cosine distance, siamese model output, and parasiamese (Psiam) output.

The obtained results show that the model is not capable to detect unrelated pairs correctly. However, it is worth noting that during the training the model did not see a single unrelated pair. According to table 11, the model seems to learn a broader relation. For example, the words safe and adverse are predicted as antonyms and although they are not antonyms, they have some oppositeness. Similarly, the combinations of *cold* and *warm* with *day* and *night* also seem to be coherent since the day tends to be warmer than the night and the night tends to be colder than the day. In the upper part of the table, we include unrelated pairs that were correctly predicted as unrelated, and below the dashed line we include failure cases on unrelated pairs.

Encoder Output

In this section, we analyze the learned base network or encoder in the parasiamese model. In figure 24 we show a 2D visualization of the original and the transformed word embeddings. The sample of words was chosen from the validation set and t-SNE [Maaten and Hinton, 2008] was used for the dimensionality reduction. It can be

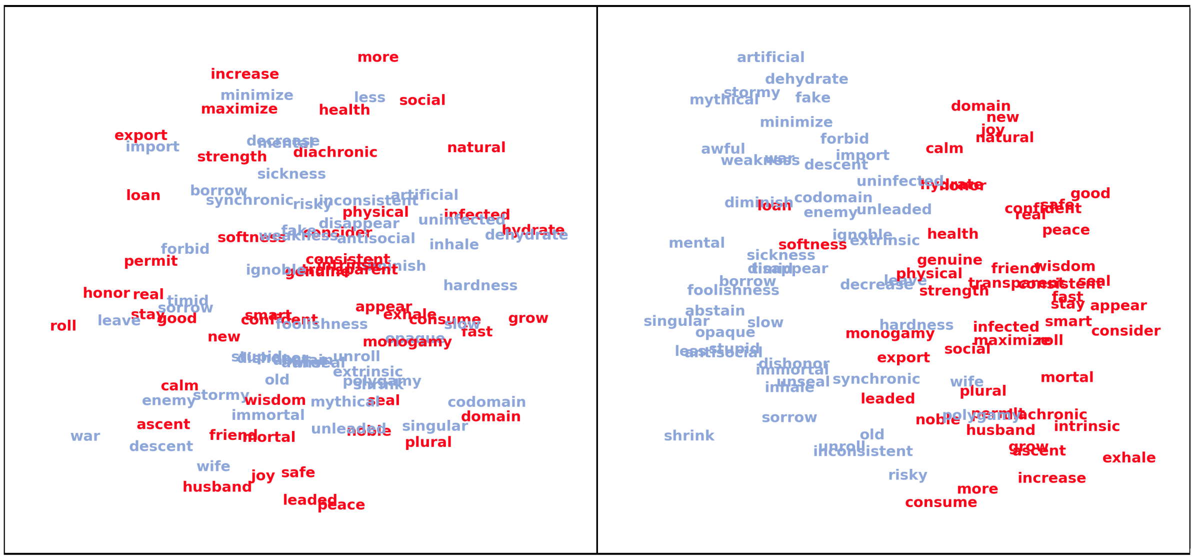


Figure 24 – 2D visualization of the original (left) and the transformed (right) word vectors. Related words are colored in red and light blue to facilitate the visualization of how the model split antonymy from original embeddings.

observed that in the original space, antonyms tend to be close and when the base network is applied the space seems to be split into two parts, corresponding to each pole of antonymy.

We also consider the resulting space from applying the transformation twice to the original word vector space, which is similar to the result of applying it only once. This behavior is coherent with the parasiamese network definition.

7 Repelling Parasiamese Network[1]

This section presents the repelling parasiamese network for antonymy-synonymy discrimination. This model can be seen as an improvement of the parasiamese network, since it presents better performance in ASD. It simply consists of contrasting a parasiamese network and its siamese counterpart in tandem as diagrammed in figure 25. The repelling constraint is argued by the fact that it can be assumed incompatibility between synonymy and antonymy. It is quite difficult, although there are exceptions, to list pairs of words that could be either synonyms or antonyms depending on the context.

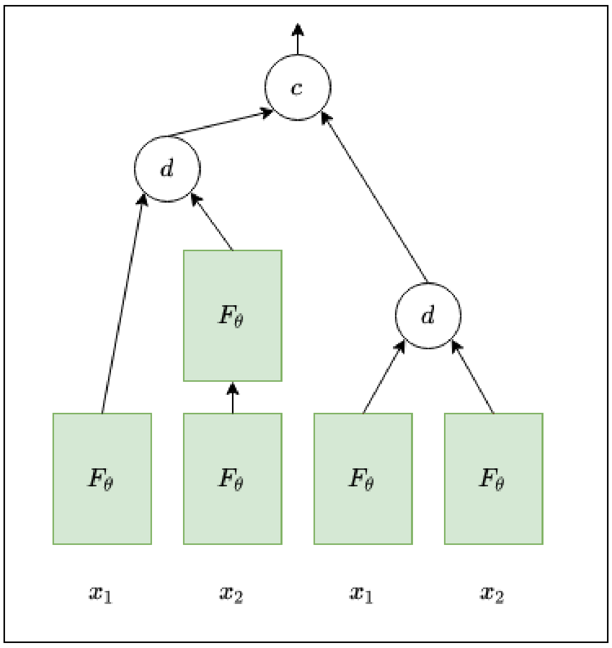


Figure 25 – Diagram of a repelling parasiamese network. d corresponds to a distance function (e.g Euclidean) and c is a function that contrasts the siamese and parasiamese outputs (e.g. contrastive loss).

To formulate the repelling parasiamese network consider the following two functions:

- $\Phi^{(p)} : \Re^d \times \Re^d \to \Re_{\geq 0}$ a function that when minimized the parasiamese output decrease (if it is greater than the margin) and the siamese network output increase (if less than the margin)

- $\Phi^{(s)} : \Re^d \times \Re^d \to \Re_{\geq 0}$ a function that when minimized the siamese output decrease (if it is greater than the margin) and the parasiamese network output increase (if less than the margin)

[1] Some results of this chapter were presented in [Etcheverry and Wonsever, 2023].

These functions, $\Phi^{(p)}$ and $\Phi^{(s)}$, are the parasiamese and siamese networks, respectively, with their respective counterparts repelled.

Then, the training is performed by minimizing:

$$L = \sum_{(x,y)\in A} \Phi^{(p)}(x,y) + \sum_{(x,y)\in S} \Phi^{(s)}(x,y),$$

where A and S are antonymic and synonymic pairs, respectively. The repelling details are on $\Phi^{(p)}$ and $\Phi^{(s)}$ definitions.

7.1 Repelling Strategies

We propose two formulations based on two of the main approaches in deep metric learning: contrastive [Hadsell et al., 2006] and triplet [Hoffer and Ailon, 2015] loss-based minimization.

7.1.1 Contrastive Loss

The contrastive loss based approach on a pair of antonyms (x,y) attempts, through $\Phi^{(p)}$, to minimize the distance between $F_\theta(x)$ and $F_\theta \circ F_\theta(y)$, and maximize the distance between $F_\theta(x)$ and $F_\theta(y)$ (analogously for synonyms and the siamese perspective $\Phi^{(s)}$).

Then, the repelling parasiamese $\Phi^{(p)}$ and $\Phi^{(s)}$ functions are written as:

$$\begin{aligned}\Phi^{(p)}(x,y) =& [\Phi(x,y) - \mu_p]_+ + [\mu_s - \Psi(x,y)]_+ \\ \Phi^{(s)}(x,y) =& [\Psi(x,y) - \mu_p]_+ + [\mu_s - \Phi(x,y)]_+\end{aligned}$$

where μ_p and μ_s are the margins used for attracting and repelling the parasiamese and siamese subnetworks, respectively; $\Phi(x,y) = ||F_\theta(x) - F_\theta(F_\theta(y))||_2^2$ is the parasiamese subnetwork, and $\Psi(x,y) = ||F_\theta(x) - F_\theta(y)||_2^2$ is its siamese counterpart.

7.1.2 Triplet Loss

The repelling parasiamese triplet-based approach on a pair of antonyms (x,y) considers, through $\Phi^{(p)}$, the triplet $(F_\theta(x), F_\theta \circ F_\theta(y), F_\theta(y))$ attempting to get $F_\theta(x)$ closer to $F_\theta \circ F_\theta(y)$ than to $F_\theta(y)$. Precisely, the repelling parasiamese functions $\Phi^{(p)}$ and $\Phi^{(s)}$ on a triplet based scheme are:

$$\begin{aligned}\Phi^{(p)}(x,y) &= [\Phi(x,y) - \Psi(x,y) + \mu]_+ \\ \Phi^{(s)}(x,y) &= [\Psi(x,y) - \Phi(x,y) + \mu]_+\end{aligned}$$

where μ is the separation margin.

