universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Unveiling Knowledge Graphs in Neural Language Models through Vector Space Transformations

verfasst von | submitted by

Giacomo Munda BA MA

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 587
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Joint-Masterstudium Multilingual Technologies
Betreut von | Supervisor:	Assoz. Prof. Mag. Dr. Dagmar Gromann BSc

Acknowledgements

My biggest thank you goes to my supervisor, Prof. Gromann, who put her trust in me and meticulously shaped my ideas into concrete findings through incredible expertise and patience. I would also like to thank Liad Magen for the joy and passion he had in teaching us. With your amazing lectures you cultivated my interest for NLP and deep learning.

To the amazing MLT24 group, especially Patricia. Thank you for making these two years, together with our friendship, something I will cherish forever. To my work colleagues and dear friends. First of all Tobias, thank your for helping me out with such dedication, patience and expertise. Your ability to break down complex topics amazes me. Dora, you already know everything I need to say to you, thank you for the laughter and the special moments we have together. Julia, thank you for being a wonderful friend and for always having an open ear. Kati, thank you for the fun times in the office and the much needed coffee breaks. To my dearest friend Alina, I am so happy to have you back in Vienna, thank you for believing in me and for always being there. To my friends in Italy, Gianmarco, Tommaso, Fabio and Thomas, thank you for being able to count on you, no matter what.

And finally, once again, to the people I love the most, my family. Mom, Dad, I cannot even put into words how grateful I am for the support you gave me during all these years. None of this would have been possible without you. Making you proud is one of my biggest motivation.

Abstract

Machine learning has been revolutionized by the introduction of deep learning and Pre-trained Language Models (PLMs), which exhibit state-of-the-art results for a variety of downstream tasks. Their success has led to the development of different methods that aim at understanding and interpreting the information learned by such models, including linguistic and factual knowledge. Approaches to unveil encoded factual associations relate to prompt-based methods, techniques to localize or edit specific model weights, and to trace the information flow during inference. Addressing a gap in existing research, this thesis proposes a methodology to directly extract factual knowledge from vector spaces of PLMs. This involves training a linear transformation with supervised data to numerically approximate geometric properties of knowledge graphs. Specifically, a labeled dataset is constructed featuring knowledge graph entities of Wikidata, both subjects and objects, each paired with a similarity score, indicating their relational proximity. The vector space is then transformed in a way that cosine similarities of knowledge graph embeddings can be effectively approximated using only contextual embeddings of PLMs, indicating a latent space in which knowledge graph properties are implicitly acquired from training data. Results show the surprising ability of BERT, GPT-2 and BART embeddings to approximate the similarity scores given the evaluation metrics, i.e., the Pearson correlation coefficient, the Spearman's Rank correlation coefficient, the mean-squared error and the root mean-squared error, with BERT achieving the most promising results in terms of correlation between predicted and actual scores as well as low error rates.

Zusammenfassung

Der Bereich des Machinellen Lernens wurde durch die Einführung vom Deep Learning und vortrainierten Sprachmodellen (engl. Pre-trained Language Models, PLMs) revolutioniert, die bei vielen Aufgaben hervorragende Ergebnisse liefern. Ihr Erfolg hat zur Entwicklung verschiedener Methoden geführt, die darauf abzielen, die von solchen Modellen gelernten Informationen zu verstehen und zu interpretieren, einschließlich linguistischem und faktischem Wissen. Ansätze zur Aufdeckung kodierter faktischer Assoziationen beziehen sich auf Prompt-basierte Methoden, Techniken zur Lokalisierung oder Bearbeitung spezifischer Modellgewichte und zur Verfolgung des Informationsflusses während der Inferenz. In dieser Masterarbeit wird eine Methode implementiert, die faktisches Wissen direkt aus Vektorräumen von PLMs extrahiert. Dazu wird eine lineare Transformation mit annotierten Daten trainiert, um geometrische Eigenschaften von Knowledge Graphs numerisch zu approximieren. Konkret wird ein annotierter Datensatz mit Entitäten von Wikidata, sowohl Subjekten als auch Objekten, konstruiert. Diese Entitäten sind jeweils mit einem Ähnlichkeitswert gepaart, der ihre relationale Nähe darstellt. Der orginale Vektorraum wird dann so transformiert, dass Kosinus-Ähnlichkeiten zwischen Vektoren im Knowledge Graph nur durch Embeddings von PLMs effektiv approximiert werden können. Das weist auf einen latenten Raum, in dem Eigenschaften von Knowledge Graphs implizit aus umfangreichen Trainingsdaten erworben werden. Die Ergebnisse zeigen die überraschende Fähigkeit von BERT, GPT-2 und BART-Vektorräume, die Ähnlichkeitswerte angesichts der Bewertungsmetriken zu approximieren, wobei BERT die vielversprechendsten Ergebnisse erzielt.

Contents

List of Tables **ix**

List of Figures **xi**

1. Introduction **1**

2. Background **5**
2.1. Deep Learning 5
2.1.1. Learning Paradigms 6
2.1.2. Neural Networks 9
2.1.3. Transformer Architecture 14
2.2. Vector Representations of Language 18
2.2.1. Statistical Word Representations 18
2.2.2. Global Word Embeddings 19
2.2.3. Contextual Embeddings 23
2.3. Language Models 26
2.3.1. Statistical Models 26
2.3.2. Neural Models 27
2.3.3. Explainability of Language Models 31
2.4. Knowledge Graphs 35
2.4.1. Knowledge Representations 36
2.4.2. Knowledge Graph Embeddings 37
2.4.3. Knowledge Graphs and Natural Language Processing 42
2.5. Vector Spaces 45
2.5.1. Transformations 46
2.5.2. Vector Similarity 48

3. Related Work **51**

4. Methods **55**
4.1. Dataset 55
4.1.1. Collecting Knowledge Graphs Entities 56
4.1.2. Computing Similarity Scores between Entities 57
4.1.3. Preparing the Dataset 57

4.1.4. Statistics . . . 58
4.2. Embedding Entities with PLMs . . . 60
4.2.1. Selecting Pre-trained Language Models . . . 60
4.2.2. Extracting Contextual Embeddings . . . 61
4.3. Transformations . . . 64
4.3.1. Transformation Models . . . 64
4.3.2. Objective Function . . . 66
4.3.3. Training and Optimization . . . 67
4.4. Evaluation Methods . . . 70
4.4.1. Quantitative Evaluation . . . 70
4.4.2. Visualizations . . . 72
4.4.3. Qualitative Evaluation . . . 74

5. Results **75**
5.1. Training Results . . . 75
5.2. Quantitative Results . . . 76
5.3. Visual Results . . . 80
5.4. Qualitative Results . . . 83

6. Discussion **85**

7. Conclusions **89**

Bibliography **91**

A. Appendix **107**
A.1. Training . . . 107
A.2. Visualizations . . . 109

List of Tables

1. Sampled subset of the Wikidata5m dataset 59
2. Technical details of the chosen PLMs, including number of parameters, hidden dimension (H-Dim), number of encoder layers (E Layers) and decoder layers (D Layers) 61
3. Training hyperparameters with corresponding values 68
4. Training results for the different vectors spaces (Emb.) together with the learning rate (LR) and the weight decay (decay) hyperparameters 75
5. Results of the original (top) and transformed (bottom) vector spaces. The scores are computed for both the embeddings containing only the entities' names (Ent.) and for the entities enhanced with their corresponding Wikipedia descriptions (Wiki.) 77
6. Results for the different dimensionalities of the output space transformed by the matrix B 79

List of Figures

1. Graphic representation of the sigmoid, tanh and ReLU activation functions 10
2. Unfolding a Recurrent Neural Network with one hidden unit S. The input sequence x is mapped to the output sequence o, where each element o_t of o depends on the previous x'_t The parameters U, V, W are shared and used at each time step (LeCun et al., 2015, 442) . . 13
3. Encoder-decoder Transformer architecture (Vaswani et al., 2017, 3) 15
4. CBOW and skip-gram architectures. CBOW aims at predicting the center word w_t given some context words, whereas skip-gram is trained to predict the context words given a center word (Mikolov et al., 2013, 5) 21
5. Rotation of word embeddings to enhance interpretability. (Above) Original skip-gram vectors. (Below) skip-gram vectors after the rotation of their dimensions, clearly showing how the vectors now have an increased notion of country and position (Park et al., 2017, 402) 33
6. Illustration of TransH and its hyperplane (Wang et al., 2017, 2726) 39
7. Overview of the neural network architectures of the (a) NTN (b) MLP models (Wang et al., 2017, 2729) 41

8. Relation distribution of the original *wikidata5m* dataset and the reduced *wikidata5m-sm* dataset using CDF (left) and KDE (right) . 59
9. Architectural overview of the model. 65

10. Plots of the losses for both the BERT-cls (left) and BERT-avg (right) models 76
11. Spectrum of the transformation matrices for BERT-avg, BERT-cls, BART, and GPT-2 81
12. Original (left) and transformed (right) vector spaces of BERT-cls . 82

13. Loss (left) and Pearson correlation coefficient (right) for the BERT-avg model. 107
14. Loss (left) and Pearson correlation coefficient (right) for the BERT-cls model. 108

15. Loss (left) and Pearson correlation coefficient (right) for the GPT2-avg model. 108
16. Loss (left) and Pearson correlation coefficient (right) for the BART-avg model. 109
17. Original (left) and transformed (right) vector spaces of BERT-avg. . 109
18. Original (left) and transformed (right) vector spaces of GPT-2. . . . 110
19. Original (left) and transformed (right) vector spaces of BART. . . . 110

1. Introduction

The advent of self-supervised learning has enabled the development of robust Pre-trained Language Models (PLMs). These models excel in acquiring nuanced hidden representations, demonstrating remarkable efficacy across various Natural Language Processing (NLP) tasks. Nevertheless, their vector spaces have become increasingly opaque, such that researchers started to develop new methods to analyze the models' outputs as well as the information encoded in their vector representations, especially the contextual embeddings produced by Transformer-based models like the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) or by Long-Short-Term-Memory-based models like ELMo (Peters et al., 2018a). Neural language models encode knowledge in their parameters through exposure to training data, but due to the continuous and dense nature of these parameters, unveiling exactly what is encoded inside of the vector spaces is an extremely complicated task. The vector spaces do not offer a transparent overview of what type of knowledge they encode and such analyses require the adoption of specialized techniques.

A common way to unveil such information from PLMs is to use probes, i.e., usually additional models that take as input the network's representations and then train on labeled data to predict properties from the hidden representations of a PLM on a task that the model was not initially trained on, without adjusting the parameters of the original model, so called zero-shot transfer (Pimentel et al., 2020). These techniques are widely used to understand the linguistic information encoded in the language model, such as syntax (Hewitt and Manning, 2019; Reif et al., 2019), morphology (Peters et al., 2018a; Belinkov et al., 2017) and semantics (Koto et al., 2021; Sorodoc et al., 2020). Nevertheless, this knowledge is not structured inside of the vector space, but distributed across its many dimensions. This poses a challenge to view PLMs as valuable alternatives to classical knowledge bases, such as knowledge graphs, which provide a structured framework for storing real-word entities and their relations, also known as facts. During training, neural language models are exposed to large amounts of language data, which can of course contain several factual information. This might suggest that PLMs encode factual knowledge implicitly during training, which can then be used during downstream tasks that require this knowledge, such as question answering. Therefore, in recent years, researchers have started to explore prompt-based methods to effectively probe the model on various tasks, including the ability of capturing factual knowledge.

The challenge is to find the best prompt that can effectively identify knowledge learned by the model and made explicit in its output (Jiang et al., 2020; Shin et al., 2020) or to directly extract complete knowledge graphs from the PLM using few-shot prompting (Cohen et al., 2023).

Despite the many approaches conducted on the exploration of linguistic and factual knowledge in PLMs, it is unclear to what extent factual knowledge is directly encoded in PLMs' vector spaces, how it can be successfully inspected and if it is encoded in a similar way as in knowledge graphs and knowledge graph embeddings. Quantifying this level of information encoded in neural language models represents an intriguing research direction for the fields of computational linguistics, deep learning and explainable Artificial Intelligence. Moreover, it can shed light on the mechanisms by which PLMs encode and retrieve factual knowledge, potentially leading to improvements in model architecture, training techniques, and interpretability. To the best of my knowledge, methods to unveil factual knowledge directly in PLMs' vector spaces remain largely unexplored and rather focus on the analysis of the representation of ontological knowledge (Michael et al., 2020) and localizing where exactly facts are stored in the embedding spaces (Meng et al., 2022; Hase et al., 2023). To address this gap, this master's thesis proposes a method that aims at answering the following research question:

> *To what extent can the transformation of vector spaces of pre-trained neural language models improve their encoding of factual information as represented in knowledge graphs?*

This thesis proposes to address this gap and to answer the research question with empirical methods and makes the following main contributions:

1. A labeled dataset is constructed featuring knowledge graph entities of Wikidata, both subjects and objects, each paired with a similarity score. This similarity score is measured based on the knowledge graph embeddings of the entities, which is an indication of their relational proximity.
2. A method is proposed for training linear transformations that takes as input contextual embeddings of the same entities extracted from PLMs and outputs transformed embeddings. The hypothesis is that factual knowledge is encoded in the PLM's vector space if the transformations are able to approximate the true similarity scores provided in the dataset using only the contextual embeddings. The assumption is that accurate similarity predictions indicate a latent vector space where contextual embeddings encapsulate factual knowledge implicitly acquired from extensive training data.
3. Different dimensionalities of the output space are tested, aiming to find a smaller subspace in which the properties of knowledge graphs are predominantly encoded in the language model. In other words, it is investigated if a

smaller vector space is capable of encoding the same properties of knowledge graphs as larger ones.

4. PLMs with different architectures, including encoder-only, decoder-only and encoder-decoder models, are chosen to test their performance and to potentially identify the best architecture and model size for factual information encoding.

By approximating the cosine similarity function through linear transformations, the goal is to bridge the gap between the raw output of PLMs and the structured knowledge contained within knowledge graphs. This approximation makes it possible to first gain more insights into the internal functionalities of language models regarding their processing of structured knowledge, but also to potentially utilize PLM embeddings in applications that require an understanding of relational knowledge, such as question answering, which is often encoded in the structure of these graphs. Analyzing whether hidden representations of language models implicitly encode factual information is of particular interest for knowledge-specific downstream tasks, neurosymbolic methods for better integration of existing knowledge, and in general for building better models. While the presented master's thesis mainly focuses on theoretical aspects and methodologies to inspect factual information within PLMs, the exploration of the transformed vector spaces in downstream tasks is left to interesting future research. The code and data are released on GitHub[1].

[1] `https://github.com/gmunda97/kg_linear_transformation.git`

2. Background

This chapter is dedicated to introducing the theoretical background necessary to develop the methodology described in Section 4. Section 2.1 is dedicated to the field of deep learning and introduces the different learning paradigms as well as the foundations of neural networks, including sophisticated architectures, such as the Transformer. Section 2.2 lays the groundwork for a deeper understanding of vector representations of language, starting with basic statistical word representations and continuing with more efficient distributional representations like global word embeddings and contextual embeddings. Section 2.3 is dedicated to language models, explaining the idea behind the concept of language modeling and proceeding with statistical language models, until reaching the state-of-the-art neural language models. Moreover, it introduces techniques used to explain the representations learned by language models as well as their predictions. In Section 2.4, the discussion shifts to knowledge graphs, presenting the methods for representing knowledge and generating embeddings for nodes and relationships, which facilitate efficient computation in downstream tasks. Section 2.5 concludes the background chapter with a formal description of vector spaces, a crucial aspect of deep learning and modern NLP. It also formalizes the intuition behind linear transformations and operations for computing vector similarities.

2.1. Deep Learning

Deep Learning (DL) is a subset of Machine Learning (ML), a field within Artificial Intelligence (AI) that strives to create systems capable of performing tasks without explicit programming. What sets deep learning apart is its emphasis on neural networks, which are mathematical models inspired by the human brain's interconnected neurons that process information by learning representations of data in a non-linear way. The term "deep" in deep learning refers to the layers that make up these neural networks. If shallow neural networks might have one or two layers of interconnected nodes, deep learning models can comprise multiple layers in a single model to efficiently learn patterns, features, and representations from the input data. The main idea behind deep learning is to remove the need of handcrafting features and to automate this process by letting the model learn the features itself (Eistenstein, 2019). This idea has garnered significant attention and acclaim for its

remarkable success in various AI applications, including computer vision, speech and natural language processing, etc. Especially in the field of NLP, deep learning has evolved as the preferred approach to process text through several neural network architectures. This section is dedicated to explaining the different learning paradigms, common to both machine learning and deep learning, spanning from traditional supervised learning to the innovative concept of self-supervision. It also introduces the background on neural networks and emphasizes the state-of-the-art Transformer architecture, which plays a crucial role in developing sophisticated Pre-trained Language Models (PLMs).

2.1.1. Learning Paradigms

In the realm of machine learning and deep learning, learning refers to the process by which a model improves its performance on a specific task through exposure to data. Learning involves the adjustment of the model's parameters through loss functions and optimization algorithms, such that the network can effectively encode patterns and relationships discerned from the training data. More details about this process are explained in Section 2.1.2. This iterative refinement allows the model to generalize its understanding and make accurate predictions or classifications on new, unseen data (Berner et al., 2022). Learning can occur through a variety of ways, be it fully supervised, semi-supervised, self-supervised, or completely unsupervised.

Supervised Learning. The most common paradigm is supervised learning, a paradigm where a model is trained on a labeled dataset, usually curated by humans. In this setup, every input is paired with a designated target output. The model is taught to associate inputs with outputs by tuning its parameters through an optimization process aimed at reducing the difference between its predicted values and the actual labels. The training samples usually have the form of $\{(\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is the feature vector and y_i is the corresponding label for the i-th training example. Given this data, the objective is to compute a scoring function $f : X \times Y \rightarrow \mathbb{R}$, where X is the input space and Y the output space, such that the output $\hat{\mathbf{y}}$ represents an approximation of the true label for a pair of training examples: $\hat{\mathbf{y}} = \underset{y}{\operatorname{argmax}}\, f(\mathbf{x}, y)$ (Berner et al., 2022). Supervised learning is commonly employed for classification tasks, including text, speech and image classification, as well as for regression tasks, such as stock price forecasting, temperature prediction, and numerical analysis.

Unsupervised Learning. Creating a labeled dataset can be a time-consuming task, since it usually requires humans to annotate each sample in the dataset, the latter often containing millions of data points. Unsupervised learning addresses this

problem and provides a paradigm in which the algorithm is capable of predicting outputs from unlabeled data. If supervised learning can be formalized with one function as described in the previous paragraph, unsupervised learning has different formalisms depending on the task. Taking for instance clustering, the task is to group unlabeled data based on their similarities or differences by computing a clustering function $C : X \rightarrow \{1, 2, \cdots, k\}$, where X is the training dataset containing $(x_1, x_2, \cdots, x_n)$ data points and k are the number of clusters, such that each data point can be assigned to a specific cluster (Eistenstein, 2019). A second common task in unsupervised learning is dimensionality reduction, useful to reduce the number of features or dimensions in a dataset while preserving its original integrity and distribution. This can be formalized by defining a transformation function $T : \mathbb{R}^d \rightarrow \mathbb{R}^k$, where $k < d$. The function T then maps the high-dimensional vector $\mathbf{x}$ into a lower dimensional vector $T(\mathbf{x})$. T can be implemented by a variety of algorithms, such as Principal Component Analysis (PCA), Linear Discriminative Analysis (LDA), and t-distributed Stochastic Neighbor Embedding (t-SNE). Learning without labeled data motivates the concept of self-supervision, which plays a crucial role in learning language models.

Self-supervised Learning. Self-supervised learning might appear an unsupervised learning technique, but it differs from the latter because it still relies on a supervisory signal. However, this signal does not come from annotations, but is generated by the model based on the input data itself. Differently from unsupervised learning, self-supervised learning makes use of a supervised loss function to predict one part of the input using the rest or to reproduce the input data. Nevertheless, the generated supervisory signal mainly serves the purpose to create a representation of the data that can be used for various downstream tasks, such as information retrieval, text generation, data compression and representation learning.

Within the context of deep learning, one of the first examples of self-supervised models are Autoencoders (Rumelhart et al., 1986b) and Variational Autoencoders (Kingma and Welling, 2013). They rely on a type of learning paradigm called autoassociative self-supervised learning, in which they are trained to reconstruct their input data. They are usually composed of an encoder and a decoder component. The encoder E_ϕ is mostly a neural network that maps input data $\mathbf{x} \in X$ to a latent space $\mathbf{z} = E_\phi(\mathbf{x})$. Given this latent representation $\mathbf{z}$, the objective of the decoder component of the model D_θ is to recreate a representation of the input data $\mathbf{x}' = D_\theta(\mathbf{z})$ (Zhai et al., 2018).

In the realm of natural language processing, self-supervised learning has been widely used in training language models and learning word embeddings. The objective in this case is to capture the semantic and syntactic structure of language

by training models to predict contextually related words or, for instance, to classify whether a sentence is the continuation of the previous one. Examples of such approaches include the skip-gram model and Continuous Bag-of-Words (CBOW) model introduced by word2vec (Mikolov et al., 2013), and more recently, Transformer-based models like BERT (Devlin et al., 2019). These models leverage the idea of predicting words based on their context in a self-supervised manner, using large-scale text corpora. Language models can also be learned in an autoregressive way using a classical language modeling objective (see Section 2.3 for more details), where, given a corpus of words $X = (x_1, \ldots, x_n)$, the objective is to minimize a loss function $\mathcal{L}$ by maximizing the log likelihood of a word x_i conditioned on a context window of size n as depicted in Equation 1.

$$\mathcal{L} = -\sum_i \log p\left(x_i \mid x_{i-n}, \ldots, x_{i-1}; \theta\right) \tag{1}$$

The probability p is computed by a machine learning model, for instance, a neural network with parameters θ. The representations learned by such models can then be fine-tuned for various downstream tasks for increased performance and generalization.

A detailed explanation of other learning paradigms, such as semi-supervised learning and reinforcement learning would go beyond the scope of this thesis. The learning paradigms described in this section can be applied to a variety of machine learning models, depending on the problem and on the type of data. Within the context of natural language processing and especially in the realm of unstructured data, neural networks have proven to be the preferred choice.

2.1.2. Neural Networks

In its most general definition, an Artificial Neural Network (ANN)[1] with one layer is a function that takes as input an n-dimensional vector $\mathbf{x} \in \mathbb{R}^n$ and produces an output $\hat{\mathbf{y}}$ by computing Equation 2.

$$\hat{\mathbf{y}} = f\left(\theta^T \mathbf{x} + \mathbf{b}\right) \tag{2}$$

f is an activation function, $\mathbf{b}$ is a bias term and θ^T are the weights of the model, which are learned during training together with $\mathbf{b}$ (Goodfellow et al., 2016). The **activation function** is one of the main components of neural networks and it is responsible for introducing non-linearity into the network. Otherwise, it would perform a simple linear transformation. Trough non-linear layers, neural networks can capture complex relationships within the data that would otherwise not be possible using linear models only. Common activation functions are sigmoid:

$$f(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \tag{3}$$

hyperbolic tangent (tanh):

$$f(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}, \tag{4}$$

and the widely used Rectified Linear Unit (ReLU):

$$f(x) = \text{ReLU}(x) = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

As illustrated in Figure 1, activation functions commonly squash the input values to a certain range, be it between 0 and 1 for sigmoid, -1 and 1 for tanh. However, ReLU sets negative values to 0 and passes positive values unchanged.

Multiple Layers. When multiple interconnected neuron layers are combined with non-linear activation functions $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$, where $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ are the first, second, and third layer respectively, so-called feedforward neural networks, deep feedworward networks, or Multilayer Perceptrons (MLPs) can be achieved (Goodfellow et al., 2016). MLPs can be formalized in matrix notation with Equation 6 (Aggarwal, 2023).

$$\mathbf{a}^n(\mathbf{x}) = f\left(W^n \mathbf{a}^{n-1}(\mathbf{x}) + \mathbf{b}^n\right) \tag{6}$$

[1]The terms *artificial neural network* and *neural network* are used interchangeably in this thesis.

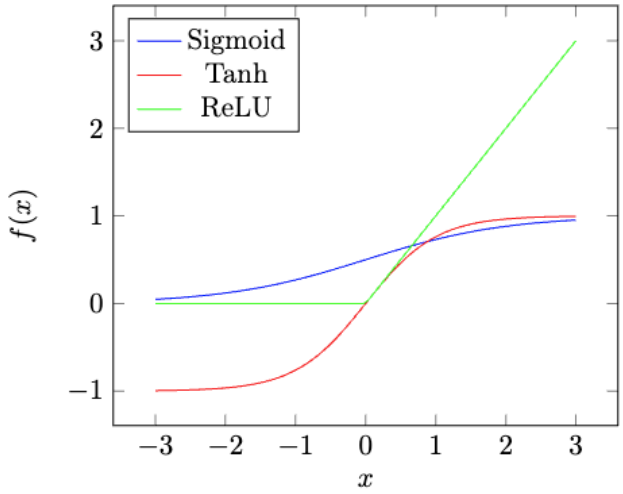

Figure 1.: Graphic representation of the sigmoid, tanh and ReLU activation functions

Compared to Equation 2, the parameters are now a weight matrix W instead of a one-column vector θ^T, n is the index of the layer, with $n = 1$ being the input layer and $n = N$ being the output layer, $\mathbf{a}^n$ is the activation vector for layer n, f is the activation function applied element-wise and $\mathbf{b}^n$ is the bias vector for layer n. For the input layer $n = 1$, $\mathbf{a}^0 = \mathbf{x}$, where $\mathbf{x}$ is the input feature vector. For the output layer $n = N$ it is common to use different activation functions, such as softmax for multiclass classification tasks to turn the output values into probability distributions. These networks are called feedforward because the information flows in a forward direction without feedback connections, which were introduced later with Recurrent Neural Networks. Every training example is accompanied by a label y, which represents the correct output that the function tries to approximate. The output layer of the neural network is responsible for outputting a value as close as possible to the value represented by y. Therefore, the training data affect only the behavior of the output layer, but not that of the intermediate layers, hence the term *hidden layers* (Goodfellow et al., 2016). The role of the hidden layers is that of transforming the network's vector space to best capture dependencies between data and they are typically represented as hyperparameters, i.e., external configuration settings that are not learned from data but are set before the training process.

