
# University of Padova

Department of Mathematics

*Master Thesis in Data Science*


# Word2Box: Analysis and Exploration of a Geometric Word Embedding Algorithm


*Supervisor*
Professor Giorgio Satta
University of Padova

*Master Candidate*
Chiara Bigarella
2004248


*Academic Year*
2023-2024

# Abstract

Vector word embeddings are powerful tools, but they do not naturally express uncertainty about the target concept. Furthermore, they do not naturally model asymmetric relations since they are compared using symmetric distance functions, such as dot product, cosine similarity, or Euclidean distance. This has led to exploring new ways of representing words based on Geometric Representations. *Word2Box* is a geometric extension of Word2Vec that learns region-based word representations that allow to perform set-theoretic operations between words. In fact, it is a fuzzy set interpretation of box embeddings, where each word is represented by an n-dimensional hyperrectangle, also called "box". This innovative approach enables modelling uncertainty and asymmetric relationships between words, offering a significant improvement over traditional vector embeddings. By leveraging the properties of fuzzy sets and geometric shapes, Word2Box provides a more nuanced and flexible representation of word meanings, allowing to capture complex linguistic relationships that cannot be represented with point-based embeddings. This thesis presents the analysis of the Word2Box algorithm and provides a comprehensive explanation of the codebase, including some considerations on its effectiveness in capturing semantic relationships and its potential to enhance natural language processing tasks.

# Contents

# Listing of figures

# Listing of tables

# Listing of acronyms

**CBOW** . . . . . Continuous Bag of Words

**K-L divergence** . Kullback–Leibler divergence

**LLM** . . . . . Large Language Model

**NLP** . . . . . Natural Language Processing

**NEG** . . . . . Negative Sampling

**NNLM** . . . . . Neural Net Language Model

**NCE** . . . . . Noise Contrastive Estimation

**OE** . . . . . Order Embedding

**POE** . . . . . Probabilistic Order Embedding

**SGNS** . . . . . SkipGram with Negative Sampling

**UTF-8** . . . . . Unicode Transformation Format - 8-bit

**XML** . . . . . Extensible Markup Language

# 1 Introduction

In this chapter, I will introduce some concepts that are necessary to fully understand the work presented in my dissertation. I will start by providing an overview of Natural Language Processing (NLP). Then, I will proceed with a brief introduction to Word Embeddings. I will explain the differences between Static and Dynamic Word Embeddings, and I will linger on one of the most famous static word embedding models, Word2Vec, and its two variants: SkipGram and CBOW.

## 1.1 Natural Language Processing

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on the interaction between computers and humans through human language. The ultimate objective of NLP is to allow machines to read, process, and derive meaning from written text.

As an interdisciplinary area of research, NLP combines several fields, such as cognitive sciences, computer science, information theory, linguistics, machine learning and mathematical logic. Its multifaceted nature allows researchers to approach the challenges using the knowledge collected in a range of research fields in order to tackle a wide variety of tasks, including but not limited to Information Retrieval, Machine Translation, Question Answering, Sentiment Analysis, Speech Recognition, Text Classification, and Word Disambiguation.

Information Retrieval Information retrieval involves the process of extracting relevant information from a vast amount of unstructured text data. This task is fundamental in various applications, such as search engines, where the goal is to find documents or data that satisfy the user's query. The challenge lies in efficiently searching through, often large, datasets to find pieces of information that are relevant to the specific needs or questions of the user, requiring sophisticated algorithms to index, search, and rank results based on relevance.

Machine Translation Machine translation is the automated process of translating text from one language to another. This field has seen significant advancements with the advent of neural networks and deep learning techniques, enabling more accurate and contextually appropriate translations. Despite these advancements, machine translation continues to face challenges, particularly with languages that have significant structural differences or when translating idioms and culturally specific references.

Question Answering Question answering systems aim to automatically provide answers to questions posed in natural language. These systems must understand the question's intent, search through a potentially vast knowledge base or the internet to find relevant information, and then present that information in a concise and understandable answer. The complexity of natural language and the broad range of possible questions make this a challenging area of NLP.

Sentiment Analysis Sentiment analysis involves analyzing text to determine the sentiment expressed within it, whether positive, negative, or neutral. This task is widely used in monitoring social media, customer feedback, and market research to gauge public opinion or emotional responses. The subtlety of language and the presence of sarcasm or irony make sentiment analysis a particularly challenging task, as the same words can be used to convey different sentiments in different contexts.

Speech Recognition Speech recognition technology converts spoken language into written text. This involves recognizing the words being spoken and understanding the speaker's intent, which can be complicated by accents, speech impediments, background noise, and the natural variability in human speech. Advances in machine learning and natural language processing have significantly improved speech recognition accuracy, making it a crucial technology in virtual assistants and voice-controlled applications.

Text Classification Text classification involves assigning predefined categories or tags to text based on its content. This task is essential for organizing and categorizing content for easier retrieval, such as filtering spam emails, news article categorization, or sentiment analysis. The challenge in text classification lies in understanding the nuances of language and the context in which words are used to accurately categorize the text.

Word Disambiguation Word disambiguation is the process of determining which meaning of a word is being used in a given context. This is crucial for many NLP tasks, as many words have multiple meanings depending on their usage. Effective word disambiguation requires understanding the context in which a word appears and how it relates to the surrounding text, a task that requires deep linguistic analysis and contextual understanding.

What emerges from the above examples is that the complexity of human languages poses several challenges to NLP. In the following paragraphs, I will briefly introduce some of the most common ones.

Understanding context Machines often struggle with understanding the context in which words or phrases are used, a critical aspect of language comprehension. Context provides essential background that influences the meaning of linguistic expressions. For humans, this understanding is intuitive, built on a complex web of cultural, situational, and historical knowledge. Machines, however, must rely on algorithms and models to infer context, a task that remains a significant challenge in the field of natural language processing (NLP).

Language ambiguity Ambiguity in language presents a considerable challenge for computational systems. A single word can carry multiple meanings, and its intended interpretation often depends on the context in which it is used. For instance, the word "bank" can refer to a financial institution or the side of a river, with the correct meaning only discernible through context. This ambiguity requires machines to not only process the language but also understand the nuances and subtleties that dictate meaning in human communication.

Understanding complex sentences Complex sentences, characterized by their multiple clauses and layers of meaning, pose a significant challenge for machine comprehension. These sentences can express nuanced ideas, conditional information, or relationships between concepts that require a deep understanding of grammar and syntax. For machines,

parsing these complex structures and extracting the intended meaning involves sophisticated algorithms that can analyze and interpret the various components and their interrelations.

Sarcasm, irony, and humor Detecting and understanding sarcasm, irony, and humor in text is a daunting task for machines. These forms of communication often rely on subtle cues, tone, or context that machines find difficult to interpret. The complexity lies in the fact that these expressions frequently convey meanings that are opposite to their literal interpretation, requiring a level of semantic and pragmatic analysis that goes beyond the current capabilities of many NLP systems.

Language evolution The constant evolution of language creates a moving target for NLP techniques. New words, phrases, and changes in meanings emerge as cultures evolve, technologies advance, and societal norms shift. This dynamism requires NLP systems to be adaptable and continually updated to understand contemporary language use. The challenge is not only in tracking these changes but also in predicting and responding to new linguistic trends, ensuring that machines can understand and process language as it is used in real-time.

Most of these challenges stem from properties of natural languages, such as ambiguity, compositionality, recursion, and hidden structure.

Ambiguity One of the fundamental challenges in understanding natural languages is ambiguity. This property manifests in various forms, such as phonetic ambiguity where the same sound can lead to different transcriptions, lexical ambiguity where words can carry multiple meanings depending on the context, and syntactic ambiguity that allows sentences to be interpreted in more than one way. Ambiguity challenges both human understanding and computational language processing, requiring sophisticated mechanisms for accurate interpretation.

Compositionality The principle of compositionality plays a crucial role in the construction and comprehension of linguistic expressions. It posits that the meaning of complex expressions can be derived from the meanings of their constituent expressions and the rules used to combine them. This hierarchical structuring allows for the creation of a vast array of sentences from a limited set of words and grammatical rules, enabling the expression of an infinite number of ideas and concepts.

Recursion Recursion is a property of language that enables the creation of new sentences by repeatedly applying grammatical rules. This feature allows for the generation of an infinite number of sentences from a finite set of elements, facilitating endless creativity and variation in language use. Through recursion, languages can build complex structures by nesting phrases within phrases, each layer adding to the overall meaning of the sentence.

Hidden Structure The concept of hidden structure in language refers to the underlying grammatical and semantic frameworks that govern sentence formation and interpretation. This invisible scaffold ensures coherence and meaning, guiding how words and phrases are assembled. However, it also implies that even minor alterations in a sentence's structure can significantly impact its overall meaning, highlighting the complexity and delicacy of language processing. Understanding and modeling this hidden structure is a significant challenge in linguistics and natural language processing.

In conclusion, the field of natural language processing faces a myriad of challenges that stem from the inherent properties of human language. The issues of ambiguity, compositionality, and recursion highlight the complexity of language and the sophisticated understanding required to process it effectively. As language continues to evolve, NLP systems must also advance, adapting to new linguistic trends and the ever-changing landscape of human communication. The ability to accurately interpret and respond to natural language is crucial for the development of intelligent systems that can seamlessly interact with humans. Thus, addressing these challenges is not only a matter of technological advancement but also a step towards more natural and intuitive human-computer interactions.

## 1.2 Word Embeddings

Word Embeddings are tools for representing words in a mathematical form, making it possible for NLP algorithms to process them. In fact, word embedding models map discrete words to numerical vectors in a continuous vector space.

These models originated from two main ideas: the first idea comes from Osgood *et al.* (1957), who suggested representing words using points in three-dimensional space; the second idea comes from linguists like Joos (1950), Harris (1954), and Firth (1957) who proposed to define the meaning of a word by its distribution in language use [1]. The latter idea is

called *distributional hypothesis* and states that words that occur in similar contexts tend to have similar meanings.

The first word embedding models used to represent words using frequency-based methods, thus creating sparse, high-dimensional embedding vectors. To address the problems caused by the so-called *curse of dimensionality* and the sparsity of the embedding vectors, Bengio *et al.* (2003) [2] proposed to use neural networks to learn dense, low-dimensional embeddings from large text corpora using unsupervised learning. This new approach has proven to significantly improve the outcomes of downstream NLP tasks, due to its ability to capture the semantic and syntactic properties of words.