7.2 Repelling Parasiemese Variants

In this section two repelling parasiamese alternatives are described. This alternatives were included in the experiments, and in some cases they presented better results.

7.2.1 Half-Twin Encoders

The repelling of the siamese part of a parasiamese network does not rely on any particular shape of its branches. The encoder double application is an alternative without adding new weights but it is only needed that both branches are not the same to be possible to repel the siamese part.

We propose the half-twin repelling parasiamese variant by substituting $F_\theta \circ F_\theta$ with a completely different encoder network.

An intuition to justify the operation of this variant is that since one of the branches is being repelled by the siamese part, it is possible to leave the parameters of the other branch free. In this way a better fit can eventually be achieved since the parameters are not bound. It will be seen in the experiments section that this did indeed happen.

7.2.2 Symmetric Variant

The parasiamese network is not symmetric. [Etcheverry and Wonsever, 2019] showed that the performance is improved when symmetry is considered by augmenting the dataset with the reversed pairs.

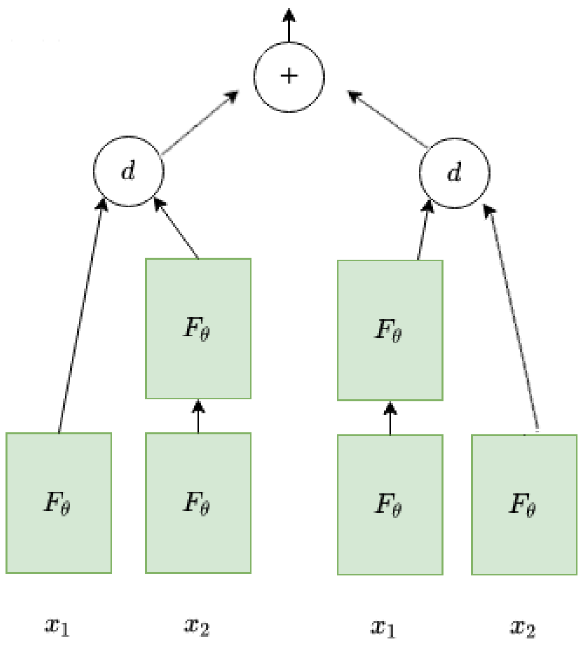


Figure 26 – Diagram of symmetric parasiamese network. d corresponds to a distance function (e.g Euclidean).

	Random Split								
	Adjective			Verb			Noun		
	P	R	F	P	R	F	P	R	F
MoE-ASD [Xie and Zeng, 2021]	.878	.907	.892	.895	.920	.908	.841	**.900**	.869
Distiller [Ali et al., 2019]	.854	**.917**	.884	.871	.912	.891	.823	.866	.844
Parasiamese	.855	.857	.856	.864	.921	.891	.837	.859	.848
Contrastive Parasiamese	.903	.870	.886	.862	.919	.889	.858	.820	.839
Contrastive Parasiamese Sym	.878	.910	.894	.863	**.945**	.902	.845	.875	.859
Contrastive Parasiamese HTwin	.875	.909	.892	.813	.907	.857	.815	.890	.851
Contrastive Parasiamese HTwin Sym	.898	.906	**.902**	.890	**.945**	**.917**	.880	.874	**.877**
Triplet Parasiamese	.887	.878	.883	.863	.916	.889	.848	.843	.846
Triplet Parasiamese Sym	.898	.859	.878	.874	.934	.903	.856	.851	.853
Triplet Parasiamese HTwin	.870	.866	.868	.854	.866	.860	.808	.800	.804
Triplet Parasiamese HTwin Sym	**.926**	.856	.890	**.905**	.925	.915	**.889**	.831	.859
	Lexical Split								
MoE-ASD [Xie and Zeng, 2021]	.808	.810	**.809**	**.830**	.693	.753	.846	.722	.776
Contrastive Parasiamese HTwin Sym	.756	**.868**	.808	.766	**.823**	**.793**	.770	.800	**.785**
Triplet Parasiamese HTwin Sym	**.831**	.643	.725	.826	.642	.723	**.860**	.634	.730

Table 12 – Results of the repelling parasiamese network and best performing related works. Non-repelling parasiamese network results are exposed in the first block. The second and third blocks correspond to the here proposed contrastive and triplet-based repelling parasiamese networks. The symmetric and half-twin variants are indicated with Sym and HTwin, respectively, in the model name column. The lower part of the table corresponds to the results obtained using [Xie and Zeng, 2021]'s lexically split dataset.

In [Etcheverry and Wonsever, 2023], we consider a symmetric variant through the model definition, by adding two variants of $\Phi^{(p)}$ and $\Phi^{(s)}$ using the same encoder through swapping the branches of the parasiamese subnetwork as showed in figure 26.

7.3 Experiments

Our experiments are performed using fastText [Joulin et al., 2016] pretrained word embeddings, available in their site[2], and the following ASD datasets:

- **Nguyen's:** Built by [Nguyen et al., 2016a] using WordNet [Miller, 1995] and Wordnik (`https://www.wordnik.com/`). It consists of 15,632 pairs of words with a balanced amount of synonyms and antonyms, over a vocabulary of 9,405 words.
- **Xie's:** Built by [Xie and Zeng, 2021] using [Nguyen et al., 2016a]'s dataset but splitting (into train, validation, and test) without any word in common between each

[2] `https://fasttext.cc/docs/en/crawl-vectors.html`

part (i.e. without lexical intersection). It counts with 12,732 balanced pairs over a vocabulary of 9,404 words.

We do not consider any explicit feature to indicate negation prefixes. The only subword information considered is by using fastText word vectors. To find a good hyperparameter configuration for each model we perform a random search against the validation set to report the best-performing configurations. We include the random search details in Appendix A.

Table 12 reports the results of the contrastive and triplet formulations, for symmetric, and half-twin combinations. The repelling parasiamese variants are compared among them and against the non-repelling one with siamese pretraining on synonyms and symmetry by augmented data. It can be observed that the symmetric variants of the repelling parasiamese network consistently outperform its non-repelling version. Regarding symmetry on the repelling variants, it obtains the best results, whereas the half-twin variants are the most benefited. It is also compared the repelling parasiamese network with the best-performing methods found in the literature: the Distiller [Ali et al., 2019] and MoE-ASD [Xie and Zeng, 2021][3]. The half-twin contrastive repelling parasiamese network achieved the best results in random and lexically split datasets.

[3] In our case, we do not consider dLCE vectors since those are already specialized to discriminate antonyms and synonyms.

8 Hypernymy and ASD in Spanish

This chapter presents the creation of datasets in Spanish for hypernymy detection [Lee et al., 2020] and antonymy-synonymy discrimination [Camacho et al., 2022]. It also presents results on these datasets using the models considered in this thesis.

It is important to note that in general, datasets in Spanish either do not exist or are of poorer quality than those for English. In this particular case, in order to work for Spanish it was necessary to generate datasets using translations of English datasets, patterns to extract terms, and Spanish WordNet.

These datasets were generated in the framework of two degree projects supervised by the author of this thesis, and were made available to the community.

8.1 Hypernymy Detection

As part of this thesis, a dataset for Spanish and experiments using the previously mentioned approach were conducted [Lee et al., 2020]. In this section, the details of the dataset and the obtained results are presented.

8.1.1 Dataset Construction

Following previous works on hypernymy detection, the dataset consists of pairs of words labeled as true when it correspond to an hyponym-hypernym pair, and false otherwise. In the dataset construction process, a variety of sources were used to obtain hypernymic and non-hypernymic instances. In the following it is described each used source and it is given an idea of the quality of the resulting dataset based on a randomly picked sampling.

8.1.1.1 Hypernyms Extraction

The extraction of hypernymic pairs was performed using: (a) Spanish WordNet, (b) patterns against a Spanish Corpus, and (c) a Shwartz's dataset translation from English to Spanish.