Optimization. Since the parameters of the neural network are often initialized randomly from normal or uniform distributions, the performance of the model after the first forward pass will most likely not be optimal. Nevertheless, it is

common to know the desired output y and optimization techniques are used to iteratively tune the parameters of the model to minimize a so-called **loss function**. Loss functions play a pivotal role in quantifying the difference between predicted outputs and actual target values, thereby guiding the model's learning process. Two commonly used loss functions are cross-entropy loss and mean-squared error. The cross-entropy loss, particularly suitable for classification tasks, measures the dissimilarity between predicted probability distributions and true class distributions. It is expressed as depicted in Equation 7.

$$\text{CrossEntropy}(\hat{y}, y) = -\sum_{i} y_i \cdot \log(\hat{y}_i) \tag{7}$$

In Equation 7 $\hat{y}_i$ represents the predicted distribution for the i-th value and y_i denotes the true distribution. For regression tasks, however, where the neural network needs to predict continuous scalar values, a common loss function is mean-squared error. It calculates the average squared difference between predicted and actual values. It is defined as in Equation 8.

$$\text{MSE}(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{8}$$

Here, N represents the number of data points, $\hat{y}_i$ is the predicted value, and y_i is the true value. Gradient-based algorithms, such as the **backpropagation** algorithm (Rumelhart et al., 1986a), have become the de-facto standard solutions to optimize neural networks and to reach small losses that imply a good overall performance of the model. After the forward computation, the backpropagation algorithm calculates the gradient of the loss function relative to the model's parameters, denoted as $\nabla_\theta \mathcal{L}(\theta)$, where θ represents the weights and biases of the network. According to Goodfellow et al. (2016), backpropagation systematically applies the chain rule of calculus to find these gradients, starting from the output layer and working backward through each layer in the network. Specifically, for each neuron, the error derivative is calculated by Equation 9.

$$\delta_j^n = \left(\sum_k w_{jk}^{n+1} \delta_k^{n+1} \right) \sigma'\left(z_j^n\right) \tag{9}$$

δ_j^n is the error term for node j in layer n, w_{jk}^{n+1} is the weight from node j in layer n to node k in layer $n+1$, and σ' is the derivative of the activation function applied to the input sum z_j^n at that node. The gradient of the loss $\mathcal{L}$ with respect to the weights is computed using the chain rule of calculus, as shown in Equation 10 (Aggarwal, 2023).

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^n} = \frac{\partial \mathcal{L}}{\partial a_j^n} \cdot \frac{\partial a_j^n}{\partial w_{ij}^n}, \tag{10}$$

In Equation 10 w_{ij}^n represents the weight for node j in layer n for the incoming node i, a_j^n is the activation of node j in layer n before passing to the activation function to generate the output. The weights and biases, denoted with θ, are then updated using the gradient and a learning rate α:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial L}{\partial \theta} \tag{11}$$

where θ^t are the parameters of the neural network at the t-th iteration of gradient descent. The parameters update rule follows the gradient descent principle and adjusts the weights and biases in the direction opposite to the gradient of the loss function, aiming to move towards the minimum of the loss surface. The learning rate α controls the step size, ensuring a balance between convergence speed and stability. This iterative process of computing gradients and updating parameters is performed across multiple epochs until the model converges to a state of minimal loss.

Recurrent Neural Networks. In the realm of natural language processing and sequence modeling, feedforward networks are not the ideal architecture to process text. Their simple structure comes with drawbacks since they do not allow for variable size of input and output neurons, the parameters are not shared and therefore fail in capturing long-range dependencies between words. Recurrent Neural Networks (RNNs) are a family of deep neural networks that remained the state-of-the art for several decades thanks to their increased performance. Unlike traditional feedforward neural networks, the potential of RNNs comes from having the ability of using the results of the previous output as context for the next input. In other words, RNNs sequentially handle input elements, preserving a so-called state vector in their hidden units that encapsulates information regarding the entire history of preceding elements in the sequence (LeCun et al., 2015). The mechanism is illustrated in Figure 2. Mathematically, an RNN updates a hidden state $\mathbf{h}_{\langle t \rangle}$ at each time step t, as depicted in Equation 12.

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, \mathbf{x}_t\right) \tag{12}$$

This ability to maintain a hidden state, or memory, empowers RNNs to effectively process and generate sequences of varying lengths. However, standard RNNs face challenges, such as the vanishing gradient problem, which hinders the learning of long-term dependencies. In this context, when optimizing RNNs through gradient-based algorithms, such as backpropagation through time, gradients diminish exponentially as they are propagated over long sequences. This leads to difficulties in updating the weights of earlier time steps during training, hindering the learning of long-term dependencies and causing information loss in the network's

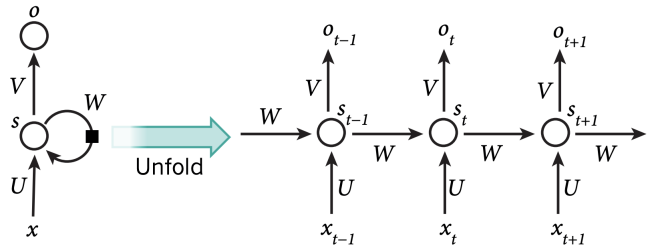

Figure 2.: Unfolding a Recurrent Neural Network with one hidden unit S. The input sequence x is mapped to the output sequence o, where each element o_t of o depends on the previous x'_t The parameters U, V, W are shared and used at each time step (LeCun et al., 2015, 442)

memory. The vanishing gradient problem is particularly pronounced when using activation functions that squash their input, such as the sigmoid or hyperbolic tangent functions. To address this limitation, more advanced variants like Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) networks and Gated Recurrent Units (GRUs) (Cho et al., 2014) have been introduced. These architectures incorporate specialized mechanisms to selectively update and forget information, mitigating the vanishing gradient issue and enhancing the modeling of long-range dependencies.

Convolutional Neural Networks. Another family of deep neural networks are Convolutional Neural Networks (CNNs) (LeCun et al., 1989). They have been predominantly used for computer vision tasks, but in recent years, they have also been applied to NLP problems, such as machine translation (Gehring et al., 2017a), text classification (Kalchbrenner et al., 2014; Zhang et al., 2015) and language modeling (Pham et al., 2016; Dauphin et al., 2017a). Moreover, they are often applied to graph data to learn useful features for nodes (Gehring et al., 2017b; Niepert et al., 2016). In the context of NLP, CNNs can be used to capture meaningful features within a sequence of words or characters, such as n-grams. The architecture of CNNs consists of multiple layers, where the core component is the convolutional layer, consisting of multiple filters or kernels that slide across the input text and perform a convolution operation. Mathematically, the convolution function S_{ij} between an input matrix X and a filter W is depicted in Equation 13.

$$S_{ij} = (X * W)_{i,j} = \sum_m \sum_n X_{i+m,j+n} W_{m,n}) \tag{13}$$

In Equation 13, $*$ denotes the convolution operation and i and j represent the row and column indices in the output matrix resulting from the convolution operation

between the input matrix X and the filter W. m and n are indices that iterate over the dimensions of the filter W. After the convolution, a non-linear activation function is applied element-wise to introduce non-linearity. To further distill the most informative features and reduce dimensionality, CNNs employ pooling layers. Max pooling, for instance, reduces the size of the feature maps by selecting the maximum value from each subregion defined by the pooling window. A CNN, depending on the task, can end with a feedforward fully connected layer, which takes as input the flattened result of the pooling operation and further enhances the vector to produce scores.

2.1.3. Transformer Architecture

The Transformer (Vaswani et al., 2017) is a powerful deep neural network originally designed for sequence-to-sequence tasks to address the limitations of RNNs and CNNs in capturing long-range dependencies. Even though LSTMs and GRUs served as a powerful alternative to vanilla RNNs, the Transformer proposes a novel approach that introduces parallelism in the processing of token sequences and therefore reduces overall sequential computation time. The architecture is illustrated in Figure 3 and is comprised of an encoder and a decoder block. The encoder-decoder architecture is necessary for sequence-to-sequence tasks, such as translation and summarization. Nevertheless, thanks to its modular design, the Transformer can be used as an encoder-only model, for tasks like representation learning and text classification, or as a decoder-only model for auto-regressive tasks, such as text generation, image captioning and music generation.

Generally speaking, the encoder takes as input a sequence of tokens $\mathbf{x} = (x_1, \ldots, x_n)$, usually sub-word tokens, such as Byte-Pair Encoding (BPE) (Sennrich et al., 2016) tokens or wordpieces (Wu et al., 2016), and maps $\mathbf{x}$ to a sequence of continuous representations $\mathbf{z} = (z_i, \ldots, z_n)$. Given these continuous representations $\mathbf{z}$, the decoder generates an output sequence $\mathbf{y} = (y_1, \ldots, y_n)$ in an autoregressive way, meaning that it accesses the previously generated tokens before outputting the next one. According to this general architecture, the Transformer stacks N multiple encoder and decoder blocks, comprised of Multi-Head Attention (MHA) and feedforward components, for better representation learning and increased performance.

Self-Attention. The main component of the Transformer architecture that introduces the ability to process sequences in parallel is **self-attention**, indicated by the Multi-Head Attention blocks in Figure 3. Attention serves as a mechanism within a neural network that allows a model to make predictions by selectively focusing on specific tokens within the input data. The degree of attention is

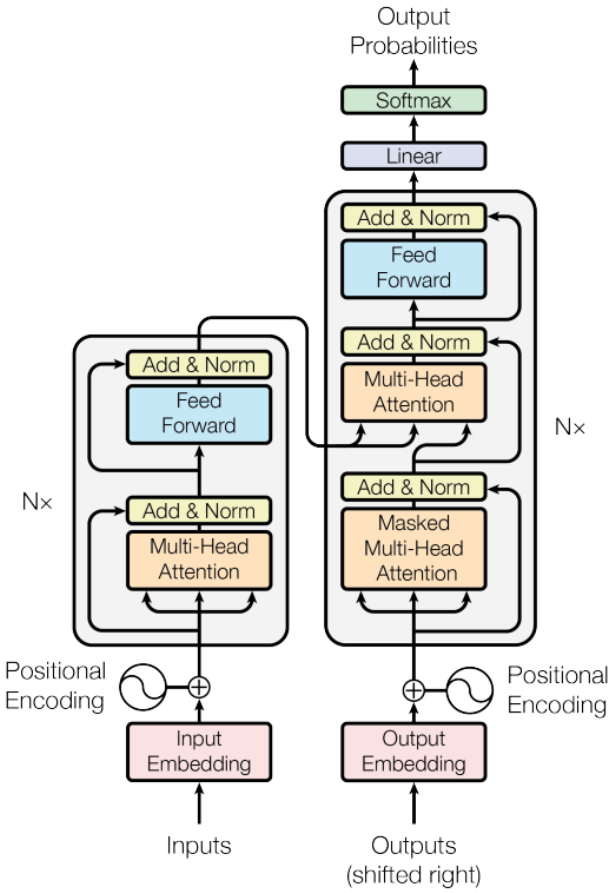

Figure 3.: Encoder-decoder Transformer architecture (Vaswani et al., 2017, 3)

determined by learned weights, resulting in an output typically generated as a weighted average. Self-attention is a specific type of attention mechanism that enables the model to predict one part of a data sample based on other parts of the same observation. The Transformer uses a specific type of self-attention called **Scaled Dot-Product Attention**. Each token in a sequence is associated with three vectors: a query vector Q, a key vector K, and a value vector V. These vectors are derived from the input embeddings of the tokens, which are multiplied with trainable weight matrices W^Q, W^K and W^V. The self-attention mechanism determines attention scores by calculating the dot product between the query vector of a specific token and the key vectors of all tokens within the sequence. The resulting scores are then scaled by the square root of the dimension of the key vectors d_k for numerical stability. A softmax function is applied to obtain normalized attention weights, which are used to compute a weighted sum of the corresponding value vectors. This process allows each tokens to attend to different

parts of the sequence, capturing contextual relationships. Mathematically, scaled dot-product attention can be expressed in a very compact form as depicted in Equation 14.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{14}$$

Here, Q, K, and V are matrices with a dimensionality $d_{\text{model}} \times seq_length$, seq_length being the length of the input sequence and d_{model} being a hyperparameter indicating the embedding size. These three matrices represent the query, key, and value vectors for all tokens in the sequence. Representing scaled dot-product attention with matrices, as in Equation 14, allows to compute the attention function simultaneously for all tokens in a sequence.

Instead of computing attention once for a whole sequence, Vaswani et al. (2017) found it beneficial to extend the capabilities of scaled dot-product attention by employing multiple attention heads in parallel. Operating on different linear projections of the input, each attention head gives the model diverse perspectives on the relationships within the input data. It then follows a concatenation and a linear transformation of the outputs from each head to obtain the expected dimensions:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}\left(\text{head}_1, \ldots, \text{head}_\text{h}\right) W^O \\ \text{where head}_\text{i} &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{15}$$

where $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are projection matrices and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ is the output linear transformation.

Position-wise Feedforward Networks. Although the self-attention mechanism does incorporate non-linear transformations through the softmax function, additional non-linearity and complexity are introduced by position-wise feedforward networks. These networks, which consist of simple multilayer perceptrons, process each position separately through fully connected layers and activation functions. In the original work, Vaswani et al. (2017) use a dimensionality of $d_{\text{model}} = 512$ for the input and output layer, 2048 as the dimensionality of the hidden layer, and employ the ReLU activation function between layers.

Positional Encoding. Since the Transformer is processing tokens in a sequence simultaneously and it does not use recurrence or convolution, it does not have a notion of the position of tokens. Positional encoding is therefore used to inject position into the embeddings before passing through the self-attention mechanism. Positional encoding $P \in \mathbb{R}^{d_{\text{model}} \times i}$ is a matrix with the same dimension of the input token embeddings so that it can be added directly to them before they are fed

into the model, making sure that the model retains information about the relative positions of the tokens. The encoding can be done using various methods, such as learned positional encoding (Gehring et al., 2017b), relative positional encoding (Shaw et al., 2018) and rotary positional encoding (Su et al., 2024), but the original work of Vaswani et al. (2017) proposes a specific sinusoidal encoding function:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{(2i/d_{\text{model}})}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{(2i/d_{\text{model}})}\right) \end{aligned} \tag{16}$$

pos is the position of the token in the sequence, i is the dimension of the positional encoding, and d_{model} is the dimension of the token embeddings. The choice of sinusoidal functions allows the model to learn to attend to relative positions since the addition of these positional encodings results in a continuous and differentiable representation that can be incorporated into the model's learning process.

2.2. Vector Representations of Language

Computers process numbers and are unable to encode meaning using text represented as strings. Therefore, when working with textual data, it would not be beneficial to input strings into any machine learning model. The text needs to be transformed into a numerical representation, ideally a vector. There are various methods to convert text into numerical representations, and this section will investigate the most significant ones. It begins by exploring fundamental statistical word representations, such as word-context matrices, Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). The discussion then transitions to distributional word representations, encompassing the word2vec and GloVe algorithms, before concluding with contextual representations learned by powerful deep neural networks.

2.2.1. Statistical Word Representations

Vector representations, especially in NLP tasks, need to have a notion of semantics, since otherwise they would be simple arrays of random numbers. It is possible to define vector semantics through the distributional hypothesis, such that words that appear in similar contexts, usually have similar meanings (Firth, 1957). This powerful idea can be applied to develop simple count-based models that leverage corpus statistics to create meaningful word representations. The main concept of count-based models is to construct matrices containing word frequencies. For instance, the matrix W in Equation 17 is a **word-context matrix** (Landauer and Dumais, 1997), where the rows represent the individual words and the columns some context words. Each cell contains a value c indicating the number of times word i appears together with context j.

$$W = \begin{pmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1n} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2n} \\ c_{31} & c_{32} & c_{33} & \dots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \dots & c_{nn} \end{pmatrix} \tag{17}$$

Since the matrix W is usually the size of the vocabulary, it can become very large and inefficient during processing. Moreover, many of the recorded counts are zeros. It is therefore common to apply dimensionality reduction techniques, such as Singular Value Decomposition (SVD) or Principal Component Analysis (PCA), to extract only the informative elements of the matrix and to reduce overall computation time.

The matrix described by Equation 17 can also be modified to capture document-level features. Instead of having the columns of the matrix representing context

words, it is possible to assign the columns to specific documents. In this way, each cell in the matrix counts the occurrences of words withing a document, such that two documents are similar if they have a similar word count. This type of model is called **word-document matrix**, or **Bag-of-Words** (BoW), since there is not a notion of the order or the structure of words within the documents.

Nevertheless, these count-based methods are very naive, since they disregard several linguistic properties, and, more importantly, they treat every word equally. Very frequent words, such as stop words, can negatively affect the semantic representations. To address this issue, measures like Pointwise Mutual Information (PMI) (Church and Hanks, 1990) can be employed to normalize the probability of co-occurrence between two words w_i and w_j by their individual occurrence probabilities, as shown in Equation 18.

$$PMI\left(w_i, w_j\right) = \log_2 \frac{P\left(w_i, w_j\right)}{P\left(w_i\right) P\left(w_j\right)} \tag{18}$$

PMI values range between $-\infty$ to $+\infty$, with negative values suggesting a co-occurrence less likely to occur by chance. When applying Equation 18 to stop words, it will typically result in high P values, but diminished overall PMI values.

Similarly, **Term Frequency-Inverse Document Frequency** (TF-IDF) is another technique used to address the issue of very frequent words. It measures the importance of a word in a document relative to a corpus by computing $f_{t,d}$, i.e., how many times term t occurs in document d. The term frequency tf:

$$\mathrm{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}, \tag{19}$$

is multiplied with the inverse document frequency idf:

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}, \tag{20}$$

where N is the total number of documents and the denominator depicts the number of documents d in the collection D where term t appears. Putting together Equation 19 and Equation 20, results in Equation 21.

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D) \tag{21}$$

This approach assigns higher weights to words that are frequent in a document but rare in the entire corpus.

2.2.2. Global Word Embeddings

The count-based models discussed in Section 2.2.1 were state-of-the-art for a long time. Thanks to their simple, yet effective structure, they enabled to create

meaningful representations of text. Nevertheless, they fail to capture complex linguistic properties and are often high-dimensional, since the vectors are typically the size of the entire vocabulary. Moreover, they are sparse, meaning they contain a lot of zeros, which make processing highly inefficient. Word embeddings have emerged as a powerful alternative to count-based models since they employ learning algorithms, such as neural networks, to embed words as dense vectors in a continuous vector space, where similar words are positioned close to each other. The general idea is to learn to predict words based on their surrounding context, encoding semantic relationships between words as continuous vectors. The continuous nature of predictive models allows for more flexible and nuanced representations of language compared to the discrete representations of count-based models.

Word2Vec. The word2vec algorithm (Mikolov et al., 2013) proposes a novel technique to learn efficient and accurate low-dimensional distributed word representations by employing shallow neural networks with a language modeling objective. Differently to previous attempts (Bengio et al., 2000; Mikolov et al., 2010), which employed more sophisticated models, but with an increased computational complexity due to inherent complexity of non-linear layers, Mikolov et al. (2013) decided to use much simpler feedforward neural networks to learn word embeddings. Their hypothesis was to take advantage of the simple model structure to train on larger datasets more efficiently. Word2vec proposes two models: Continuous Bag-Of-Words (CBOW) and skip-gram.

The **CBOW** model is trained to predict a single center word given the surrounding words in a predifined context window. It minimizes the negative log likelihood loss function $\mathcal{L}$, depicted by Equation 22 (Mikolov et al., 2017).

$$\mathcal{L} = -\sum_{t=1}^{T} \log P\left(w_t \mid W_t\right) \tag{22}$$

In Equation 22, $W_t = (w_{t-n}, \ldots, w_t, \ldots, w_{t+n})$ is the sequences of words in the slicing window n, T is the total number of target words in the training dataset and w_t is the target word that needs to be predicted.

The **skip-gram** model takes a different approach and predicts the surrounding context words given a single center word. In this model, the objective is to maximize the probability of observing the context word w_{t+j} located j positions away from the center word w_t. This is achieved by minimizing once again the negative log likelihood loss function $\mathcal{L}$ as depicted in Equation 23.

$$\mathcal{L} = -\sum_{t=1}^{T} \sum_{-m \leq j \leq m} \log P\left(w_{t+j} \mid w_t\right) \tag{23}$$

m is the size of the context window, which determines the range of context words to consider around the target word w_t, T is the total number of target words in the training dataset and $P(w_{t+j} \mid w_t)$ can be formulated as a softmax function, as shown in Equation 24.

$$p(o \mid c) = \frac{\exp\left(\mathbf{u}_o^T \mathbf{v}_c\right)}{\sum_{w \in V} \exp\left(u_w^T \mathbf{v}_c\right)}. \tag{24}$$

Here V is the vocabulary of words w in the dataset, $\mathbf{u}_o$ and $\mathbf{v}_c$ are the vector representations of the context word o and the central word c, respectively and u_w are all context words in the vocabulary. The skip-gram model aims to learn embeddings, such that the probability of observing context words given the center word is maximized. Figure 4 offers a graphical overview of the CBOW and skip-gram models.

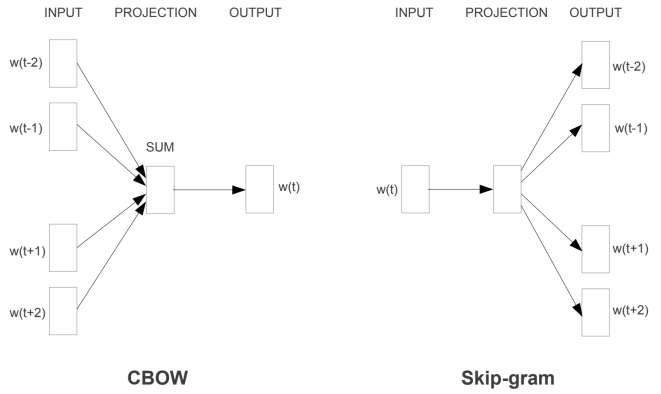

Figure 4.: CBOW and skip-gram architectures. CBOW aims at predicting the center word w_t given some context words, whereas skip-gram is trained to predict the context words given a center word (Mikolov et al., 2013, 5)

GloVe. Another powerful algorithm for learning word embeddings is GloVe (Pennington et al., 2014). GloVe is an alternative unsupervised method that combines count-based models like word-context matrices and neural network-based models like word2vec, but instead of focusing only on local word co-occurrence statistics, it leverages global word co-occurrence statistics obtained from the entire corpus. The main idea is to use the ratio of co-occurrence probabilities between two words w_i and w_j with a third word w_k $P(w_i, w_k)/P(w_j, w_k)$ to gauge their semantic associations, rather than focusing solely on the co-occurrence probability $P(w_i, w_j)$. While GloVe does not employ neural networks for learning word vectors, it can

still be considered a predictive model since it adopts optimization algorithms like Stochastic Gradient Descent (SGD) to optimize a non-convex function (Pilehvar and Camacho-Collados, 2020). The optimization objective is to solve Equation 25.

$$w_i^T w_k + b_i + b_k = \log(1 + X_{ik}) \tag{25}$$

where b_i and b_k denote bias terms for words w_i and w_k, respectively and X_{ik} represents the count of co-occurrence between w_i and w_k. To prevent divergence when X_{ik} is zero and to train solely on non-zero elements, a 1 is added to the logarithm. Through optimization, GloVe constructs vectors w_i and w_k whose dot product provides an effective estimate of their transformed co-occurrence counts (Pilehvar and Camacho-Collados, 2020). Nevertheless, the optimization objective described by Equation 25 weighs all entries equally and to overcome this problem, Pennington et al. (2014) introduced a weighting function $f(X_{ij})$ into the loss function $\mathcal{L}$ to transform Equation 25 into a least squares regression model:

$$\mathcal{L} = \sum_{i,j=1}^{V} f\left(X_{ij}\right) \left(w_i^T w_j + b_i + b_j - \log X_{ij}\right)^2 \tag{26}$$

where V is the size of the vocabulary. The function f should follow certain conditions: $f(0) = 0$ and $f(x)$ should be non-decreasing to avoid overweighting rare co-occurrences while also not becoming overly large for large x values to avoid overweighting frequent words. A suitable choice for $f(x)$ is as follows:

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise,} \end{cases} \tag{27}$$

where α and x_{max} are hyperparameters. α governs the weighting assigned to values relative to x_{max}, while x_{max} represents a threshold for determining which values should be weighted differently. Values below this threshold will be scaled according to the weighting function, whereas values equal or above it will receive a fixed weight of 1. Pennington et al. (2014) found optimal performance with $\alpha = 3/4$ and $x_{max} = 100$.

Interesting capabilities that emerge after the training of word embeddings, such as the word2vec anf GloVe models, see the application of algebraic manipulations on the word vectors to reveal latent linguistic correlations (Mikolov et al., 2013). For instance, a well-trained model, is able to approximate the embedding E of word "mother" by performing the operation $E(\text{father}) - E(\text{man}) + E(\text{woman})$. Other properties of these vector spaces are the ability to perform analogical reasoning, such that $E(\text{sushi}) + E(\text{Germany}) - E(\text{Japan}) \approx E(\text{Bratwurst})$, or $E(\text{Paris}) + E(\text{Italy}) - E(\text{France}) \approx E(\text{Rome})$.

2.2.3. Contextual Embeddings

Global word embeddings have significantly advanced NLP tasks by capturing semantic and syntactic relationships between words based on their co-occurrence statistics. However, they suffer from several limitations. One major drawback is their inability to capture word meanings in different contexts, leading to the same word being represented by a single vector regardless of its usage across diverse linguistic contexts. This limitation poses challenges for tasks requiring nuanced understanding of word semantics. When using global word embeddings in downstream tasks, the model has the task of trying to disambiguate the word representation for different contexts and to derive its meaning, leading to a reduced performance and accuracy (Pilehvar and Camacho-Collados, 2020). Additionally, the model's ability to capture compositionality and long-term dependencies is constrained by the limitations of individual word barriers. To overcome this limitation, contextual embeddings have emerged as a transformative solution. Unlike global word embeddings, contextual embeddings generate representations dynamically based on the specific context in which the word appears.

One main difference of contextual embeddings is how they process words. While global word embeddings represent entire, single words, contextual embeddings typically represent sub-word tokens to handle out-of-vocabulary words and enhance compositionality. Instead of inputting a single word into the model, contextualized models receive a sequence comprising the target token along with its context. When computing an embedding for a specific input token $\mathbf{x} \in X$, where $X = (x_1, \ldots, x_n)$, the model analyzes the whole context X and produces an n-dimensional contextual vector representation $\mathbf{z} \in \mathbb{R}^n$, which is generated dynamically based on its context. Consequently, the representation of the word *bank* in the sentence "I sat by the *bank* of the river" would be different than the representation of the same word in the sentence "I went to the *bank* to take some money".

Contextual representations are commonly computed using neural language models instead of simple feedforward neural networks. One of the earlier methods to compute contextual embeddings was **Context2Vec** (Melamud et al., 2016), an extension of word2vec's CBOW model. Context2Vec employs bidirectional LSTM language models to generate vector representations of context words in both directions. These LSTM output vectors are then concatenated and fed into a multi-layer perceptron to compute the context embedding.

Similarly, **ELMo** (Peters et al., 2018a) uses a forward LSTM and a backward LSTM to encode left and right contexts and to train the language model by minimizing the negative log likelihood loss function $\mathcal{L}$ using the objective described

by Equation 28.

$$\mathcal{L} = -\sum_{i=1}^{n} \Big(\log p \left(x_i \mid x_1, \ldots, x_{i-1}; \theta_e, \overrightarrow{\theta}_{\text{LSTM}}, \theta_s \right) + \log p \left(x_i \mid x_{i+1}, \ldots, x_n; \theta_e, \overleftarrow{\theta}_{\text{LSTM}}, \theta_s \right) \Big) \tag{28}$$

In this setup, an input sequence of n tokens $(x_1, \ldots, x_n)$ is fed into the forward LSTM $\overrightarrow{\theta}_{\text{LSTM}}$ and then concatenated with the representations from the backward LSTM $\overleftarrow{\theta}_{\text{LSTM}}$. The variables θ_e and θ_s denote the parameters of the token representation and the softmax layer, respectively. At each layer j, the resulting contextual representations are formed by concatenating the left-to-right and right-to-left hidden states, yielding n hidden representations $(\mathbf{h}_{1,j}, \mathbf{h}_{2,j}, \ldots, \mathbf{h}_{n,j})$.