A common way of classifying the existing types of Word Embeddings consists of dividing them into two categories: Static Word Embeddings and Dynamic Word Embeddings. I will briefly introduce both of them in the following sections. However, since this dissertation focuses on a static word embedding algorithm, I will delve deeper into this type of word embeddings.

### 1.2.1 Static Word Embeddings

Static Word Embeddings are vector representations of words that univocally associate a fixed, precomputed embedding vector to each word *type* * in the vocabulary $\mathcal{V}$. This means that the same word will always have the same embedding, regardless of the context in which it appears.

This characteristic makes static embedding models easy to implement, train and use. In fact, they are usually based on matrix factorization or on shallow neural networks. However, it also constitutes a big limitation when it comes to word sense disambiguation and *polysemic word* * representation. For example, the word "bank" would have the same embedding vector whether it's used in the context of a financial institution or in the context of the side of a river, making it impossible for the model to distinguish between the two meanings. Another issue is represented by the incapability of static embedding models to generate an embedding for words that are not present in the training corpus. This makes dealing with out-of-vocabulary words very problematic.

Learning from a fixed corpus, these embeddings are designed to approximate the statis-

*Word *types* are unique words in the vocabulary, while word *tokens* are the occurrences of these words in a text.

**Polysemic words* are words that have multiple meanings. An example of polysemic word is the word "bank". This word can refer to a financial institution or to the side of a river.

tical patterns of word co-occurrences in the corpus. Several embedding algorithms have been developed for the purpose, each one of them exploiting different strategies. In section 1.3, I will talk about **Word2Vec**, which is the most famous and widely used model for learning static word embeddings. Thereafter, I will explain how its two variants, **SkipGram** and **CBOW** work. CBOW is particularly relevant for the work presented in this dissertation, as it is the model on which the analyzed algorithm is based.

### 1.2.2 Dynamic Word Embeddings

Dynamic Word Embeddings, also called **Contextualized Word Embeddings**, associate each *token* (i.e. each word occurrence) with a different embedding vector that takes into account the context in which the word appears. Thus, the same word *type* will have different embeddings, depending on the neighbouring words (i.e. the *context* [†]) of its occurrences.

For this reason, dynamic embeddings are able to deal with *polysemic words* and are successful in word sense disambiguation tasks.

On the other hand, they are computationally more expensive than static embeddings, since they are computed dynamically (i.e. at the moment of the task execution) by Large Language Models (LLMs) that have been pre-trained on large text corpora and then fine-tuned on the downstream NLP task at hand.

## 1.3 Word2Vec

Word2Vec consists of two architectures for learning static word embeddings, proposed by Mikolov *et al.* (2013). The primary goal of these architectures is to minimize the computational complexity of the training process, while still producing high-quality word embeddings. To achieve this, the authors proposed two models based on a feedforward neural network, where they removed the non-linear hidden layer. This way, they were able to obtain large improvements in accuracy at a much lower computational cost [3].

Word2Vec embeddings are learnt in a way that the embeddings of words with similar meanings (e.g. "king" and "queen") are closer than the embeddings of words with completly different meanings (e.g. "king" and "carpet"). Nontheless, Word2Vec embeddings allow to

[†]The *target* word is the word for which we want to compute the embeddings, while the *context* words are the words surrounding the target word.

compute word analogies using simple mathematic operations on vectors. For example, king — man + woman = queen [3].

In the following sections, I will explain how the two Word2Vec architectures, CBOW and SkipGram, work. Then, I will introduce the concept of *Negative Sampling* and how it has been combined with SkipGram to form a new model called **SkipGram with Negative Sampling (SGNS)**.

### 1.3.1 CBOW

The first architecture that has been proposed is CBOW, which stands for Continuous Bag of Words. The name "Bag Of Words" comes from the fact that, as in standard bag-of-words models, the order of *context* words does not influence the resulting embedding. However, unlike standard bag-of-words models, CBOW uses a continuous distributed representation of the *context* words to predict the *target* word [3].

Figure 1.1 shows a visual representation of the CBOW architecture. It is similar to the feedforward NNLM (Neural Net Language Model) model proposed by Bengio *et al.* (2003) in [2]. However, the non-linear hidden layer has been removed in order to improve the computational efficiency. The input words (e.g. the context) are encoded using one-hot encoding, and are selected using a window of size $L$, centered in the target word. As you can see in Figure 1.1, the projection layer is shared for all words, thus their vectors are averaged [3]. Finally, a *hierarchical softmax* [‡] activation function is used in the output layer to predict the target word.

### 1.3.2 SkipGram

SkipGram is similar to the CBOW architecture, but instead of predicting the *target* word based on the *contex*, it predicts the *context* words given the *target* word [3].

Figure 1.1 shows a visual representation of the SkipGram architecture. The context words are predicted whithin a window of size $L$, centered in the target word, using a hierarchical softmax activation function in the output layer [3].

[‡] *Hierarchical softmax* is a computationally efficient approximation of the full softmax activation function.

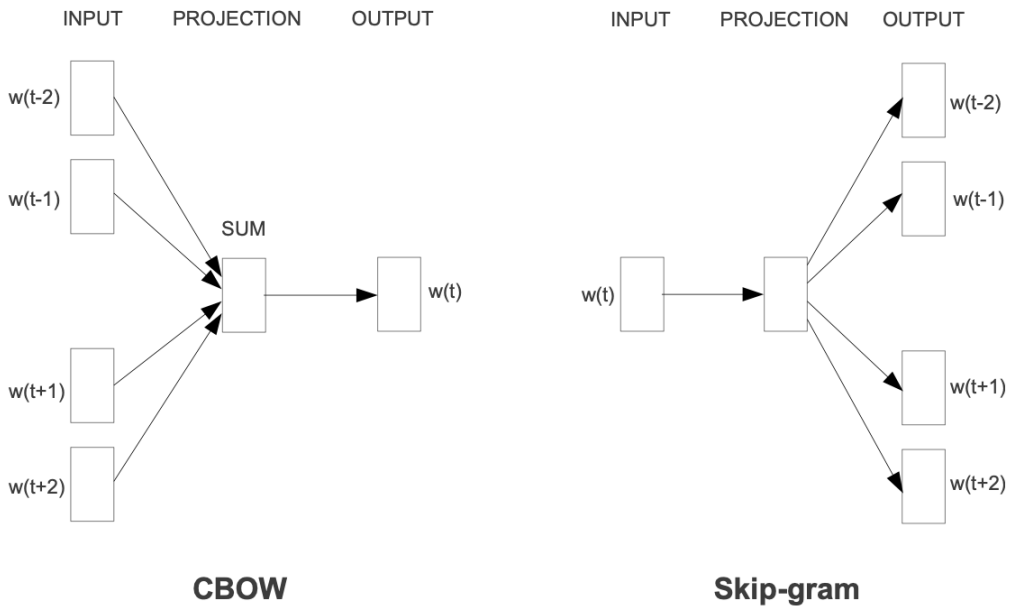



**Figure 1.1:** A comparison between CBOW and SkipGram architectures. This picture is taken from paper [3].

### 1.3.3 Negative Sampling

Negative Sampling (NEG) is a NLP technique inspired by *Noise Contrastive Estimation* (NCE), which states that a good model should be able to distinguish between data and noise using logistic regression [4]. To do so, the dataset is augmented with negative examples drawn from a noise distribution $P_n(w)$. There are $k$ negative examples for each data sample and the optimal value of $k$ depends on the dataset size [4]:

- If the dataset is small, $k$ should be in the range of 5-20.
- If the dataset is large, $k$ should be in the range of 2-5.

#### SkipGram with Negative Sampling

*SkipGram with Negative Sampling* (SGNS) is a variant of the SkipGram model that implements the Negative Sampling technique together with a logistic regression classifier trained on a binary classification task [1]. The goal is to discern between the target word $w_0$ and a set of

negative examples drawn from a noise distribution $P_n(w)$ [4]. The NEG objective function is a simplified version of the NCE objective function, however it guarantees that the quality of the learned vectors is not compromised [4].

The advantages of using Negative Sampling to train the Skip-Gram model are the following [4]:

- First of all, it reduces the computational complexity of the training, making it faster.
- Secondly, it improves the vector representation accuracy for rare words.

Finally, Negative Sampling can be used also with the CBOW model [4], as it will be shown later in this dissertation.

# 2

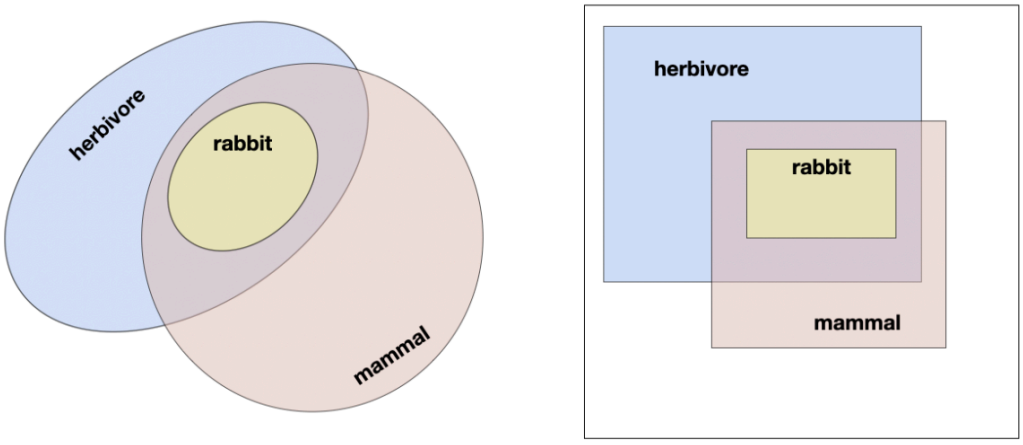
# Geometric Representations

Vector word embeddings are powerful tools, but they do not naturally express uncertainty about the target concept. Furthermore, they do not naturally model asymmetric relations since they are compared using symmetric distance functions, such as dot product, cosine similarity, or Euclidean distance [5]. This has led to the exploration of new ways of representing words. In this chapter, I will define the concept of Geometric Representation and its advantages compared to other approaces. Subsequently, I will introduce the main models based on this idea and, for each model, I will list the pros and cons.