In the following, each source use is described:

- Spanish WordNet:
  The main source of positive instances of the dataset is the Spanish version of the WordNet of the Open Multilingual Wordnet (OMW). The hypernymy relation in

	Size (# pairs)		% Correct	
k	no freq.	freq.	no freq.	freq.
1	15695	10103	83.9	84.3
2	29180	19258	82.2	83.3
3	35103	22851	77.6	83.5

Table 13 – Sizes and percentages of correct hypernyms (from a sample) of the resulting pairs considering 1, 2, and 3 most frequent words of each synset (i.e. k value). Values on the left correspond to apply only the first heuristic (i.e. pairs from most frequent words of the synsets holding hypernymy), and values on the right correspond to apply both heuristics (i.e. pairs whose hyponyms have a higher frequency than the hyperonym, from most frequent words of the synsets holding hypernymy).

WordNet is defined between synsets, hence a selection of pairs is needed to be performed, taking one word of each synset, to obtain hypernymic pairs that will belong to the dataset.

The following two heuristics were considered:

1. Most frequently used word form each synset according to their frequency in the corpus of [Cardellino, 2016][1].
2. Pairs that the hyponyms have a frequency greater than the frequency of it proposed hypernym, based on [Santus et al., 2014] work.

Regarding the first heuristic, the result of considering the pairs from an all-vs-all of the k most frequent words of each synset, for different values of k, was studied. Table 13 reports the respective sizes and percentages of correct pairs of a random sample. It can be observed that taking into account more than the two most frequent words of each synset the results degrade considerably.

The output of the first heuristic is filtered using the second heuristic. It can be observed a quality improvement in the resulting pairs after applying it (table 13). Based on these results, the most three frequent words of each synset were used, and the pairs where the hyponym is more frequent than the hypernym were filtered. To finish with WordNet extracted hypernyms, the cycles were eliminated, since they are generated due to the multiple senses of certain words. The resulting pairs conform to the WordNet positive instances of the dataset.

- Pattern-based:
  Relying on the importance of the pattern-based approaches to detect and discover hypernyms [Hearst, 1992], positive instances extracted using high-confidence patterns

[1] Spanish Billion Word Corpus and Embeddings by Cristian Cardellino: `https://crscardellino.github.io/SBWCE/`

were included in the dataset. To obtain confident tuples, the following two patterns for Spanish that present high confidence were used [Ortega et al., 2011]:

1. "el <hyponym> es el único <hyperonym>"
2. "de <hyponym> y otras <hyperonym>"

These patterns were used to extract candidate pairs from [Cardellino, 2016]' corpus. Despite using high-confidence tuples, the quality of the resulting pairs was poor. Subsequently, some improvement was achieved by filtering these tuples using the most probable part of speech of the words in the tuple to be the same. Even so, the obtained quality was not good enough to be included in the final dataset. However, despite the poor quality of the tuples, it was considered useful to study the behavior when they are included as noisy data. For this purpose, this tuples are included along with the dataset in a separate file.

- Shwartz dataset translation:
  In the dataset built by [Shwartz et al., 2016a], the hypernymy relation instances were obtained from English WordNet, DBPedia, Wikidata, and Yago. The translation was performed through Google's translation library. The resulting candidates were included as positive instances of the dataset.

In addition to these sources, it is possible to consider the transitive links as potential hypernymic instances, since the hypernym relation fulfills the transitive property. However, this assumption may not be satisfied when different senses are faced in a transitive links, introducing incorrectly labelled pairs. So, it was decided to not include transitive instances in the dataset.

8.1.1.2 Non-Hypernymic Pairs

The non-hypernymic pairs, or negative instances of the dataset, are pairs of words that do not hold an hypernymic relation between them. The procurement of unrelated pairs was done by the following approaches:

- Random sampling:
  Since most of the words are not hypernym between them, a random pick of two words from a given vocabulary will probably get a non-hypernymic pair. So, the noun words with at least 4 characters and a frequency greater than 200 from Cardellino's Corpus were obtained, jointly with the vocabulary of the positive part, above mentioned, of the dataset. Then, random tuples that were not already included in the dataset were included till completed the desired ratio of 1:3 of positive:negative instances.

  The dataset resulting from the positive tuples from WordNet, Shwartz's translation, and the negative tuples obtained through random pairs is referred to as **base** dataset.

- Cohyponyms:
  Cohyponymy is the relation between hyponyms that share the same hypernym. They are words that have properties in common, but which in turn have characteristics that differentiate them well from each other. Cohyponymy can be seen as words belonging to the same class (e.g. male-female, march-november). Given a pair of cohyponyms, it is highly probable that a hypernymy relation is not fulfilled between them. Therefore, it is possible to obtain negative pairs from cohyponymic relations entailed from the positive instances.

- Inverted links:
  The hypernym relation is asymmetric. Therefore, if a tuple satisfies the hypernym relation, its inverse does not. Then, once having the positive tuples of the dataset already, a simple way to obtain potentially negative tuples is by exchanging the order of the pairs of the positive dataset. However, some synonyms may become a problem in this assumption when they are wrongly considered as hypernyms. Synonyms can be extracted by patterns as hypernyms fulfilled in both directions (e.g. neat-tidy).

- Antonymy:
  Words that have opposite meanings are called Antonyms. It can be assumed that if there is an antonymy relationship, the hypernym relationship is not satisfied. Therefore, antonyms extracted from WordNet are considered as negative instances.

8.1.1.3 Dataset Details

The resulting amount of positive and negative pairs in the dataset are shown in table 14. There were *16835* words from WordNet in the dataset and *26443* words from Cardellino.

Positive Pairs		
WordNet	Pattern	Shwartz
27861	2731	3798

Negative Pairs			
Random	Cohyponym	Antonym	Meronym
$\sim$ 90000	$\sim$ 45000	1107	5940

Table 14 – Total amount of positive and negative instances from where each version of the dataset is built.

Following the work of [Shwartz et al., 2016a] and [Etcheverry and Wonsever, 2020], it is considered two splits of the data: random and lexical split. While the random split is performed randomly, the lexical split avoids lexical intersection between the partitions.

The random split consists in splitting the dataset randomly, without taking into account any consideration. A random split was performed with the following ratio: 70 % for the training set, 25 % for the test set, and 5 % for the validation set. To avoid the phenomenon of lexical memorization, the training, validation, and test sets are split without words in common. The split was performed with the methodology of [Shwartz et al., 2016a]. The approximate division ratio was 70-25-5. The respective sizes of the random and lexical splits of the base dataset are shown in table 15.

		Train	Val	Test	Total
Rnd. Split	P	18654	1332	6662	106592
	N	55962	3996	19986	
Lex. Split	P	8221	513	2506	44960
	N	24663	1539	7518	

Table 15 – Spanish dataset sizes for lexical and random splits. The sizes are discriminated in terms of positive (P) and negative (N) instances (i.e. hypernymic and non-hypernymic, respectively). These reported sizes do not contain cohyponyms as negative pairs or pattern-extracted positive instances.

8.1.2 Experiments

A feed-forward network was trained as an order embeddings [Vendrov et al., 2015b] on hypernymic and non-hypernymic pairs. This network takes the word vectors to non-negative vectors with a partial order relation between them. It is trained to take hypernym pairs as ordered vectors.

The hyperparameter configuration is obtained through random search. They were tuned using the validation set, and the best configuration was reported using the test partition. For simplicity, only feed-forward networks were considered as encoders for the order embedding and fastText [Joulin et al., 2016] word vectors for Spanish and English.

The models were evaluated using precision, recall, and F measures. The best configuration consisted of a three-layered feed-forward network, with 150 neurons and SELU activation functions on the first two layers and 100 ReLU units for the output layer. For training Adam [Kingma and Ba, 2014] strategy with a learning rate of 0.005. The training is concluded by early stopping with the patience of 5, returning the best-performing snapshot against the validation set.