The Transformer architecture described in Section 2.1.3 is widely used for implementing language models capable of producing high-quality contextual representations. The self-attention mechanism is particularly suitable to capture dependencies in the input sequence and to therefore adjust the representations based on the surrounding context. Following this novel architecture, several Transformer-based language models have been trained on large datasets using different self-supervised objectives. The Generative Pre-trained Transformer (**GPT**) family (Radford et al., 2018, 2019; Brown et al., 2020) employs N Transformer decoder blocks to compute contextual hidden representations in an autoregressive way using a classical, autoregressive language modeling objective. Their training procedure involves a pre-training stage for learning the language model and a supervised fine-tuning stage. The Bidirectional Encoder Representations from Transformers (**BERT**) model (Devlin et al., 2019) is another innovative technique to learn contextual embeddings. Unlike the autoregressive approach of GPT, BERT employs a bidirectional architecture based on Transformer encoders only, enabling it to consider both left and right contexts simultaneously. Instead of reading the sequence left-to-right and predicting the future token as in generative models, BERT employs two self-supervised tasks to predict tokens and sentences. The initial task involves the Masked Language Modeling (MLM) objective, in which a portion of the input tokens is randomly masked, and the model is tasked with predicting these masked tokens using the context provided by their surrounding words. Unlike directional models, which are limited to reading the input sequence left-to-right or right-to-left, the MLM task allows BERT to freely consider the context on both sides of a masked token. Moreover, the Next Sentence Prediction (NSP) objective is used during training to predict whether a second sentence is the continuation of the previous one, aiming to improve tasks like question answering and natural language inference. Similar to GPT, BERT undergoes pre-training on large corpora to learn robust language representations, followed by fine-tuning on specific downstream

tasks for optimal performance.

Other powerful language models, such as RoBERTa (Liu et al., 2019), TransformerXL (Dai et al., 2019), XLNet (Yang et al., 2019), T5 (Raffel et al., 2020), and BART (Lewis et al., 2020), have been developed to learn high-quality contextual representations, but a detailed explanation of their architectures and objectives would go beyond the scope of this section.

2.3. Language Models

Language modeling has gained massive attention because of its simple, yet very powerful objective. It turns out that predicting the next word in a sequence or some missing words in a sequence enables the model to learn powerful features and characteristics of natural language and therefore of how humans communicate in the real world. Moreover, training language models does not require annotated datasets, since their training objectives rely on the concept of self-supervised learning explained in Section 2.1.1. The previous sections introduced the concept of language models and their vital role for learning meaningful representations of text. This section is dedicated to introducing a formal description of language models, together with their different architectures and training objectives.

2.3.1. Statistical Models

In its more general definition, a language model assigns probabilities to words or sequences of words to determine how likely a sequence of words is in a specific context (Eistenstein, 2019). The goal is to approximate the distribution of natural language and to encode linguistic properties by training the model on large text corpora. The most basic approach of statistical language modeling is to calculate the probability of the next word based on counts, i.e., the number of times the sequence appears in the corpus. For instance, given a sequence $X = (x_1, x_2, x_3)$, the probability of x_4 to occur, denoted as the conditional probability $P(x_4 \mid x_1, x_2, x_3)$, is computed as depicted in Equation 29.

$$P(x_4 \mid x_1, x_2, x_3) = \frac{\text{count}(x_1, x_2, x_3, x_4)}{\text{count}(x_1, x_2, x_3)} \tag{29}$$

However, this naive method faces the problem of the curse of dimensionality because of the very large context needed to predict one single word, ending up in a model with too many parameters and reduced efficiency. The **Markov assumption** can help in these cases, since it states that the probability of a word depends only on a limited, fixed-size history. This assumption simplifies the complexity of language modeling by considering only a small window of context. Equation 30 provides a description of this method used to compute probabilities for a sequence of words $(w_1, w_2, \ldots, w_n)$.

$$P(w_1, w_2, \ldots, w_n) \approx \prod_{i=1}^{n} P\left(w_i \mid w_{i-(n-1)} \ldots w_{i-1}\right) \tag{30}$$

This approach, even though more efficient, still faces dimensionality challenges and problems of data sparsity, since the probability of encountering specific sequences

of words can be exceedingly low in large text corpora. As a result, many possible word combinations are not observed during training, making it difficult for the model to accurately predict the likelihood of such sequences occurring in real-world language use. This issue necessitates the use of smoothing techniques and more advanced modeling strategies (Witten and Bell, 1991; Katz, 1987; Ney et al., 1994).

A more concrete example of models based on the Markov assumption are **n-gram** language models, which rely on the conditional probability of a word given the previous $n-1$ words (Rosenfeld, 2000). Mathematically, an n-gram model estimates the probability of a word w_i given its history $w_{i-(n-1)} \dots w_{i-1}$ as $P\left(w_i \mid w_{i-(n-1)}, \dots, w_{i-1}\right)$. This is achieved using the chain rule of probability, as shown in Equation 31.

$$P\left(w_1, w_2, \dots, w_n\right) = P\left(w_1\right) P\left(w_2 \mid w_1\right) \dots P\left(w_n \mid w_1, w_2, \dots, w_{n-1}\right) \tag{31}$$

N-gram language models with $n = 1$ are commonly referred to as unigrams, where the probability $P(w_i)$ of a word occurring is considered in isolation, without regard to its preceding or following words. Bigrams take the analysis a step further by setting $n = 2$. In this case, the model computes the probability of a word based on the immediately preceding word, capturing pairwise word dependencies. Trigrams extend the context window to $n = 3$, conditioning the probability of word w_i on the previous two words: $P\left(w_i \mid w_{i-2}, w_{i-1}\right)$. As n increases, the model becomes more complex, moving on to 4-grams, 5-grams, and so on. These higher-order n-grams allow for an even broader context to be considered when predicting the probability of a word. However, they also introduce greater challenges in terms of computational resources and data sparsity, as the likelihood of encountering the exact sequence of words in the training corpus decreases with larger n Rosenfeld (2000). The actual parameters of the model would be the probabilities associated with each n-gram sequence within the corpus, i.e., the frequency counts of n-grams and (n-1)-grams in the training data.

2.3.2. Neural Models

Neural language models represent a significant shift in approach from traditional statistical models to one that leverage the power of neural networks. These models aim to overcome the limitations of data sparsity and high dimensionality mentioned in Section 2.3.1 by using dense vector representations, or embeddings, for words. This allows them to capture more complex patterns and relationships between words in a more compact form and to better generalize on unseen data. As explained in Section 2.2.2, embeddings represent words as vectors positioned closely to semantically similar words within the vector space, which facilitates the model's ability to understand and process novel sequences by drawing on known linguistic

relationships. Like statistical language models, neural language models learn to predict the probability of a sequence of words by training on large corpora of text, but they do so by learning a continuous space representation of words and sequences. One of the first attempts to learn distributed representations with a language modeling objective was conducted by Bengio et al. (2000) with a simple multilayer perceptron. This allowed them to simultaneously learn a distributed representation of words while computing the probability for the next words in a sequence, successfully training the language model.

The difference to early statistical approaches is that the model is now a more complex neural network with parameters θ, which has the task of computing the function described by Equation 32, with a softmax function to transform the unnormalized log probabilities y, also called logits, to probabilities summing to 1.

$$P\left(w_t \mid w_{t-1}, \cdots w_{t-n+1}, \theta\right) = \frac{\exp^{y_{w_t}}}{\sum_i \exp^{y_i}} \tag{32}$$

The words w_t, represented as one-hot encoded representations, undergo a mapping E to a real-valued vector $E(w_t) \in \mathbb{R}^d$, where E is a $V \times d$ matrix, V being the size of the vocabulary and d the dimension of the vector. Following the embedding matrix E, the model typically employs one or more hidden layers with non-linear activation functions. In the case of Bengio et al. (2000), the model consisted in just one hidden layer with a tanh non-linearity, but more powerful alternatives to MLPs can be used, such as RNNs, LSTMs, or GRUs, depending on the specific implementation of the language model. These layers are responsible for processing the sequence of word embeddings, maintaining a hidden state of the words that have been processed so far to predict the next word in the sequence.

Consider a neural language model with one hidden layer h that has the task of predicting the next word given three previous words, the algorithm would look as following:

1. **Embeddings selection from** E: The one-hot encoded representations of the three words are multiplied with the embedding matrix E. Since E has a dimensionality of $V \times d$, the resulting vector for word w_i would be $\mathbf{e}_i \in \mathbb{R}^d$. The three embeddings for the three words are concatenated, resulting in the embedding layer C.
2. **Matrix multiplication with** W: The embedding layer C is now mulitiplied with the weight matrix W and passed through a non-linear activation function to obtain the hidden layer h. Recall that the parameters of the matrix W also have the additional bias terms $\mathbf{b}$.
3. **Matrix multiplication with** U: The hidden layer h is now multiplied with an additional weight matrix $U \in \mathbb{R}^{d \times V}$, which stretches the input back to the size of the vocabulary V.

4. **Application of softmax**: To get a probability distribution for the next possible word in the vocabulary, the result of the multiplication with U is passed to a softmax activation function to transform the logits into probabilities. The word with the highest probability is chosen and compared to the ground truth using a loss function.
5. **Backpropagation of the error**: The error computed by the loss function is sent back to the previous layers and the parameters of E, W, and U are adjusted before repeating the next forward pass.

Feedforward neural networks like MLPs have a fixed architecture, meaning that they do not allow for variable size of input and output neurons. To this extent, RNNs, as explained in Section 2.1.2, offer a more advanced architecture that is specifically suitable for learning language models. RNNs offer a theoretically unlimited history length thanks to neurons with recurrent connections, which allow to maintain a memory of previous inputs by incorporating a hidden state that is passed from one step of the sequence to the next. Similarly to MLPs, each word embedding in the sequence is fed into the RNN, one after another. The RNN computes the current hidden state $\mathbf{h}_t$ based on the current word embedding $\mathbf{e}_t$. This is often represented as $\mathbf{h}_t = f(W \cdot \mathbf{h}_{t-1} + U \cdot \mathbf{e}_t + \mathbf{b})$, where f is a non-linear activation function, W is the weight matrix applied to the previous hidden state $\mathbf{h}_{t-1}$, U is the weight matrix applied to the current input embedding $\mathbf{e}_t$ and $\mathbf{b}$ is the bias vector. This step iteratively updates the hidden state with information from each word in the sequence, thereby capturing the temporal dependencies between words (Mikolov et al., 2010).

Transformer-based Language Models. With the introduction of the Transformer architecture, described in Section 2.1.3, the field of language modeling has being completely revolutionized. While the objective still relies on the self-supervised learning paradigm, this novel deep neural network architecture allows for faster training, better accuracy, and improved representations (Vaswani et al., 2017; Devlin et al., 2019). As for other neural network architectures, the input to the Transformer is a sequence of embeddings. To extract the initial embeddings, a sequence of text is transformed into numerical tokens using a trained tokenizer, such as BPE. The tokens now represent the index needed to select the corresponding embedding from the embedding matrix E, which, once again, has a dimensionality of $V \times d$. The positional embedding is added to the token embeddings and the resulting vector is fed into the Transformer, which applies its several components, described in Section 2.1.3, to compute a contextually enriched representation.

To adapt the basic Transformer architecture for language modeling, one crucial enhancement is the addition of a language model head atop of its last layer (Vaswani et al., 2017). This specialized component is tasked with calculating the conditional

probabilities of subsequent words in a sequence, thereby learning a distribution across the entire vocabulary. Specifically, it focuses on the output of the last hidden state corresponding to the final token in the sequence, N, and utilizes this output to predict the next token $N + 1$. The prediction process involves a linear transformation of the last hidden state's vector, which has a dimension of $1 \times d$, into a logit vector with dimensions $1 \times V$. This linear transformation is facilitated by E^T, the transpose of the initial embedding matrix E with dimension $V \times d$. This design choice allows E^T to map the embeddings from $1 \times d$ back to $1 \times V$. Since E^T essentially shares weights with the initial embedding matrix E, it eliminates the need for learning a separate set of weights for this transformation (Vaswani et al., 2017; Press and Wolf, 2017). The logits, as explained before, are transformed into probabilities using a softmax function. One more benefit of Transformer-based models, compared to statistical n-gram models, is that they allow for much longer sequences to be processed without being subject to the curse of dimensionality problems, especially because of the usage of distributed vector representations and a higher model complexity. Therefore, they are able to compute the probability of an entire sequence, rather than relying solely on the frequencies of fixed-length word sequences. The sequences length, often referred to as context size, can vary significantly depending on the specific model and its configuration, and it can reach up to 2048 (Brown et al., 2020) 4096 (Beltagy et al., 2020) and even 32,000 (Jiang et al., 2024) tokens.

The advent of Transformer-based language models has led to the emergence of sophisticated **Pre-trained Language Models** (PLMs). These models harness the advanced capabilities of the Transformer architecture to generate high-quality, context-sensitive representations, as detailed in Section 2.2.3. Initially, PLMs undergo a pre-training stage with a specific language modeling objective, during which they acquire robust embeddings that capture a wide range of linguistic features (Hewitt and Manning, 2019; Reif et al., 2019; Peters et al., 2018a; Koto et al., 2021; Sorodoc et al., 2020) and other information, including factual knowledge (Petroni et al., 2019; Jiang et al., 2020; Shin et al., 2020). Subsequently, these models can be adapted for a variety of downstream tasks through the process of fine-tuning. This involves augmenting the foundational language model with additional, task-specific layers, allowing the fine-tuned model to leverage the deep, contextualized representations learned during pre-training. This methodology eliminates the need to construct a new model for each task, offering a flexible approach to harnessing the latent representations within the PLM for diverse applications. Section 2.2.3 introduced some of these models, such as the GPT family and BERT and its variants, which, through different language modeling objectives, offer a basis of computational language understanding.

Research showed that scaling both training data and model's parameters increases

the performance of PLMs and enables the outstading abilities to solve tasks on which the model was not explicitly trained on (Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2023). This research avenue started the development of **Large Language Models** (LLMs). The scalability of the Transformer architecture has been crucial in accommodating the ever-increasing model sizes, facilitating models with billions of parameters (Brown et al., 2020; Touvron et al., 2023) capable of capturing nuanced patterns in data. The benefits of scaling enable so-called emergent abilities, i.e., abilities that large models posses that are not observed in smaller models (Wei et al., 2022a). A comprehensive survey by Zhao et al. (2023) categorizes these abilities into three primary groups:

1. **In-context learning**: The language model is shown examples of how to solve a specific task through natural language prompts and it can generate the expected output without relying on fine-tuning or other gradient updates. GPT-3 was among the first models to exhibit this capability (Brown et al., 2020).
2. **Instruction following**: After the pre-training stage, LLMs that undergo fine-tuning with different multi-task datasets, structured using natural language instructions, demonstrate proficiency in tackling new, unseen tasks articulated through instructions (Sanh et al., 2021; Ouyang et al., 2022). This process, known as instruction tuning, gives LLMs the ability to comprehend and execute tasks without the need for explicit examples, thereby enhancing their generalization capacity.
3. **Step-by-step reasoning**: Employing a technique known as chain-of-thought prompting, LLMs can deconstruct a problem into intermediate steps as part of their problem-solving strategy, leading to the derivation of the final answer. This method of systematic reasoning is particularly evident in models trained on programming and logical tasks (Wei et al., 2022b).

Given the substantial resource requirements of LLMs, this research will concentrate on medium-sized PLMs, such as BERT, GPT-2, and BART. The focus of this thesis is to examine the hidden representations within these models, rather than their output capabilities. Therefore, PLMs represent a powerful alternative to conduct the research.

2.3.3. Explainability of Language Models

The use of embeddings instead of count-based text representations, thus more efficient and effective, has led to more opaque models because of their continuous and distributed nature. Research has focused on explaining deep learning models for many years through several approaches (Lee et al., 2022; Roy et al., 2022;

Frye et al., 2020). On the one hand, the decisions made by classification models have become easier to interpret thanks to techniques, such as Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016) or SHapley Additive exPlanations (SHAP) (Lundberg and Lee, 2017), which enable, for instance, the identification of specific words that led to a certain prediction. On the other hand, analyzing models trained through self-supervision with non-linear activation functions, specifically models that encode text in low-dimensional vector spaces, is more complex and necessitates the usage of specialized techniques to unveil specific information encoded in the hidden representations.

Models developed for training global word embeddings, such as word2vec and GloVe, facilitate the application of linear algebraic techniques to understand their properties. Due to their static nature, these embeddings provide a stable foundation for various transformations and manipulations aimed at enhancing their interpretability and efficiency. A significant area of research within this domain focuses on transforming or sparsifying the embedding matrix to derive a more interpretable vector space, potentially reducing its dimensionality without losing significant semantic information (Xing et al., 2015; Rothe and Schütze, 2016; Rothe et al., 2016; Park et al., 2017). Techniques such as matrix factorization, linear transformations, vector space rotations, and dimensionality reduction methods are explored for this purpose. This often involves the training of an auxiliary matrix that transforms the embeddings in a way that certain types of information are represented by a small number of dimensions (Rothe and Schütze, 2016). Another possible method is to apply rotation algorithms to the embedding spaces with the goal of adjusting the word vectors so that each vector and its dimensions exhibit a sparse distribution of values, ideally with a few large values and many values close to zero (Park et al., 2017). This property is visualized in Figure 5. The sparsity makes the transformed vector space more interpretable, as each dimension becomes more meaningful and distinct, facilitating the identification of semantic and syntactic relationships between words.

Techniques aimed at enhancing the interpretability of deep learning models, particularly through the use of linear transformations and other model-based manipulations, serve as the groundwork for the development of **probes**. Probing techniques have emerged to interpret the complex properties encoded within deep language models. These probing models are specifically designed to isolate and evaluate the presence and quality of various linguistic features, such as syntax and semantics, or any other property that may be embedded within the layers of neural networks, using only the representations learned by the deep models, especially those obtained through self-supervised learning like word embeddings or contextual embeddings from Transformer models (Hewitt and Liang, 2019). By assessing how well these auxiliary models can predict predefined properties, it is possible to

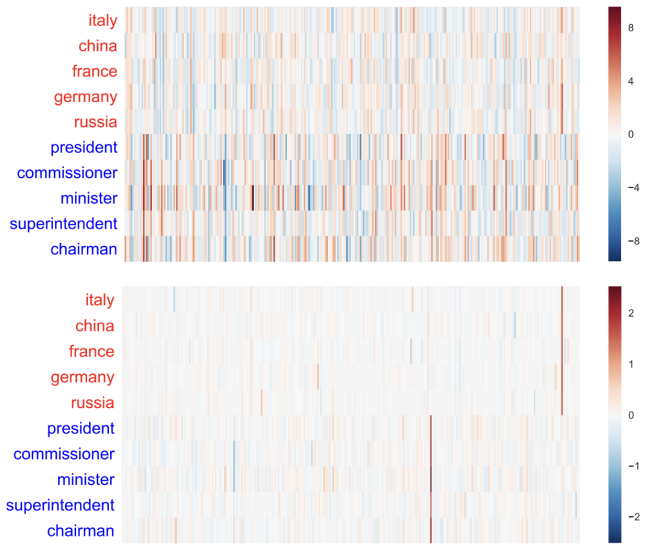

Figure 5.: Rotation of word embeddings to enhance interpretability. (Above) Original skip-gram vectors. (Below) skip-gram vectors after the rotation of their dimensions, clearly showing how the vectors now have an increased notion of country and position (Park et al., 2017, 402)

infer the degree to which such properties are inherently captured by the original language models.

Consider a deep learning model that generates representations $\mathbf{h}$ for input data $\mathbf{x}$, such that $\mathbf{h} = f(\mathbf{x}; \theta)$, where f is the function modeled by the neural network with parameters θ. A probe is then another model G, with parameters ϕ, trained to predict a specific property $\hat{\mathbf{y}}$ from $\mathbf{h}$. The objective of the probe is to learn a mapping $G(\mathbf{h}; \phi \rightarrow \mathbf{y})$ that is as accurate as possible, where $\mathbf{y}$ represents the true label for the property of interest, usually coming from an annotated dataset. The training process $\mathcal{T}$ for the probe involves minimizing a loss function $\mathcal{L}$ that measures the difference between the predicted properties $\hat{\mathbf{y}} = G(\mathbf{h}; \phi)$ and the true properties $\mathbf{y}$ over a dataset D, as depicted by Equation 33.

$$\mathcal{T} = \min_{\phi} \mathcal{L}(\mathbf{y}, G(f(\mathbf{x}; \theta); \phi)), \quad \mathbf{x}, \mathbf{y} \in D \tag{33}$$

During training, the parameters θ of the original model are kept fixed and only the probe's parameters ϕ are adjusted to capture the relationship between $\mathbf{h}$ and $\hat{\mathbf{y}}$. High accuracy suggests that $\mathbf{h}$ contains information relevant to $\hat{\mathbf{y}}$, whereas lower accuracy indicates that the property may not be well-represented or easily decodable from $\mathbf{h}$. When designing probes, it is particularly important to pay attention to the architecture of the model G and to find the correct balance between

simplicity and effectiveness, since otherwise, there could exist the risk that the probe learns the task without guaranteeing that the representations **h** genuinely encode the property of interest (Hewitt and Liang, 2019).

Recall that large enough language models posses abilities that smaller models do not show, as explained in Section 2.3.2. These emergent capabilities allow to take advantage of the model itself to unveil properties through **prompting**. This is particularly useful, for instance, to understand different knowledge categories encoded in the model. If it is able to answer the question with a high enough accuracy, it might be safe to assume that the model indeed possess the property that needs to be measured. The concept is very similar to that of probes, but does not require the design and training of an additional model to unveil the property. For example, given a fact $(subject, relation, object) =$ `[:Vienna :CapitalOf :Austria]`, it is possible to reformulate it as "Vienna is the capital of [MASK]" and if the model predicts the missing word correctly, the assumption is that it possesses factual knowledge (Petroni et al., 2019).

Even tough prompting can be seen as a valuable alternative to probing, because of its less invasive nature, it does not inspect vector spaces directly, but rather the ability of the model to give the correct output for a specific problem. Nevertheless, research investigate also methods to make embeddings more interpretable by enabling direct interaction through natural language (Tennenholtz et al., 2023; Li and Liang, 2021; Wu et al., 2023). Such methods see, for instance, the incorporation of embeddings through adaptive layers, which enable the language model to act as a decoder to engage in discourse about the entities represented by the vectors (Tennenholtz et al., 2023).

2.4. Knowledge Graphs

Mathematically, a knowledge graph G is a set of triples (s, r, o), where s and o are subjects and objects, and r signifies the relation connecting them, encapsulating diverse domains and enabling comprehensive semantic exploration. These triples, often referred to as facts, collectively construct the graph when aggregated. Subjects and objects are referred to as entities and represent the primary objects of the graph, such as people, places, organizations, or events. They are typically depicted as nodes. Relationships, or edges, connect pairs of nodes, representing various types of associations or interactions between entities. Each relationship can be directed and labeled to reflect the nature of the connection accurately.

The term knowledge graph was popularized by Google with the introduction of semantic knowledge technologies in web search. Although there is no universally accepted formal definition of a knowledge graph beyond the mathematical description, Paulheim (2017) suggests that a knowledge graph should meet several criteria to be considered as such:

- **Coverage across different domains**: It should span a variety of topics or fields.
- **Description of real-world entities and their interconnections**: It should accurately reflect complex real-world networks organized in a graph structure.
- **Structured schema**: It should include a schema defining classes of entities and types of relationships.
- **Interrelationships among entities**: It should demonstrate the interconnected nature of entities through explicit links.

A knowledge graph not only serves as a static repository of information, but also as a dynamic framework that can evolve and expand over time. This dynamic nature allows knowledge graphs to adapt to new information and insights, continually refining and expanding their utility across various applications. Moreover, thanks to their inherent structure, knowledge graphs also enhance the discoverability of information through efficient traversal and querying mechanisms that can uncover hidden relationships and insights not readily apparent in unstructured data (Hogan et al., 2021). Following the definition of knowledge graphs, Section 2.4.1 goes over alternative ways of representing knowledge, Section 2.4.2 explains a pivotal concept of this thesis, i.e., knowledge graph embeddings, whereas Section 2.4.3 introduces the role of knowledge graphs in natural language processing.

2.4.1. Knowledge Representations

There exist multiple ways to formally represent knowledge, especially structured knowledge. Knowledge graphs, as explained in Section 2.4, are collection of interconnected nodes and relations that can be traversed using specific query languages, such as SPARQL, and different graph traversal algorithms. Knowledge graphs build upon the foundational concept of **semantic networks**, which are graph structures utilized to represent knowledge within specific domains. Similar to knowledge graphs, semantic networks comprise nodes that denote classes, concepts, or individuals, along with relations that link these nodes and describe their interconnections. Their main focus is on representing conceptual relationships and modeling domain knowledge for reasoning and cognitive processes, but they are not particularly suitable for for modeling complex and large datasets, especially when containing numeric and string values (Grimm, 2009).

This is where schemas, ontologies and standards play an important role to transform simpler structures like semantic networks into complex data structures like knowledge graphs that integrate and interlink vast amounts of structured and unstructured data from various domains to enable advanced data retrieval and reasoning. The Resource Description Framework (RDF) (Cyganiak et al., 2014) enable such complex representations by introducing a standard to formally represent interconnected data on the web. RDF can be seen as a directed graph and uses triples in the form of (s, r, o) to describe data and knowledge. In fact, RDF can potentially model the world knowledge using its schema, with the additional benefit of interchanging data on the web. This is done by representing subjects, relations and objects as Uniform Resource Identifiers (URIs) to ensure that each element in the RDF data model can be unambiguously referenced (Hitzler et al., 2009). Moreover, objects can be either URIs or literals, which are values such as strings, numbers, or dates, providing concrete data associated with resources.

The RDF structure allows for flexible and extensible data modeling because new information can be added without disrupting the existing schema. This property has been crucial for the development of Linked Data, which aims to interconnect data across the web, thus enabling a graph representation of all web, which can be traversed to gather information in a more efficient way. Through RDF, data can be linked and enriched with additional context and relationships, creating a web of interconnected information.

RDF is a rather recent concept of knowledge representation, but this field has been scientifically investigated for many decades. Knowledge has often been represented using logic and rules, which play a crucial role in formalizing and automating reasoning processes. **Rules** are often expressed in the form of if-then statements, enabling systems to infer new information or take actions based on specific conditions. These rules can be simple, such as "`IF` some professor lectures

a course attended by a particular student `THEN` the professor teaches this student" (Grimm, 2009, 113) or complex, involving multiple conditions and nested logic. **Logic-based representations** encompass various forms of formal logic, such as propositional logic and first-order logic, which provide a mathematical foundation for representing and manipulating knowledge. For instance, first-order logic allows the representation of relationships between objects and the properties of those objects using quantifiers, predicates, and logical connectives (Russell and Norvig, 2016). Given a triple `(Professor, lectures, Class)`, the entities can be used to represent more expressive knowledge in logic, such as the following:

$$\forall x, y : (\text{lectures}(x, y) \rightarrow \text{Professor}(x) \wedge \text{Class}(y))$$
$$\forall x : (\text{Professor}(x) \rightarrow \exists y(\text{Class}(y) \wedge \text{lectures}(x, y)))$$

The first statement formalizes that for all entities x and y, if x lectures y, then x must be a Professor and y must be a Class. The second statement means that for every entity x, if x is a Professor, then there must exist some entity y, such that y is a Class and x lectures y, asserting that every Professor must lecture at least one Class to be considered as such.