Geometric Representations are ways to represent entities as geometric objects in a high-dimensional space. Different kinds of geometric embeddings have been developed, based on different geometric objects. Nontheless, they all share the ability to represent polysemy and asymmetry, and to answer complex queries [6]. Furthermore, they have a geometry that is more suitable to express relationships in the domain [7] and provide a strong inductive bias when labelled data is scarce [6].

Figure 2.1 displays two examples of geometric representations of words in a $2D$ space. Figure 2.1 (a) shows the $2D$ representation of Gaussian embeddings, while Figure 2.1 (b) shows the $2D$ representation of Box embeddings. It is clearly visible how entailment can be represented as inclusion among geometric objects [7].

In the following sections, we will explore the some geometric representation models useful for the analysis of the algorithm analyzed in this dissertation.




(a) Gaussian embeddings (b) Box embeddings

**Figure 2.1:** Geometric representations of word embeddings in a 2D space. The embeddings are computed using 2 different algorithms: (a) Gaussian Embeddings; (b) Box Embeddings. This picture is taken from Vilnis doctoral thesis [6].

## 2.1 Gaussian Embeddings

Gaussian Embeddings is a geometric word embedding algorithm proposed in 2015 by Vilnis and McCallum in [5]. The goal of the algorithm is to map words to Gaussian distributions in an infinite-dimensional space, in such a way that the linguistic properties of the words are represented by the properties of and the relationships between the distributions [5].

The model is trained on a set of positive and negatives examples using *KL-divergence* between Gaussian distributions as the energy function, and *max-margin* as the loss function [5]. Max-margin is a ranking-based loss that pushes scores of positive pairs above negatives by a margin. Thus, the model learns Gaussian distributional embeddings to predict the context words given the target word and rank the context words by their likelihood [5].

More formally, the KL-divergence between Gaussian distributions defines the following negative energy that the model tries to optimize:

$$
\begin{aligned}
-E(P_i, P_j) = D_{KL}(\mathcal{N}_j||\mathcal{N}_i) &= \int_{x\in\mathbb{R}^n} \mathcal{N}(x;\mu_i,\Sigma_i) log \frac{\mathcal{N}(x;\mu_j,\Sigma_j)}{\mathcal{N}(x;\mu_i,\Sigma_i)} dx \\
&= \frac{1}{2}\left( tr(\Sigma_i^{-1}\Sigma_j) + (\mu_i-\mu_j)^T\Sigma_i^{-1}(\mu_i-\mu_j) - d - \log\frac{|\Sigma_j|}{|\Sigma_i|}\right)
\end{aligned}
\tag{2.1}
$$

The advantages of representing words through Gaussian embeddings are mostly two:

- Probability densities better represent asymmetry and uncertainty on the representation, compared to point vectors [5].
- KL-divergence is straightforward to compute, naturally asymmetric and has a geometric interpretation as an inclusion between families of ellipses [5].

However, Gaussian Embeddings are not a truly probabilistic model. In fact, this kind of geometric representation does not model asymmetry and relations in terms of probabilities, but in terms of asymmetric comparison functions, such as KL-divergence [8].

## 2.2 Order Embeddings

Order Embeddings (OE) is a method introduced in 2016 by Vendrov *et al.* [9] to learn ordered representations in order to explicitly model the partial order structure of hierarchical data.

The goal of OE is to learn an order-preserving mapping from the partially-ordered data domain to some other partially-ordered space that will enable generalization. This provides the model with the capacity of solving *Partial Order Completion* tasks, such as hypernym or entailment prediction [9]. Furthemore, OE also represents a standard constructions of a vector *lattice* * in which the operations of *meet* † and *join* ‡ are defined as the pointwise maximum and minimum of two vectors, respectively [8].

More formally, a function $f : (X, \preceq_X) \to (Y, \preceq_Y)$ is an orded embedding if $\forall u, v \in X$, $u \preceq_X v \Leftrightarrow f(u) \preceq_Y f(v)$. The choice of $Y$ and $\preceq_Y$ depends on the application [9].

*A *lattice* is a partially ordered set where any subset of elements has a single unique least upper bound and greatest lower bound. It is equipped with two binary operations: $\vee$ (join) and $\wedge$ (meet) [6].

†*Meet* is a binary operation that computes the greates lower bound of two elements $a, b \in P$ and it is denoted as $a \wedge b$ [6].

‡*Join* is a binary operation that computes the least upper bound of two elements $a, b \in P$ and it is denoted as $a \vee b$ [6].

The main advantage of representing entities through Order Embeddings is its ability to impose transitivity and antisymmetry of the partial order, making it suitable to model transitive relational data, e.g. entailment graphs [9]. However, this model learns a deterministic knowledge base, thus posing a limit to the expressiveness of queries and to the usage of uncertainty during learning and inference [8].

## 2.3 Probabilistic Order Embeddings

Probabilistic Order Embeddings (POE) have been proposed in 2017 by Lai and Hockenmaier in [10] as part of a framework that aims to capture the *denotational probabilities* of words and phrases. Denotational probabilities are based on the truth-conditional semantics concept of denotanion of a declarative sentence $s$, which refers to the set of possible worlds in which the sentence is true, and are computed as the number of images in the visual denotation of $s$ over the size of the corpus [10]. These probabilities can be used to improve the outcome of textual entailment prediction tasks, where the goal is to predict whether a sentence is true, false or neither given a premise sentence [10].

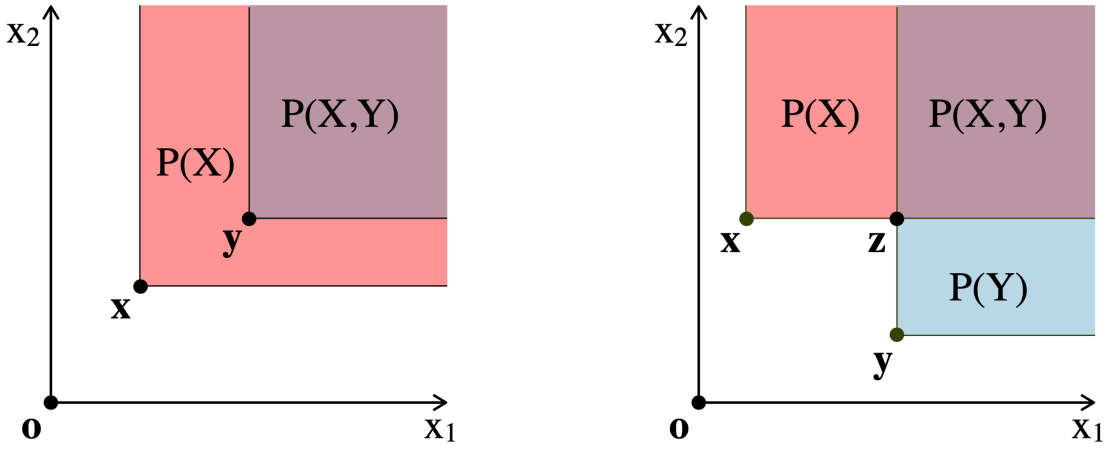



**Figure 2.2:** A 2D embedding space that expresses the individual probabilities of events $X$ and $Y$ and their joint probability $P(X, Y)$. This picture is taken from paper [10].

In more formal words, POE models learn a mapping from a phrase $x$ to a n-dimensional vector $\mathbf{x} \in \mathbb{R}^n_+$ such that the vector $\mathbf{x} = (x_1, ..., x_n)$ defines the denotational probability of $x$ as $P_{[\![}(x) = exp(-\sum_i x_i)$, which is a Bernoulli random variable [10]. Figure 2.2 shows that a phrase vector $\mathbf{x}$ defines a region in the embedding space that is proportional to the

phrase's denotational probability. In fatc, the Bernoulli probability of each concept or joint set of concepts is given by the volume of this region under the exponential measure, thus offering a probabilistic interpretation of the cone lattice's volume [8].

Representing entities through Probabilistic Order Embeddings has the following advantages:

- POE is a valid probability distribution over concepts [8];
- It is able to model asymmetric transitive relations [8];
- And finally, it has the ability to perform rich joint and conditional queries over arbitrary sets of concepts [8].

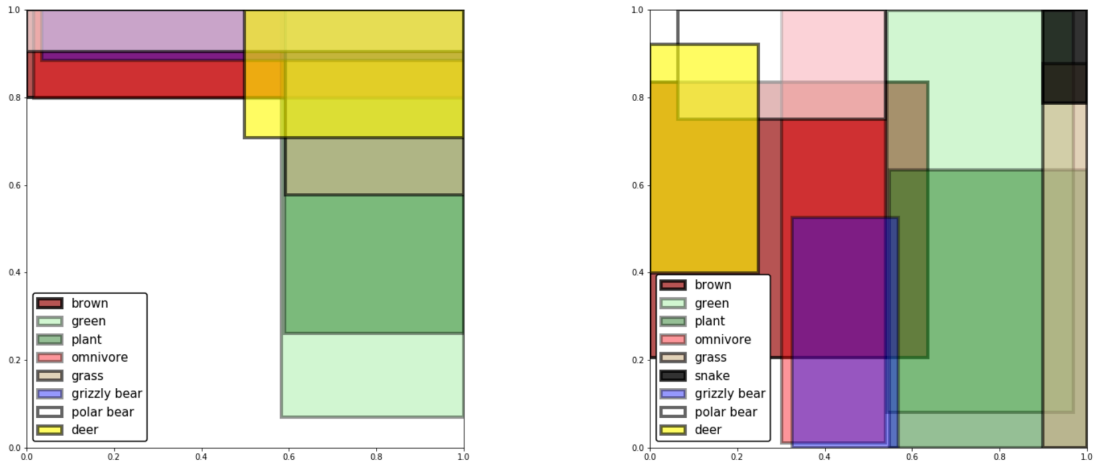
However, POE has also a limitation: it is not able to model negative correlations between concepts [8].