To show the performance of the previously commented subsets that are not included in the base dataset, such as pattern-extracted tuples and co-hyponyms, the following arrange were considered as training data:

- As a base, the positive instances from WordNet and the translated instances of the Shwartz dataset, and the randomly sampled words from the vocabularies of

(a)

	P_{rand}	R_{rand}	F_{rand}	P_{lex}	R_{lex}	F_{lex}
OrdEmb base	0.855	0.904	0.879	**0.823**	0.674	0.741
OrdEmb +cohyp	0.857	**0.932**	**0.893**	0.809	**0.827**	**0.818**
OrdEmb +pattern	**0.860**	0.885	0.872	0.798	0.766	0.782
OrdEmb +pattern +cohyp	0.859	0.930	0.893	0.802	0.821	0.811

(b)

	P_{rand}	R_{rand}	F_{rand}	P_{lex}	R_{lex}	F_{lex}
OrdEmb base	0.719	**0.946**	0.817	0.744	0.841	**0.789**
OrdEmb +cohyp	0.847	0.869	0.858	**0.781**	0.716	0.747
OrdEmb +pattern	0.742	0.931	0.826	0.666	**0.857**	0.749
OrdEmb +pattern +cohyp	**0.848**	0.870	**0.859**	0.759	0.678	0.716

Table 16 – Results on the test set for Spanish. The upper table (a) shows the result of evaluating without introducing inferred cohyponymy instances in the test partition and the lower table (b) shows the results including cohyponymy instances in the test partition. The labels +cohyp and +pattern stand for cohyponymy and pattern-extracted instances in the training data of the base dataset.

Cardellino and WordNet as negative instances.

- The base dataset with positive instances extracted by patterns.
- The base dataset with the co-hyponyms as negative instances.
- The base dataset with cohyponyms as negative and pattern-extracted tuples as positive instances.

Results

The results are in the table 16. The model was evaluated against the base test partition and included co-hyponymy instances on the test data (in addition to the train data). In the results it can be observed that both co-hyponyms and pattern-extracted instances give some improvement in most cases, being the co-hyponyms the most beneficial except for the lexical split.

8.2 Antonymy-Synonymy Discrimination

Antonymy-synonymy discrimination has been focused mainly in English and as far as we know there is not any specific work for Spanish. In a broader context of lexical relations, there is some research for Spanish, such as [Glavaš and Vulić, 2018] that addressed lexical relations detection and performed a manually curated automatic translation to Spanish and German (consists of 1850 pairs of antonyms), and a popular publicly available

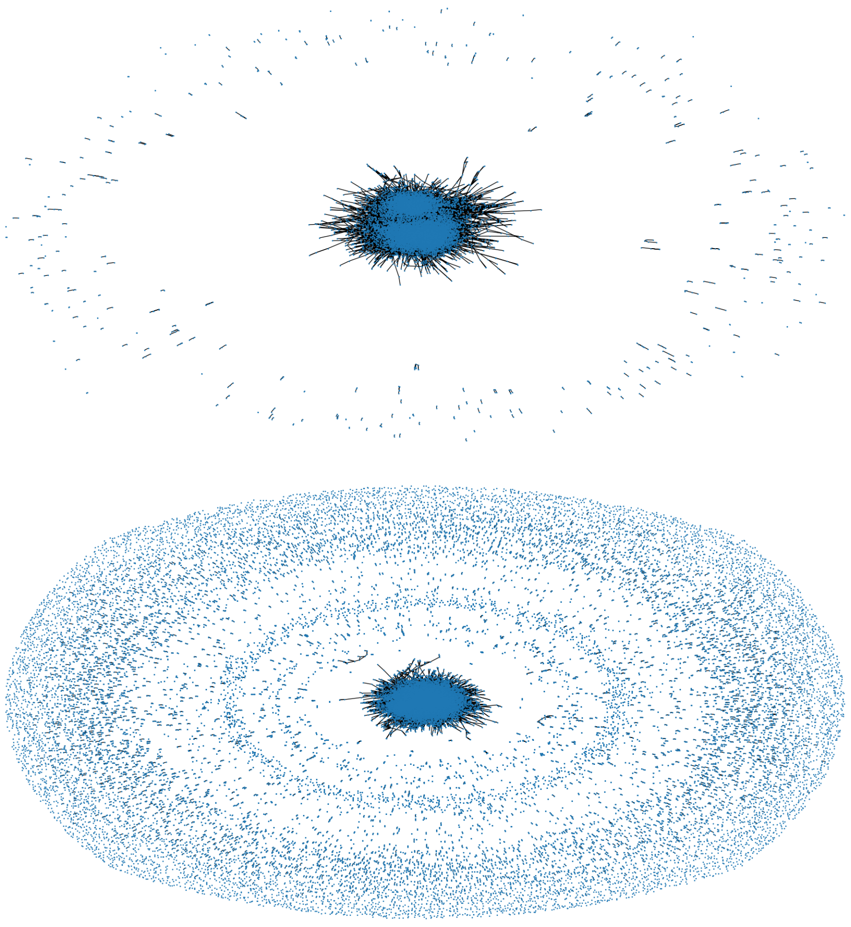


Figure 27 – Antonymy (top) and synonymy (bottom) graphs. Words are represented as nodes (blue dots) and the relations between words are represented as edges (black lines). Notice that what looks like isolated blue dots are actually two very close nodes and the edge is not visible due to the proximity. This proximity is due to the arrangement of the nodes when drawing the graph and not to the semantic proximity of the words. They denote words related to a single word, connected components of two elements. On the other hand, the central component, in both cases, is a connected component of nodes, that are connected between them through paths formed by one or more edges.

source for lexical relations for Spanish is WordNet [Miller, 1995], through the Multilingual Central Repository (MCR) [Gonzalez-Agirre et al., 2012], that contains a translation to Spanish from the original WordNet in English [Fernández-Montraveta et al., 2008].

In the framework of this thesis, a dataset has been built for Spanish considering Spanish WordNet and online dictionaries [Camacho et al., 2022]. The dataset is imparted in 3 versions according to its partitioning criterion: (1) random, (2) lexical split using the algorithm of [Shwartz et al., 2016b], and (3) lexical split using the information of the graph of the relation (see figure 27). Finally, experiments have been conducted using the parasiamese network.

8.2.1 Dataset for Spanish

The dataset was created using WordNet for Spanish[Fernández-Montraveta et al., 2008] and three sites of synonyms and antonyms. An extraction process was performed on each resource and the results were combined.

Firstly, we used the Spanish version of WordNet and we extracted all its words with their respective antonyms and synonyms.

Then, we considered the following online dictionaries:

- wordreference.com
- antonimos.net
- sinónimo.es

For each source, we have built a scraper, that has extracted each word with its respective antonyms and synonyms. Due to the nature of the online dictionaries, an initial vocabulary was required to scrape them. Each scraper uses this initial vocabulary to query the online dictionary. The query result is processed by the scraper to generate tagged tuples with the queried word and the ones in the response. If a word in the response is not present in the vocabulary that word is added to it, expanding the current vocabulary. The initial vocabulary was from WordNet for Spanish. After that, we also used the vocabulary from Cardellino corpus [Cardellino, 2016] in a second run of the scrapers. So, as a result, two datasets are obtained, one generated with WordNet vocabulary and the other with Cardellino's; in addition to the pairs from WordNet.

Despite Cardellino's vocabulary being bigger than WordNet (3225303 vs 36681), the datasets obtained from each one were very similar, resulting in 1105441 and 977408 tuples, respectively, with around 10% difference between them. After merging the different sources, the symmetric pairs were removed; then, two versions of each dataset is created regarding symmetry, generating the symmetrical tuples in each partition.

An analysis of the topology in the antonymy and synonymy relations was performed using Networkx [Aric Hagberg, 2005]. We draw each relation as an undirected graph, considering words as nodes, and the relations between words as the edges.

In figure 27 we present the graph representation of the antonymy relation. It can be observed that there is a massive connected component that contains 16651 nodes. This means that through the antonymy relation, there are 16651 words connected between them in the graph. The remaining graph contains 298 components ranging between 2 and 11 nodes each. In the disposition of the graph drawn, it seems like a kernel and a nebula that surrounds it. Regarding synonymy, similar to the antonymy graph, there is a component that contains 44576 nodes and 9770 components containing between 2 and 20 nodes.

With the information provided by the graph representation, we can conclude that the words in Spanish are highly related to each other in the antonymy and synonymy relations, due to the existence of paths in those relations that connect the majority of them. This can be the reason why the Cardellino and WordNet vocabularies generate similar datasets.

To study the quality of the dataset a sample of 200 tuples (100 antonyms and 100 synonyms shuffled) was taken and then were manually classified by two native speakers, looking for the meaning of unknown words [2].

Once we have the tuples classified, we used Cohen Kappa Score [Cohen, 1960] to calculate the concordance between the classifications obtaining a score of 0.8999, which means a high concordance between the annotators.

When analyzing the annotation mismatches, we found that many of them were cases of tuples that could be classified as antonyms or synonyms depending on the meaning of the words that the annotator considered. An example of this is (*inhospito*, *selvático*) in which one annotator considered *selvático* as a place hostile to life and the other as a place full of life. Other examples are (*enervar*, *espabilar*), (*lateral separado*) and (*entregarse*, *ocuparse*).