2.4.2. Knowledge Graph Embeddings

Given the description of knowledge graphs and knowledge representation in Sections 2.4 and 2.4.1, Knowledge Graph Embeddings (KGEs) aim at representing entities and relations in a low-dimensional continuous vector space, while keeping the original semantic meaning. Given a fact represented as a triple $(h, r, t) \in D^2$, being the set of all facts in the KG, the goal is to embed all head and tail entities $h, t \in \mathcal{E}$ and relations $r \in \mathcal{R}$ to a vector $\mathbf{e} \in \mathbb{R}^d$. $\mathcal{E}$ denotes the set of entities, whereas $\mathcal{R}$ is the set of relations. According to Wang et al. (2017), the process of computing KGEs usually involves three steps:

1. Agreeing on a representation of entities and relations.
2. Defining and computing a scoring function.
3. Learning the actual entities and relations embeddings.

The first step is common to many approaches, since it usually consists of representing h and t in a continuous vector space and considering r as operations in the vector space, also represented as vectors or matrices. The the plausibility of a fact (h, r, t) is measured by a scoring function, such that facts observed in the KG will have

[2]The notation (h, r, t) is interchangeable with (s, r, o), which represents subjects, relations, and objects. Since literature on knowledge graph embeddings commonly describes triples as (h, r, t), this notation is used throughout this section.

higher scores than facts not observed in the KG. Learning the embeddings requires an optimization to maximize the total plausibility of observed facts in D. This is usually done through a process called negative sampling, in which correct triples present in D get corrupted by switching the head or tail entities, such that the fact becomes incorrect. The model's task is to score these triples in a way that valid triples get higher scores than invalid triples, which are considered negative samples. The loss function then aims to separate the scores of positive and negative triples by some margin γ. The initialization of the vectors is commonly done using normal or uniform distributions, but can also be performed by first averaging the word vectors of the entities or relations name using pre-trained models and then feeding these vectors into KGE models (Socher et al., 2013). Wang et al. (2017) categorizes techniques to compute KGEs into two main groups: translational distance models and semantic matching models.

Translational Distance Models. One of the most significant translational distance model is **TransE** (Bordes et al., 2013). TransE relies on a simple model architecture to compute embeddings of entities and relations by considering relations as translations operating on the vectors $\mathbf{h}$ and $\mathbf{t}$ of the head and tail entities respectively. Similar to Mikolov et al. (2013), TransE aims to approximate $\mathbf{t}$ by computing $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$, meaning that when the fact holds, $\mathbf{r}$ can be seen as a vector connecting $\mathbf{h}$ and $\mathbf{t}$. Given a triple `(AlfredHitchcok, DirectorOf, Psycho)`, the trained vectors should have the following property: $\mathbf{h}$`(AlfredHitchcok)` + $\mathbf{r}$`(DirectorOf)` $\approx$ $\mathbf{t}$`(Psycho)` (Wang et al., 2017). This property is achieved by maximizing the scoring function $f_r(h, t)$, as depicted in Equation 34.

$$f_r(h, t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{1/2} \tag{34}$$

Equation 34 should yield a large score if the fact holds and a small score otherwise, because the negative distance is computed. More intuitively, $\mathbf{t}$ should be as near as possible to $\mathbf{h} + \mathbf{r}$ for correct facts and further away for incorrect facts. Nevertheless, operating on a single vector space leads to challenges with TransE when handling one-to-many and many-to-one relations.

TransH(Wang et al., 2014) partly solves this issue by introducing translations on relation-specific hyperplanes, as illustrated in Figure 6. This model projects $\mathbf{h}$ and $\mathbf{t}$ onto a hyperplane defined by the normal vector $\mathbf{w}_r$, for each relation $\mathbf{r}$. The projection is computed as follows.

$$\mathbf{h}_\perp = \mathbf{h} - \mathbf{w}_r^T \mathbf{h} \mathbf{w}_r, \quad \mathbf{t}_\perp = \mathbf{t} - \mathbf{w}_r^T \mathbf{t} \mathbf{w}_r \tag{35}$$

where $\mathbf{h}_\perp$ and $\mathbf{t}_\perp$ are the projections of $\mathbf{h}$ and $\mathbf{t}$ onto the hyperplane. The scoring function of TransE is modified in TransH as shown in Equation 36.

$$f_r(h, t) = -\|\mathbf{h}_\perp + \mathbf{r} - \mathbf{t}_\perp\|_2^2 \tag{36}$$

This enables TransH to introduce a sort of contextualized mechanism, allowing different vector representations of an entity depending on the specific relation involved.

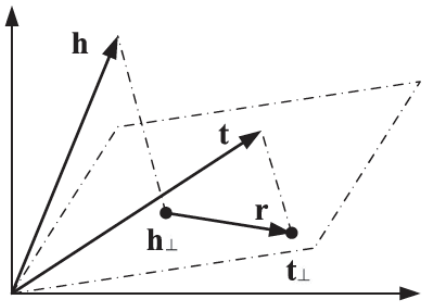

Figure 6.: Illustration of TransH and its hyperplane (Wang et al., 2017, 2726)

Alternatively, **TransR** (Lin et al., 2015) models relations in specific spaces instead of hyperplanes. The model does so by projecting $\mathbf{h}$ and $\mathbf{t}$ onto a space, which is specific to the relation, using a projection matrix $M_r \in \mathbb{R}^{k \times d}$ as shown in Equation 37.

$$\mathbf{h}_\perp = M_r \mathbf{h}, \quad \mathbf{t}_\perp = M_r \mathbf{t} \tag{37}$$

The scoring function remains the same as Equation 36.

Semantic Matching Models. Semantic matching models differ from distance-based models in that they use operations like dot product and cosine similarity to compute scores that consider the latent semantic properties of head and tail entities. **RESCAL** (Nickel et al., 2011) is one of the early examples of semantic matching models, representing entities as vectors and each relation as a matrix M_r, tasked with modeling pairwise interactions between latent features. Each element $w_{i,j}$ of the matrix M_r quantifies the interaction level between the i-th latent factor of the vector $\mathbf{h}$ and the j-th latent factor of the vector $\mathbf{t}$. Therefore, the plausibility of a fact is given by Equation 38.

$$f_r(h,t) = \mathbf{h}^T M_r \mathbf{t} = \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij}^{(r)} \mathbf{h}_i \mathbf{t}_j \tag{38}$$

In Equation 38, d represents the dimensionality of the latent space. However, because RESCAL captures interactions between every pair of components in $\mathbf{h}$ and $\mathbf{t}$, it requires a significant number of parameters, specifically, $\mathcal{O}(d^2)$ parameters per relation.

DistMult (Yang et al., 2014) solves this issue by representing M_r as a diagonal matrix, such that the interactions between $\mathbf{h}$ and $\mathbf{t}$ are only captured along the

same dimension. It embeds h, t and r in the same real dimensional space and uses a dot product based scoring function to determine the plausibility of a given triple. The scoring function is described by Equation 39.

$$f_r(h,t) = \mathbf{h}^T \operatorname{diag}(\mathbf{r})\mathbf{t} = \sum_{i=0}^{d-1} \mathbf{r}_i \mathbf{h}_i \mathbf{t}_i \tag{39}$$

Here, $\operatorname{diag}(\mathbf{r})$ is a diagonal matrix derived from the relation vector $\mathbf{r}$. The goal of this scoring function is to multiply $\mathbf{h}$ with $\operatorname{diag}(\mathbf{r})$, which outputs another vector $\mathbf{t}'$. The dot product is then computed between $\mathbf{t}$ and $\mathbf{t}'$. The resulting score ideally maximizes the alignment between the true tail entity $\mathbf{t}$ and the predicted tail entity $\mathbf{t}' = \mathbf{h}^T \cdot \operatorname{diag}(\mathbf{r})$, such that if the fact is correct, the dot product is as high as possible.

Combining the rich interactions of RESCAL with the efficiency of DistMult results in the Holographic Embeddings (**HolE**) model (Nickel et al., 2016). HolE introduces the circular correlation operator to combine the head and tail entity representations into $\mathbf{h} \star \mathbf{t} \in \mathbb{R}^d$, where the circular correlation $\star : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is defined as shown in Equation 41.

$$[\mathbf{h} \star \mathbf{t}]_i = \sum_{k=0}^{d-1} \mathbf{h}_k \cdot \mathbf{t}_{(i+k) \bmod d} \tag{40}$$

This way, each component $[\mathbf{h} \star \mathbf{t}]_i$ represents a summation over a fixed partition of pairwise interactions. This design enables HolE to group semantically similar interactions into a shared partition, allowing weights to be shared through the relation vector $\mathbf{r}$. The scoring function for HolE is described by Equation 41.

$$f_r(h,t) = \mathbf{r}^T (\mathbf{h} \star \mathbf{t}) = \sum_{i=0}^{d-1} \mathbf{r}_i \sum_{k=0}^{d-1} \mathbf{h}_k \cdot \mathbf{t}_{(i+k) \bmod d} \tag{41}$$

The inner sum represents the circular correlation between the head and tail embeddings, while the outer sum represents the dot product with the relation vector $\mathbf{r}$.

Semantic matching models can also be implemented using neural networks. An example is the use of a Multi-Layer Perceptron (**MLP**) (Dong et al., 2014), which concatenates the entities and relations vectors in the input layer, before passing them to a second layer with a non-linear activation function σ to score the triple, as depicted by Equation 42.

$$f_r(h,t) = \sigma(\mathbf{w}^T \sigma(W[\mathbf{h}, \mathbf{r}, \mathbf{t}])) \tag{42}$$

$W \in \mathbb{R}^{n \times 3d}$ are the the three weight matrices for the first layer and $[\mathbf{h}, \mathbf{r}, \mathbf{t}]$ the concatenated vectors for head, relation and tail. $\mathbf{w} \in \mathbb{R}^d$ is a vector representation of the weights of the second layer.

The Neural Tensor Network (**NTN**) (Socher et al., 2013) model takes a different approach by initially projecting entities into their vector embeddings $\mathbf{h}$ and $\mathbf{t}$ in the first layer. It then combines these embeddings with a tensor $\underline{M}_r \in \mathbb{R}^{d \times d \times k}$, which is specific to the relation. The model proceeds to compute the score through a relation-specific linear layer as follows:

$$f_r(h,t) = \mathbf{r}^T \tanh\left(\mathbf{h}^T \underline{M}_r \mathbf{t} + M_r^1 \mathbf{h} + M_r^2 \mathbf{t} + \mathbf{b}_r\right) \tag{43}$$

where M_r^1 and M_r^2 are matrices in $\mathbb{R}^{k \times d}$ and $\mathbf{b}_r$ is a bias vector in $\mathbb{R}^k$, all of which are specific to each relation. The bilinear tensor product $\mathbf{h}^T \underline{M}_r \mathbf{t}$ yields a vector whose i-th entry is calculated as $\mathbf{h}^T \underline{M}_r^{[:,:,i]} \mathbf{t}$. Figure 7 gives an overview of the architecture of both the NTN and MLP models.

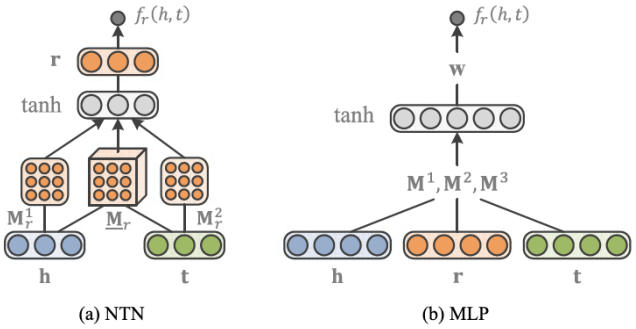

Figure 7.: Overview of the neural network architectures of the (a) NTN (b) MLP models (Wang et al., 2017, 2729)

RDF2Vec. **RDF2Vec** (Ristoski and Paulheim, 2016) is an attempt to apply neural network architectures used for word embeddings, such as word2vec, to RDF data to compute meaningful graph embeddings. The main idea is to treat RDF graphs like documents in which RDF triples, consisting of a subject, relation, and object, are analogous to words in text.

Since word2vec processes sequences of text, it is important to transform a graph into a set of sequences. RDF2Vec first involves walking through the RDF graph to generate sequences of nodes similar to sentences in natural language. These walks can be either random or deterministic. In random walks, the algorithm chooses the next node randomly among the connected nodes, while deterministic walks might use specific patterns or strategies for node selection. For a graph $G = (\mathcal{V}, \mathcal{E})$ the

graph walk algorithm involves traversing all the edges $e \in \mathcal{E}$ for the vertices $v \in \mathcal{V}$ and then visiting all the vertices connected to the previously explored edges.

Once these walks are generated, they are treated as sentences of entities and can be used as input to the word2vec models. Initially, these sequences might be simple, containing only single-hops or direct triples from the knowledge graph, but they can already be used to computed vectors with word embedding models (Nielsen, 2017). However, such basic sequences provide limited contextual information as they only reflect direct relationships. To enrich the context captured by the embeddings, it is beneficial to extend these sequences beyond single triples. Paulheim et al. (2023) describes how to construct two-hop sequences S_{2hop} from the set of all triples $\mathcal{T} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ as follows:

$$S_{2hop} := \{(v_1, r_1, v_2, r_2, v_3) : (v_1, r_1, v_2) \in \mathcal{E} \wedge (v_2, r_2, v_3) \in \mathcal{E}\} \tag{44}$$

By incorporating more hops and therefore combining more triples, the sequence can capture more extensive contextual relationships, potentially leading to embeddings with a richer semantic understanding of the graph's structure. The hops are then inserted into the word2vec model. The skip-gram architecture, as explained in Section 2.2.2, can be used as following:

$$\mathcal{SG} = \sum_{w \in W} \sum_{c \in C(w)} \log P\left(c \mid w\right) \tag{45}$$

where W is the set of nodes in the walk, $C(w)$ is the context of node w and $P(c \mid w)$ is modeled using the softmax function, as shown in Equation 46.

$$P\left(c \mid w\right) = \frac{\exp\left(\mathbf{v}_c^T \mathbf{v}_w\right)}{\sum_{u \in V} \exp\left(v_u^T \mathbf{v}_w\right)} \tag{46}$$

Here, $\mathbf{v}_w$ and $\mathbf{v}_c$ are the vector representations of the node and context, respectively, and V are all the unique nodes in the graph. Conversely, CBOW predicts a target node from the surrounding nodes, using the average of the context node vectors to predict the target node vector.

2.4.3. Knowledge Graphs and Natural Language Processing

Sections 2.4.1 and 2.4.2 presented, amongst other things, the benefits of representing factual information through structured knowledge. The application of this trend to NLP becomes natural, since language is the primary mean for the communication of facts. A way of integrating structured knowledge represented by knowledge graphs and contextual knowledge represented by language models has become an intriguing research avenue, as well as the usage of knowledge graphs to improve

information retrieval, question answering, relation extraction, etc. A survey of Schneider et al. (2022) summarizes the main applications of such technology to NLP.

The most common NLP tasks associated with KGs revolve around their construction. This means extracting structured knowledge from unstructured data, a task commonly referred to as **knowledge graph construction**, which can be further subcategorized into entity extraction, relation extraction, entity linking, entity alignment, etc. Entity extraction is usually employed to extract real-world entities from text, which can be then linked to each other through the relation extraction task. These two tasks already can create triples, the fundamental concept of knowledge graphs. Nevertheless, some of the extracted entities can already exist in the KG and entity linking can therefore be used to connect such entities to the already existing ones.

Reasoning over KGs is another task used to infer new knowledge by performing reasoning and logical operations over these graph structures (Schneider et al., 2022). For instance, one can classify entities or predict links between entities when missing, a task known as **link prediction**. The knowledge graph embedding techniques described in Section 2.4.2 are also methods used to reason over a knowledge graph by transforming it into dense vector representations for better performance on downstream tasks.

According to the results of Schneider et al. (2022), the most popular NLP task where KGs are applied is **Question Answering** (QA). In this context, a KG acts as a semantic network that models real-world entities, their attributes, and the relationships between them, such that when a question is posed, the QA system utilizes the KG to interpret the query and to identify relevant entities and their relationships that satisfy the query's conditions. This can be done using traditional methods involving predefined rules and logic to parse the question and to map the elements to the corresponding entities and relations within the KG (Berant et al., 2013; Bast and Haussmann, 2015). More efficient and modern approaches rely on information retrieval-based systems to efficiently retrieve entities from natural language questions and then scoring them using functions to predict the final answer (Fu et al., 2020). One efficient method is to use representation learning, such as KGEs, embeddings learned through language models, etc. to transform questions and candidate answers into vectors and then performing semantic operations on these vectors to retrieve the most probable answer. The latter strategy is also employed for a further task, which is commonly referred to as **semantic search**, where the goal is to use semantics to search for intents and context and not only for literal matches as in classical information retrieval (Bast et al., 2016). In this context, KGs can be used as a basis to perform search, especially large networks like ConceptNet (Speer et al., 2017).

In recent years, thanks to the continuous development of powerful language models, one of the most prominent research avenue sees the combination of KG technologies with PLMs to enhance their structured knowledge. As explained in Sections 2.1 2.2.3 and 2.3.2, PLMs encode knowledge in their parameters through exposure to unstructured data. **Augmented language models** are a family of PLMs that seek to integrate structured knowledge contained in KGs into the usual architecture of the baseline models. The integration usually happens through mechanisms that inject KG-derived features into the input and/or output layers or through attention mechanisms that specifically focus on using relationships and entity attributes during the model's inference phase (Colon-Hernandez et al., 2021). Examples of such models are Enhanced language RepresentatioN with Informative Entities (ERNIE) (Zhang et al., 2019), K-BERT (Liu et al., 2020), COMET (Bosselut et al., 2019) and KEPLER (Wang et al., 2021).

2.5. Vector Spaces

A vector space, also known as a linear space, is a fundamental concept in linear algebra and mathematics at large, characterized by a collection of vectors which can be added together and multiplied by scalars. Scalars are often taken from the field of real numbers $\mathbb{R}$ but can also come from other number systems such as complex numbers $\mathbb{C}$. The key properties that define a vector space are captured in a set of axioms that vectors must adhere to. According to Liesen (2015), a vector space $V \in \mathbb{R}$ must satisfy some properties for vectors $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$, and for scalars $c, d \in \mathbb{R}$:

- **Additive closure**: $\mathbf{u} + \mathbf{v} \in V$.
- **Additive commutativity**: $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$.
- **Additive associativity**: $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$.
- **Zero**: there exist a neutral element of addition in form of a 0-vector $0_V \in V$, such that $\mathbf{u} + 0_V = \mathbf{u}$.
- **Multiplicative closure**: $c \cdot \mathbf{v} \in V$.
- **Distributivity**: $(c + d) \cdot \mathbf{v} = c \cdot \mathbf{v} + d \cdot \mathbf{v}$.
- **Associativity**: $(c \cdot d) \cdot \mathbf{v} = c \cdot (d \cdot \mathbf{v})$.
- **Unity**: there is a neutral element 1, such that $1 \cdot \mathbf{v} = \mathbf{v}$ for all $\mathbf{v} \in V$.

These axioms ensure that vector addition and scalar multiplication operate in a way that is consistent and predictable. Vector spaces can have any dimension, from the simple two-dimensional plane to infinite-dimensional function spaces. The concept is crucial in various areas of mathematics and applied sciences, as it provides a structured framework for analyzing geometric and algebraic problems. In the field of deep learning, vector spaces play an important role in the structure and function of neural networks. Sections 2.1.2 and 2.2 explained how in deep learning, data are represented as high-dimensional vectors. These representations allow the network to process and learn from complex data by mapping it from one vector space to another through several layers of computations. Especially in neural language models and in modern NLP, vector spaces enable words to be represented as high-dimensional vectors, learning the latent features of language for better representations and performance. The following sections explain the concept of vector space transformations, a crucial aspect of deep learning and mathematics, and introduce common operations used to quantify similarities between sets of vectors.

2.5.1. Transformations

In linear algebra, the primary goal of a **linear transformation** is to map input data from one vector space to another, potentially changing its dimensionality while preserving the operations of vector addition and scalar multiplication (Liesen, 2015). The advantage of linear transformations lies in their ability to maintain the vector space's structural properties. As a result, characteristics observed in the domain of a linear transformation often apply to its range, under specific circumstances. A linear transformation $T : V \in \mathbb{R}^n \rightarrow W \in \mathbb{R}^m$ takes as input n-dimensional vectors and outputs m-dimensional vectors in a transformed vector space W, which nonetheless, adhere to the following properties:

$$T(a\mathbf{h} + b\mathbf{v}) = aT(\mathbf{h}) + bT(\mathbf{v}), \tag{47}$$

where $\mathbf{h}$ and $\mathbf{v}$ are two vectors in the domain of the linear transformation and a and b are scalars.

If the vector spaces V and W are finite-dimensional and there exists a basis for each vector space, the transformation can be represented as a matrix $B \in \mathbb{R}^{n\times m}$ that maps from V to W. Given an input vector $\mathbf{x} \in \mathbb{R}^n$, the transformation applies matrix B to produce an output vector $T(\mathbf{x}) \in \mathbb{R}^m$, adhering to the linear transformation equation $T(\mathbf{x}) = B\mathbf{x}$. Note that the matrix-vector multiplication is only possible if the number of columns of the matrix matches the dimension of the input vector. This ensures that for each element in the output vector $T(\mathbf{x})$, there is a corresponding linear combination of the columns of B weighted by the elements of $\mathbf{x}$.

If $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is a basis for V, every vector $\mathbf{v} \in V$ can be expressed as $\mathbf{v} = c_1\mathbf{v}_1 + \ldots + c_n\mathbf{v}_n$ using coefficients $c_1, \ldots, c_n$ in the space $\mathbb{R}$. Thus, if $T : V \rightarrow W$ is a linear transformation, it follows that:

$$T(\mathbf{v}) = T(c_1\mathbf{v}_1 + \ldots + c_2\mathbf{v}_2) = c_iT(\mathbf{v}_1) + \ldots + c_nT(\mathbf{v}_n), \tag{48}$$

demonstrating that T is fully determined by its action on the basis vectors. Assume $\{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ is a basis for W, then each vector $T(\mathbf{v}_j)$ can be expressed as:

$$T(\mathbf{v}_j) = a_{1j}\mathbf{w}_1 + \ldots + a_{mj}\mathbf{w}_m. \tag{49}$$

The matrix representation of T, denoted by B, captures these coefficients such that the j-th column of B, corresponding to $T(\mathbf{v}_j)$, contains the coefficients $a_{1j}, \ldots, a_{mj}$:

$$B = \begin{pmatrix} \cdots & a_{1j} & \cdots \\ & \vdots & \\ & a_{mj} & \end{pmatrix}, \tag{50}$$

where each column $j = 1, \ldots, m$ aligns with the vector $T(\mathbf{v}_j)$. This matrix B efficiently maps any vector from V to its image in W under the transformation T.

Several variants of vanilla linear transformations exist, such as **orthogonal transformations**, i.e., a specific case of a linear transformation where the transformation matrix is orthogonal. An orthogonal transformation Q has the property that its transpose Q^T is also its inverse: $Q^TQ = QQ^T = I$, where I is the identity matrix. This property ensures that the transformation does not alter the dimensionality of the original vector space while preserving the dot product between vectors, and hence the length and angle between vectors are unchanged by the transformation. The transformation only rotates the space around an axis or multiple axes without altering the magnitude of vectors, as well as flipping or reflecting the space across a plane or other geometric features. Therefore, for a pair of vectors $\mathbf{h}, \mathbf{v} \in V$, the orthogonal transformation $Q : V \to V$ has the property $\langle \mathbf{h}, \mathbf{v} \rangle = \langle Q\mathbf{h}, Q\mathbf{v} \rangle$.

Another common transformation applied in NLP and in the context of language models is **Singular Value Decomposition** (SVD). The key idea is to decompose a matrix $W \in \mathbb{R}^{n \times m}$ into three matrices, as depicted by Equation 51.

$$W = U\Sigma V^T \tag{51}$$

$U \in \mathbb{R}^{N \times l}$ and $V \in \mathbb{R}^{M \times l}$, l being the rank of the original matrix W. Σ is a diagonal matrix with non-negative real numbers, also called singular values, i.e., $\text{diag}(\sigma_1, \ldots, \sigma_l)$ (Hua et al., 2022). For a real-valued matrix W, U and V are orthogonal. Therefore, U represents the left singular vectors, comprised of the eigenvectors of WW^T and V is the transpose of an orthogonal matrix containing the right singular vectors, composed of the eigenvectors W^TW. Given these properties, SVD has the ability to capture the inherent structure and variance within data while facilitating dimensionality reduction. By retaining only a subset of the highest-ranked singular values and corresponding singular vectors, one can effectively approximate the original matrix with reduced dimensionality, thus extracting the most salient features while discarding noise. This is particularly useful in neural networks and in language models with many parameters, since it becomes possible, for instance, to approximate the weight matrix in the linear layers of neural networks with the three corresponding matrices created by SVD:

$$\hat{\mathbf{y}} = XW + \mathbf{b} \approx (XU_r\Sigma_r)V_r^T + \mathbf{b} \tag{52}$$

r being the parameters of the matrices U and Σ, $\mathbf{b}$ the bias terms and $\hat{\mathbf{y}}$ the output vector (Hua et al., 2022; Hu et al., 2021).

Linear transformations play a crucial role in data representation and manipulation across various NLP tasks and model architectures. They are usually employed to convert embeddings from one vector space to another, facilitating the processing of textual data in a way that captures semantic and syntactic relationships. For

instance, when training neural networks, including those used in language models, linear transformations are applied at various layers to project input data into spaces that the model can more effectively learn from, as described by Equation 52 in the case of SVD applied to neural networks. Moreover, as explained in Section 2.3.3, linear transformations are not only used as part of neural network architectures, but also as stand-alone models to unveil properties from hidden representations because of their simplicity and effectiveness in maintaining the structure of vector spaces. For instance, one can train linear transformations with supervised data to transform neural networks' vector spaces in a way that they can encode the property of interest. If the transformation is orthogonal, the vector space maintains the dot product operations between vectors as well as their lengths and it is even more safe to assume that the properties are encoded and only need to be linearly transformed to become explicit. Moreover, numerical operations such as matrix vector products can be efficiently computed in parallel, highlighting the importance of linear functions in modern machine learning and deep learning.

2.5.2. Vector Similarity

In the realm of vector spaces, a fundamental task is the assessment of the relationships between vectors. Vector similarity measures are crucial in fields such as machine learning, deep learning, and NLP, as they provide a quantifiable metric to gauge the similarity between objects, data points, or vectors from a mathematical perspective. These measures are especially important when working with embeddings, which are designed to encapsulate the semantic properties of language. By computing similarity scores between sets of embeddings, it is possible to gain insight into their semantic relatedness.

One of the basic operation to compute vector similarity is **dot product**. Given two vectors $\mathbf{h}$ and $\mathbf{v}$ with dimension n, the dot product is computed by multiplying the vector components and adding the results, as described by Equation 53.

$$\mathbf{h} \cdot \mathbf{v} = \sum_{i=1}^{n} \mathbf{h}_i \mathbf{v}_i = h_1 v_1 + h_2 v_2 + h_3 v_3 + \ldots + h_n v_n \tag{53}$$

The operation returns a single scalar value, which is an indication of both the direction and the magnitude of vectors. Geometrically, it can also be represented as $\mathbf{h} \cdot \mathbf{v} = \|\mathbf{h}\| \|\mathbf{v}\| \cos(\theta)$, where $\|\mathbf{h}\|$ and $\|\mathbf{v}\|$ are the magnitudes of the two vectors and $\cos(\theta)$ is the angle between them. The dot product can be positive for $\cos(\theta) < 90$, negative for $\cos(\theta) > 90$, or zero if $\cos(\theta) = 90$, indicating the degree of alignment between the vectors.