## 2.4 Probabilistic Box Embeddings

Probabilistic Box Embeddings are a region-based representation proposed in 2018 by Vilnis *et al.* in [8], similar to POE of Lai *et al.*. Likewise POE, also this model adopts a dual approach: on one hand, a probabilistic approach is used to model uncertainty about relationships and attributes, while, on the other hand, a geometric approach is used to base the probability on a latent space of geometric objects that have natural structural biases for modeling transitive, asymmetric relations [8].

Box Embeddings aims at representing objects, concepts, and events as high-dimentional products-of-intervals (e.g. hyperrectangles, also called *boxes*), where the event's unary probability comes from the box volume and the joint probabilities of events come from the box overlaps. Box Embeddings is defined as a pair of vectors $(\mathbf{x}^-, \mathbf{x}^+)$ such that $\mathbf{x}^-, \mathbf{x}^+ \in [0,1]^n$, where $\mathbf{x}^-$ and $\mathbf{x}^+$ represent the minimum and maximum at each coordinate, respectively [8]. The model is trained by minimizing the weighted cross-entropy of both the unary marginals and the pairwise conditional probabilities [8].

In Figure 2.3, we can see a comparison between POE lattices and the box lattices produced by the Box Embeddings model. The examples are in a $2D$ space to allow for visualization, but both models can be extended to higher dimensions [8]. We can clearly see that POE is anchoring the lattices to a corner, while Box Embeddings are able to occupy the entire




(a) POE lattice (b) Box lattice

**Figure 2.3:** A 2D comparison between POE lattices (a) and Box lattices (b). This picture is taken from paper [8].

space, thus proving more expressivity and a richer capacity to model positive and negative correlations [8].

We can summarize the main advantages of using a Probabilistic Box Embeddings model as the following:

- First of all, the model is able to capture anticorrelation and disjoint concepts [8];
- Secondly, it has the ability to perform rich joint and conditional queries over arbitrary sets of concepts [8];
- And finally, it is able to learn from and predict calibrated uncertainty [8].

However, this model has also some limitations:

- First of all, it cannot represent all possible probability distributions or concepts as embedded box. For example, the complement of a box is not a box. Nontheless, the cases in which representing true complements is required are very rare [8].
- Secondly, requiring the total probability mass covered by boxes to equal 1 is a difficult box-packing problem and not generally possible [6].
- Thirdly, learning is not straightforward. In fact, the gradient of the joint probability with respect to the box embedding parameters is nonzero when boxes intersect, but it has zero derivative otherwise [8]. Thus, the model is unsuitable for optimization and the author proposes to optimize a lower bound instead [8].

# 3
# Word2Box

In this chapter, I will present the Word2Box model, a geometric extension of Word2Vec that learns region-based word representations that allow to perform set-theoretic operations between words [11]. I will introduce the concepts of Gumbel Boxes and Fuzzy Sets, and I will explain how they are combined together in Word2Box to offer an improvement to the box embeddings proposed by Vilnis *et al.* (2018).

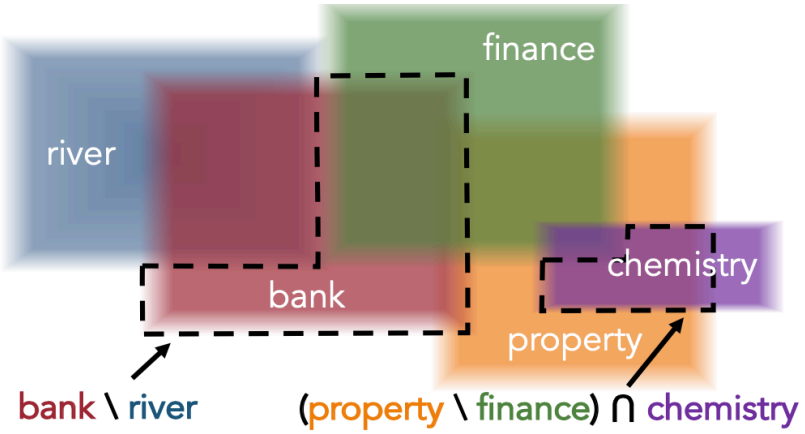



**Figure 3.1:** A 2D representation of the word embeddings generated by the Word2Box algorithm and examples of set-theoretic operations that can be performed on these embeddings. This picture is taken from paper [11].

Word2Box has been proposed by Dasgupta *et al.* (2022) in [11] as a fuzzy set interpretation of box embeddings for words. The model is based on a variant of box embeddings called

GumbelBox, introduced in [7]. Each word is represented by an $n$-dimensional hyperrectangle, also called box. However, unlike the original box embeddings and the GumbelBox model, the training objective of Word2Box is set-theoretic. This choice is motivated by the fact that many relationships between words can be naturally expressed set-theoretically [11]. As shown in figure 3.1, boxes are intepreted as *fuzzy sets*, making it possible to perform set-theoretic operations such as union, intersection, and difference between words. This allows to capture relationships between words that cannot be grasped with traditional methods like Word2Vec [11].

## 3.1 Gumbel Boxes

As anticipated in Section 2.4, Vilnis et al. (2018) [8] defined the box embeddings of an element $a$ as a Cartesian product of intervals:

$$Box(\mathbf{x}) := \prod_{i=1}^{d} [x_i^-, x_i^+] = [x_1^-, x_1^+] \times \ldots \times [x_d^-, x_d^+] \subseteq \mathbb{R}^d \tag{3.1}$$

where $\mathbf{x} = (x_1^-, ..., x_d^-, x_1^+, \ldots, x_d^+) \in \mathbb{R}^{2d}$ and $x_i^- \leq x_i^+$.

The unary probability of an event $a$ is defined as the volume of the box $Box(\mathbf{x})$ associated with it:

$$P(a) = |Box(\mathbf{x})| = \prod_{i=1}^{d} max(0, x_i^+ - x_i^-) \tag{3.2}$$

While the joint probability of two events $a, b$ is defined as the volume of the intersection of their boxes, i. e. $Box(\mathbf{x})$ and $Box(\mathbf{y})$ respectively:

$$P(a, b) = |Box(\mathbf{x}) \cap Box(\mathbf{y})| = |\prod_{i=1}^{d} [max(x_i^-, y_i^-), min(x_i^+, y_i^+)]| \tag{3.3}$$

However, these hard $max$ and $min$ operations are not differentiable, resulting in large areas of the parameter space where the gradient is zero [11]. To address this problem, Dasgupta *et al.* (2020) [7] proposed to model the corners of the boxes $\{x_i^{\pm}\}$ with Gumbel random variables $\{X_i^{\pm}\}$ and named the resulting boxes Gumbel Boxes. The Gumbel distribution

$$f(x; \mu, \beta) = \frac{1}{\beta} \exp\left(-\frac{x - \mu}{\beta} - e^{-\frac{x-\mu}{\beta}}\right) \tag{3.4}$$

is min/max stable, meaning that the minimum and maximum of two Gumbel random variables are also Gumbel distributed [7]. This property assures that the intersection between two Gumbel boxes is still a Gumbel box [11]. Furthermore, Dasgupta *et al.* (2020) [7] showed that the GumbelBox method outperforms other variants of box embeddings, and stated that it should be the default method for training probabilistic box embeddings [7]. Finally, Boratko *et al.* (2021) [12] proved that Gumbel boxes embedded in a space of finite measure have a rigorous probabilistic interpretation.

## 3.2 Fuzzy Sets

Fuzzy sets (also called *uncertain sets*) are sets whose elements have degrees of membership, and therefore they can be considered a generalization of the classical definition of set. In fact, in classical set theory an element either belongs or does not belong to the set. By contrast, fuzzy set theory permits the gradual assessment of the membership of elements in a set, using a membership function valued in $[0, 1]$.

More formally, a fuzzy set is a pair $(U, m)$ where $U$ is a set (often required to be non-empty) and $m : U \to [0, 1]$ is a membership function. Let $x \in U$. Then $x$ is called:

- **not included** in the fuzzy set $(U, m)$ if $m(x) = 0$;
- **fully included** in $(U, m)$ if $m(x) = 1$;
- **partially included** in $(U, m)$ if $0 < m(x) < 1$.

The definition of set intersection for fuzzy sets depends on the concept of *t-norm*, a binary operation $T : [0, 1] \times [0, 1] \to [0, 1]$ that satisfies the commutative, associative and monotonic properties. Examples of t-norms are the minimum and product operations. Given any t-norm $T$, the intersection of fuzzy sets $A$ and $B$ has membership function $m_{A\cap B}(x) = T(m_A(x), m_B(x))$ [11]. Likewise, the definition of the union between fuzzy sets depends on the concept of *t-conorm*, a binary operation defined as $\perp\ (a, b) = 1 - T(1 - a, 1 - b)$. Each t-norm has a corresponding t-conorm; for example, the t-conorm of the minimum t-norm is the maximum operation. The union of fuzzy sets $A$ and $B$ has membership function $m_{A\cup B}(x) =\perp\ (m_A(x), m_B(x))$ [11]. Finally, the complement of a fuzzy set $A$ has membership function $m_{A^c}(x) = 1 - m_A(x)$ [11].

One of the main reason to use fuzzy sets instead of the classical sets is that it is not possible to learn a set representation in a gradient-based model using a hard membership function

[11]. Furthermore, fuzzy sets can be used to model the concept of *graded similarity* [11]. Graded similarity in NLP refers to the capabilty of measuring the degree of similarity between words, phrases, sentences, or documents, not just in binary terms (similar or not similar), but along a continuum or scale. This approach acknowledges that linguistic items can share varying degrees of meaning or context, rather than being entirely identical or completely different.

## 3.3 Gumbel Boxes as Fuzzy Sets

In [11], Dasgupta *et al.* interpreted Gumbel boxes as fuzzy sets where the soft membership function $m(\mathbf{z})$ of a point $\mathbf{z} \in \mathbb{R}^d$ is given by the probability of $\mathbf{z}$ being inside the Gumbel box. Therefore, a Gumbel box $Box_G(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^{2d}$, corresponds to the fuzzy set $(\mathbb{R}^d, m)$ where $m : \mathbb{R}^d \to [0, 1]$ is defined as:

$$m(\mathbf{z}) = P(\mathbf{z} \in Box_G(\mathbf{x})) = \prod_{i=1}^{d} P(z_i > X_i^-)P(z_i < X_i^+) \tag{3.5}$$

where $\{X_i^\pm\}$ are Gumbel random variables that define the corners of the box. For simplicity, the authors called this fuzzy set $Box_F(\mathbf{x})$ [11].