8.2.1.1 Dataset Splits

Once the dataset was completed we proceeded to partition it into three different sets for training, validation, and test. These partitions were made with different criteria: one randomly preserving the proportion between antonyms and synonyms, and two lexical splits. A partition with lexical split means that the intersections between the vocabularies of training, validation, and test sets are empty (i.e. each word can only occur in one of them).

We decided to make partitions with lexical split to evaluate the performance of a model on words that have not been seen during training or hyperparameter tuning, setting a more challenging task than the random split. Another motivation is the lexical memorization phenomenon already commented on hypernymy detection [Levy et al., 2015]. This phenomenon seems much more relevant in hypernymy detection because the relation is prone to having one word repeated many times on one side of the tuple (e.g. a massive hyperonym such as *animal*), however, it is also interesting to study the behavior of ASD models in this scenario.

[2] The meanings were obtained through the *Real Academia Española* dictionary (`https://dle.rae.es/`)

Table 17 – Quantity of tuples in the datasets without symmetric counterparts

		Ant	**Syn**	**Total**
Rand	Train	47720	244378	292098
	Val	3592	18394	21986
	Test	12828	65693	78521
Lex Shwartz	Train	28571	132772	161343
	Val	3684	18803	22487
	Test	7498	37503	45001
Lex Graph	Train	27124	77200	104324
	Val	6647	18919	25566
	Test	13357	38017	51374

8.2.1.1.1 Random Split

To create the random partition the library scikit-learn [Pedregosa et al., 2011] was used. This library has methods to create random stratified partitions of datasets, thus we decided to use these methods to generate the random partition of the dataset. As a result of the partition we obtained the train, val, and test sets the number of tuples in each set is shown in table 17

8.2.1.1.2 Lexical Split by Shwartz

This partition of the dataset is inspired by [Shwartz et al., 2016a]. We iterate over the dataset and each tuple is assigned to train, val, or test trying to preserve the 70%, 10%, and 20% of the original dataset, respectively. If a tuple could not be assigned to one of the three sets because it would break the lexical split, then that tuple is discarded.

8.2.1.1.3 Lexical Split by Graph

For the creation of this partition, we focused primarily on the antonymy relation graph. We created an algorithm based on its structure to attempt to reduce the amount of discarded tuples. First, we take the small components of the graph and add them into the test set, because these components contain less than 10% of the antonyms. Then, to divide the big component of the graph, we perform a breadth-first search (BFS) starting on the node with the most edges and stopping once it reaches the target size of the test set. The reached nodes are removed from the graph and the edges from the removed nodes that were not added to the test set are discarded. This process is repeated for the validation set and the remaining graph forms the training set. The test and validation sets have 20% and 10%, respectively, of the antonymy tuples, and the training set contains less than 70% due to the discarded tuples. Finally, synonyms are added, preserving lexical split, to each set to achieve a 25% ratio of antonyms/synonyms.

8.2.2 Experiments

For the experiments, we considered three models: (1) feed-forward neural networks receiving the concatenation of the word vectors as input, (2) siamese networks, and (3) parasiamese networks.

We used FastText [Joulin et al., 2016] word embeddings with a vector dimension of 300. The publicly available pretrained model of FastText was used, which was trained on a Spanish Wikipedia dump.

For each dataset variant and model, a random search was performed to obtain a suitable hyperparameter configuration. This was done using the train and validation datasets with and without symmetric tuples, but in all cases, the test set with symmetric tuples was used.

In total, 33 random searches with 200 trials each were executed. Table 19 contains the results for the executions performed using train and validation datasets without symmetrics, and table 18 contains the results for the executions with symmetrized train and validation datasets. Both tables report precision (P), recall (R), and F1-score (F1) scores.

Table 18 – Results on the three versions of the dataset with symmetric tuples in train and validation partitions. The name of the models are abbreviated where Siam means siamese, PSiam means parasiamese, R means right and L means left. PSiam Pre means pretrained parasiamese.

Models	Random			Lex Shwartz			Lex Graph		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Concat	0.80	0.76	0.78	**0.58**	0.50	0.53	**0.64**	0.33	0.44
Siam	0.41	0.58	0.48	0.30	0.45	0.36	0.26	**1.00**	0.41
PSiam R	0.88	0.85	0.87	0.55	0.57	0.56	0.61	0.45	0.52
PSiam L	0.89	0.87	0.88	0.47	0.65	0.55	0.58	0.40	0.47
PSiam Pre R	0.92	0.88	**0.90**	0.46	**0.66**	0.55	0.61	0.50	**0.55**
PSiam Pre L	0.88	**0.89**	0.89	0.56	0.58	**0.57**	0.60	0.49	0.54

As we can see in table 19 the best model in terms of F1 score is the Parasiamese model across all the dataset partitions. The left or right versions of the parasiamese model are very close to each other in terms of F1 scores, this indicates that the side of the double application of the base network does not significantly affect the results. It can be noticed that the pretrained parasiamese variants perform better than their non-pretrained counterparts, with an improvement in the F1 score, ranging from 0.03 and 0.08.

In the table 18 we can see that almost all the models benefited from the dataset containing the symmetric tuples, the only exception being the pretrained version of parasiamese model. Due to the inherently symmetric nature of the Siamese model, the

Table 19 – Results for the three versions of the dataset on the test partition with symmetric tuples and without symmetric tuples in train and validation. The names of the models are abbreviated, where Siam means siamese, PSiam means parasiamese, R and L mean right and left, respectively, and PSiam Pre means pretrained parasiamese.

Models	Random			Lex Shwartz			Lex Graph		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Concat	0.69	0.66	0.68	**0.59**	0.41	0.48	0.59	0.34	0.43
Siam	0.41	0.58	0.48	0.30	0.45	0.36	0.26	**1.00**	0.41
PSiam R	0.85	0.83	0.84	0.57	0.51	0.53	0.57	0.40	0.47
PSiam L	0.79	0.84	0.81	0.48	**0.62**	0.54	**0.62**	0.37	0.46
PSiam Pre R	**0.93**	0.87	**0.90**	0.52	**0.62**	0.56	0.59	0.47	0.52
PSiam Pre L	0.91	**0.88**	0.89	0.58	0.58	**0.58**	0.58	0.48	0.53

random search was not done in the symmetrized datasets, the results are presented in table 18. The parasiamese model continues to be the best model in terms of F1-score. The versions of the parasiamese models with or without pretraining, are closer in terms of F1-score than in the table 19. The performance gained by the pretrained variants ranges from 0.01 to 0.03 in most cases, except the graph-based partition which shows an improvement of 0.07.

8.2.2.1 Results Analysis

As we can see in table 19 the best model in terms of F1 score is the Parasiamese model across all the dataset partitions. The left or right versions of the parasiamese model are very close to each other in terms of F1 scores, this indicates that the side of the double application of the base network does not significantly affect the results. It can be noticed that the pretrained parasiamese variants perform better than their non-pretrained counterparts, with an improvement in F1 score, ranging from 0.03 and 0.08.

In the table 18 we can see that almost all the models benefited from the dataset containing the symmetric tuples, the only exception being the pretrained version of parasiamese model. Due to the inherently symmetric nature of the siamese model, the random search was not done in the symmetrized datasets, the results are presented in table 18. The parasiamese model continues to be the best in terms of F1-score. The versions of the parasiamese models with or without pretraining, are more close in terms of F1-score than in the table 19. The performance gained by the pretrained variants ranges from 0.01 to 0.03 in most cases, excepting the graph-based partition which shows an improvement of 0.07.

Comparing the results of the same model in both tables, we can see that both concat and non-pretrained parasiamese perform better when trained with symmetric tuples.

The reason for this is that those models are not symmetric. Parasiamese will apply the base network twice only on one side of the tuple, so if we have the symmetric tuple the model will generate a new pair of transformed vectors for the same pair of vectors. In the concat model, the explanation is similar, as it uses the concatenation of the word vectors to create a new vector which is the network input, so the symmetrical is a different input vector.

On the other hand, pretrained parasiamese seems to not improve when symmetric tuples are included. We think that this can be explained by the model pretraining.

9 Conclusion and future work

In this thesis, we studied lexical semantics within word embeddings, using and extending techniques inspired by deep metric learning, that consider the algebraic properties of each lexical relation. We conducted experiments that show the capability of static word embeddings to distinguish different lexical-semantic relations in a supervised machine learning scheme, without including other information such as text patterns or morphology features. Particularly, we take into consideration two lexical semantic tasks from natural language processing: (1) hypernymy detection and (2) antonymy-synonymy discrimination.