A similar metric is **cosine similarity**, which measures the cosine of the angle $\cos(\theta)$ between two vectors $\mathbf{h}$ and $\mathbf{v}$, providing a similarity score independent of

their magnitude. The dot product between two vectors is normalized by their magnitudes and is defined by Equation 54.

$$\text{similarity}(\mathbf{h}, \mathbf{v}) = \cos(\theta) = \frac{\mathbf{h} \cdot \mathbf{v}}{\|\mathbf{h}\|\|\mathbf{v}\|} = \frac{\sum_{i=1}^{n} \mathbf{h}_i \mathbf{v}_i}{\sqrt{\sum_{i=1}^{n} \mathbf{h}_i^2} \cdot \sqrt{\sum_{i=1}^{n} \mathbf{v}_i^2}} \tag{54}$$

The output is a scalar value ranging from -1 to 1, which is easier to interpret compared to the results of dot product, which can be any real number. A value of -1 indicates that the vectors are pointing in opposite directions, values close to 0 indicate orthogonal vectors, whereas values close to 1 mean that the angle is close to 0 and thus the vectors are very similar. Because of the the normalization by the magnitudes, cosine similarity focuses solely on the orientation of vectors and it is therefore scale-invariant, meaning that it does not change if the magnitude of the vectors is scaled up or down. This is particularly useful when working with text embeddings, where the orientation of the vectors is more important than their length. It is sometimes used between layers of neural networks as an alternative to dot product (Luo et al., 2018).

A third known similarity metric is the **L2 distance**, also called Euclidean distance. It quantifies the straight-line distance between two points, or vectors, in the Euclidean space. It is defined as depicted in Equation 55.

$$\begin{aligned} \|\mathbf{h} - \mathbf{v}\|_2 &= \sqrt{\sum_{i=1}^{n} (\mathbf{h_i} - \mathbf{v_i})^2} \\ &= \sqrt{(h_1 - v_1)^2 + (h_2 - v_2)^2 + (h_3 - v_3)^2 + \ldots + (h_n - v_n)^2} \end{aligned} \tag{55}$$

L2 distance is sensitive to scale, meaning that vectors with large values will have a larger L2 distance than vectors with smaller values.

The choice of vector similarity metrics depends on the specific application and on the nature of the data. Even though cosine similarity is often the preferred approach in natural language processing, it is important to understand its limitations, such as the underestimation of the similarity of very frequent words or tokens observed in the training data (Zhou et al., 2022).

3. Related Work

Analyzing the extent to which knowledge is encoded in language models and in their vector spaces has become a prominent research avenue. Current studies employ supervised probing tasks and prompt-based methods to uncover information from both model outputs and embeddings, but approaches to inspect factual knowledge directly from vector spaces remain largely unexplored.

Hewitt and Manning (2019) propose a structural probe to investigate the presence of syntax in neural hidden representations. Differently to other methods (Kuncoro et al., 2018; Peters et al., 2018b; Tenney et al., 2019b), they employ a linear transformation to determine whether syntax trees are consistently embedded in ELMo's and BERT's vector spaces. A significant finding in their work is the observation that the tree structure is embedded when the squared L2 distance between two word vectors corresponds to the number of edges between those words in the parse tree. To delve deeper into the analysis of syntax, they introduce another linear transformation in which the L2 norm of word vectors corresponds to the depth of the word in the parse tree, enabling to reconstruct edge directions. Results show a surprising extent to which both distance and depth are encoded in the models' vector space after the probe transformations, with BERT-Large exhibiting the best performance. However, open issues that are not addressed by their experiments remain the investigations of the applicability of their methods to different languages and in the context of multilingual models. Additionally, the authors propose validating their techniques to identify the potential presence of alternative graph structures within neural hidden representations. This aligns with the central theme of this thesis, namely, investigating the representation of factual knowledge within the vector spaces of pre-trained language models through knowledge graphs.

Rothe et al. (2016) use a similar strategy as Hewitt and Manning (2019), but design a method to create ultradense word embeddings by learning an orthogonal transformation that compacts task-relevant information into a smaller subspace. The core hypothesis is similar to that of this thesis, since the objective is to find a vector space in which specific properties can be predominantly encoded. Moreover, the transformation of Rothe et al. (2016) reduces the dimensionality of the original vector space with the assumption that the information necessary for a given task is concentrated in a smaller, ultradense subspace of the original embedding space. This thesis also experiments with this dimensionality reduction technique, with the

difference that the core properties are related to factual associations and knowledge graphs, rather than linguistic knowledge. Their model, called Densifier, has been demonstrated to excel in lexicon creation tasks, particularly in annotating words with sentiment, concreteness, and frequency information across multiple languages. The method has shown state-of-the-art performance on tasks such as the SemEval2015 Task 10B for sentiment analysis, achieving these results without any loss of information despite the reduced subspace size.

Building on the work of Rothe et al. (2016), Rothe and Schütze (2016) propose an alternative method to enhance the interpretability and functionality of word embeddings by decomposing the vector space into orthogonal subspaces through linear transformations. Their work focuses on four interpretable dimensions: polarity, concreteness, frequency, and part-of-speech. The decomposition of standard word embedding spaces into orthogonal subspaces allows for increased interpretability, since each subspace captures a specific type of information, such as polarity or part-of-speech, while the remaining subspace contains the residual information. Moreover, decomposing the full space into these ultradense subspaces allows for a range of new embedding operations that extend traditional analogy computations, such as finding the antonym of a word by performing the operation "$-1 \times \text{hate} = \text{love}$".

Michael et al. (2020) investigate the extent to which PLMs, in their case ELMo and BERT, inherently learn to encode meaningful linguistic structures, or ontologies, without explicit supervision. Their study introduces a novel technique called Latent Subclass Learning (LSL), which modifies traditional classifier-based probing to induce latent categorizations of input representations. This method allows for the extraction of emergent structures in an interpretable and quantifiable manner. In their experiments, Michael et al. (2020) apply LSL to various linguistic tasks, including named entity recognition, nonterminal labeling, syntactic dependency labeling, and semantic role labeling. The findings reveal that ELMo and BERT's hidden representations can discover familiar categories like personhood and dates, as well as novel ontological distinctions such as fine-grained semantic roles. These results demonstrate that PLMs encode meaningful linguistic distinctions that align with both traditional categories and novel structures, providing deeper insights into the latent representations of language within these models.

Petroni et al. (2019) conducted an analysis aimed at understanding the level of factual and relational knowledge present in baseline PLMs and introduced to this extent the LAnguage Model Analysis (LAMA) probe to effectively test and measure their performance. They use different knowledge sources composed of a corpus of facts either stored as (s, r, o) triples or as question-answer pairs and transform these data structures into cloze-style prompts. The models are evaluated based on how accurately they can predict the missing word against a classical knowledge

base built with a relation extractor. The hypothesis is that the more accurate a model is, the better it is at capturing factual knowledge. The probe is tested on 6 different models: FairSeq FConv based on gated convolutional networks (Dauphin et al., 2017b), TransformerXL large, ELMo original, ELMo 5.5B, BERT-Base and BERT-Large. Results highlight BERT-Large's superior performance, since relational knowledge can effectively be retrieved as successfully as with the classical knowledge base, especially in open-domain question answering. Challenges persist, however, when handling many-to-many relations, with language models struggling to extract optimal objects.

Petroni et al. (2019) illustrated how it is possible to view language models as knowledge bases, but their methods do not allow for facts to be represented explicitly. Thus, Cohen et al. (2023) propose to obtain an interpretable representation by means of prompting to understand what exactly PLMs encode by extracting a knowledge graph of facts from a given model. Given the probably large size of a complete knowledge graph, the study focuses on several key entities, building the knowledge graph recursively around them. The model used for the experiments is GPT-3 (Brown et al., 2020). The process involves learning a set of relations relevant to the entity and generating the corresponding object. To enhance precision, they deliberately instruct the model to output "Don't know" when unsure about the object fit. These two processes are achieved by using few-shot prompts exemplifying the desired outputs and by employing a paraphrasing procedure to generate possible replacements for both the entity and the relations to improve recall. Experiments on 140 key entities reveal a surprising ability to extract knowledge graphs from language models, reaching a precision of up to 92%. Despite their promising findings, open questions persist about the applicability of their methods to smaller or bidirectional models, as they only experimented with the autoregressive GPT-3, one of the largest PLMs containing 175 billion parameters.

To automate prompt generation and discovering the level of factual information encoded in PLMs, Shin et al. (2020) propose AutoPrompt, a technique for autonomously determining optimal prompts for specific tasks using a gradient-guided search approach (Wallace et al., 2019). Automating prompt constructions involves learning a set of tokens $w \in V$ to be inserted in a prompt template, where V is the vocabulary. Initially set to [MASK], these tokens are updated by maximizing label probabilities y given the token. Once all tokens in the template are computed, the model's output for the [MASK] token with the highest probability p is selected. The study utilizes the LAMA benchmark introduced by Petroni et al. (2019) to retrieve facts about real-world entities. In contrast to the original work, they employ AutoPrompt to automatically create prompts. Their approach involves learning the relations of triples (s, r, o) to effectively retrieve the object from the model. The findings suggest that automatically generated prompts, also called soft

prompts, outperform manually created ones in extracting knowledge from BERT and RoBERTA.

Alternatively, Li et al. (2022) conduct a causal-inspired analysis to not only find optimal prompts for knowledge retrieval, but especially to understand on which types of associations PLMs rely for generating accurate factual predictions. Such words associations are: knowledge-dependent, positionally close and highly co-occurred. By using a Structured Causal Model (SCM), i.e., a custom model relying on PLMs, for the process of generating missing words, they successfully measure the causal effect between the masked words and the remaining words with specific associations between them. Results reveal that BERT, RoBERTA, SpanBERT (Joshi et al., 2020) and ALBERT depend more on the less effective positionally close and highly co-occurred associations, rather than the more effective knowledge-dependent associations, to capture factual knowledge. Importantly, this trend persists even with larger or more efficient models, suggesting that PLMs are less effective in grasping factual knowledge due to their reliance on inadequate associations. However, the study, due to its analytical nature, does not provide suggestions for the enhancement of knowledge-dependent associations, leaving this area to interesting future research directions.

4. Methods

This chapter outlines the methodology used in this research to investigate the degree to which contextual embeddings derived from pre-trained language models are able to encode information relevant to knowledge graphs, especially geometric aspects of knowledge graph embeddings. Section 4.1 begins by describing the dataset used in this study, including the process of collecting knowledge graph entities, computing similarity scores between them, and preparing the dataset for further processing. Next, Section 4.2 is dedicated to explaining how entities are embedded using PLMs, detailing the selection process for appropriate models and the steps taken to extract contextual embeddings from these models. Section 4.3 is dedicated to the design of the linear transformation that translates contextual embeddings into a new vector space where the true similarity scores from the dataset can be effectively approximated by PLMs. This part includes a discussion of the model architecture, the objective function guiding the transformation, and the training and optimization procedures that ensure the model's effectiveness in capturing the desired relationships. Finally, Section 4.4 explains the evaluation methods adopted to assess the quality of the trained models, both quantitatively and qualitatively.

4.1. Dataset

The goal is to create a dataset D in the following format: $D = \{(X_i, y_i), \ldots, (X_n, y_n)\}$, where $X = \{(e_s, e_o), (d_s, d_o)\}$, (e_s, e_o) being the subject and object entities respectively and (d_s, d_o) their Wikipedia descriptions. $y \in \mathbb{R}^1$ is a cosine similarity score between e_s and e_o after being transformed into knowledge graph embeddings.

To create the final dataset used to train the models explained in Section 4.3.1 multiple steps are followed to transform the initial collection of triples into a final, curated, labeled dataset. During this processing pipeline, intermediate datasets with different properties are computed. Sections 4.1.1, 4.1.2 and 4.1.3 will go over these steps, explaining the methodologies adopted to construct the dataset.

4.1.1. Collecting Knowledge Graphs Entities

In order to create a dataset with similarity scores between knowledge graph entities, a collection of triples is needed. Wang et al. (2021) constructed the Wikidata5m dataset, i.e., a collection of 4,579,609 entities, 822 relations and 20,496,514 triples coming from the Wikidata knowledge graph (Vrandečić and Krötzsch, 2014). The dataset comes in the following format: `[:Q7825714 :P31 :Q55488] = [:SUB :REL :OBJ]`, with each triple on a new line of a Tab-separated Value (TSV) file. IDs denoted by "Q" represent head or tail entities, whereas IDs denoted by "P" represent relations.

Due to its large size, a reduced version of it needs to be created for performance purposes. Instead of randomly sampling N triples from the original dataset, an algorithm is implemented to perform proportional sampling. The goal is to reduce the dataset size while maintaining the original distribution of relations as much as possible. The algorithm calculates the proportion of each relation in the original dataset and uses these proportions to determine how many triples to sample for each relation, based on the target size of the reduced dataset. It also ensures that each selected relation has a minimum number of samples to avoid losing rare relations.

Recall that the Wikidata5m dataset stores triples with their Wikidata IDs. IDs are not informative enough to compute neither knowledge graph embeddings, nor contextual embeddings. A textual value for entities and relations is needed. Wikidata provides a free Application Programming Interface (API)[1] to retrieve information for both entities and relations using the `wbgetentities` action. The IDs present in the dataset can therefore be sent to the API using a GET request to collect the necessary information. Since Wikidata is a multilingual knowledge graph, only information in English is retrieved. Given the large size of the dataset, even after the reduction, it would not be efficient to send a single request for each ID in each triple to the API. To solve this issue, optimization techniques can be adopted to decrease processing time and to make the scripts more efficient. Parallel processing is implemented to fetch entity information for each ID concurrently, which significantly speeds up the process, especially when making network requests, and improves overall performance. As a final strategy to improve efficiency and reduce memory usage, the dataset is split into smaller chunks that fit comfortably in memory and are within API rate limits. Each chunk is written to a temporary file incrementally appended to the final output file. This strategy also allows for partial re-runs in case of failures. The final output is written to a JSONL file with the following information as attributes:

- `"sub_id"`: holds the ID of the head entity.

[1] `https://www.wikidata.org/w/api.php`

- `"pred_id"`: holds the ID of the relation.
- `"obj_id"`: holds the ID of the tail entity.
- `"sub_value"`: holds the textual value of the head entity.
- `"pred_value"`: holds the textual value of the relation.
- `"obj_value"`: holds the textual value of the tail entity.

4.1.2. Computing Similarity Scores between Entities

The next step in the pipeline to create the final dataset is to compute a similarity score between the subject and object entities. Training a knowledge graph embedding model, such as TransE, DistMult, or RDF2Vec, with the graph represented by the intermediate dataset created in Section 4.1.1 would have been computationally unfeasible without Graphic Processing Units (GPUs). Fortunately, Nielsen (2017) introduced a web API known as Wembedder[2], which facilitates the querying of embeddings for entities within the Wikidata knowledge graph. These embeddings are derived using a streamlined adaptation of the RDF2Vec algorithm explained in Section 2.4.2, which employs word2vec models, such as skip-gram or CBOW, to generate vectors based on graph walks. This process involves extracting RDF triples, consisting of subjects, relations and objects, from the knowledge graph, in this case Wikidata, and subsequently training the word2vec models using these triples. This approach enables the generation of vectors for all entities and relations within the graph.

The `/api/similarity/<entity1>/<entity2>` endpoint of Wembedder is particularly useful for obtaining a cosine similarity score between the embeddings of any two entities in the knowledge graph. To achieve this, requests are submitted to the API for all pairs of entity IDs in the dataset, employing the same optimization techniques outlined in Section 4.1.1. The retrieved cosine similarity scores are then appended to the JSONL dataset under the attribute `sim_score`. The similarity scores represent the relational proximity, as well as the semantic and factual similarity between knowledge graph entities. They are the true scores Y the models described in Section 4.3 need to approximate.

4.1.3. Preparing the Dataset

After computing a similarity score between each entity pair in the dataset, the complete dataset is compiled into a Comma-Separated Value (CSV) file because of the structural simplicity offered by this format. In addition to the simple triples, Wang et al. (2021) created an additional dataset with Wikipedia descriptions for

[2] `https://wembedder.toolforge.org`

each entity and relation in the Wikidata knowledge graph. Such descriptions are very beneficial for the scope of this research, since they could significantly enhance the contextual performance of the embeddings extracted from PLMs. Nevertheless, the descriptions are rather long and may contain redundant or irrelevant information. Thus, all the descriptions are tokenized using the sentence tokenizer from the Natural Language Toolkit (NLTK) library (Bird and Loper, 2004) and only the first two sentences for each entry are kept, since the most important factual information is usually contained in the first sentences of Wikipedia articles. Recall that tokenization is an unsolved problem in NLP and, therefore, it could be possible that some descriptions may contain less or more than two sentences. The tokenized descriptions for the collected entities are compiled into the same CSV file.

The final file includes pairs of subjects e_s and objects e_o, along with their corresponding Wikipedia descriptions d_s, d_o. The dataset also contains the similarity score y between e_s and e_o. The data format is as follows:

$$\begin{pmatrix} e_{s,i} & , & d_{s,i} & , & e_{o,i} & , & d_{o,i} \\ \vdots & & \vdots & & \vdots & & \vdots \\ e_{s,i} & , & d_{s,i} & , & e_{o,i} & , & d_{o,i} \end{pmatrix}, \begin{pmatrix} y_i \\ \vdots \\ y_n \end{pmatrix}$$

where $e_{s,i}$ and $e_{o,i}$ represent the subject and object entities for the i-th data point and $d_{s,i}$ and $d_{o,i}$ the Wikipedia descriptions for subjects and objects for the i-th data point.

To ensure the robustness of the model trained on this data, the dataset is divided into a training, validation, and test set with the 80 to 10 and 10 ratio. The split is performed using Scikit-Learn's (Pedregosa et al., 2011) `train_test_split` method. The training set is used to fit the model, while the validation set is used during training to monitor the performance of the model and to avoid overfitting on the training data. Additionally, the validation set is used to conduct multiple iterations to tune the model's hyperparameters. Lastly, the test set is reserved for evaluating the trained model's generalization capability on unseen data, providing an assessment of its predictive accuracy.

4.1.4. Statistics

Differently to common procedures involved in the creation of labeled datasets, which usually require human annotations, the dataset constructed for this research sees the mapping of subjects and objects embeddings $\mathbf{x}_s, \mathbf{x}_o \in X$ to a similarity score y automatically, drastically reducing the effort of manually annotating data. This is done by using the methods described in Sections 4.1.1 and 4.1.2, which involve sending requests to the Wikidata API for the collection of the entities and

relations' names and to the Wembedder API for the computation of the cosine similarity between knowledge graph embeddings.

Table 1.: Sampled subset of the Wikidata5m dataset

Type	**Training**	**Validation**	**Test**	**Total**
Triples	33,937	4,242	4,243	42,422
Entities	45,343	6,883	6,874	55,344
Relations	190	146	156	191

Through the proportional sampling algorithm described in Section 4.1.1, the original Wikidata5m dataset is decreased to a manageable size, eliminating the need for GPUs during training, yet retaining a sufficient number of samples for hypothesis testing. Moreover, this approach aims to reduce the dataset size while preserving the original distribution of relations as closely as possible. Table 1 provides the final numbers of the Wikidata5m dataset. The reduction algorithm successfully decreased the dataset to a manageable size, eliminating the need for GPUs during training, yet retained a sufficient number of samples for hypothesis testing.

Figure 8 illustrates the distribution of relations in both datasets, analyzed through a Cumulative Distribution Function (CDF) and a Kernel Density Estimation (KDE)

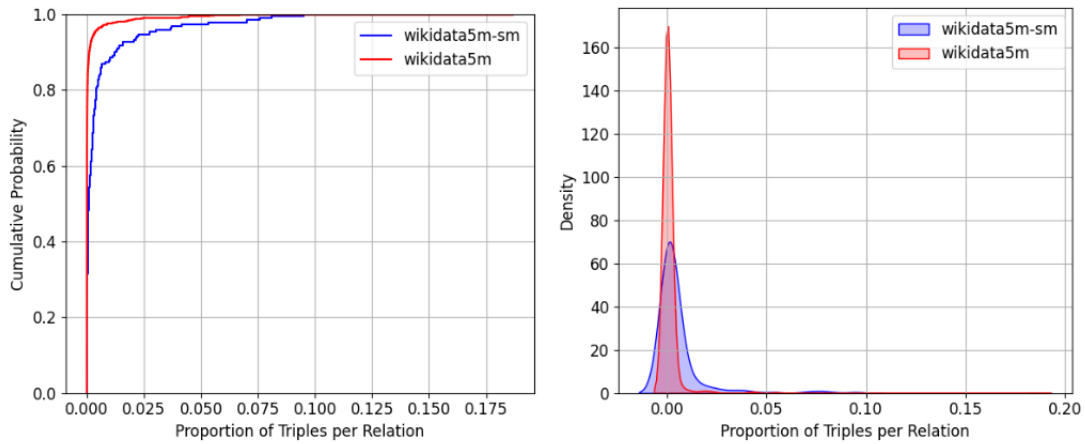

Figure 8.: Relation distribution of the original *wikidata5m* dataset and the reduced *wikidata5m-sm* dataset using CDF (left) and KDE (right)

4.2. Embedding Entities with PLMs

This section examines the process of selecting a family of pre-trained language models and subsequently extracting contextual embeddings out of them, which are then utilized as inputs to the model described in Section 4.3.

4.2.1. Selecting Pre-trained Language Models

The objective of this thesis is to develop a methodology for uncovering knowledge graphs and factual knowledge from Pre-trained Language Models (PLMs). To achieve this, it is essential to access open-source models along with their weights and hidden states. The machine learning platform Hugging Face (Wolf et al., 2020) provides a comprehensive repository of trained models, including notable models such as BERT and GPT-2, which are readily accessible and versatile for various applications. For this thesis, these models are primarily utilized to extract contextual embeddings.

BERT has been extensively studied in recent research, which has investigated the latent properties of its hidden representations (Reif et al., 2019; Hewitt and Manning, 2019; Koto et al., 2021). BERT is chosen for its robust contextual bidirectional representations and the extensive body of comparative research available. As BERT is an encoder-only Transformer model, the hypothesis is tested on different architectures as well. GPT-2, the only open-source model from OpenAI's GPT family available on Hugging Face, offers interesting generative capabilities and it represents the choice for a decoder-only model in this study. Nevertheless, because of its auto-regressive architecture, the hidden states are subject to masking to avoid accessing future tokens and can therefore lack rich contextual knowledge. Additionally, BART, an encoder-decoder model, is included to complete the examination of the Transformer architecture spectrum. BART combines a BERT-like encoder with a GPT-like decoder, making it particularly suitable for tasks, such as machine translation and summarization. Notably, its final hidden state is derived from the last layer of the decoder, providing a rich source of contextual information for analysis. Only the base versions of the chosen pre-trained language models are used, mainly for two reasons:

1. **Computational resources**: The base versions of BERT, GPT-2 and BART offer powerful, yet slightly smaller models compared to the large versions. Since the experiments are not run with GPUs, the reduced number of parameters allows for faster training on CPU.
2. **Embedding size**: Since all three base versions of the chosen PLMs have the same hidden size, i.e., 768, it becomes possible to have a direct comparison of performance, without incurring in the assumption that a higher embedding

Table 2.: Technical details of the chosen PLMs, including number of parameters, hidden dimension (H-Dim), number of encoder layers (E Layers) and decoder layers (D Layers)

Model	Params	H-Dim	E Layers	D Layers
BERT-base	110M	768	12	/
GPT2-small	124M	768	/	12
BART-base	140M	768	6	6

dimension leads to better results.

Table 2 offers an overview of the technical details of each model. For the BERT-base model, the version of the uncased pretraining was chosen.

4.2.2. Extracting Contextual Embeddings

Once the pre-trained language models have been selected, it is necessary to use them to extract meaningful contextual embeddings for the specific entities in the dataset. Challenges occur when trying to extract a single contextual embedding for multi-word entities. To this end, different methodologies can be adopted, such as averaging and pooling the embeddings of all tokens in the sequence. This is achieved by calculating the mean of the last hidden state across the sequence dimension, which aggregates the information contained across all token embeddings into a single vector representation. Given an input sequence $S = (w_1, w_2, \ldots, w_n)$, which could represent an entity in the dataset, where each w_i represents a token in the sequence and n is the sequence length, a PLM transforms each token w_i into a vector representation $\mathbf{h}_i$. Each vector $\mathbf{h}_i$ is of dimension d, so $\mathbf{h}_i \in \mathbb{R}^d$. The mean pooled vector representation $\mathbf{h}_{\text{mean}}$ of the sequence S is computed as the average of all vectors $\mathbf{h}_i$ in the sequence:

$$\mathbf{h}_{\text{mean}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{h}_i, \tag{56}$$

where n is the total number of token vectors in the sequence. In essence, all vector representations $\mathbf{h}_i$ of the words in the sequence are summed up and then divided by n to obtain a single representation.

An alternative method to extract single vector representations from sequences of text is by using the so-called CLS token. This is a special classification token present in numerous Transformer-based language models, especially encoder-only models like BERT, RoBERTa, and ALBERT. This token is added to the beginning of each input sequence, serving as a special marker for the model and is commonly

used for sentence classification or sentiment analysis tasks. During training, the representation of the CLS token is optimized to aggregate the entire semantic content of the input sequence. Therefore, the final state of the CLS token, after training, encapsulates a high-level, context-aware representation of the entire input. This is particularly useful for extracting single vector representations of multi-word entities and phrases, as the model's attention mechanism ensures that the CLS token's representation is informed by the interactions and relationships between all parts of the input text. The CLS token can be easily accessed in the last hidden state of the PLM by extracting the very first position of the sequence, resulting in an embedding with the same dimensionality of the model's hidden dimension.

For BERT-base, experiments are conducted with both techniques, i.e., averaged and pooled embeddings and using the CLS token, whereas for GPT-2 and BART, the contextual embeddings are extracted using only the average and pooling method because of the absence of the CLS token. By doing this, a direct comparison becomes possible, potentially finding the best performing methodology.

Given that the dataset outlined in Section 4.1 includes entities that are composed of single or multiple words, there is a concern that the contextual information of single entities input in the PLM might result in a diminished capacity to leverage the contextual richness inherent in the Transformer's architecture. This could lead to the overlooking of contextual cues within sequences. Fortunately, as outlined in Section 4.1.3, Wang et al. (2021) incorporated Wikipedia descriptions for each entity in their Wikidata5m dataset to support entity disambiguation and to enhance contextual understanding. Thus, two distinct methodologies are evaluated:

1. **Entities only**: This method extracts contextual embeddings from PLMs based solely on the entities themselves, without additional context. It relies on the intrinsic contextual information encapsulated within the entity's text.
2. **Description-enhanced embeddings**: In this approach, the entity names are combined with the initial two sentences of their Wikipedia descriptions as follows: `entity + '[SEP]' + description`. This combination aims to enrich the context surrounding the entities, thereby potentially generating more detailed and contextually rich embeddings. The reason for keeping only the first two sentences of each Wikipedia description is twofold. First, the most important factual information is usually in the first sentences, thus avoiding to insert non-relevant context. Second, limiting the number of sentences helps to stay within the token limits set by the PLM. By integrating the Wikipedia descriptions, the goal is to provide the PLMs with a broader context that includes both the entity and its relevant factual background.