Set operations on $Box_F(\mathbf{x})$ can be easily defined as follows:

- **Complement**: $Box_F(\mathbf{x})^c$ has membership function $m_c(\mathbf{z}) = 1 - m(\mathbf{z}) = 1 - P(\mathbf{z} \in Box_G(\mathbf{x}))$, which is the probability of $\mathbf{z}$ not being inside the Gumbel box $Box_G(\mathbf{x})$.
- **Intersection**: $Box_F(\mathbf{x}) \cap Box_F(\mathbf{y})$ has membership function defined by the *product t-norm* of the membership functions of the two fuzzy sets: $m(\mathbf{z}) = P(\mathbf{z} \in Box_G(\mathbf{x}) \cap Box_G(\mathbf{y})) = P(\mathbf{z} \in Box_G(\mathbf{x}))P(\mathbf{z} \in Box_G(\mathbf{y}))$.
- **Union**: $Box_F(\mathbf{x}) \cup Box_F(\mathbf{y})$ has membership function defined by the *t-conorm* of the membership functions of the two fuzzy sets: $m(\mathbf{z}) = P(\mathbf{z} \in Box_G(\mathbf{x}) \cup Box_G(\mathbf{y})) = P(\mathbf{z} \in Box_G(\mathbf{x})) + P(\mathbf{z} \in Box_G(\mathbf{y})) - P(\mathbf{z} \in Box_G(\mathbf{x}))P(\mathbf{z} \in Box_G(\mathbf{y}))$. This corresponds to the probability of $\mathbf{z}$ being inside at least one of the two Gumbel boxes.

The size of $Box_F(\mathbf{x})$ is obtained by integrating the membership function $m(\mathbf{z})$ over $\mathbb{R}^d$:

$$|Box_F(\mathbf{x})| = \int_{\mathbb{R}^d} P(\mathbf{z} \in Box_G(\mathbf{x}))d\mathbf{z} \tag{3.6}$$

As prooved in [7] and [12], the integral in equation 3.6 can be approximated by the following formula:

$$|Box_F(\mathbf{x})| \approx \prod_{i=1}^{d} \beta log(1 + exp(\frac{\mu_i^+ - \mu_i^-}{\beta} - 2\gamma)) \tag{3.7}$$

Paper [11] shows that also the size of the fuzzy set intersection $|Box_F(\mathbf{x}) \cup Box_F(\mathbf{y})|$ and the size of the fuzzy set difference $|Box_F(\mathbf{x}) \setminus Box_F(\mathbf{y})|$ can be approximated in the same way.

## 3.4 Training

Before describing how the Word2Box model is trained, I will clarify some concepts and terminology that are needed to understand the training process. I will then talk about the training dataset used by Dasgupta *et al.* (2022) [11].

Let's start with some notation: the *vocabulary*, indexed in a fixed but arbitrary order, is indicated as $V = \{v_i\}_{i=1}^{N}$, where $N$ is the number of words in the vocabulary. A *sentence* $\mathbf{s} = (s_1, \ldots, s_j)$, $s_i \in V$ is a variable-length sequence of words in the vocabulary, and the *corpus* $C = \{\mathbf{s}_i\}_{i=1}^{M}$ can be defined as the multiset of all the sentences in the corpus [11]. Given a fixed *window size* $\ell$ and a sentence $\mathbf{s}$, the *window* centered at the $i$-th word $s_i$ is defined as $\mathbf{w}_i = [s_{i-\ell}, \ldots, s_i, \ldots, s_{i+\ell}]$. Given a window $\mathbf{w_i}$, the *center word* is indicated as $cen(\mathbf{w}_i) = s_i$ and the *context*, which is made of all the remaining words in the window, is indicated as $con(\mathbf{w}_i)$. Finally, $C_W$ is the multiset of all the windows in the corpus $C$ [11].

In paper [11], Dasgupta *et al.* also define the set of windows centered at a word $v \in V$ as:

$$cen_W(v) := \{w \in C_W : cen(w) = v\} \tag{3.8}$$

and the set of windows whose context contains the word $v \in V$ as:

$$con_W(v) := \{w \in C_W : v \in con(w)\} \tag{3.9}$$

However, to allow the representation of a word $u$ to overlap with the representation of a word $v$ when they have a similar meaning, the authors define the following fuzzy sets:

$$\begin{aligned} \widetilde{cen_W}(v) &:= \{w \in C_W : cen(w) = u, u \text{ is similar to } v\} \\ \widetilde{con_W}(v) &:= \{w \in C_W : u \in con(w), u \text{ is similar to } v\} \end{aligned} \tag{3.10}$$

The aim of the Word2Box model is to learn the following center and context box rep-

resentations, based on the fuzzy sets defined in 3.10, for each word $v$ in the vocabulary $V$ [11]:

$$\begin{aligned} cen_B(v) &:= Box_F(\widetilde{cen_W}(v)) \\ con_B(v) &:= Box_F(\widetilde{con_W}(v)) \end{aligned} \tag{3.11}$$

To do so, Word2Box uses a *max-margin* training objective where the score for a given window $\mathbf{w}$ is defined as [11]:

$$f(\mathbf{w}) := \left| cen_B(w_0) \cap \bigcap_{i \neq 0} con_B(w_i) \right| \tag{3.12}$$

Max-margin loss is used to encourage the model to rank the correct output higher than any of the incorrect outputs. This is achieved by maximizing the margin between the score of the correct output and the scores of the incorrect outputs. Thus, the max-margin loss helps the model improve prediction accuracy by making the correct output clearly distinguishable from the incorrect ones.

Finally, the model has been compared with a baseline consisting in the Word2Vec model trained using CBOW with negative sampling. This version of Word2Vec has been implemented in Pytorch using Stochastic Gradient Descent with varying batch sizes and trained using a GPU [11].

### 3.4.1 Dataset

Dasgupta *et al.* (2022) [11] trained the Word2Box model using the **ukWaC** corpus, which is a large English corpus created by Baroni *et al.* (2019) [13] using web crawling. They further preprocessed the corpus by removing all punctuation, replacing numbers with the `<num>` token, and lowercasing all words. The resulting dataset is composed of 0.9 billion tokens and has a vocabulary of more than 112k unique words [11].

During the training process, the dataset is augemented using the *negative sampling* technique. Negative examples $\mathbf{w}'$ are sampled from the vocabulary applying the negative sampling procedure used for CBOW in [4]. This procedure consists in replacing the center word $w_0$ with a word sampled from the unigram distribution raised to the power of $3/4$ [11]. *Subsampling* of context words is also performed as in [4].

## 3.5 Model evaluation

To better undestand the model's performances, Dasgupta *et al.* (2022) [11] compared a 64-dimensional Word2Box model with a 128-dimensional Word2Vec model, considering that a $n$-dimensional box embedding has $2n$ parameters. The two models have been evaluated on both quantitative and qualitative tasks related to semantic similarity, relatedness, lexical ambiguity, and uncertainty [11].

### 3.5.1 Word Similarity Benchmarks

The first set of evaluations has been done using word similarity benchmarks, which consist in datasets made of word pairs (both nouns and verbs) annotated by humans with a similarity score [11]. As we can see in Table 3.1, Word2Box outperforms Word2Vec in most of the word similarity tasks, especially in those involving rare words. In particular, the authors noticed that the differences in performances between the two models are more pronounced as the amount of frequent words in the similarity datasets dicreases. This can be seen in the plots of Figure 3.2, that show the Spearman's correlation computed on word pairs vs the word treshold frequency in log scale for some of the similarity datasets used for the models' evaluation [11].

| | Stanford RW | RG-65 | YP-130 | MEN | MC-30 | Mturk-287 | SimVerb-3500 | SimLex-999 | Mturk-771 | WS-353 (Sim) | WS-353 (All) | WS-353 (Rel) | VERB-143 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Poincaré | — | 75.97 | — | — | 80.46 | — | 18.90 | 31.81 | — | — | 62.34 | — | — |
| *Gaussian | — | 71.00 | 41.50 | 71.31 | 70.41 | — | — | 32.23 | — | 76.15 | 65.49 | 58.96 | — |
| Word2Vec | 40.25 | 66.80 | 43.77 | 68.45 | 75.57 | 61.83 | 23.58 | 37.30 | 59.90 | 75.81 | **69.01** | **61.29** | 31.97 |
| Word2Box | **45.08** | **81.45** | **51.6** | **73.68** | **87.12** | **70.62** | **29.71** | **38.19** | **68.51** | **78.60** | **68.68** | 60.34 | **48.03** |

**Table 3.1:** Results of the word similarity task on word similarity benchmarks. For comparison, also the reported results of Gaussian and Poicaré embeddings are shown, however they might not be directly comparable. This table is taken from paper [11].

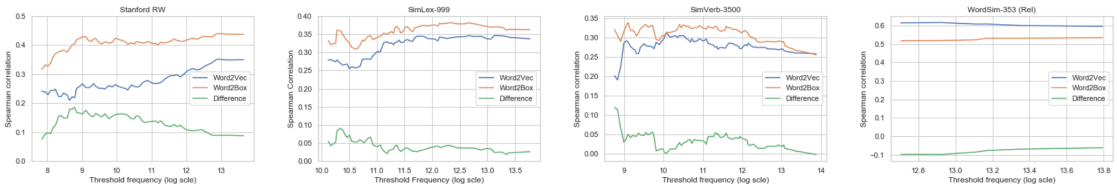



**Figure 3.2:** Spearman's correlation vs the Treshold Frequency in log scale for Standord RW, SimLex-999, SimVerb-3500, and WordSim-353 datasets. This picture is taken from paper [11].

The Word2Box better performances might be explained by the fact that box embeddings are a more flexible representation of words, capable of handling sets of mutually disjoint words, such as rare words, which co-occure with a more common word. In fact, the many corners of box embeddings lead to a great flexibility in representing the intersections also in cases of complicated co-occurences [11].

### 3.5.2 Set Theoretic Operations

The second part of the evaluation has been done performing set-theoretic operations on the word embeddings generated by Word2Box and Word2Vec with the aim of evaluating the models' capabilty of representing sets. To do so, the authors created a dataset of homographs and polysemous words, consisting of triples of words (A, B, C) where A $\circ$ B should yield a set similar to C, for some set-theoretic operations $\circ$. The dataset contains 52 examples for both Union and Negation, and 20 examples for Intersection [11].