Chapter 2 introduces concepts of lexical semantics, with particular emphasis on the considered relations. Chapters 3 and 4 are dedicated to the topics of word embeddings, neural networks and deep metric learning. These chapters form the theoretical basis of the models applied and developed in the remaining chapters.

The main contributions of the thesis are depicted in the remaining chapters.

In chapter 5, we deal with hypernymy detection. In particular, we propose to encode pretrained word embeddings using the order embedding learning approach proposed by [Vendrov et al., 2015a]. An encoder neural network is trained to transform pretrained word vectors to a partial order relation established by the reversed product order.

Chapters 6 and 7 are dedicated to antonymy-synonymy discrimination. In chapter 6, we introduce the parasiamese network. This model is inspired by the siamese network, weakening its tendency to learn transitive relations. Guided by the characteristics of this model, we introduce a pretraining method using synonyms and relation symmetry considerations. The experiments that we carried out with this model in the task of antonymy-synonymy discrimination show promising results.

In chapter 7 we improve the results obtained in antonymy-synonymy discrimination by proposing the repelling parasiamese neural network. This model, based on the results obtained by its predecessor, consists of simultaneously contrasting a siamese and a parasiamese network using the same encoder. We call this model, the repelling parasiamese network. With this model, we propose some variants concerning the branches of the model and the symmetry of the relationship, which yield state-of-the-art results.

Finally, in chapter 8 we address the hypernymy detection and antonymy-synonymy discrimination tasks in Spanish. In this part, we built and published datasets for Spanish in the framework of two theses on computer engineering, along with experiments conducted with these datasets and the models previously mentioned.

In the course of this thesis, the development of Large Language Models using Transformers has been remarkable. LLMs used as chatbots have presented very interesting results and can be used for a great diversity of tasks, leading to a certain predominance in the field of natural language processing and becoming popular in general to society due to commercial products such as ChatGPT. Given this scenario, it is interesting to perform some experiments with LLMs and lexical semantics. As a concluding observation for this thesis, the task of antonym-synonym discrimination is faced using **llama2-7b-chat-hf** through the prompt depicted in appendix C. The results obtained were significantly lower than those obtained using the model proposed in this thesis. However, this is a very initial experiment for ASD task using LLMs, a variety of prompt techniques and more powerful models remain to be considered.

For future work, we propose the following lines of work:

Regarding word embeddings, we propose as future work the study of dynamic word embeddings from the point of view of lexical semantic relations. A main challenge to highlight is the confection of datasets where lexical relations are annotated over the occurrences of words in a corpus instead of words independently of their contexts as in our work.

Concerning lexical relations, particularly antonymy, we propose as a line of future work to take into consideration the different types of antonyms, such as gradable (e.g. hot-cold), complementary (e.g. on-off) and relational antonyms (e.g. teacher-student). These concepts regarding the antonymy relation were introduced in section 2.3.2 but were not considered in the experiments conducted for this thesis.

It is also worth mentioning as a possible future line of work to consider other lexical relations, such as cohyponymy, meronymy, and semantic fields. For instance, cohyponyms are related to antonyms in the sense of mutual exclusion, but unlike antonyms, they are a transitive relationship.

Finally, we propose as future work the application of the results and models obtained in this thesis in tasks such as ontology completion. Also, we propose to use the parasiamese and repelling parasiamese neural networks in other tasks than lexical semantics. For example, the parasiamese models could be used for alliance and opposition detection in general, such as predicting these types of relationships between candidates in political elections or between commercial brands.

Bibliography

[Akbik et al., 2019] Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019). Flair: An easy-to-use framework for state-of-the-art nlp. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59.

[Ali et al., 2019] Ali, M. A., Sun, Y., Zhou, X., Wang, W., and Zhao, X. (2019). Antonym-synonym classification based on new sub-space embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6204–6211.

[Almeida and Xexéo, 2019] Almeida, F. and Xexéo, G. (2019). Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*.

[Aric Hagberg, 2005] Aric Hagberg, Dan Schult, P. S. (2005). Networkx.

[Auer et al., 2007] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg. Springer-Verlag.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Bar and Dershowitz, 2010] Bar, K. and Dershowitz, N. (2010). Using synonyms for arabic-to-english example-based translation. In *Proceedings of the Ninth Conference of the Association for Machine Translation in the Americas (AMTA 9)*.

[Baroni et al., 2012] Baroni, M., Bernardi, R., Do, N., and Shan, C. (2012). Entailment above the word level in distributional semantics. In *EACL*, pages 23–32. The Association for Computer Linguistics.

[Bengio et al., 2000] Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.

[Bergmanis and Goldwater, 2018] Bergmanis, T. and Goldwater, S. (2018). Context sensitive neural lemmatization with Lematus. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1391–1400, New Orleans, Louisiana. Association for Computational Linguistics.

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

[Bertinetto et al., 2016] Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. S. (2016). Fully-convolutional siamese networks for object tracking. *CoRR*, abs/1606.09549.

[Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146.

[Bonato et al., 2017] Bonato, A., Infeld, E., Pokhrel, H., and Prałat, P. (2017). Common adversaries form alliances: modelling complex networks via anti-transitivity. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 16–26. Springer.

[Camacho et al., 2022] Camacho, J., Camera, J., and Etcheverry, M. (2022). Antonymy-synonymy discrimination in spanish with a parasiamese network. In *17th Ibero-American Conference on Artificial Intelligence, Iberamia.*

[Cardellino, 2016] Cardellino, C. (2016). Spanish billion words corpus and embeddings. *Spanish Billion Words Corpus and Embeddings.*

[Che et al., 2018] Che, W., Liu, Y., Wang, Y., Zheng, B., and Liu, T. (2018). Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

[Chen et al., 2017] Chen, Q., Zhu, X., Ling, Z., Inkpen, D., and Wei, S. (2017). Natural language inference with external knowledge. *CoRR*, abs/1711.04289.

[Chi and Zhang, 2018] Chi, Z. and Zhang, B. (2018). A sentence similarity estimation method based on improved siamese network. *Journal of Intelligent Learning Systems and Applications*, 10(04):121.

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

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://keras.io`.

[Clark et al., 2007] Clark, P., Fellbaum, C., and Hobbs, J. (2007). Using and extending wordnet to support question-answering. In *GWC 2008: 4th Global WordNet Conference, Proceedings.*

[Cohen, 1960] Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537.

[Cruse, 1986] Cruse, D. A. (1986). *Lexical semantics.* Cambridge university press.

[Dash et al., 2020] Dash, S., Chowdhury, M. F. M., Gliozzo, A., Mihindukulasooriya, N., and Fauceglia, N. R. (2020). Hypernym detection using strict partial order networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7626–7633.

[Deng et al., 2019] Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4690–4699.

[Eberhard et al., 2021] Eberhard, D. M., Simons, G. F., and (eds.)., C. D. F. (2021). *Ethnologue: Languages of the World.* SIL International, Twenty-fourth edition. Dallas, Texas.

[Edmundson, 1967] Edmundson, H. P. (1967). Axiomatic characterization of synonymy and antonymy. In *COLING 1967 Volume 1: Conference Internationale Sur Le Traitement Automatique Des Langues.*

[Etcheverry and Wonsever, 2016] Etcheverry, M. and Wonsever, D. (2016). Spanish word vectors from wikipedia. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3681–3685.

[Etcheverry and Wonsever, 2019] Etcheverry, M. and Wonsever, D. (2019). Unraveling antonym's word vectors through a siamese-like network. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3297–3307.

[Etcheverry and Wonsever, 2020] Etcheverry, M. and Wonsever, D. (2020). Order embeddings for supervised hypernymy detection. *Computación y Sistemas*, 24(2):565–574.

[Etcheverry and Wonsever, 2023] Etcheverry, M. and Wonsever, D. (2023). Antonymy-synonymy discrimination through repelling parasiamese neural networks. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 01–07. IEEE.

[Faruqui et al., 2014] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. (2014). Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166.*

[Fernández-Montraveta et al., 2008] Fernández-Montraveta, A., Vázquez, G., and Fellbaum, C. (2008). The spanish version of wordnet 3.0.

[Frege, 1948] Frege, G. (1948). Sense and reference. *The philosophical review*, 57(3):209–230.