Normalization. L2 normalization is used to normalize the extracted contextual embeddings, ensuring that each embedding vector has a unit length in the vector

space. This process involves dividing each vector by its own L2 norm. For an embedding vector $\mathbf{h}$, its L2 norm is $\|\mathbf{h}\|_2$. The normalized vector $\mathbf{h}'$ is then obtained by $\mathbf{h}' = \frac{\mathbf{h}}{\|\mathbf{h}\|_2}$. This ensure that $\mathbf{h}'$ has a magnitude of 1, lying on the unit sphere in the embedding space. Combining the operations of averaging and normalizing the contextual embeddings, the complete process of extracting embeddings $\mathbf{h}' \in \mathcal{X}$ for the set of knowledge graph entities $\mathcal{X}$ in the dataset can be described by Equation 57.

$$\mathbf{h}' = \frac{\mathbf{h}_{\text{mean}}}{\|\mathbf{h}_{\text{mean}}\|_2} = \frac{\frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i}{\left\|\frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i\right\|_2} \tag{57}$$

Normalizing embeddings in this way is crucial for several reasons. Firstly, it allows for a consistent scale among all embeddings, making it possible to compare them directly based on their directionality. This is particularly important when using cosine similarity as a measure of similarity between vectors, as it ensures that the similarity score reflects the angle between vectors. Secondly, normalized embeddings can help mitigate issues related to varying scales in the data, which can affect the performance of machine learning models, especially those relying on distance measures. Lastly, by ensuring all embeddings have the same magnitude, normalization can contribute to the stability and convergence speed of the algorithms that use these embeddings.

Saving the embeddings. To enhance the efficiency and speed of the training process, the contextual embeddings of the training, validation and test set are precomputed and saved on disk using PyTorch, specifically through the `torch.save` method. This method serializes the tensor objects containing the embeddings and facilitates their storage in a compressed format that can be rapidly accessed. When the training or evaluation process is initiated, these serialized objects can be loaded into memory using the `torch.load` method, significantly reducing the overhead associated with recalculating the embeddings from scratch for each training iteration. This approach also allows for the exploration of various model architectures and hyperparameters without the repetitive computation of the embeddings.

4.3. Transformations

This section is dedicated to formally introducing the models designed to answer the research question and to unveil factual knowledge as represented in knowledge graphs from PLMs' vector spaces. It also describes the objective function as well as the training and optimization algorithms.

The methods draw inspiration from Hewitt and Manning (2019) and Rothe and Schütze (2016), since they involve the implementation of a factual probe in form of linear transformations to unveil specific properties from PLMs' vector spaces.

4.3.1. Transformation Models

Figure 9 shows a high level overview of the methodology designed to unveil factual knowledge from PLMs. The contextual embeddings of the subjects and objects of knowledge graph triples are obtained from the PLM's last layer. The obtained contextual embeddings $\mathbf{h}_s$ and $\mathbf{h}_o$ undergo a transformation by an additional model, which outputs transformed embeddings $\mathbf{t}_s$ and $\mathbf{t}_o$. A similarity score $\hat{y}$ between the transformed vectors is then computed and compared to the true similarity scores y present in the dataset.

The models designed to unveil factual knowledge from PLMs do not rely on neural networks, since employing deep and non-linear models would go against the whole idea of designing simple probes and any non-linear model might inadvertently learn the task without guaranteeing that the representations genuinely encode the phenomenon (Hewitt and Liang, 2019). Thus, the chosen model is a simple linear transformation tasked with transforming the original vector space into a new, learned space in which the desired properties are successfully encoded.

Linear Transformation. Given the definition of linear transformations in Section 2.5.1, the model is designed as a linear transformation $T : \mathbb{R}^n \to \mathbb{R}^m$ that takes as input n-dimensional contextual embeddings and outputs embeddings with dimension m in a transformed vector space. The model itself consists of the matrix B, which is multiplied with the input vector $\mathbf{h}$ to output a transformed vector $\mathbf{t}$, therefore $\mathbf{t} = B\mathbf{h}$. Each element t_i of the output vector $\mathbf{t}$ is a linear combination of the columns of B weighted by the elements of $\mathbf{h}$. The model can be thought of as a linear layer in a neural network without non-linear activation functions and bias terms. The parameters of the matrix B are initialized using a uniform distribution $U\left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right)$, n being the size of the input features, i.e., 768.

Since the objective is to probe whether factual knowledge as represented in knowledge graphs is encoded in PLMs' vector spaces, it is essential to ensure the integrity of the vector geometry by maintaining the initial dimensionality. In other

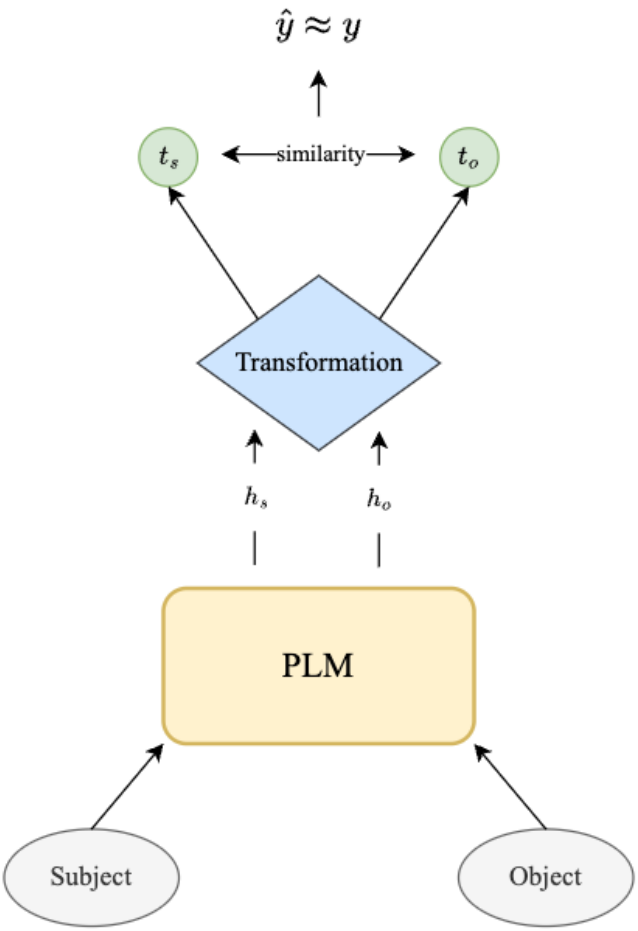

Figure 9.: Architectural overview of the model.

words, no additional information should be added. The underlying assumption is that such knowledge is represented in the original vector space, which is evaluated by a linear transformation of the learned vector space. As explained in Section 2.2.3, the training of contextual embeddings specifically aims at capturing contextual relationships from textual sequences. While properties of knowledge graphs may be incorporated in language, it is possible to assume that the relevant information could be encapsulated in fewer dimensions and not in the whole 768 dimensions of BERT-base, GPT2-small and BART-base. To this end, two experimental designs of the linear transformation model are proposed:

1. **Unaltered dimensionality**: The first transformation matrix is designed as $B \in \mathbb{R}^{n \times n}$, which maintains the original dimensionality of the embeddings, under the hypothesis that the information relevant to the knowledge graphs is distributed across all dimensions. Preserving the embedding dimensionality

is crucial for downstream tasks that may rely on the full-dimensional space for effective computation.

2. **Dimensionality reduction**: The second transformation is designed as $B \in \mathbb{R}^{n \times m}$, where $m < n$. Therefore, the transformed vector $\mathbf{t}$ is in $\mathbb{R}^m$. This approach aims to reduce the dimensionality of the vector space to find a subspace where the properties of the knowledge graphs are predominantly encoded. The objective is to investigate whether a reduced number of dimensions is capable to encode the same amount of factual associations as the larger counterpart. Therefore, given a collection of vectors $\mathcal{V}$, B would be chosen as:

$$\max_{B \in \mathbb{R}^{n \times m}} \mathcal{I}(B\mathcal{V}), \quad \text{such that} \quad m < n$$

where $\mathcal{I}(B\mathcal{V})$ maximize the reduction of m such that the factual associations in the knowledge graph remain to be encoded.

By experimenting with different ranks, it becomes possible to compare the performance of the different models and to potentially find the matrix B that yields the best results while aiming for the smallest possible matrix. This experiment is also motivated by the findings of Hewitt and Manning (2019), who reported that increasing m beyond 64 or 128 dimensions leads to unchanged results in the syntactic parsing accuracy of the PLM.

4.3.2. Objective Function

The goal is to learn the best transformation matrix $B \in \mathbb{R}^{n \times m}$ under which the similarity scores between knowledge graph embeddings depicted in the training dataset can be approximated. The matrix B has a dimensionality of $\mathbb{R}^n \times \mathbb{R}^m$, where n is the size of the input embeddings and m is the size of the output space. If the transformation does not alter the dimensionality of the output embeddings, B would have a dimensionality of 768×768, resulting in a square matrix with 589,824 trainable parameters. The parameters of the transformation are trained by computing a similarity score between transformed embeddings and calculating the difference between predicted scores and true scores coming from the annotated dataset using a loss function. Following the work of Hewitt and Manning (2019), it is possible to define a similarity function $s(e_s, e_o)$, which is the cosine similarity between subject and object entity in the dataset after being transformed into knowledge graph embeddings. The hypothesis is that factual knowledge is embedded in the language model if the linear transformation is able to reproduce the similarity scores of the knowledge graph embeddings using only the contextual embeddings $\mathbf{h}_s$ and $\mathbf{h}_o$, which undergo a transformation by B to become $\mathbf{t}_s$ and $\mathbf{t}_o$. Thus, the function that is approximated is $s_B(\mathbf{t}_s, \mathbf{t}_o) \approx s(e_s, e_o)$. The similarity

function $s_B(\mathbf{t}_s, \mathbf{t}_o)$ is defined as cosine similarity. For an unchanged dimensionality of the output space, this can be achieved by minimizing the loss function $\mathcal{L}$ using a regression objective and by training the parameters of the matrix B using gradient descent in a supervised setting, as depicted by Equation 58.

$$\mathcal{L} = \min_{B} \frac{1}{N} \sum_{\ell=1}^{N} |s_T(e_s^\ell, e_o^\ell) - s_B(\mathbf{t}_s^\ell, \mathbf{t}_o^\ell)|^2 \tag{58}$$

Here s_T are the true similarity scores between knowledge graph embeddings of subject e_s and object e_o from the annotated dataset. Then, s_B represents the predicted similarity scores between the transformed subject entity vector $\mathbf{t}_s$ and transformed object entity vector $\mathbf{t}_o$. ℓ indexes the different subject-object pairs in the dataset and N is the total number of subject-object pairs. However, when reducing the dimensionality of B, the risk is that the matrix overfits on the training data without being able to generalize to unseen data. Thus, a regularization $\|B\|_F^2$ based on the Frobenius norm is added to the loss function to penalize the complexity of the transformation matrix. If during training a particular weight in the matrix tends to large values, the squared term in the Frobenius norm becomes significant, increasing the overall loss. Consequently, the optimization process actively works to reduce such weights to manage and minimize the total loss. A set of possible values for m is defined and described in Table 3 alongside the hyperparameter settings, whereas the objective function is updated as depicted by Equation 59.

$$\mathcal{L} = \min_{B,m} \frac{1}{N} \sum_{\ell=1}^{N} |s_{T^\ell}(e_s^\ell, e_o^\ell) - s_B(\mathbf{t}_s^\ell, \mathbf{t}_o^\ell)|^2 + \lambda \|B\|_F^2 \tag{59}$$

λ is a regularization parameter that controls the trade-off between fitting the data and minimizing the matrix complexity based on the Frobenius norm $\|B\|_F^2$ of the matrix B.

4.3.3. Training and Optimization

All models are implemented and optimized using the deep learning library PyTorch (Paszke et al., 2019). The models are trained for 100 epochs using an initial learning rate between 1e-3 and 5e-3, depending on the PLM used. In order to increase the probability to reach the minimum of the function, learning rate decay techniques are applied. Such techniques enable to start training the model with a large learning rate and then to slowly reduce it until the minimum is reached. To this end, a method called reduce learning rate on plateau is used. This scheduler reduces the learning rate when a metric stops improving. A metrics quantity is read by the scheduler and if no improvement is seen for a certain number of epochs, the learning

Table 3.: Training hyperparameters with corresponding values

Hyperparameter	**Value**
Nr. of epochs	100
Weight decay	5e-3
Learning rate	1e-3 $\leq x \leq$ 5e-3
Learning rate scheduler factor	1e-1
Learning rate patience	5
Early stopping patience	10
Dimensionality of m	$\{32, 64, 128, 256, 512\} \cup \{768\}$
λ	1e-4

rate is reduced by some factor. The patience parameter, i.e., the number of epochs to wait until the rate is reduced, is initially set to 5 and and the reduction factor is set to 0.1. The AdamW optimizer (Loshchilov and Hutter, 2017) is used to effectively minimize the loss function and adjust the parameters at each backward pass of gradient descent. An additional regularization term ψ provided by the AdamW optimizer in form of a weight decay is added to the objective function described by Equation 58 to prevent overfitting by penalizing large weights. The updated loss function is described by Equation 60.

$$\mathcal{L} = \min_{B,m} \frac{1}{N} \sum_{\ell=1}^{N} |s_{T^\ell}(e_s^\ell, e_o^\ell) - s_B(\mathbf{t}_s^\ell, \mathbf{t}_o^\ell)|^2 + \lambda \|B\|_F^2 + \psi(\theta) \tag{60}$$

θ are the parameters of the transformation matrix B. Before the training process is started, ψ is set to 5e-3. Additionally, an early stopping mechanism is implemented to avoid overfitting. This involves monitoring the loss of both the training set and validation set and halting the training process when the validation performance no longer improves for a specified number of training epochs. By doing so, it is possible to balance the trade-off between underfitting and overfitting by stopping the training at the point where the model is generalized enough to perform well on both the training and validation sets. The early stopping patience hyperparameter is set to 10, meaning that the model will wait for 10 epochs of no improvement before stopping the training process. Table 3 offers an overview of the different hyperparameters used during training and Algorithm 1 provides a description of the training process.

Algorithm 1 Training Algorithm

Given training data $D_t = \{X, Y\}$
Given validation data $D_v = \{X, Y\}$
Initialize transformation matrix B with $U\left(\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right)$
Define $\mathcal{L}$ as in Equation 60
Define val_loss as in Equation 60
epoch $\leftarrow 0$
best_loss $\leftarrow \infty$
patience_counter $\leftarrow 0$
while epoch < 100 and patience_counter < 10 **do**
    for each pair ℓ in D_t **do**
        Compute $\mathbf{t}_s^\ell$ and $\mathbf{t}_o^\ell$ as $B(\mathbf{h}_s^\ell, \mathbf{h}_o^\ell)$
        Calculate similarity $s_B(\mathbf{t}_s^\ell, \mathbf{t}_o^\ell) = \frac{\mathbf{t}_s \cdot \mathbf{t}_o}{\|\mathbf{t}_s\| \|\mathbf{t}_o\|}$
        Compute loss $\mathcal{L}$ for the pair ℓ given $s_B(\mathbf{t}_s^\ell, \mathbf{t}_o^\ell)$ and true score y
        Perform backpropagation to update B
    end for
    Evaluate on validation data D_v using val_loss
    if learning_rate_patience $\leftarrow 5$ **then**
        Apply learning rate scheduler
    end if
    if val_loss $<$ best_loss **then**
        best_loss $\leftarrow$ val_loss
        patience_counter $\leftarrow 0$
    else
        patience_counter $\leftarrow$ patience_counter $+ 1$
    end if
    epoch $\leftarrow$ epoch $+ 1$
end while

4.4. Evaluation Methods

In order to asses the performance of the probing models, a set of evaluation methods is needed to compute both quantitative metrics and perform qualitative analysis. Quantitative metrics are useful to obtain a score representing the numeric performance of the trained models, which are usually easy to interpret and can be computed automatically, thus reducing human intervention. Nevertheless, they do not offer a complete overview of the performance, which is why qualitative methods are needed to analyze the behavior of the trained models and to identify strengths and weaknesses. In Section 4.4.1 a set of metrics to perform a quantitative analysis is described and Section 4.4.2 details the visualizations implemented to further inspect both the transformation matrices and the vector spaces. Section 4.4.3 concludes the evaluation by providing a qualitative analysis of the worst performing data points.

4.4.1. Quantitative Evaluation

Recall that the output of the models are transformed vectors, which are then passed trough a cosine similarity function to obtain a numerical value indicating the relational proximity between two entities in the knowledge graph. To effectively quantify the accuracy of the predictions, established metrics to evaluate regression models are used. To this end, the Pearson correlation coefficient, the Spearman's rank correlation coefficient, the mean-squared error and the root mean-squared error are chosen.

Pearson Correlation Coefficient. The Pearson Correlation Coefficient (PCC), denoted as r, measures the linear correlation between two sets of data. In the context of this thesis, these two sets of data are the predicted similarity scores $\hat{Y}$ coming from the transformed contextual embeddings and the true similarity scores Y coming from the knowledge graph embeddings depicted in the labeled dataset. The PCC is computed for pairs of similarity scores $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, as depicted by Equation 61.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}. \tag{61}$$

x_i and y_i are the predicted and true scores, respectively, whereas $\overline{x}$ and $\overline{y}$ are the means of these scores. In simple words, PCC is calculated by dividing the covariance of two variables x and y by the product of their standard deviations. The values produced by Equation 61 can vary between -1 and 1. A value of 1 signifies a perfect positive linear correlation, while -1 denotes a perfect negative linear correlation. A

result of 0 suggests that there is no correlation between the predicted and actual scores. This metric is also used during training and validation to monitor the performance of the model in addition to the loss value. The `pearsonr` function of the open-source scientific computing library SciPy (Virtanen et al., 2020) is used for an efficient computation of the PCC.

Spearman's Rank Correlation Coefficient. Additionally to PCC, Spearman's Rank Correlation Coefficient (SRCC) is added to the family of evaluation metrics. SRCC, unlike PCC, which assesses linear relationships based on raw data, determines the correlation by assessing the rankings of the two variables x and y. This method is used to assess how effectively the relationship between two variables, such as the predicted and true scores in the case of this work, can be represented by a monotonic function. Given the two scores, the SRCC is computed as described by Equation 62.

$$p = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{62}$$

$d_i = R(X_i) - R(Y_i)$ is the discrepancy between the ranks of corresponding values of the two variables, and n is the number of observations. The output ranges once again from -1 to 1, with 1 indicating a perfect positive association of rankings, -1 a perfect negative association of rankings, and 0 no association at all. As expected, the goal is to reach a value as high as possible.

In the context of this thesis, especially because the computation of cosine similarity is not a linear operation in the algebraic sense, SRCC is a useful metric because it does not assume that the relationship between the predicted and true similarity scores is linear or that the data is normally distributed. It simply assesses how well the rankings of the predicted similarity scores from PLM embeddings match the rankings of the actual similarity scores from the knowledge graph. Again, SciPy is used for the computation of the SRCC, specifically, the `spearmanr` function.

Mean-Squared Error. To directly assess the error values in the similarity predictions themselves, the Mean-Squared Error (MSE) metric is used, which is also the chosen loss function. MSE has already been introduced with Equation 8 in Section 2.1.2 and it is commonly used to assess the performance of regression models. It gives an indication of the overall error and not only an assessment of the directionality or correlation of the predictions, as Pearson and Spearman correlation coefficients do. Lower values indicate better performance, as they reflect smaller average squared errors between predicted and actual values. A low MSE suggests that the model is capable of accurately predicting similarity scores, with minimal

error, thus aligning well with the true data. Scikit-learn's `mean_squared_error` function is used to compute the metric.

Root Mean-Squared Error. The final evaluation metric is the Root Mean-Squared Error (RMSE), which is the square root of the original MSE. The advantage of RMSE is that it measures the average magnitude of the errors between predicted and actual similarity scores, giving the error in the same units as the data. This makes RMSE particularly intuitive since it aligns directly with the units of the similarity scores that are being predicted. For computation, the results of Scikit-learn's `mean_squared_error` function are square-root before returning the value.

4.4.2. Visualizations

To further assess the methodology, the transformation matrix B is examined to understand its impact on various embeddings used to test the hypothesis. Specifically, the spectrum, i.e., the eigenvalues of B, is computed for each PLM employed. Eigenvalues of a linear transformation represented by B act as scaling factors along directions defined by their corresponding eigenvectors. By computing them it becomes possible to analyze how the transformation enlarges, shrinks, or reverses the direction of vectors and to possibly understand how much the embeddings need to be transformed to encode the properties of interest. If B is the square matrix, its spectrum p is found by solving the characteristic polynomial given by the determinant of $B - \lambda I$, where λ represents the eigenvalue and I is the identity matrix. This process is formalized by Equation 63.

$$p(\lambda) = \det(B - \lambda I) \qquad \lambda \in \mathbb{C} \tag{63}$$

Given the large size of B, scientific computing libraries, in this case PyTorch, are used to compute the spectrum. Such libraries provide more efficient solutions involving numerical methods to solve Equation 63. The specific function applied here is the `torch.linalg.eigvals` method, which accepts a square matrix as input, i.e., the trained matrix B for each PLM, and outputs its spectrum.

Eigenvalues, being part of the complex number set $\mathbb{C}$, can be visualized by plotting their real parts on the x-axis and their imaginary parts on the y-axis, using a symmetrical plot generated through Matplotlib (Hunter, 2007). Given that the cosine similarity, i.e., the score the model approximates, reflects the cosine of the angle between two vectors, the transformation is expected to predominantly involve rotations rather than scaling.

Once the spectrum of B has been computed and the effect on the embeddings has been analyzed, the entire vector space is projected to a two-dimensional space to visualize how it has been transformed and to potentially identify anomalies,

outliers and clusters. The 768-dimensional embeddings need to be reduced to two dimensions. Several dimensionality reduction algorithms exist, such as Principal Component Analysis (PCA) (Hotelling, 1933), t-distributed Stochastic Neighbor Embedding (t-SNE) (Hinton and Roweis, 2002), and Uniform Manifold Approximation and Projection (UMAP) (McInnes et al., 2018).

The unsupervised t-SNE algorithm is chosen because of its non-linear nature, its ability to preserve the local structure of the data and the better performance compared to PCA or UMAP (Arora et al., 2018). Hinton and Roweis (2002) describes t-SNE using the following equations:

$$p_{j|i} = \frac{\exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2 / 2\sigma_i^2\right)} \tag{64}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{65}$$

$$q_{ij} = \frac{\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \left\|\mathbf{y}_k - \mathbf{y}_l\right\|^2\right)^{-1}} \tag{66}$$

$$D_{KL}\left(P \| Q\right) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{67}$$

where $p_{j|i}$ is the conditional probability modeling the similarity of datapoint $\mathbf{x}_j$ to datapoint $\mathbf{x}_i$ and σ_i is the variance of the Gaussian distribution centered on datapoint $\mathbf{x}_i$. The variance is adapted for each point differently and is chosen in a way that the perplexity of the conditional distribution equals a specified hyperparameter, which is set to 30. p_{ij} are the symmetrized joint probabilities used to capture the pairwise similarity between datapoints in the high-dimensional space, with N specifying the number of datapoints. q_{ij} measures the similarity between $\mathbf{y}_i$ and $\mathbf{y}_j$, which are the low-dimensional embeddings of the datapoints $\mathbf{x}_i$ and $\mathbf{x}_j$. D_{KL} is the Kullback-Leibler divergence (Kullback and Leibler, 1951), which is minimized during several iterations of gradient descent, which are set to 1000. The `TSNE` class of the Scikit-learn library encapsulates the learning process described by Equations 64, 65, 66 and 67 and can be easily used to reduce the dimensionality of the vector space and to plot it as a scatter plot using Matplotlib.

Visualizing several thousands of data points can become confusing and not very informative. In order to solve this problem and to gain more information about how the vector space gets transformed by the matrix, two random entities are highlighted before and after the transformation to inspect how they move in the space and how their angles get changed. The general assumption is that the two entities get closer, not in terms of magnitude because cosine similarity does not take magnitude into consideration, but in terms of angular proximity.

4.4.3. Qualitative Evaluation

Since the methodology does not involve the application of the trained models to downstream tasks, quantitative evaluations play the most vital role to test the hypothesis and to analyze the performance of the linear transformation. Nevertheless, it is still possible to evaluate the results in a qualitative way. To this extent, once the metrics introduced in Section 4.4.1 have been computed on the entire test set, the subject-object pairs with the lowest scores can be analyzed. Such an analysis aims to identify potential relations or entity types, for which the linear transformation is unable to approximate the true similarity scores, indicating flaws in the encoding of factual knowledge in the PLM tested.

To extract the worst-performing data points, the predicted similarity scores $\hat{Y}$ are subtracted from the actual similarity scores Y and then sorted in descending order. This methodology displays the subject-object pairs with the highest discrepancy between $\hat{Y}$ and Y. Because of the large effort of manually analyzing all data in the test set, only the bottom-30 subject-object pairs for all three PLMs are taken into consideration for the qualitative evaluation. This in-depth review is carried out to determine common characteristics or patterns that may contribute to the model's failure. This involves factors, such as the rarity of the entities, their length and the exact nature of the label of entities as provided in the English version of Wikidata. The most important analysis regards the type of relations present in the analyzed pairs, including their complexity and ambiguity. Since the cosine similarity score that is approximated by the transformed vector space represents the relational proximity between entities, it is most interesting to see if any specific types of relations keep on recurring in the bottom-30 entity pairs, indicating that their encoding might be most challenging.

5. Results

This chapter presents the results derived by applying the methods described in Chapter 4. Sections 5.1 and 5.2 are dedicated to the main results of the linear transformation, both the training and the quantitative results. Section 5.3 presents the visualization of the spectrum of the transformation matrices and the vector space, whereas Section 5.4 is dedicated to a qualitative analysis of a small subset of knowledge graph triples.

5.1. Training Results

As outlined in Section 4.3.3, the objective was to train the linear transformation model across 100 epochs. Each transformation successfully completed 100 epochs of gradient descent using various learning rates, achieving low losses and promising Pearson Correlation Coefficient (PCC) scores. The training results are collected in Table 4, reporting the loss values, the PCC scores, the learning rates and the weight decays for all models at the 100th epoch.

Table 4.: Training results for the different vectors spaces (Emb.) together with the learning rate (LR) and the weight decay (decay) hyperparameters

Emb.	**Loss**	**PCC**	**LR**	**Decay**
BERT-cls	0.0084	0.9038	1e-3	5e-3
BERT-avg	0.0084	0.9038	1e-3	5e-3
GPT2-avg	0.0163	0.8017	4e-3	5e-3
BART-avg	0.0117	0.8656	5e-3	5e-3

Figure 10 displays the training plots for the BERT-base model, comparing the use of the CLS token and the average of the token embeddings, denoted by BERT-cls and BERT-avg respectively. Noteworthy differences are observed early in the training phase. BERT-avg exhibits a more turbulent training process up to epoch 30 before stabilizing and reaching the function's minimum. Conversely, BERT-cls demonstrates a smoother and more consistent training trajectory, with similar performance metrics between training and validation datasets evident from epoch 20 onward. This suggests that the CLS token possesses stronger averaging and

contextual capabilities, whereas averaging the token embedding by performing a simple averaging does not provide the same results, especially with a lower epoch count. Nevertheless, both embedding strategies seem to converge to the minimum of the function in a similar way. Detailed plots for training and validation losses, as well as the PCC scores for both training and validation phases for the other PLMs tested, are presented in Appendix A.1. Neither the learning rate decay nor the early stopping mechanism were activated for any PLMs, indicating a steady and effective optimization process.