Defining union and intersection in the vector space is not trivial, therefore the authors tried different strategies. Nonehtless, the results show that Word2Box yields a higher rank for the target C than Word2Vec over 90% of the time, thus proving the model's capabilty to capture the underlying set theoretic aspects of the words in the corpus [11]. Finally, Word2Box shows a better consistency in the results of logical queries compared to Word2Vec, as can be seen in the example queries of Table 3.2 [11].

| Word2Box | | | Word2Vec | | | | | |
|---|---|---|---|---|---|---|---|---|
| (bank$\cap$river) $\cap X$ | (bank$\cup$river) $\cap X$ | (bank\river) $\cap X$ | (bank+river) $\cdot X$ | (bank$-$river) $\cdot X$ | max(bank,river) $\cdot X$ | max(bank,river) $\cdot X$ | max(bank$\cdot X$,river$\cdot X$) | min(bank$\cdot X$,river$\cdot X$) |
| headwaters | tributary | barclays | tributaries | cheques | tributary | vipava | tributaries | gauley |
| tributary | valley | hsbc | tributary | tymoshenko | tributaries | quabbin | headwaters | pymatuning |
| lake | headwaters | banking | headwaters | receivables | prut | irwell | tributary | 'utricularia |
| basin | reservoir | citigroup | nakdong | citibank | chambal | trabajadores | headwater | luangwa |
| estuary | gorge | citibank | vipava | eurozone | headwaters | chattahoochee | distributaries | vipava |
| creek | lake | firm | estuary | brinks | larrys | tributaries | larrys | guadalquivir |
| valley | dam | ipo | larrys | defrauded | nakdong | belait | kobuk | suir |
| reservoir | headwater | brokerage | headwater | courtaulds | waterway | bougouriba | estuary | meenachil |
| canal | junction | interbank | distributary | refinance | loyalsock | canal | ijssel | tributary |
| floodplain | creek | kpmg | luangwa | mortgage | 'hyperolius | glomma | distributary | battuta |

**Table 3.2:** Output of Word2Box and Word2Vec for various set operations. This table is taken from paper [11].

# 4

# Experiments

In this chapter, I will present my work on the Word2Box algorithm. I will start by presenting the dataset I used to train the Word2Box model and the preprocessing steps I executed to prepare the data for training. I will then talk about the exploratory data analysis I did to better understand the data. Finally, I will present the code analysis and explanation of the Word2Box algorithm.

## 4.1 Dataset

To train the Word2Box model, I used an extract of the **enwik8** dataset created by Matt Mahoney [*], to which I applied the preprocessing steps described in Section 4.2. `enwik8` contains the first $10^8$ bytes of the English Wikipedia dump made on Mar. 3, 2006. The text is primarily in English and it consists of UTF-8 encoded XML, as shown in Figure 4.1.

Mahoney also provides a preprocessed version of the `enwik8` dataset, called **text8**. `text8` is a 100 MB file, where XML tags, citations, footnotes, and markup are removed. Furthermore, hyperlinks are converted to ordinary text, numbers are spelled out (e.g. "20" becomes "two zero"), letters are lowercased, and all sequences of characters not in the range $a - z$ are converted to a single space, including punctations. Figure 4.2 shows an extract of the preprocessed dataset and the differences between the two datasets are evident.

[*] The `enwik8` dataset is freely available at `https://mattmahoney.net/dc/textdata.html`

```
 <mediawiki xmlns="http://www.mediawiki.org/xml/export-0.3/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" x
 si:schemaLocation="http://www.mediawiki.org/xml/export-0.3/ http://www.mediawiki.org/xml/export-0.3.xsd" version="0
 .3" xml:lang="en">
<siteinfo>
  <sitename>Wikipedia</sitename>
  <base>http://en.wikipedia.org/wiki/Main_Page
 </base>
  <generator>MediaWiki 1.6alpha</generator>
  <case>first-letter</case>
    <namespaces>
    <names
 pace key="-2">Media</namespace>
    <namespace key="-1">Special</namespace>
    <namespace key="0" />
    <na
 mespace key="1">Talk</namespace>
    <namespace key="2">User</namespace>
    <namespace key="3">User talk</name
 space>
    <namespace key="4">Wikipedia</namespace>
    <namespace key="5">Wikipedia talk</namespace>
    <na
 mespace key="6">Image</namespace>
    <namespace key="7">Image talk</namespace>
    <namespace key="8">MediaWik
 i</namespace>
    <namespace key="9">MediaWiki talk</namespace>
    <namespace key="10">Template</namespace>

     <namespace key="11">Template talk</namespace>
    <namespace key="12">Help</namespace>
    <namespace key="
```

**Figure 4.1:** An extraxt of the `enwik8` dataset.

```
 anarchism originated as a term of abuse first used against early working class radicals including the diggers of t
he english revolution and the sans culottes of the french revolution whilst the term is still used in a pejorative
way to describe any act that used violent means to destroy the organization of society it has also been taken up as
 a positive label by self defined anarchists the word anarchism is derived from the greek without archons ruler chi
ef king anarchism as a political philosophy is the belief that rulers are unnecessary and should be abolished altho
ugh there are differing interpretations of what this means anarchism also refers to related social movements that a
dvocate the elimination of authoritarian institutions particularly the state the word anarchy as most anarchists us
e it does not imply chaos nihilism or anomie but rather a harmonious anti authoritarian society in place of what ar
e regarded as authoritarian political structures and coercive economic institutions anarchists advocate social rela
tions based upon voluntary association of autonomous individuals mutual aid and self governance while anarchism is
```

**Figure 4.2:** An extraxt of the `text8` dataset.

After an initial analysis of the `enwik8` and `text8` datasets, I chose to use the `enwik8` one. The main reason of my choice is the fact that I would like to perform the preprocessing in a slightly different way, in order to keep the information provided by the punctuation marks. Also, I would like to represent numbers using the `<NUM>` token, as it has been done in [11].

## 4.2 Data preprocessing

The first phase of the data preprocessing consists in applying part of the preprocessing steps used by Mahoney to create the `text8` dataset. These steps are contained in a Perl script provided by Mahoney, together with the datasets. In particular, I removed the part of the script that spells out the numbers and the part that converts punctations into spaces. The

resulting script is the following:

```
#!/usr/bin/perl

$/=">";                     # input record separator
while (<>) {
    if (/<text /) {$text=1;} # remove all but between <text> ... </text>
    if (/#redirect/i) {$text=0;} # remove #REDIRECT
    if ($text) {
        # Remove any text not normally visible
        if (/<\/text>/) {$text=0;}
        s/<.*>//;           # remove xml tags
        s/&amp;/&/g;        # decode URL encoded chars
        s/&lt;/</g;
        s/&gt;/>/g;
        s/<ref[^<]*<\/ref>//g; # remove references <ref...> ... </ref>
        s/<[^>]*>//g;       # remove xhtml tags
        s/\[http:[^] ]*/[/g;   # remove normal url, preserve visible text
        s/\|thumb//ig;      # remove images links, preserve caption
        s/\|left//ig;
        s/\|right//ig;
        s/\|\d+px//ig;
        s/\[\[image:[^\[\]]*\|//ig;
        # show categories without markup
        s/\[\[category:([^|\]]*)[^]]*\]\]/[[$1]]/ig;
        s/\[\[[a-z\-]*:[^\]]*\]\]//g;  # remove links to other languages
        s/\[\[[^\|\]]*\|/[[/g; # remove wiki url, preserve visible text
        s/{{[^}]*}}//g;     # remove {{icons}} and {tables}
        s/{[^}]*}//g;
        s/\[//g;            # remove [ and ]
        s/\]//g;
        s/&[^;]*;/ /g;      # remove URL encoded chars

        # Convert to lowercase letters and spaces
        $_=" $_ ";
        tr/A-Z/a-z/;
        tr/a-z.,;:"-()!?0-9/ /cs;
        chop;
        print $_;
    }
}
```

In Figure 4.3 you can see how the dataset looks like after the first part of the preprocessing is applied.

```
 '''anarchism''' originated as a term of abuse first used against early working class radicals including the digger
s of the english revolution and the ''sans culottes'' of the french revolution. whilst the term is still used in a
pejorative way to describe '' any act that used violent means to destroy the organization of society '', it has als
o been taken up as a positive label by self defined anarchists. the word '''anarchism''' is derived from the greek
'' '' ( without archons (ruler, chief, king) ). anarchism as a political philosophy, is the belief that ''rulers''
are unnecessary and should be abolished, although there are differing interpretations of what this means. anarchism
 also refers to related social movements) that advocate the elimination of authoritarian institutions, particularly
 the state. the word anarchy, as most anarchists use it, does not imply chaos, nihilism, or anomie, but rather a ha
rmonious anti authoritarian society. in place of what are regarded as authoritarian political structures and coerci
ve economic institutions, anarchists advocate social relations based upon voluntary association of autonomous indiv
```

**Figure 4.3:** An extraxt of the `enwik8` dataset after the first part of the preprocessing is applied.

The second phase of data preprocessing consists in the following steps:

- All numbers are replaced with the `<NUM>` token, as in [11].
- All punctuation marks are replaced with the corresponding token (e.g. the question mark `?` is replaced with the `<QUESTION_MARK>` token, and so on). I decided to keep all the punctuation marks in the dataset because they provide useful information about the text structure. However, this is different to what has been done in [11], where the punctuations are removed.
- Stopwords are defined as words that carry very little information and therefore are not essential to understand the meaning of a text. They are usually very frequent words, such as articles, prepositions, and conjunctions. They have been removed from the dataset in order to let the algorithm focus on more meaningful words.
- Low-frequency words which appear in the dataset with a frequency lower than 5 are removed.