[Fu et al., 2014] Fu, R., Guo, J., Qin, B., Che, W., Wang, H., and Liu, T. (2014). Learning semantic hierarchies via word embeddings. In *ACL (1)*, pages 1199–1209. The Association for Computer Linguistics.

[Glavaš and Vulić, 2018] Glavaš, G. and Vulić, I. (2018). Discriminating between lexico-semantic relations with the specialization tensor model.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org.

[Gonzalez-Agirre et al., 2012] Gonzalez-Agirre, A., Laparra, E., and Rigau, G. (2012). Multilingual central repository version 3.0. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2525–2529, Istanbul, Turkey. European Language Resources Association (ELRA).

[Hadsell et al., 2006] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE.

[Haghighi et al., 2005] Haghighi, A., Ng, A., and Manning, C. (2005). Robust textual inference via graph matching. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*.

[Harris et al., 1954] Harris, Z. et al. (1954). Distributional hypothesis. *Word World*, 10(23):146–162.

[Hearst, 1992] Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992*, pages 539–545.

[Hill et al., 2014] Hill, F., Cho, K., Jean, S., Devin, C., and Bengio, Y. (2014). Embedding word similarity with neural machine translation. *CoRR*, abs/1412.6448.

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

[Hoffer and Ailon, 2015] Hoffer, E. and Ailon, N. (2015). Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, pages 84–92. Springer.

[Hu et al., 2014] Hu, J., Lu, J., and Tan, Y.-P. (2014). Discriminative deep metric learning for face verification in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1875–1882.

[Jadhav et al., 2020] Jadhav, A., Amir, Y., and Pardos, Z. (2020). Lexical relation mining in neural word embeddings. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1299–1311.

[Joulin et al., 2016] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016). Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651.*

[Jurafsky and Martin, 2009] Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (2Nd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Jurafsky and Martin, 2021] Jurafsky, D. and Martin, J. H. (2021). Speech and language processing (3rd edition, draft). *Available from: https://web. stanford. edu/˜ jurafsky/slp3.*

[Kaya and Bilge, 2019] Kaya, M. and Bilge, H. Ş. (2019). Deep metric learning: a survey. *Symmetry*, 11(9):1066.

[Khatri et al., 2018] Khatri, C., Singh, G., and Parikh, N. (2018). Abstractive and extractive text summarization using document context vector and recurrent neural networks. *CoRR*, abs/1807.08000.

[Kiela et al., 2015] Kiela, D., Hill, F., Clark, S., et al. (2015). Specializing word embeddings for similarity or relatedness. In *EMNLP*, pages 2044–2048.

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

[Lee et al., 2020] Lee, G. W., Etcheverry, M., Sánchez, D. F., and Wonsever, D. (2020). Supervised hypernymy detection in spanish through order embeddings. In *Proceedings of the 7th Workshop on Linked Data in Linguistics (LDL-2020)*, pages 75–81.

[Lenci, 2008] Lenci, A. (2008). Distributional semantics in linguistic and cognitive research. *Italian journal of linguistics*, 20(1):1–31.

[Levy et al., 2015] Levy, O., Remus, S., Biemann, C., and Dagan, I. (2015). Do supervised distributional methods really learn lexical inference relations? In Mihalcea, R., Chai, J. Y., and Sarkar, A., editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 970–976. The Association for Computational Linguistics.

[Liu et al., 2020] Liu, Q., Kusner, M. J., and Blunsom, P. (2020). A survey on contextual embeddings. *arXiv preprint arXiv:2003.07278*.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[McCartney et al., 2022] McCartney, B., Devereux, B., and Martinez-del Rincon, J. (2022). A zero-shot deep metric learning approach to brain–computer interfaces for image retrieval. *Knowledge-Based Systems*, 246:108556.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133.

[Medela and Picón, 2019] Medela, A. and Picón, A. (2019). Constellation loss: Improving the efficiency of deep metric learning loss functions for optimal embedding. *CoRR*, abs/1905.10675.

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

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119.

[Mikolov et al., 2013c] Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751.

[Miller, 1995] Miller, G. A. (1995). Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

[Miller and Charles, 1991] Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). Perceptron: an introduction to computational geometry.

[Mueller and Thyagarajan, 2016] Mueller, J. and Thyagarajan, A. (2016). Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[Nakashole et al., 2012] Nakashole, N., Weikum, G., and Suchanek, F. M. (2012). PATTY: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*, pages 1135–1145. ACL.

[Navigli and Velardi, 2010] Navigli, R. and Velardi, P. (2010). Learning word-class lattices for definition and hypernym extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1318–1327, Uppsala, Sweden. Association for Computational Linguistics.

[Navigli et al., 2011] Navigli, R., Velardi, P., and Faralli, S. (2011). A graph-based algorithm for inducing lexical taxonomies from scratch. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[Neculoiu et al., 2016] Neculoiu, P., Versteegh, M., and Rotaru, M. (2016). Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157.

[Nguyen, 2018] Nguyen, K. A. (2018). Distinguishing antonymy, synonymy and hypernymy with distributional and distributed vector representations and neural networks.

[Nguyen et al., 2017] Nguyen, K. A., Schulte im Walde, S., and Vu, N. T. (2017). Distinguishing antonyms and synonyms in a pattern-based neural network. *CoRR*, abs/1701.02962.

[Nguyen et al., 2016a] Nguyen, K. A., Walde, S. S. i., and Vu, N. T. (2016a). Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. *arXiv preprint arXiv:1605.07766*.

[Nguyen et al., 2016b] Nguyen, K. A., Walde, S. S. i., and Vu, N. T. (2016b). Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. *arXiv preprint arXiv:1605.07766*.

[Ono et al., 2015] Ono, M., Miwa, M., and Sasaki, Y. (2015). Word embedding-based antonym detection using thesauri and distributional information. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 984–989.

[Ortega et al., 2011] Ortega, R. M. A.-a., Aguilar, C. A., VillaseÃ, L., Montes, M., and Sierra, G. (2011). Hacia la identificaciÃde relaciones de hiponimia/hiperonimia en Internet. *Revista signos*, 44:68 – 84.

[Paradis et al., 2009] Paradis, C., Willners, C., and Jones, S. (2009). Good and bad opposites: using textual and experimental techniques to measure antonym canonicity. *The Mental Lexicon*, 4(3):380–429.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Peng et al., 2007] Peng, F., Ahmed, N., Li, X., and Lu, Y. (2007). Context sensitive stemming for web search. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 639–646.

[Peng et al., 2021] Peng, K., Roitberg, A., Schneider, D., Koulakis, M., Yang, K., and Stiefelhagen, R. (2021). Affect-dml: Context-aware one-shot recognition of human affect using deep metric learning. In *2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021)*, pages 1–8. IEEE.

[Pennington et al., 2014a] Pennington, J., Socher, R., and Manning, C. D. (2014a). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Pennington et al., 2014b] Pennington, J., Socher, R., and Manning, C. D. (2014b). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543.

[Peters et al., 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of NAACL*.

[Roller et al., 2014] Roller, S., Erk, K., and Boleda, G. (2014). Inclusive yet selective: Supervised distributional hypernymy detection. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING 2014)*, pages 1025–1036, Dublin, Ireland.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Rubenstein and Goodenough, 1965] Rubenstein, H. and Goodenough, J. B. (1965). Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

[Ruder et al., 2019] Ruder, S., Vulić, I., and Søgaard, A. (2019). A survey of cross-lingual word embedding models. *Journal of Artificial Intelligence Research*, 65:569–631.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., McClelland, J. L., et al. (1986). A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(45-76):26.

[Sahlgren, 2008a] Sahlgren, M. (2008a). The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53.

[Sahlgren, 2008b] Sahlgren, M. (2008b). The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53.

[Santus et al., 2014] Santus, E., Lu, Q., Huang, C.-R., et al. (2014). Taking antonymy mask off in vector space. In *Proceedings of the 28th Pacific Asia Conference on Language, Information and Computing.*

[Saussure, 1959] Saussure, F. d. (1959). Curso de lingüística general (trad, y prólogo de a. alonso), ed. *Losada, Buenos Aires.*

[Scheible et al., 2013] Scheible, S., Im Walde, S. S., and Springorum, S. (2013). Uncovering distributional differences between synonyms and antonyms in a word space model. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 489–497.

[Schwartz et al., 2015] Schwartz, R., Reichart, R., and Rappoport, A. (2015). Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of the nineteenth conference on computational natural language learning*, pages 258–267.