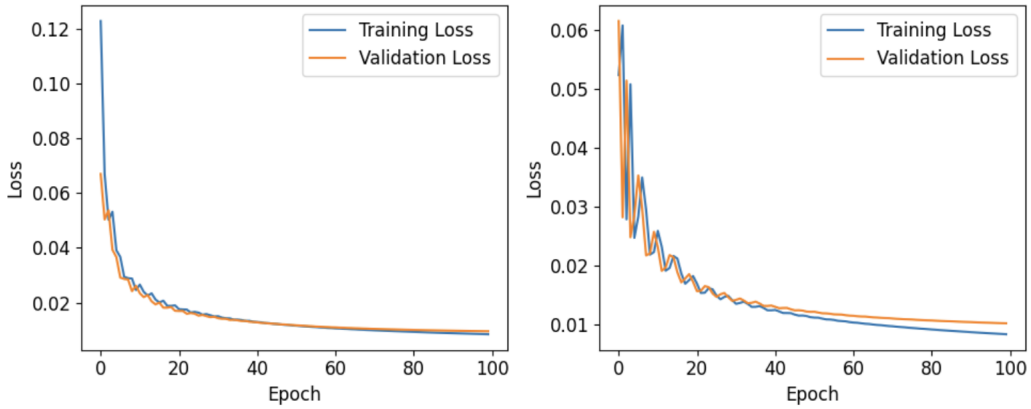

Figure 10.: Plots of the losses for both the BERT-cls (left) and BERT-avg (right) models

5.2. Quantitative Results

As outlined in Section 4.4.1, the family of quantitative metrics chosen to evaluate the performance of the trained linear transformation are the Pearson Correlation Coefficient (PCC), the Spearman's Rank Correlation Coefficient (SRCC), the Mean-Squared Error (MSE) and the Root Mean-Squared Error (RMSE). These metrics are applied to the test set, which contains 4,243 triples unseen during training.

Table 5 presents the performance of the original vector space using vanilla embeddings, which have not been processed through the transformation matrix B, as well as the transformed vector spaces. As anticipated, embeddings enriched with Wikipedia descriptions of the entities, denoted by *Wiki.*, significantly outperform those that contain only the entity names, labeled *Ent.*. This superior performance is due to the fact that given a triple (s, r, o), the Wikipedia description of s often contains relevant information about o as well, and vice versa. Consequently, when the cosine similarity is computed between vectors $\mathbf{h}_s$ and $\mathbf{h}_o$, the resulting scores

Table 5.: Results of the original (top) and transformed (bottom) vector spaces. The scores are computed for both the embeddings containing only the entities' names (Ent.) and for the entities enhanced with their corresponding Wikipedia descriptions (Wiki.)

	PCC		**SRCC**		**MSE**		**RMSE**	
Orig.	Wiki.	Ent.	Wiki.	Ent.	Wiki.	Ent.	Wiki.	Ent.
BERT-cls	0.6157	0.0289	**0.6208**	0.0478	0.0548	0.1254	0.2341	0.3542
BERT-avg	**0.6340**	0.2428	0.6155	0.1917	0.0565	**0.0520**	0.2378	**0.2281**
GPT2-avg	0.3949	0.4186	0.4590	0.4231	0.2341	0.2283	0.4838	0.4778
BART-avg	0.6000	0.1643	0.5801	0.1461	0.0446	0.0568	0.2113	0.2383
	PCC		**SRCC**		**MSE**		**RMSE**	
Transf.	Wiki.	Ent.	Wiki.	Ent.	Wiki.	Ent.	Wiki.	Ent.
BERT-cls	**0.9261**	0.8839	**0.9108**	0.8696	**0.0067**	0.0103	**0.0821**	0.1013
BERT-avg	0.9166	0.8796	0.9019	0.8674	0.0076	0.0106	0.0872	0.1030
GPT2-avg	0.8547	0.7988	0.8411	0.7882	0.0145	0.0185	0.1206	0.1427
BART-avg	0.9080	0.8431	0.8928	0.8329	0.0083	0.0138	0.0909	0.1360

are predictably higher. This property is absent in embeddings that only incorporate entity names, where the challenge lies in extracting latent semantic properties and relationships solely from these names, resulting in notably lower performance for the *Ent.* embeddings.

Interestingly, the original GPT-2 vector space of the entities only demonstrate strong positive linear and non-linear correlations between predicted and actual scores, denoted by PCC and SRCC. However, these embeddings also exhibit high error rates in predictions, indicating an inconsistent behavior. Conversely, BERT's and BART's vector spaces align with the initial hypotheses, showing weaker performance with only entity names but improved outcomes when Wikipedia descriptions are included.

The semantics and relation between the entities s and o in the vector space might be latent, when no additional information beside their names is given. The task of the linear transformation is to make these properties explicit and to transform the embeddings in a way that they align with the geometric properties of the knowledge graph embedding space. Notably, analyzing the vector space containing only the entities' names offers a unique perspective, as it demonstrates how facts can be encoded in a language model without additional information. In Table 5, noticeable improvements of the transformed vector space compared to the original vector space can be seen, with BERT-cls reaching the highest scores for both PCC and SRCC, as well as the lowest error rates. With a PCC of 0.8839, a SRCC of 0.8696, a MSE of 0.0103 and a RMSE of 0.1013 for the *Ent.* version, it is possible to

assume that BERT indeed possesses a high factual knowledge hidden in its vector space and the linear transformation successfully extracted it. BART ranks as the second-best performing PLM, displaying an average PCC performance drop of 3.7%, and 3.5% for the MSE relative to BERT-cls for the *Ent.* version. GPT2- is the worst performing PLM, aligning with the initial hypothesis described in Section 4.2.1 of its strong generative capabilities, but potentially weaker performance in creating rich contextual representations due to its autoregressive architecture. The average PCC performance drop compared to BERT-cls is of 7.2% and 3.5% compared to BART. The drop in MSE is of 8.2% compared to BERT-cls and 4.7% compared to BART. The relatively poor performance of GPT-2 compared to BERT and BART is further illustrated in the training plots presented in Appendix A.1. These plots reveal a notably turbulent optimization process for GPT-2, particularly in terms of optimizing the Pearson correlation coefficient. The progression of the PCC score throughout the epochs is inconsistent, displaying fluctuations rather than a steady increase. This volatility persists until around epoch 60, after which a distinct divergence between training and validation performance becomes apparent.

Reduced Dimensionality. The second experiment was conducted by reducing the rank of the matrix B to analyze how compactly knowledge graph properties are encoded in PLMs vector spaces. The tested dimensionalities, beside the full-rank tested and analyzed so far, are $\{32, 64, 128, 256, 512\}$.

Table 6 illustrates that a reduced vector space can still adequately approximate similarity scores, compared to the full-dimensional results presented in Table 5, strongly aligning with the results of Hewitt and Manning (2019). This trend is evident in both BERT and BART models. However, for GPT-2, it remains unclear how and where the knowledge graph might be encoded, as indicated by inconsistent performance metrics across different dimensionalities.

A small increase in performance observed as the dimensionality increases from 64 to 512 suggests that the core properties of the knowledge graph may be sufficiently captured within just 64 of the total 768 dimensions of BERT embeddings. The additional dimensions may contribute marginally to improving the similarity scores, potentially due to other linguistic or contextual properties encoded in the vectors, which are not the focus of this thesis. A similar trend is noticeable when incorporating the Wikipedia descriptions in the entities' embeddings, denoted by *Wiki.* in Table 6, with dimension 32 performing only 2.2% worse than dimensions 512 and 768 for the BERT-cls embeddings and 1.8% worse for the BERT-avg embeddings in terms of PCC.

BART demonstrates a similar pattern to BERT, potentially encoding essential properties within as few as 32 dimensions out of 768. The highest performance is noted at a dimensionality of 128 for the *Ent.* embeddings and at 512 for the

Table 6.: Results for the different dimensionalities of the output space transformed by the matrix B

		PCC		**SRCC**		**MSE**		**RMSE**	
Dim.	**Emb.**	Wiki.	Ent.	Wiki.	Ent.	Wiki.	Ent.	Wiki.	Ent.
16	BERT-cls	0.8908	0.8399	0.8760	0.8300	0.0108	0.0152	0.1040	0.1235
	BERT-avg	0.8787	0.8263	0.8650	0.8182	0.0121	0.0167	0.1101	0.1292
	GPT2-avg	0.8032	0.7863	0.7890	0.7753	0.0219	0.0205	0.1481	0.1432
	BART-avg	0.8600	0.8135	0.8460	0.8069	0.0137	0.0173	0.1168	0.1316
32	BERT-cls	0.9037	0.8493	0.8860	0.8372	0.0092	0.0137	0.0959	0.1169
	BERT-avg	0.8987	0.8539	0.8833	0.8443	0.0097	0.0139	0.0983	0.1150
	GPT2-avg	0.8206	0.7942	0.8057	0.7838	0.0190	0.0192	0.1379	0.1386
	BART-avg	0.8859	0.8349	0.8709	0.8252	0.0106	0.0147	0.1030	0.1213
64	BERT-cls	0.9120	0.8704	0.8962	0.8573	0.0081	0.0115	0.0904	0.1074
	BERT-avg	0.9064	0.8629	0.8912	0.8508	0.0087	0.0122	0.0933	0.1103
	GPT2-avg	0.8196	0.7982	0.8048	0.7875	0.0193	0.0192	0.1390	0.1386
	BART-avg	0.8950	0.8434	0.8801	0.8316	0.0096	0.0137	0.0978	0.1170
128	BERT-cls	0.9197	0.8804	0.9041	0.8664	0.0074	0.0106	0.0860	0.1030
	BERT-avg	0.9117	0.8754	0.8969	0.8637	0.0081	0.0110	0.0901	0.1049
	GPT2-avg	0.8197	0.8042	0.8053	0.7929	0.0192	0.0182	0.1385	0.1349
	BART-avg	0.9028	0.8506	0.8874	0.8400	0.0088	0.0130	0.0936	0.1140
256	BERT-cls	0.9244	0.8804	0.9089	0.8666	0.0069	0.0105	0.0832	0.1028
	BERT-avg	0.9167	0.8782	0.9024	0.8662	0.0076	0.0107	0.0872	0.1036
	GPT2-avg	0.8230	0.8034	0.8084	0.7921	0.0187	0.0183	0.1366	0.1352
	BART-avg	0.9054	0.8468	0.8905	0.8357	0.0085	0.0133	0.0922	0.1153
512	BERT-cls	**0.9260**	0.8837	**0.9111**	0.8695	**0.0068**	0.0103	**0.0823**	0.1015
	BERT-avg	0.9168	0.8807	0.9022	0.8688	0.0076	0.0105	0.0871	0.1025
	GPT2-avg	0.8232	0.7981	0.8089	0.7872	0.0186	0.0187	0.1363	0.1368
	BART-avg	0.9077	0.8376	0.8927	0.8356	0.0083	0.0132	0.0910	0.1150

Wiki. embeddings, though the increase in performance from 32 dimensions is slight, suggesting a plateau in the gain obtained from additional dimensions. Similarly, for the description-enhanced embeddings, the PCC performance increases only 2.2% from dimension 32 to dimension 512. The MSE also decreases just 1.2% from dimension 32.

As shown in Table 5, GPT-2 embeddings generally underperform compared to BERT and BART. This trend persists even with reduced dimensionalities, with no clear performance trend evident with changes in dimension. The best results are observed at 128 dimensions when training the linear transformation with entities only, similar to BART, yet without a significant or consistent improvement or decline at other tested dimensionalities. Similar results are observed with the description-enhanced embeddings, with dimension 32 showing very similar results to dimension 512, whereas the performance slightly decreases at dimension 64 and 128, indicating a relatively inconsistent behavior.

5.3. Visual Results

As explained in Section 4.4.2, the results are further analyzed trough visualizations. Specifically, the spectrum of the transformation matrix B for all three PLMs is plotted to understand the extent to which the vector space needs to be transformed to encode the properties of interest. Figure 11 shows such visualizations for the linear transformation trained with embeddings containing solely information about the entities's names.

As depicted in Figure 11, both BERT-avg and BERT-cls display eigenvalues that suggest a relatively random matrix structure, though unlike typical random matrices whose eigenvalues are constrained to the unit sphere, these eigenvalues range approximately from -0.6 to 0.6. This indicates that the transformation effects applied to BERT embeddings are relatively moderate. The distribution of these eigenvalues implies that the embeddings from BERT may already closely encode the necessary properties, and thus, the transformation matrix B may not require substantial modifications to these embeddings. Instead, B might only need to apply moderate scaling and rotations to achieve the desired characteristics commonly seen in random matrices, aligning with its initial configuration. This moderation in transformation suggests an inherent alignment of BERT embeddings with the structural properties needed for the task, without the necessity for aggressive adjustments.

Conversely, the transformation of BART's and GPT-2's vector spaces involves more adjustments and the eigenvalues are more dense towards the intersection of the real and imaginary parts. Even though the values are still grouped primarily between -0.5 and 0.5, they show more outliers and a less uniform distribution,

especially in the case of GPT-2.

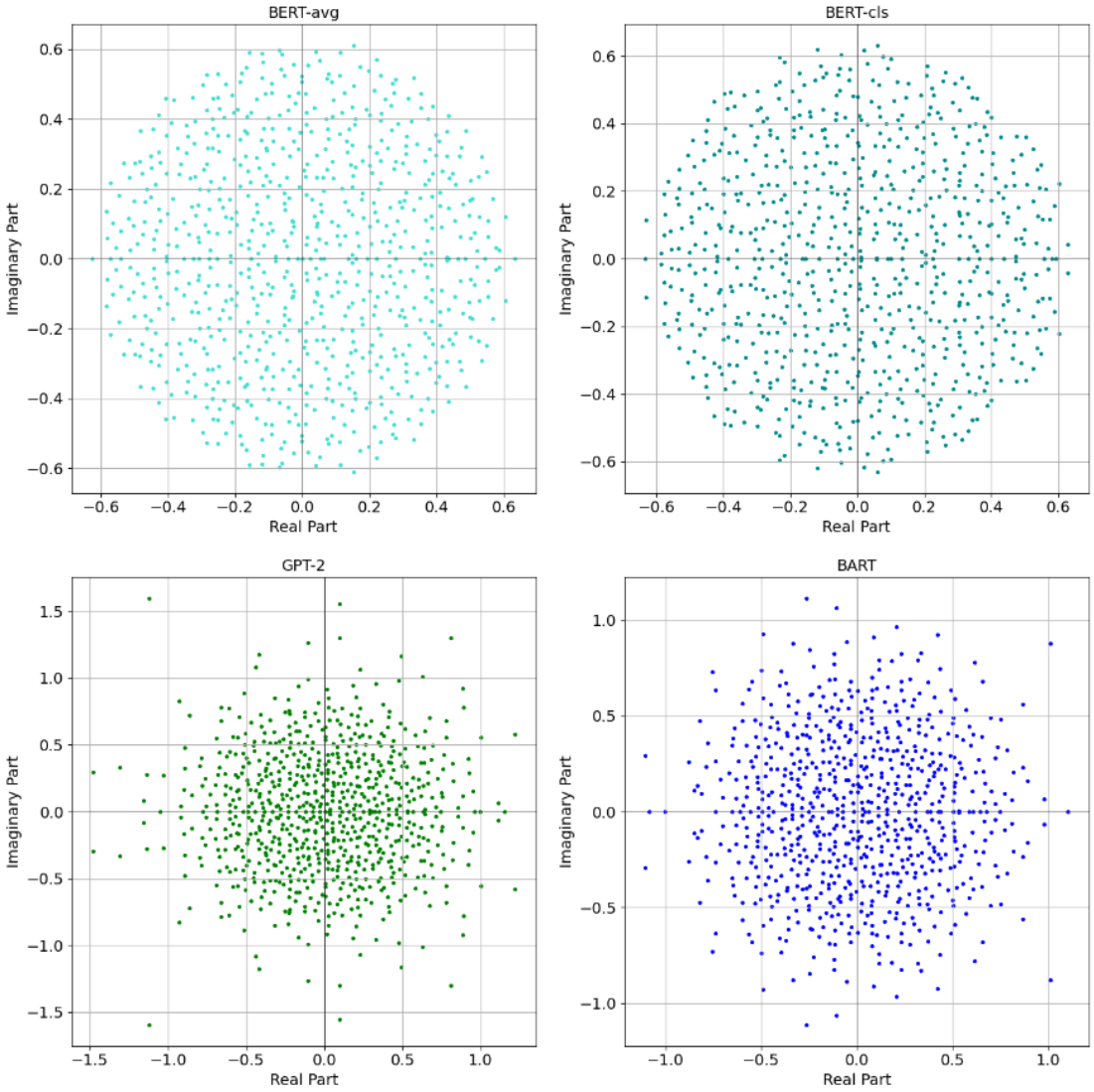

Figure 11.: Spectrum of the transformation matrices for BERT-avg, BERT-cls, BART, and GPT-2

Vector Space. As discussed in Section 4.4.2, one further method for visually inspecting the results involves the visualization of the vector space both before and after applying the transformation. For this purpose, the embeddings from the test set are reduced to 2 dimensions using the t-SNE algorithm and then displayed on a Cartesian plane. This section focuses exclusively on the vector space transformations of BERT-cls, while the visualizations for other PLMs are provided

in Appendix A.2. Figure 12 illustrates both the original and transformed vector spaces of BERT-cls. Initially, the original vector space appears more compact, with few discernible clusters and some outliers at the lower part of the plane. In this visualization, orange dots represent the subjects from the knowledge graph, and blue dots denote the objects. In contrast, the transformed vector space exhibits a less compact arrangement with a more distinct separation between subjects and objects, alongside enhanced clustering of objects in the peripheral areas. The magnitudes of the vectors also seem to increase compared to the non-transformed vector space, with several vectors approaching -100 and $+100$.

To gain more insights into how the transformation matrix B has transformed the vector space and moved the embeddings, two random entities are highlighted. These entities are visible in Figure 12 and they are marked with a large red dot for the subject and a large yellow dot for the object. The chosen entities are `The Day After` and `Cold War`, which are connected by the relation `main subject`. In the original vector space, the vectors corresponding to these entities are nearly orthogonal, suggesting a minimal semantic relationship. However, when the trained transformation matrix B is applied, the vectors move from the upper to the lower half of the plane, with one of the two dimensions being a negative one, but they also move significantly closer in angle. Despite their continued spatial separation, their angular proximity suggests a stronger semantic connection post-transformation.

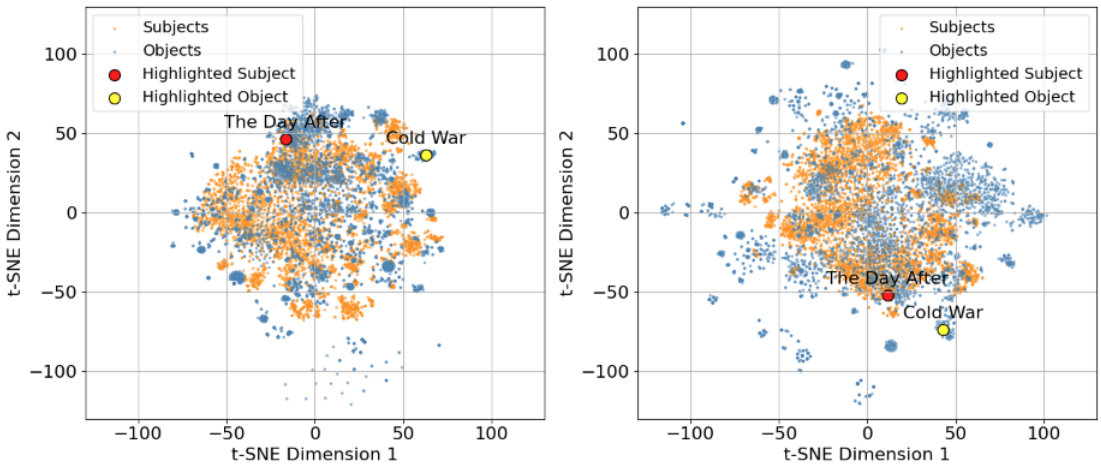

Figure 12.: Original (left) and transformed (right) vector spaces of BERT-cls

5.4. Qualitative Results

The qualitative results suggest that both variants of BERT, i.e., BERT-cls and BERT-avg, struggle with geographic relations, particularly the `shares border with` relation. Within the 30 analyzed triples, BERT-cls showed a great discrepancy between predicted and true scores for 8 instances of these relations, alongside three additional mispredictions related to other geographic categories: `located in the administrative territorial entity`, `country` and `capital`. A further triple depicting a geographic fact for which BERT-cls was not able to correctly approximate the similarity score is `(Guildford, instance of, borough in the United Kingdom)`. BERT-avg demonstrated even less proficiency with geographical facts, erroneously predicting 10 instances of the `shares border with` relation and encountering difficulties with other geographic relations, especially the `located in the administrative territorial entity`, `twinned administrative body`, `country`, and `capital` relations. This underperformance may be attributable to several factors. First, it is a prevalent relation within the dataset, increasing the likelihood of erroneous similarity predictions due to its large number. Second, many entities within this relation category have non-English names in spite of the explicit label as English on Wikidata, which presents a challenge for BERT, primarily trained on English data. This can potentially hinder its ability to accurately encode these specific facts. Lastly, the entities involved in this relation are often defined by purely factual, rather than linguistic, similarities, making it difficult for the model to use linguistic cues to accurately predict relationships, even after the transformation. For example, in the triple `(Pozzo d'Adda, shares border with, Basiano)`, BERT would need to have a detailed understanding of Italian geography, particularly the municipalities in the Lombardy region. Even though geographical facts are generally encoded in BERT's hidden representations, this small-scale qualitative analysis reveals that there are still notable gaps in its performance.

GPT-2 shares similarities with BERT, with the `shares border with` relation showing the weakest result with 11 erroneous predictions, potentially because of the same reasons as for BERT. Additional difficulties for GPT-2 are observed with familial and historical relationships. For instance, the model struggles with triples like `(Marcus Aurelius, child, Fadilla)`, or `(Elizabeth of Austria, spouse, Sigismund II Augustus)`, indicating a challenge in accurately encoding the nature of personal relationships between subjects and objects. Other problematic relations in this category include `father` and `sibling`. A potential reason for these difficulties is that the model must handle proper nouns, some of which are presented in their original languages, such as `Sigismund II Augustus` or `Germanicus`.

The pattern of weak performance with geographic relations continues with BART

as well. Out of the 30 analyzed triples, 13 are geographic facts, 10 of which belong to the `shares border with` relation. The remaining three are `located in the administrative territorial entity`, `twinned administrative body`, and the relation `instance of` for the fact `(Juiz the Fora, instance of, municipality of Brazil)`. Like GPT-2, BART struggles as well with personal facts about the entities, such as the `father`, `mother` and `family name` relations.

The only triple with weak performance common to all four vector spaces, i.e., BERT-cls, BERT-avg, GPT2-avg, and BART-avg is `(Ziguinchor, shares border with, West Coast Division)`. Other triples present in multiple vector spaces are:

- `(Brong-Ahafo Region, shares borders with, Eastern Region)`
- `(Hongzhou, father, Yongzheng Emperor)`
- `(Jóhann, said to be the same as, Ieuan)`

6. Discussion

Based on the results presented in Chapter 5, the methodology adopted to unveil factual knowledge as represented in knowledge graphs from pre-trained language models has proven to be successful. The custom dataset constructed for the purpose of this research provided a unique perspective to answer the research question and to test the hypothesis on the chosen PLMs. Learning a linear transformation of the vector spaces allowed them to align with the properties of knowledge graph embeddings and to unveil a latent space in which these properties are predominantly encoded. To avoid introducing too much complexity and to preserve the structural properties of the original space, it was decided that a simple linear transformation would be employed. Even though other methods propose the use of more complex models involving non-linear transformations (White et al., 2021; Sorodoc et al., 2020; Koto et al., 2021), this thesis focused on linearly transforming the original space to make the desired properties explicit without incurring in the assumption that the model has learned the underlying patterns in the data but the representations do not genuinely encode the phenomenon. This assumption is supported by the promising results in terms of automatic evaluation metrics, with the Pearson correlation and Spearman's rank correlation coefficients reaching scores over 0.80 in the case of all PLMs, approaching 0.90 in the case of BERT and even surpassing the 0.90 barrier when enhancing the vector space with Wikipedia descriptions for the corresponding knowledge graph entities. The descriptions-enhanced vectors demonstrate satisfactory performance when the vector space is not transformed by the linear transformation. This actually suggest that equipping factual associations with fact-specific definitions further aids the knowledge representation of models trained on textual sequences, which comes as little surprise. The difference between description-enhanced embeddings and embeddings containing solely the names of the entities is notable. In the case of BERT-cls, this gap is of 59% in terms of PCC. Nevertheless, more interesting findings are given by the vectors containing solely the entities' names, since, as stated in Section 5.2, the goal is to understand how much factual information a language model's vector space encodes when comparing it to how knowledge graphs represent this information, which is often in terms of simple triples. Linear transformations seem to be an efficient method to identify subspaces with factual associations and to quantify factual knowledge hidden in PLMs' vector spaces. Such method can potentially eliminate the need to inject new knowledge into the model while relying solely on the information learned during

pre-training.

Inspired by the work of Hewitt and Manning (2019), one further experiment involved changing the dimensionality of the transformed vector space to analyze how compactly the knowledge graph can be encoded in the language model. The findings reveal very similar performance in the vector space containing only 64 dimensions and increasing it beyond 64 to the standard 768 dimensions leads to very similar results. These results strongly align with the experiments conducted by Hewitt and Manning (2019), suggesting that the remaining dimensions might be dedicated to additional knowledge, but do not contribute to the successful encoding of factual information.

Other interesting findings are given by the qualitative evaluation, where the 30 worst-performing data points for each PLM were manually analyzed to discover patterns for which the linear transformation cannot deliver good results. As presented in Section 5.4, common relations shared by all three PLMs, i.e., BERT, GPT-2, and BART, are geographic relations and familiar/historical relations. Although the analysis is not statistically significant due to the small sample size, the language models seem to have difficulties, for instance, in capturing the relation between cities that share borders or to encode the information that an object has a familiar relation with the subject. In such cases, the discrepancy between predicted and actual similarity scores is high, suggesting a poor alignment with the properties of the knowledge graph embeddings.

An additional objective of this thesis was to understand which type of PLMs are most capable of encoding factual knowledge. To this end, three models with different architectures were chosen. All three are Transformer-based models, but BERT has an encoder-only architecture, GPT-2 is a decoder-only model, whereas BART is an encoder-decoder Transformer. The findings reveal that BERT outperforms the other PLMs in all experiments, making it the best-performing model in this thesis. In fact, the vector spaces of BERT and its variants have proven to encode the most linguistic knowledge compared to other deep neural language models (Hewitt and Manning, 2019; Reif et al., 2019; Koto et al., 2021; Tenney et al., 2019a). Therefore, it is possible to assume that this is the case for factual knowledge as well. It seems that the bidirectional architecture of BERT makes it particularly suitable to encode information from both sides of the sequence and to increase its contextual capabilities. This may be one of the reasons explaining its better performance compared to the other models, since a fact contained in a triple can be present in multiple sequences seen by BERT during training and the relative positions influence each other, successfully encoding information about subjects and objects in the contextual representations. Another reason for BERT's superior results compared to GPT-2 and BART might be correlated to the data used during pre-training. BERT was trained using a mixture of English Wikipedia and the

BooksCorpus (Zhu et al., 2015). No common crawl was involved, as it is the case of GPT-2 (Radford et al., 2019). This limitation to specific resources might have influenced the final representations of BERT in the sense of having a more pure representation and closer relation to Wikipedia data. Since the Wikidata knowledge graph was constructed from Wikipedia, the assumption exists that the embeddings have learned the underlying structure of entities and their relations.