The resulting dataset contains $8,858,098$ words and $53,693$ unique words. Figure 4.4 shows how the dataset looks like after the second part of the preprocessing is applied.

```
<QUOTATION_MARK> anarchism <QUOTATION_MARK> originated term abuse first used early working class radicals including
 diggers english revolution <QUOTATION_MARK> sans <QUOTATION_MARK> french revolution <PERIOD> whilst term still use
d pejorative way describe <QUOTATION_MARK> act used violent means destroy organization society <QUOTATION_MARK> <CO
MMA> taken positive label self defined anarchists <PERIOD> word <QUOTATION_MARK> anarchism <QUOTATION_MARK> derived
 greek <QUOTATION_MARK> <QUOTATION_MARK> <LEFT_PAREN> without archons <LEFT_PAREN> ruler <COMMA> chief <COMMA> king
 <RIGHT_PAREN> <RIGHT_PAREN> <PERIOD> anarchism political philosophy <COMMA> belief <QUOTATION_MARK> rulers <QUOTAT
ION_MARK> unnecessary abolished <COMMA> although differing interpretations means <PERIOD> anarchism refers related
social movements <RIGHT_PAREN> advocate elimination authoritarian institutions <COMMA> particularly state <PERIOD>
word anarchy <COMMA> anarchists use <COMMA> imply chaos <COMMA> nihilism <COMMA> <COMMA> rather harmonious anti aut
horitarian society <PERIOD> place regarded authoritarian political structures coercive economic institutions <COMMA
```

**Figure 4.4:** An extraxt of the `enwik8` dataset after the second part of the preprocessing is applied.

I implemented the preprocessing steps using Python; the complete code snippet is available in a Colab notebook.

## 4.3 Exploratory Data Analysis

Once the data has been fully preprocessed, I performed an exploratory data analysis to better understand the dataset. First of all, I computed the frequency of each word type in the corpus. Then, I extracted the maximun and minimum frequencies, I computed the average frequency, the average word length and the weighted average word length. For simplicity, I didn't exclude the punctuation tokens and the `<NUM>` token from the analysis. The results are shown in Table 4.1.

| **Metric** | **Value** |
|---|---|
| Maximum Frequency | 718,889 |
| Minimum Frequency | 1 |
| Average Frequency | 165 |
| Average Word Length | 7.42 |
| Weighted Average Word Length | 7.12 |

**Table 4.1:** Statistics of the enwik8 dataset after preprocessing.

We can see that the average word length and the weighted average word length are very similar. The average word frequency is 165, which means that the dataset is not too sparse. Nonetheless, the maximum frequency is 718, 889, which is very high compared to the average frequency. This is due to the fact that the dataset contains a lot of punctuation marks and numbers, which are repeated many times.

After that, I investigated which are the most frequent and less frequent words. Table 4.2 displays the 20 most frequent words in the dataset, together with their frequency. As we can see, the 7 most frequent word types are punctations and numbers. Table 4.3 shows the 20 least frequent words. As expected, if we exclude the `<eos>` token (i.e. the token that indicates the end of the dataset), the smallest word frequency is 6. In fact, less frequent words have been removed during the data preprocessing. It's interesting to note that, among the rarest words, there are some mispelled words, such as `transfered` and `corea`.

To conclude, I implemented the exploratory data analysis using the Python libraries `torchtext` and `pandas`. The complete code snippet is available in the same Colab notebook used for the data preprocessing.

| Word Type | Frequency |
|---|---|
| <COMMA> | 718,889 |
| <PERIOD> | 569,330 |
| <N> | 418,395 |
| <QUOTATION_MARK> | 333,538 |
| <RIGHT_PAREN> | 169,757 |
| <LEFT_PAREN> | 169,649 |
| <APOSTROPH> | 93,119 |
| <COLON> | 84,702 |
| <SEMICOLON> | 31,773 |
| first | 20,970 |
| used | 17,629 |
| new | 17,196 |
| time | 15,303 |
| see | 14,744 |
| world | 13,198 |
| american | 12,571 |
| people | 10,920 |
| states | 10,868 |
| use | 10,836 |
| known | 10,800 |

**Table 4.2:** Most frequent word types in the dataset.

| Word Type | Frequency |
|---|---|
| <eos> | 1 |
| citium | 6 |
| regimentation | 6 |
| girondins | 6 |
| communitarian | 6 |
| climaxed | 6 |
| realign | 6 |
| luddites | 6 |
| leftism | 6 |
| agitators | 6 |
| woodworth | 6 |
| bayonets | 6 |
| caplan | 6 |
| freetown | 6 |
| bleuler | 6 |
| kanner | 6 |
| unwinding | 6 |
| transfered | 6 |
| shyness | 6 |
| corea | 6 |

**Table 4.3:** Less frequent word types in the dataset.

## 4.4 Code analysis

The last section of this chapter is dedicated to the analysis and explanation of the code of the Word2Box algorithm. The original code is written in Python and it is freely available on GitHub at the following link: `https://github.com/iesl/word2box`. However, since it contains several bugs and it is not well documented, on one hand I decided to fork the repository and fix the bugs, and on the other hand I created a ready-to-run Colab notebook to train and test the model, with a detailed code explanation. The (hopefully) bug-free codebase is available at the following GitHub repository: `https://github.com/ChiaraBi/word2box`.

In the following subsections, I will provide an insight into the project structure. Then, I will dive deep into the models and loss functions available in the codebase. Finally, I will explain how the training is performed and how to evaluate the model.

### 4.4.1 Project structure

The Word2Box codebase is organized into four modules: `datasets`, `box`, `models` and `train`. I will briefly describe each module in the paragraphs below.

DATASETS The `datasets` module contains the classes that represent the dataset and the dataset loader, `Word2VecDatasetOnDevice` and `LazyDatasetLoader` respectively. It also contains the functions to read and tokenize the dataset, extract the vocabulary and perfom a subsampling.

BOX The `box` module contains the classes that represent the box tensors (`BoxTensor` and `DeltaBoxTensor`) and the box embedding layer (`BoxEmbedding`). `BoxTensor` exposes the methods to compute the intersection between two boxes, the volume of a box, the volume of the intersection and the volume of the intersection of Gumbel boxes.

MODELS The `models` module contains the classes that represent the models for Word2Box, Word2Vec and Word2Gauss. I will provide a more in depth explanation of the Word2Box models in Section 4.4.2.

TRAIN The `train` module contains the loss functions, the classes that perfom the negative sampling for both CBOW (`RandomNegativeCBOW`) and SkipGram (`RandomNegativeSkipGram`), the trainers (`Trainer` and `TrainerWordSimilarity`) and the `training` function, which takes in input the hyperparameters and other model configurations, and executes the training of the selected model.

### 4.4.2 Models

In this section I will focus on the two variants of the Word2Box algorithm available in the codebase, i.e. `Word2Box` and `Word2BoxConjunction`.

#### Word2Box

`Word2Box` is the SkipGram implementation of the Word2Box algorithm. It inherits from `BaseModule`, a class that exposes the methods needed to save and load the model's parameters and the training checkpoints. The constructor (`__init__`) initializes the model and creates

the `BoxEmbedding` layers for the target and context words. In addition, `Word2Box` exposes three methods: `word_similarity`, `conditional_similarity` and `forward`.

`word_similarity` takes in input two words and computes the box embedding for each one of the words. Then, it computes the volume of the intersection between the two box embeddings.

`conditional_similarity` is similar to `word_similarity`, and in addition it computes the `_log_soft_volume_adjusted` defined in the `BoxTensor` class.

However, it is not really possible to use the `Word2Box` model because, when used, it gives the following error message: `"Gumbel intersection is not possible".` Therefore, I decided not to report the implementation of the `forward` method in the code below.

```
class Word2Box(BaseModule):
    def __init__(self, TEXT=None, embedding_dim=50, batch_size=10, n_gram=4,
        volume_temp=1.0, intersection_temp=1.0, box_type="BoxTensor", **kwargs):
        super(Word2Box, self).__init__()

        # Model
        self.batch_size = batch_size
        self.n_gram = n_gram
        self.vocab_size = len(TEXT.itos)
        self.embedding_dim = embedding_dim

        # Box features
        self.volume_temp = volume_temp
        self.intersection_temp = intersection_temp
        self.box_type = box_type

        # Create embeddings
        self.embeddings_word = BoxEmbedding(
            self.vocab_size, self.embedding_dim, box_type=box_type
        )
        self.embedding_context = BoxEmbedding(
            self.vocab_size, self.embedding_dim, box_type=box_type
        )

    def forward(self, idx_word, idx_context, context_mask, train=True):
        pass

    def word_similarity(self, w1, w2):
        with torch.no_grad():
```

```
            word1 = self.embeddings_word(w1)
            word2 = self.embeddings_word(w2)
            score = word1.gumbel_intersection_log_volume(word2,
                    volume_temp=self.volume_temp,
                    intersection_temp=self.intersection_temp)
            return score

    def conditional_similarity(self, w1, w2):
        with torch.no_grad():
            word1 = self.embeddings_word(w1)
            word2 = self.embeddings_word(w2)
            score = word1.gumbel_intersection_log_volume(word2,
                    volume_temp=self.volume_temp,
                    intersection_temp=self.intersection_temp)
            # Word1 Word2  queen   royalty 5.93
            # Word2 is more general P(royalty | queen) = 1
            # Thus we need p(w2 | w1)
            score -= word1._log_soft_volume_adjusted(word1.z, word1.Z,
                temp=self.volume_temp, gumbel_beta=self.intersection_temp)
            return score
```

### Word2BoxConjunction

`Word2BoxConjunction` is the CBOW implementation of the Word2Box algorithm. It inherits from the `Word2Box` class and overrides the `forward` method of `Word2Box`. Furthermore, it defines a new method called `intersect_multiple_box`, which applies a mask to the input boxes and turns them into Gumbel random variables. Note that `intersection_temp`, which represent the $\beta$ parameter of the Gumbel distribution, should always be $! = 0$, in order to avoid a division by 0.

```
class Word2BoxConjunction(Word2Box):
    def intersect_multiple_box(self, boxes, mask):
        beta = self.intersection_temp
        z = boxes.z.clone()
        Z = boxes.Z.clone()
        z[~mask] = float("-inf")
        Z[~mask] = float("inf")
        z = beta * torch.logsumexp(z / beta, dim=1, keepdim=True)
        Z = -beta * torch.logsumexp(-Z / beta, dim=1, keepdim=True)

```

```
        return BoxTensor.from_zZ(z, Z)

    def forward(self, idx_word, idx_context, mask_context, train=True):
        context_boxes = self.embedding_context(idx_context)  # Batch_size * 2 * dim
        word_boxes = self.embeddings_word(idx_word)  # Batch_size * ns+1 * 2 * dim
        # Note that the context is not masked yet. We need to mask it as well:
        pooled_context = self.intersect_multiple_box(context_boxes, mask_context)

        score = word_boxes.gumbel_intersection_log_volume(
            pooled_context,
            volume_temp=self.volume_temp,
            intersection_temp=self.intersection_temp,
        )
        return score
```

### 4.4.3 Loss functions

The Word2Box codebase provides three different loss functions: Noise Contrastive Estimation, Negative Log Likelihood and Max Margin. The default loss function for Word2Box, and the one used in [11], is Max Margin, however I trained the model using the NCE loss function. The type of loss function to be used by the model should be specified in the configurations passed in input to the `training` function.