[Shwartz et al., 2016a] Shwartz, V., Goldberg, Y., and Dagan, I. (2016a). Improving hypernymy detection with an integrated path-based and distributional method. *CoRR*, abs/1603.06076.

[Shwartz et al., 2016b] Shwartz, V., Goldberg, Y., and Dagan, I. (2016b). Improving hypernymy detection with an integrated path-based and distributional method. *arXiv preprint arXiv:1603.06076.*

[Snow et al., 2004] Snow, R., Jurafsky, D., and Ng, A. Y. (2004). Learning syntactic patterns for automatic hypernym discovery. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 1297–1304.

[Snow et al., 2006] Snow, R., Vanderwende, L., and Menezes, A. (2006). Effectively using syntax for recognizing false entailment. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 33–40. Association for Computational Linguistics.

[Tissier et al., 2017] Tissier, J., Gravier, C., and Habrard, A. (2017). Dict2vec: Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263.

[Ustalov et al., 2017] Ustalov, D., Arefyev, N., Biemann, C., and Panchenko, A. (2017). Negative sampling improves hypernymy extraction based on projection learning. In *EACL (2)*, pages 543–550. Association for Computational Linguistics.

[Vasudeva et al., 2021] Vasudeva, B., Deora, P., Bhattacharya, S., Pal, U., and Chanda, S. (2021). Loop: Looking for optimal hard negative embeddings for deep metric learning. *CoRR*, abs/2108.09335.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Vendrov et al., 2015a] Vendrov, I., Kiros, R., Fidler, S., and Urtasun, R. (2015a). Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*.

[Vendrov et al., 2015b] Vendrov, I., Kiros, R., Fidler, S., and Urtasun, R. (2015b). Order-embeddings of images and language. *CoRR*, abs/1511.06361.

[Vinyals et al., 2016] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. *Advances in neural information processing systems*, 29:3630–3638.

[Vulić, 2018] Vulić, I. (2018). Injecting lexical contrast into word vectors by guiding vector space specialisation. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 137–143.

[Vulić and Mrkšić, 2017] Vulić, I. and Mrkšić, N. (2017). Specialising word vectors for lexical entailment. *arXiv preprint arXiv:1710.06371*.

[Vulic and Mrksic, 2017] Vulic, I. and Mrksic, N. (2017). Specialising word vectors for lexical entailment. *CoRR*, abs/1710.06371.

[Vylomova et al., 2016] Vylomova, E., Rimell, L., Cohn, T., and Baldwin, T. (2016). Take and took, gaggle and goose, book and read: Evaluating the utility of vector differences for lexical relation learning. In *ACL (1)*. The Association for Computer Linguistics.

[Wang et al., 2019] Wang, J., Wang, K.-C., Law, M. T., Rudzicz, F., and Brudno, M. (2019). Centroid-based deep metric learning for speaker recognition. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3652–3656. IEEE.

[Weeds et al., 2014] Weeds, J., Clarke, D., Reffin, J., Weir, D. J., and Keller, B. (2014). Learning to distinguish hypernyms and co-hyponyms. In *COLING*, pages 2249–2259. ACL.

[Xie and Zeng, 2021] Xie, Z. and Zeng, N. (2021). A mixture-of-experts model for antonym-synonym discrimination. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 558–564.

[Xu et al., 2019] Xu, X., Cao, H., Yang, Y., Yang, E., and Deng, C. (2019). Zero-shot metric learning. In *IJCAI*, pages 3996–4002.

[Yi et al., 2014] Yi, D., Lei, Z., Liao, S., and Li, S. Z. (2014). Deep metric learning for person re-identification. In *2014 22nd International Conference on Pattern Recognition*, pages 34–39. IEEE.

[Zhitomirsky-Geffet and Dagan, 2005] Zhitomirsky-Geffet, M. and Dagan, I. (2005). The distributional inclusion hypotheses and lexical entailment. In *ACL*.

[Zhong et al., 2018] Zhong, D., Yang, Y., and Du, X. (2018). Palmprint recognition using siamese network. In *Chinese Conference on Biometric Recognition*, pages 48–55. Springer.

A Repelling Parasiamese Network Random Search Details

Each run was early stopped with a patience of 5. Every model was trained using adam kingma2014adam and a learning rate of 0.001. The batch size was chosen from [32,64,128,256]. We perform 100 trials for each model.

Regarding the thresholds, they were selected using the following criteria:

- The real interval between 0 and 10 is discretized using a step of 0.1
- 3 values of this sequence are sampled with replacement.
- The sampled values are ordered to become positive, acceptance, and negative thresholds, respectively.

The encoder network (in the half-twin variant, both encoders) stays fixed. It consists of a 3-layered feedforward network with ReLU activation and a hidden layer with an input and output size of 900.

In the following, we detail for each reported result, the best F score obtained on validation, along with the batch size, positive, acceptance, and negative thresholds. In the case of the triplet variant, there is only one threshold, given that the same triplet loss threshold is considered for acceptance.

In the random split dataset, for the contrastive repelling parasiamese network variants we get the following values:

- Vanilla: .903, 32, 2.3, 3.6, 5.2
- Symmetric: .907, 64, 1.8, 2.4, 2.9
- Half-twin: .888, 256, 1.0, 2.2, 2.4
- Half-twin and symmetric: .919, 32, 5.5, 7.2, 9.1

In the random split dataset, for the triplet repelling parasiamese network variants we get the following values:

- Vanilla: .896, 128, 1.3
- Symmetric: .904, 256, 2.4

- Half-twin: .889, 64, 0.6
- Half-twin and symmetric: .912, 256, 2.3

In the lexically split dataset we consider only half-twin symmetric variants with each repelling strategy, and we obtain the following value:

- Contrastive: .814, 256, 6.2, 8.4, 8.9
- Triplet: .714, 128, 7.5

B Antitransitive and anti-Euclidean Relations Equivalence

In a non-symmetric scenario, the parasiamese models will tend to learn a relation that satisfies the following property:

$$a\ R\ b \wedge c\ R\ b \implies a \not R\ c \tag{B.1}$$

We suggest the name of antieuclidean relation. Particularly, left antieuclidean in this case.

It is interesting to note that the anti-Euclidean property is equivalent to antitransitivity. We give the proof for the direct part of left antieuclidean. The other proofs are analogous.

Property. If R is a left antieuclidean relation $\implies$ R is antitransitive

Proof. Let R be not antitransitive, then it must exist a,b,c such that $a\ R\ b \wedge b\ R\ c \wedge a\ R\ c$. But, given that R is left anti-Euclidean and $a\ R\ c \wedge b\ R\ c$, then $a \not R\ b$, which is contradictory.

This result is useful for a better understanding of antintransitive (or antieuclidean) non-symmetric relations.

C Synonymy-Antonymy Discrimination using Llama 2

We use **llama2-7b-chat-hf** with the following prompt:

```
I want you to do a task in lexical semantics. I´ll give you a list
of pairs of words. The two words of a pair are either synonyms or
antonyms. As each word can have more than one meaning, the words will
be antonyms or synonyms in one of these meanings. I want you to read
a list of pairs, where each pair is a list of two words, and tell me,
for each pair, if they are synonyms (S) or antonyms(A)  in one of
their senses. For instance: if the input is [[good, bad], [hot, warm]]
the output should be [[good, bad, A], [hot, warm, S]].
Answer using the same format. Replace X by A if the words are antonyms
or by S in case of synonyms for the following pairs:
[efficient, incompetent, X],
[precipitous, steep, X],
[mystical, mysterious, X],
...
```

We performed every run in batches of 30 pairs to consider the 4096 context tokens of the model. We initially performed two runs to compare results. As we noticed that for some batches we were not getting an answer in the expected format, even though we had previously made changes to the prompt to improve the results, we performed two more runs on those batches to try to get answers in the expected format. Thus, of the **3914** pairs in the test set, we obtained responses for **3702** pairs, and of these pairs, **916** responses differed between runs. For the **2786** pairs where the outputs were the same, the **precision**, **recall** and **f** score were **57.2**, **56.6** and **56.9**, respectively; in a test set of 1423 synonyms and 1363 antonyms. This was the best result obtained using **llama2-7b-chat-hf**. The results for individual runs were slightly lower (between 1 and 5 points) than the previous result. It is worth mentioning that although this result is much lower than those obtained with the methods presented in this thesis, in the case of this experiment using LLMs there is no supervision given by related and unrelated pairs, only the description of the task and a few examples as shown in the prompt above.