Beside the promising results, this thesis has several limitations. One of the most important limitation regards the generation of knowledge graph embeddings for the entities using the Wembedder API. Even though this strategy strongly facilitates the access of embeddings generated using RDF2Vec, the resulting vectors may not have an increased notion of the structure and semantics of the knowledge graph used in this thesis, leading to potentially less nuanced vectors. A further limitation is given by the employment of cosine similarity as a metric to quantify the factual information between subjects and objects in the knowledge graph. Although cosine similarity is a valuable metric to evaluate the relatedness between vectors, it may not capture the factual relations in their entirety and it is sensitive to how the embeddings are trained and what their scoring functions are. Furthermore, the usage of English-only PLMs poses another limitation, since it remains unclear if multilingual models and datasets can yield the same results. A large number of entities in the knowledge graph are named entities that are also in the original language in the English version of Wikidata, making the encoding particularly challenging. Even though such models see non-English words during training because of the absence of English equivalents, comparing the results with multilingual models might offer interesting insights. Moreover, the lack of GPUs constrained the present work to the usage of small-scale models, the largest one containing 140 million parameters. Lastly, due to the analytical nature of this thesis, there has been a lack of practical testing on downstream tasks, which could further validate the effectiveness of the proposed methods in real-world applications.

7. Conclusions

The starting point of the present research was that Pre-trained Language Models (PLMs) have proven to encode extensive information in their hidden representations, but to this day it remains unclear how they encode factual associations in their vector spaces and if such associations are represented in a similar way as in knowledge graphs and knowledge graph embeddings. To address the research question, a linear transformation of the original vector spaces of three PLMs, i.e., BERT-base, GPT-2-small and BART-base, was learned to uncover a latent vector space in which knowledge graphs properties are predominantly encoded. To this end, a labeled dataset featuring subjects and objects from the Wikidata knowledge graph was constructed, together with a similarity score between the two entities, representing their relational proximity. The structural probe in form of a linear transformation, inspired by Hewitt and Manning (2019), had the task of taking as input the contextual embeddings of the knowledge graphs' entities and then transforming the vector space to approximate the true cosine similarity scores depicted in the dataset. The assumption was that accurate similarity predictions indicate a latent space in which factual knowledge is implicitly acquired by the language models through training on vast amounts of language data.

Results have shown promising correlations between predicted and true scores indicated by the Pearson and Spearman's rank correlation coefficients, as well as very low error rates in terms of mean-squared errors. BERT-base achieved the most important results in all experiments compared to GPT-2-small and BART-base, potentially making it the PLM with the most factual knowledge hidden in its vector space. Nevertheless, a small-scale qualitative analysis reported difficulties for all three PLMs to accurately align with the geometric properties of knowledge graph embeddings when dealing with geographic relations and familiar/historical relations, indicating a gap in their encoded facts.

The present work contributed to the vibrant research fields of deep learning, computational linguistics and explainable Artificial Intelligence by proposing a simple, yet effective method to uncover hidden properties of knowledge graphs from PLM's vector spaces. Furthermore, it provided interesting insights into the field of combining knowledge-based and neural approaches, called neuro-symbolic AI. The findings can potentially suggest an alternative approach to enhance the explicitness of specific information within PLMs without the need for injecting new knowledge or extensive fine-tuning of parameters. Instead, this method utilizes the

original model structure without substantial modifications. Furthermore, the study demonstrated that encoder-only Transformers excel over other variants in encoding factual information, potentially because of their bidirectional architecture and in the case of BERT, the specific data on which it has been trained.

Future research may overcome some of the limitations introduced in Chapter 6. Firstly, training custom knowledge graph embeddings models, such as TransE, DistMult, and NTN on the graph represented by the dataset used in this research could result in more customized vectors with a stronger notion of the graph structure given by the triples, including their relations and underlying semantics. Secondly, instead of relying on cosine similarity as a metric to quantify the factual relationships between two entities, an alternative method would be to follow the strategy implemented by Hewitt and Manning (2019). This could involve counting the number of hops needed to go from one entity to the other, which of course could pose its own challenges given by the structural differences between knowledge graphs and syntax trees. Thirdly, with the ongoing advancements in larger and more sophisticated PLMs, future studies could explore applying these enhanced methodologies to Large Language Models (LLMs), potentially yielding superior results. Additionally, extending the research to incorporate multilingual knowledge graphs and models could further substantiate the hypothesis and broaden its applicability across different languages. Finally, future work could also explore the practical applications of the trained transformations in downstream tasks that demand factual knowledge, such as question answering. This would not only demonstrate the real-world utility of the transformations but also provide deeper insights into their effectiveness across various knowledge-driven applications.

Bibliography

Charu C. Aggarwal (2023). *Neural Networks and Deep Learning, A Textbook*. Springer Cham.

Sanjeev Arora, Wei Hu, and Pravesh K Kothari (2018). An analysis of the t-sne algorithm for data visualization. In *Conference on learning theory*, pages 1455–1462. PMLR.

Hannah Bast, Björn Buchhold, Elmar Haussmann, et al. (2016). Semantic search on text and knowledge bases. *Foundations and Trends® in Information Retrieval*, 10(2-3):119–271.

Hannah Bast and Elmar Haussmann (2015). More accurate question answering on freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 1431–1440, New York, NY, USA. Association for Computing Machinery.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass (2017). What do neural machine translation models learn about morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 861–872, Vancouver, Canada. Association for Computational Linguistics.

Iz Beltagy, Matthew E. Peters, and Arman Cohan (2020). Longformer: The long-document transformer. *CoRR, abs/2004.05150*.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent (2000). A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang (2013). Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544. Association for Computational Linguistics.

Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen (2022). *The Modern Mathematics of Deep Learning*, page 1–111. Cambridge University Press.

Steven Bird and Edward Loper (2004). NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko (2013). Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi (2019). COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779, Florence, Italy. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. (2023). Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Kenneth Ward Church and Patrick Hanks (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

Roi Cohen, Mor Geva, Jonathan Berant, and Amir Globerson (2023). Crawling the internal knowledge-base of language models. In *Findings of the Association for*

Computational Linguistics: EACL 2023, pages 1856–1869, Dubrovnik, Croatia. Association for Computational Linguistics.

Pedro Colon-Hernandez, Catherine Havasi, Jason Alonso, Matthew Huggins, and Cynthia Breazeal (2021). Combining pre-trained language models and structured knowledge. *CoRR, abs/2101.12294*.

Richard Cyganiak, David Wood, and Markus Lanthaler (2014). Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978—-2988, Florence, Italy. Association for Computational Linguistics.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier (2017a). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier (2017b). Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 933–941. PMLR.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610.

Jacob Eistenstein (2019). *Introduction to Natural Language Processing*. The MIT Press.

J.R. Firth (1957). A synopsis of linguistic theory 1930-1955. *Studies in Linguistic Analysis*, pages 1–32.

Christopher Frye, Colin Rowat, and Ilya Feige (2020). Asymmetric shapley values: incorporating causal knowledge into model-agnostic explainability. In *Advances in Neural Information Processing Systems*, volume 33, pages 1229–1239. Curran Associates, Inc.

Bin Fu, Yunqi Qiu, Chengguang Tang, Yang Li, Haiyang Yu, and Jian Sun (2020). A survey on complex question answering over knowledge base: Recent advances and challenges. *CoRR, abs/2007.13069*.

Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin (2017a). A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 123–135, Vancouver, Canada. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin (2017b). Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Stephan Grimm (2009). Knowledge representation and ontologies. In *Scientific data mining and knowledge discovery: Principles and foundations*, pages 111–137. Springer.

Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun (2023). Does localization inform editing? surprising differences in causality-based localization vs. knowledge editing in language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 17643–17668. Curran Associates, Inc.

John Hewitt and Percy Liang (2019). Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, Hong Kong, China. Association for Computational Linguistics.

John Hewitt and Christopher D. Manning (2019). A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Geoffrey E Hinton and Sam Roweis (2002). Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press.

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph (2009). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC.

Sepp Hochreiter and Jürgen Schmidhuber (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann (2021). Knowledge graphs. *ACM Comput. Surv.*, 54(4).

Harol Hotelling (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 6(23):417–441.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen (2021). Lora: Low-rank adaptation of large language models. *CoRR, abs/2106.09685*.

Ting Hua, Yen-Chang Hsu, Felicity Wang, Qian Lou, Yilin Shen, and Hongxia Jin (2022). Numerical optimizations for weighted low-rank estimation on language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1404–1416, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

J. D. Hunter (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. (2024). Mixtral of experts. *CoRR, abs/2401.04088*.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig (2020). How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy (2020). SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland. Association for Computational Linguistics.

S. Katz (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.

Diederik P Kingma and Max Welling (2013). Auto-encoding variational bayes. *CoRR, abs/1312.6114*.

Fajri Koto, Jey Han Lau, and Timothy Baldwin (2021). Discourse probing of pretrained language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3849–3864, Online. Association for Computational Linguistics.

Solomon Kullback and Richard A Leibler (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom (2018). LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, Melbourne, Australia. Association for Computational Linguistics.

Thomas Landauer and Susan Dumais (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychol Rev*, 104.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2015). Deep learning. *Nature*, 521:436–444.

Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel (1989). Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Seungeon Lee, Xiting Wang, Sungwon Han, Xiaoyuan Yi, Xing Xie, and Meeyoung Cha (2022). Self-explaining deep models with logic rule reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pages 3203–3216. Curran Associates, Inc.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Shaobo Li, Xiaoguang Li, Lifeng Shang, Zhenhua Dong, Chengjie Sun, Bingquan Liu, Zhenzhou Ji, Xin Jiang, and Qun Liu (2022). How pre-trained language models capture factual knowledge? a causal-inspired analysis. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1720–1732, Dublin, Ireland. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang (2021). Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Jörg Liesen (2015). *Linear Algebra*, 1st ed. 2015. edition. Springer Undergraduate Mathematics Series. Springer International Publishing : Imprint: Springer,, Cham :.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu (2015). Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang (2020). K-bert: Enabling language representation with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2901–2908.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov (2019). Roberta: A robustly optimized bert pretraining approach. *CoRR*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter (2017). Decoupled weight decay regularization. *CoRR, abs/1711.05101*.

Scott M Lundberg and Su-In Lee (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

Chunjie Luo, Jianfeng Zhan, Xiaohe Xue, Lei Wang, Rui Ren, and Qiang Yang (2018). Cosine normalization: Using cosine similarity instead of dot product in

neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 382–391. Springer.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger (2018). Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861.

Oren Melamud, Jacob Goldberger, and Ido Dagan (2016). context2vec: Learning generic context embedding with bidirectional LSTM. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 51–61, Berlin, Germany. Association for Computational Linguistics.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov (2022). Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc.

Julian Michael, Jan A. Botha, and Ian Tenney (2020). Asking without telling: Exploring latent ontologies in contextual representations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6792–6812, Online. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). Efficient estimation of word representations in vector space. *CoRR, abs/1301.3781*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin (2017). Advances in pre-training distributed word representations. *CoRR, abs/1712.09405*.

Tomas Mikolov, Martin Karafiát, Luká Burget, Jan Honza ernocký, and Sanjeev Khudanpur (2010). Recurrent neural network based language model. In *Interspeech*.

Hermann Ney, Ute Essen, and Reinhard Kneser (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech Language*, 8(1):1–38.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio (2016). Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Maximilian Nickel, Volker Tresp, Hans-Peter Kriegel, et al. (2011). A three-way model for collective learning on multi-relational data. In *Icml*, volume 11, pages 3104482–3104584.

Finn Årup Nielsen (2017). Wembedder: Wikidata entity embedding web service. *CoRR, abs/1710.04099*.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov (2016). Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Sungjoon Park, JinYeong Bak, and Alice Oh (2017). Rotated word vector representations and their interpretability. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 401–411, Copenhagen, Denmark. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Heiko Paulheim (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508.

Heiko Paulheim, Petar Ristoski, and Jan Portisch (2023). *Embedding Knowledge Graphs with RDF2vec*. Synthesis Lectures on Data, Semantics, and Knowledge. Springer Cham.

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

Jeffrey Pennington, Richard Socher, and Christopher Manning (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018a). Deep contextualized word representations. 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 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih (2018b). Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller (2019). Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Ngoc-Quan Pham, German Kruszewski, and Gemma Boleda (2016). Convolutional neural network language models. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 1153–1162. Association for Computational Linguistics.

M.T. Pilehvar and J. Camacho-Collados (2020). *Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning*. Synthesis Lectures on Human Language Technologies Series. Morgan & Claypool Publishers.

Tiago Pimentel, Josef Valvoda, Rowan Hall Maudslay, Ran Zmigrod, Adina Williams, and Ryan Cotterell (2020). Information-theoretic probing for linguistic structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4622, Online. Association for Computational Linguistics.

Ofir Press and Lior Wolf (2017). Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. (2018). Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Emily Reif, Ann Yuan, Martin Wattenberg, Fernanda B Viegas, Andy Coenen, Adam Pearce, and Been Kim (2019). Visualizing and measuring the geometry of bert. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Petar Ristoski and Heiko Paulheim (2016). Rdf2vec: Rdf graph embeddings for data mining. In *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15*, pages 498–514. Springer.

Ronald Rosenfeld (2000). Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278.

Sascha Rothe, Sebastian Ebert, and Hinrich Schütze (2016). Ultradense word embeddings by orthogonal transformation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 767–777, San Diego, California. Association for Computational Linguistics.

Sascha Rothe and Hinrich Schütze (2016). Word embedding calculus in meaningful ultradense subspaces. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 512–517, Berlin, Germany. Association for Computational Linguistics.

Nicholas A. Roy, Junkyung Kim, and Neil Rabinowitz (2022). Explainability via causal self-talk. In *Advances in Neural Information Processing Systems*, volume 35, pages 7655–7670. Curran Associates, Inc.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams (1986a). Learning representations by back-propagating errors. *Nature*, 323:533–536.

David E. Rumelhart, James L. McClelland, and PDP Research Group (1986b). *Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations*. The MIT Press.

Stuart J Russell and Peter Norvig (2016). *Artificial intelligence: a modern approach*. Pearson.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. (2021). Multitask prompted training enables zero-shot task generalization. *CoRR, abs/2110.08207*.

Phillip Schneider, Tim Schopf, Juraj Vladika, Mikhail Galkin, Elena Simperl, and Florian Matthes (2022). A decade of knowledge graphs in natural language processing: A survey. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 601–614, Online only. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani (2018). Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh (2020). Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 4222–4235. Association for Computational Linguistics.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng (2013). Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26.

Ionut-Teodor Sorodoc, Kristina Gulordava, and Gemma Boleda (2020). Probing for referential information in language models. In *Proceedings of the 58th Annual*

Meeting of the Association for Computational Linguistics, pages 4177–4189, Online. Association for Computational Linguistics.

Robyn Speer, Joshua Chin, and Catherine Havasi (2017). Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu (2024). Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Guy Tennenholtz, Yinlam Chow, Chih-Wei Hsu, Jihwan Jeong, Lior Shani, Azamat Tulepbergenov, Deepak Ramachandran, Martin Mladenov, and Craig Boutilier (2023). Demystifying embedding spaces using large language models. *CoRR, abs/2310.04475*.

Ian Tenney, Dipanjan Das, and Ellie Pavlik (2019a). Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick (2019b). What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. (2023). Llama: Open and efficient foundation language models. *CoRR, abs/2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero,

Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Denny Vrandečić and Markus Krötzsch (2014). Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh (2019). Universal adversarial triggers for attacking and analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China. Association for Computational Linguistics.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang (2021). Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen (2014). Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. (2022a). Emergent abilities of large language models. *CoRR, abs/2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. (2022b). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jennifer C. White, Tiago Pimentel, Naomi Saphra, and Ryan Cotterell (2021). A non-linear structural probe. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 132–138, Online. Association for Computational Linguistics.

I.H. Witten and T.C. Bell (1991). The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Junda Wu, Tong Yu, Rui Wang, Zhao Song, Ruiyi Zhang, Handong Zhao, Chaochao Lu, Shuai Li, and Ricardo Henao (2023). Infoprompt: Information-theoretic soft prompt tuning for natural language understanding. In *Advances in Neural Information Processing Systems*, volume 36, pages 61060–61084. Curran Associates, Inc.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR, abs/1609.08144*.

Chao Xing, Dong Wang, Chao Liu, and Yiye Lin (2015). Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1006–1011, Denver, Colorado. Association for Computational Linguistics.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng (2014). Embedding entities and relations for learning and inference in knowledge bases. *CoRR, abs/1412.6575*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He (2018). Autoencoder and its various variants.

Xiang Zhang, Junbo Zhao, and Yann LeCun (2015). Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu (2019). ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451, Florence, Italy. Association for Computational Linguistics.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. (2023). A survey of large language models. *CoRR, abs/2303.18223*.

Kaitlyn Zhou, Kawin Ethayarajh, Dallas Card, and Dan Jurafsky (2022). Problems with cosine as a measure of embedding similarity for high frequency words. *CoRR, abs/2205.05092*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

A. Appendix

The appendix that follows includes supplementary material not presented in the main body of this thesis. It contains additional plots and results that offer a broader overview of the performance and outcomes of the factual probe discussed throughout this study.

A.1. Training

Figures 14, 13, 15, 16 display the training and validation losses, as well as the training and validation Pearson Correlation Coefficients (PCCs) for the BERT-cls, BERT-avg, GPT2-avg and BART-avg models.

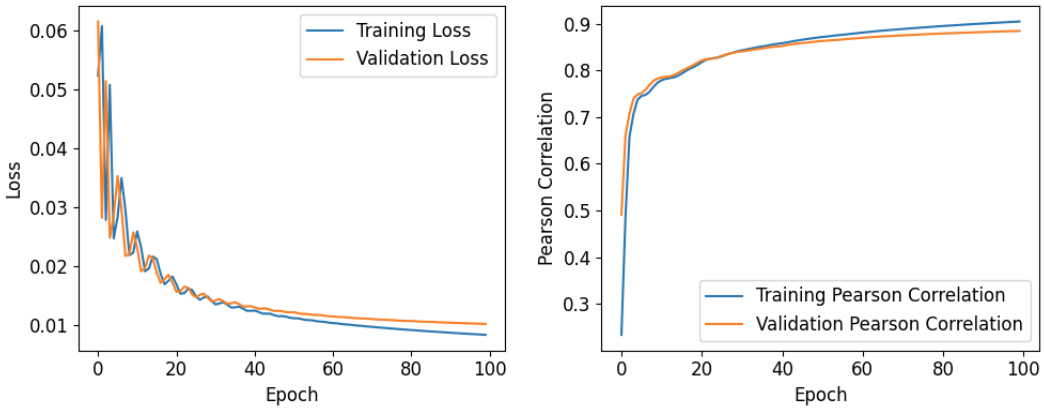

Figure 13.: Loss (left) and Pearson correlation coefficient (right) for the BERT-avg model.

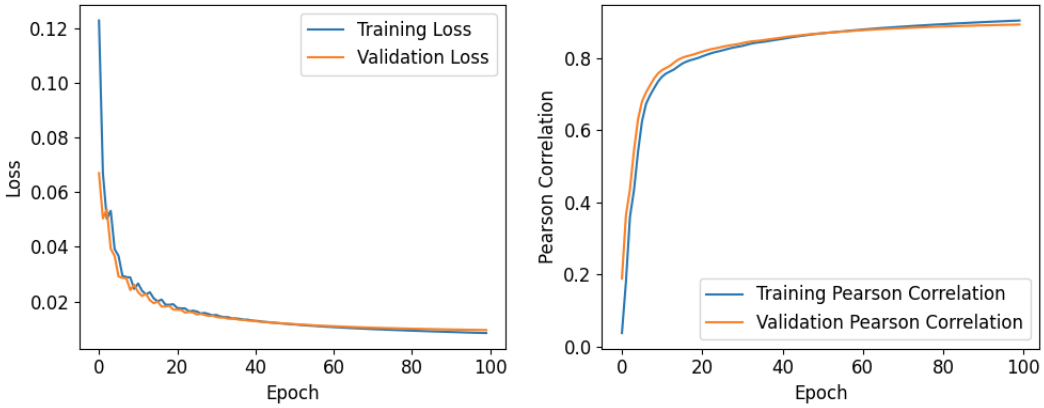

Figure 14.: Loss (left) and Pearson correlation coefficient (right) for the BERT-cls model.

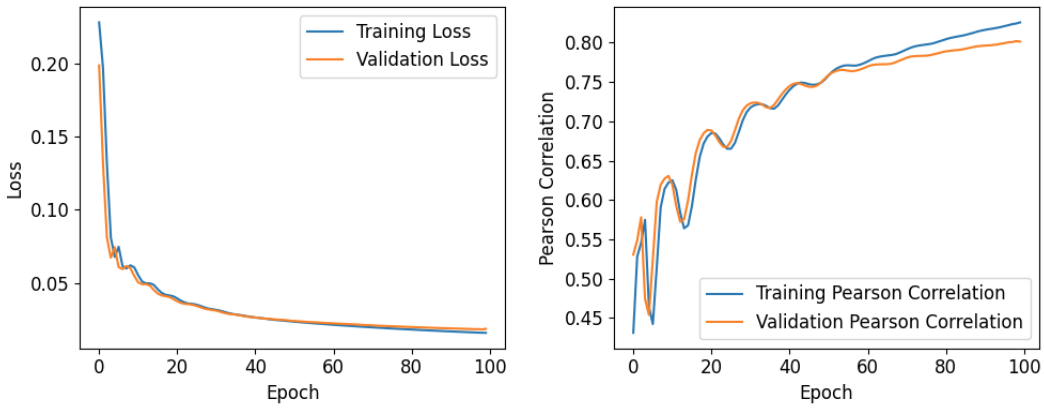

Figure 15.: Loss (left) and Pearson correlation coefficient (right) for the GPT2-avg model.

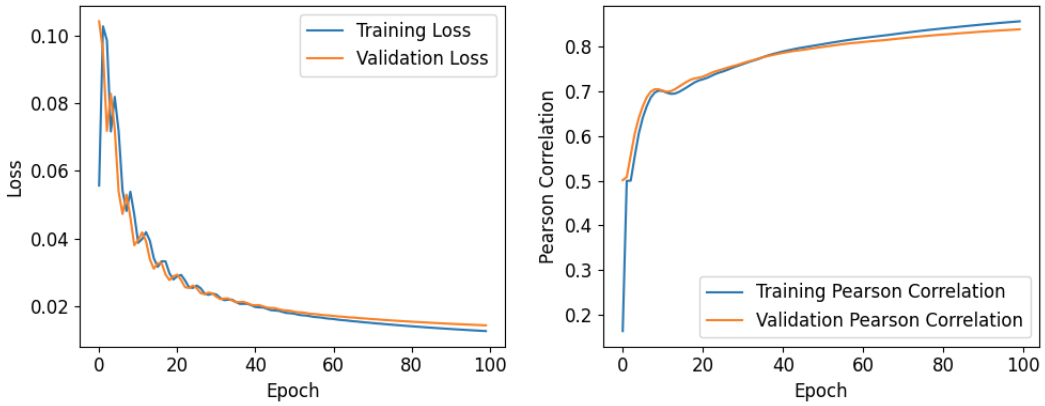

Figure 16.: Loss (left) and Pearson correlation coefficient (right) for the BART-avg model.

A.2. Visualizations

Figures 17, 18, 19 show the original vector spaces and the corresponding transformed vector space of BERT-avg, GPT-2 and BART, respectively. The highlighted entities serve to better understand how the space evolves and how the embeddings get closer in terms of angular proximity.

Figure 17.: Original (left) and transformed (right) vector spaces of BERT-avg.

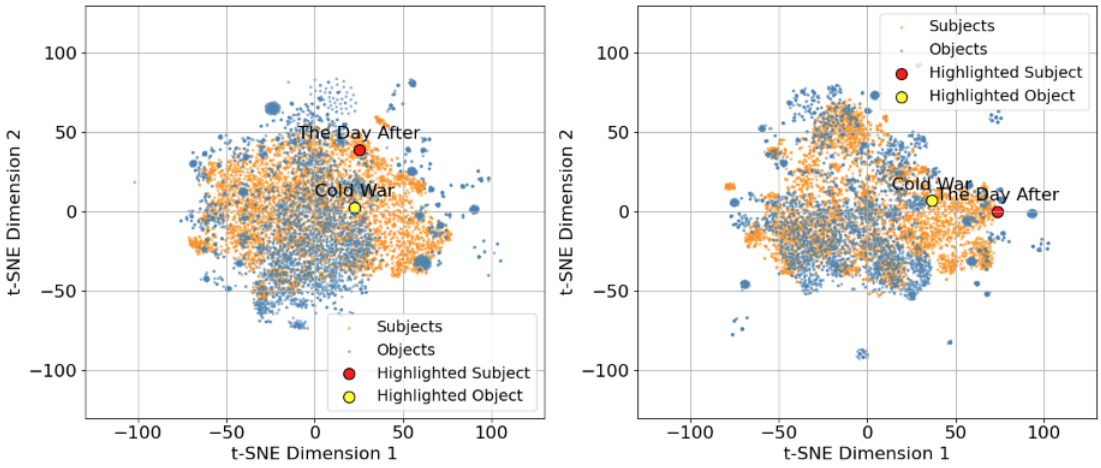

Figure 18.: Original (left) and transformed (right) vector spaces of GPT-2.

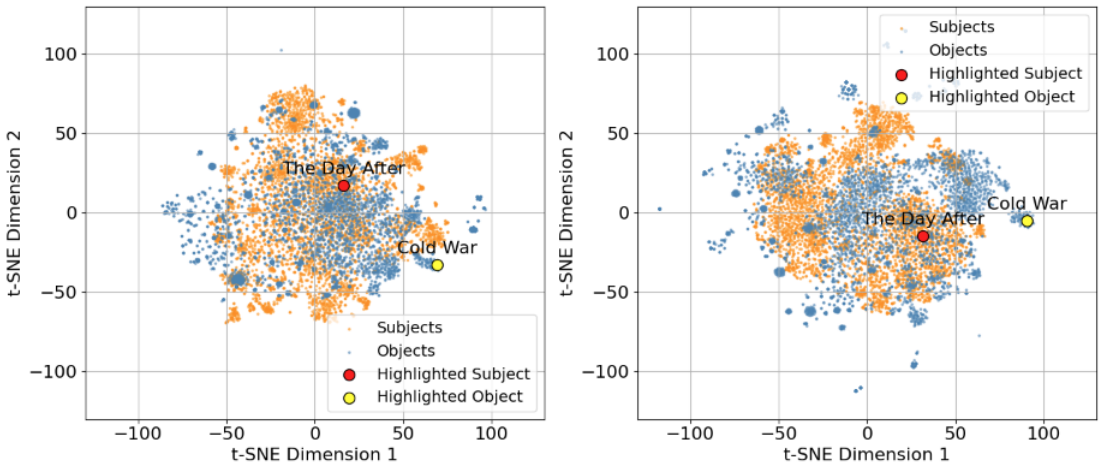

Figure 19.: Original (left) and transformed (right) vector spaces of BART.