#### Max Margin

The **Max Margin** (`max_margin`) loss function provided in the codebase is specific for box embeddings and its goal is to increase the similarity score of the positive examples more than a margin from the negative examples' scores. If that margin is satisfied then the loss is zero. The input scores are the unnormalised similarity scores for the positive and negative examples (`pos` and `neg` respectively).

```
    def max_margin(pos, neg, margin=5.0):
        zero = torch.tensor(0.0).to(device)
        return torch.sum(torch.max(zero, neg - pos + margin), dim=1)
```

#### Noise Contrastive Estimation

**Noise Contrastive Estimation** (`nce`) is the loss function used by Word2Vec, however it can be used also for box embeddings. NCE takes in input the unnormalised probabilities of the posi-

tive and negative examples (`pos` and `neg` respectively). These probabilities are passed through a sigmoid function (`nn.functional.logsigmoid`) in order to get a normalised score.

```
    def nce(pos, neg, **kwagrs):
        return -(nn.functional.logsigmoid(pos)
            + torch.sum(nn.functional.logsigmoid(-neg), dim=1))
```

#### Negative Log Likelihood

The **Negative Log Likelihood** (`nll`) loss function provided in the codebase is specific for box embeddings. It takes in input the log probabilities of the positive and negative examples (`pos` and `neg` respectively).

```
    def nll(pos, neg, **kwagrs):
        assert (pos < 0).all(), "Log probabiltiy can not be positive"
        assert (neg < 0).all(), "Log probabiltiy can not be positive"

        # Note that log1mexp(neg) = log(1-exp(neg))
        return -(pos + torch.sum(log1mexp(neg), dim=1))
```

### 4.4.4 Training

The `training` function is the training entry point. It takes in input the configurations and hyperparameters needed to train the model. After setting the seed, it creates an instance of `LazyDatasetLoader` to load and preprocess the training dataset. Then, it creates the correct model, based on the `config["model_type"]` value. After that, it creates a trainer of type `TrainerWordSimilarity` and calls the `trainer.train_model` method to start the model's training.

Note that I slightly modified the `training` function to make it return the `model` and the `trainer`, since they are necessary to test the model. Furthermore, I kept only the `Word2BoxConjunction` model and the CBOW `model_mode`, since I didn't use the other models.

```
def training(config):
    # Set the seed
    if config["seed"] is None:
        config["seed"] = random.randint(0, 2**32)
    torch.manual_seed(config["seed"])
    random.seed(config["seed"])
```

```
    # get_iter_on_device is defined in datasets.utils
    TEXT, train_iter = get_iter_on_device(
        config["batch_size"],
        config["dataset"],
        config["model_type"],
        config["n_gram"],
        config["subsample_thresh"],
        config["data_device"],
        config["add_pad"],
        config["eos_mask"],
    )

    if config["model_type"] == "Word2BoxConjunction":
        model = Word2BoxConjunction(
            TEXT=TEXT,
            embedding_dim=config["embedding_dim"],
            batch_size=config["batch_size"],
            n_gram=config["n_gram"],
            intersection_temp=config["int_temp"],
            volume_temp=config["vol_temp"],
            box_type=config["box_type"],
        )
    else:
        raise ValueError("Model type is not valid. Please enter a valid model type")

    if use_cuda:
        model.cuda()

    # Instance of trainer
    trainer = TrainerWordSimilarity(
        train_iter=train_iter,
        val_iter=None,
        vocab=TEXT,
        lr=config["lr"],
        n_gram=config["n_gram"],
        loss_fn=config["loss_fn"],
        negative_samples=config["negative_samples"],
        model_mode="CBOW",
        log_frequency=config["log_frequency"],
        margin=config["margin"],
```

```
        similarity_datasets_dir=config["eval_file"],
        subsampling_prob=None,  # pass: subsampling_prob, when you want to adjust neg_sampling distn
    )

    trainer.train_model(
        model=model,
        num_epochs=config["num_epochs"],
        path=config.get("save_dir", False),
        save_model=config.get("save_model", False),
    )

    return model, trainer
```

The core of the training process happens inside the `train_model`. This method takes in input the model to train, the number of epochs, the path where to save the model and the flag `save_model`. It uses the Adam optimizer and a custom loss function. The method also includes a training loop that iterates over the training data, computes the loss, and updates the model parameters. It also checks for `NaN` or infinite values in the loss or parameters, which could indicate problems with the training. The method also provide intermediate and a final evaluations of the model. It computes the Spearman's correlation on the **Simlex-999** dataset and keeps track of the best value of this metric. If the `save_model` flag is set, it saves to disk the current model and the best model according to the metric.

To train the `Word2BoxConjunction`, I mainly used the default values provided in [11]. However, I changed the training dataset and the loss function. Furthermore, I used only 1 training epoch due to time and hardware constraints.

```
SAVED_MODELS_DIR = '/content/saved-models'
DATASET_NAME = 'enwik8-preprocessed'

config = {
    "alpha_dim": 32,
    "batch_size": 1024,
    "cuda": True,
    "data_device": 'cpu',
    "diag_context": 0,
    "eval_file": "./data/similarity_datasets/",
    "eos_mask": True,
    "embedding_dim": 50,
    "int_temp": 0.01,
```

```
    "log_frequency": 100,
    "lr": 0.01,
    "margin": 10,
    "n_gram": 4,
    "negative_samples": 5,
    "sep_output": 0,
    "subsample_thresh": 0.001,
    "vol_temp": 1.0,
    "num_epochs": 1,
    "box_type": "DeltaBoxTensor",
    "dataset": DATASET_NAME,
    "loss_fn": "nce",
    "model_type": "Word2BoxConjunction",
    "save_dir": SAVED_MODELS_DIR,
    "save_model": True,
    "seed": 5,
}
```

### 4.4.5 Model Evaluation

After the model has been trained, it can be evaluated using the `trainer.model_eval` method. This method takes in input the trained model and, if the `similarity_datasets_dir` is defined, it evaluates the model using `model.word_similarity` method on each row of each dataset present in the folder. Then, it computes the Spearman's correlation between the predicted scores and the target scores. Finally, it returns a dictionary containing the correlation value obtained for each similarity dataset.

# 5
# Conclusion

The Word2Box algorithm represents a significant advancement in the field of natural language processing by introducing a geometrically inspired method for word representation. By representing words as n-dimensional hyperrectangles called "boxes", it offers a novel way to express uncertainty and model asymmetric relationships between words, overcoming some limitations of traditional vector word embeddings. This approach not only enhances the representation of semantic relationships, but also facilitates set-theoretic operations on word meanings, providing a richer linguistic representation of polysemic words. Furthemore, Dasgupta *et al.* (2022) [11] have shown that Word2Box outperforms Word2Vec in both the word similarity task and on logical queries based on set operations, proving the effectiveness of the algorithm in capturing the word semantic.

Due to time and resource constraints, I was not able to train the Word2Box model on the full dataset and I had to limit the number of epochs to 1. As a future improvement, it would be interesting to investigate how a properly trained Word2Box model would perform in downstream NLP tasks, in order to fully understand the algorithm's potential. In particular, it would be interesting to compare Word2Box with the static word embedding model proposed by Ahmet Onur Akman in his Master's Thesis "Design of a Third-Order Word Embedding Model Using Vector Projections" (2023) [14].

# References


[1] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 1st ed. Standford University, 2021.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A Neural Probabilistic Language Model," 2003.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," 2013.

[4] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[5] L. Vilnis and A. McCallum, "Word Representations via Gaussian Embedding," *ICLR*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6623

[6] L. Vilnis, "Geometric Representation Learning," 2021. [Online]. Available: https://scholarworks.umass.edu/dissertations_2/2146

[7] S. Dasgupta, M. Boratko, D. Zhang, L. Vilnis, X. Li, and A. McCallum, "Improving Local Identifiability in Probabilistic Box Embeddings," *NeurIPS*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/01c9d2c5b3ff5cbba349ec39a570b5e3-Abstract.html

[8] L. Vilnis, X. Li, S. Murty, and A. McCallum, "Probabilistic Embedding of Knowledge Graphs with Box Lattice Measures," *ACL*, 2018. [Online]. Available: http://arxiv.org/abs/1805.06627

[9] I. Vendrov, R. Kiros, S. Fidler, and R. Urtasun, "Order-Embeddings of Images and Language," *ICLR*, 2016. [Online]. Available: http://arxiv.org/abs/1511.06361

[10] A. Lai and J. Hockenmaier, "Learning to Predict Denotational Probabilities For Modeling Entailment," *ACL*, 2017. [Online]. Available: https://aclanthology.org/E17-1068

[11] S. S. Dasgupta, M. Boratko, S. Mishra, S. Atmakuri, D. Patel, X. L. Li, and A. McCallum, "Word2Box: Capturing Set-Theoretic Semantics of Words using Box Embeddings," *ACL*, 2022. [Online]. Available: http://arxiv.org/abs/2106.14361

[12] M. Boratko, J. Burroni, S. S. Dasgupta, and A. McCallum, "Min/max stability and box distributions," *UAI*, 2021. [Online]. Available: https://proceedings.mlr.press/v161/boratko21a.html

[13] M. Baroni, S. Bernardini, A. Ferraresi, and E. Zanchetta, "The WaCky wide web: A collection of very large linguistically processed web-crawled corpora," 2019. [Online]. Available: http://link.springer.com/10.1007/s10579-009-9081-4

[14] A. O. Akman, "Design of a third-order word embedding model using vector projections," 2023.