Università di Pisa

Department of Computer Science
Master's Degree in Computer Science

Regularising Transformers By Symbolic Knowledge And Deep Graph Networks

Author:
Matteo Medioli

Supervisor:
Prof. Dr. Davide Bacciu

Co-Supervisor:
Andrea Valenti
Drs. Lucia Passaro

AA 2020/2021 - April 2022

Contents

1 Introduction **3**

2 Background **5**
2.1 Graphs 5
2.2 Knowledge Graph 8
2.3 Deep Learning for Graphs 11
2.3.1 Early Recursive Approaches 11
2.3.2 Handling Cycles with Deep Graph Networks 12
2.3.3 Topological Ordering Issue in Neighborhood Aggregation 16
2.4 BERT 17
2.4.1 Self-Attention and Multi-Head attention 17
2.4.2 Architecture Overview 21
2.4.3 Input Representation 22
2.4.4 Masked Language Modeling 23

3 Related Work **26**
3.1 DGNs and KGs in Language Modeling 26
3.1.1 Knowledge Graph Embeddings 26
3.1.2 Context Diffusion in Knowledge Graphs 28
3.1.3 Applications of DGNs in Natural Language Processing 32
3.2 Adapts Different Embeddings Spaces 36
3.2.1 Enriching Word Representations 36
3.2.2 Embeddings Regression 38

4 Proposed Framework **42**
4.1 KGE Sentence Dictionary 42
4.1.1 Learning KGEs 43
4.1.2 Word Sense Disambiguation 45
4.2 MLM KGE Regression Regularization 46

5 Experiments **50**
5.1 Training Dataset 50
5.1.1 WN18RR 50
5.1.2 FB15K-237 52
5.1.3 Wikipedia 53
5.2 Experimental Setup 54

5.2.1 KBGAT 54
5.2.2 BERT-KG and BERT-BASELINE 56
5.3 Language Model Probing Tasks 59
5.3.1 Datasets 59
5.3.2 Word Similarity 60
5.3.3 Knowledge Base Completion 61

6 Results and Analysis **63**
6.1 KBGAT Evaluation 63
6.2 Language Model Training 64
6.3 MEN Word Similarity 66
6.4 SQuAD KBC 69

7 Conclusions **71**

1 Introduction

Graphs are particularly useful data structures in real-life applications and are the focus of several research domains, from medicine (Barbiero et al. [1]) to logistics (Gutierrez et al. [2]), from social networks (Ruiz-Frau et al. [3]) to autonomous driving (Hegedűs et al. [4]). Graphs are a good way to structure databases that store ontological knowledge. In particular, Knowledge Graphs are one of the main resources for representing knowledge, reasoning about facts, and retrieving information efficiently (Färber [5], Fellbaum [6], Bordes et al. [7], Toutanova et al. [8], Matthew [9], Auer et al. [10], Suchanek and Weikum [11]). With the advent of Deep Graph Networks (Bacciu et al. [12]), it is possible to represent and analyze the complex structured information collected in a graph. Language models, in particular Transformers-based (Vaswani et al. [13]) are trained on plain text corpora, generalizing natural languages based solely on word distribution and attention mechanisms (Bahdanau et al. [14]) to capture part of the context in the text. Furthermore, word embeddings learned from a transformers-based language model have no structure for diversifying distinct entities which often appear in the same context (Zhou et al. [15]). The objective of this thesis is to define a regularization term to transfer symbolic knowledge from knowledge graphs to language models, through the use of Deep Graph Networks during Masked Language Modeling pre-training. In particular, the proposed framework performs a regression between word-embeddings obtained from the encoder of the Transformers architecture, specifically BERT (Devlin et al. [16]), and the node-embeddings computed by Graph Attention Network (Velickovic et al. [17]). In addition, this work analyzes two different language model probing tasks (Bruni et al. [18], F. Petroni and Riedel [19]) to evaluate the impact of the proposed regularization term in the symbolic knowledge transfer process in language modeling. In Section 2 a comprehensive background for the main concepts explored in the proposed work is defined, including basic notions of graph theory, deep learning for graphs given an overview of Deep Graph Network models, Transformer architecture and BERT model. Section 3 describes work that have addressed similar tasks, such as the application of DGN on Knowledge Graphs in NLP and embeddings regression. In the third section, the BERT regularization framework is described, specifically analyzing the mapping function between word-embeddings to node embeddings from a word sense disambiguation problem perspective. The section concludes with a description of the concepts behind the regression, defining

the regularization term for BERT. Then, in Section 4 defines the elements necessary for the proposed regularization including how the knowledge graph embeddings are learned, the word sense disambiguation mechanism, and the proposed regularization term in masked language modeling. Section 5 introduces the experimental setup and the datasets applied in the phase of training of the models, both linguistic and deep graph neural networks. In addition, the two probing tasks used to evaluate the impact of the regularization term on the BERT model are analyzed. Finally, Section 6 presents the comparison between the baseline and the proposed regularized model on the obtained results performing the probing tasks.

2 Background

In this section are introduced all the main concepts underlying the regularization of language models through deep graph network embeddings. Some basics of graph theory, the definition of graph and knowledge graph are given. Furthermore, the methodologies behind deep learning for graphs and deep graph neural networks are described. The section continues with the description of the Transformers architecture, focusing on attention mechanism. Finally, the inputs and outputs structure of language models are presented with a summary of Masked Language Modeling.

2.1 Graphs

A graph is a structure defined as a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of entities called *vertices* or *nodes*, and $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ is a set of pairs representing the relations between elements of $\mathcal{V}$. A pair $(u, v) \in \mathcal{E}$, with $u, v \in \mathcal{V}$ is called *edge*. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a *directed graph* when pairs of nodes are *ordered*, i.e $\mathcal{E} \subseteq \{(u, v) \mid u, v \in \mathcal{V}\}$ (Bondy and Murty [20]).

In the rest of this thesis, each graph will be considered to be *directed*. It is possible to enrich a graph $\mathcal{G}$ with additional sets of *feature vectors* $\mathcal{X}$ and $\mathcal{A}$: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{A})$. Every node $u \in \mathcal{V}$ has a corresponding feature vector $\mathbf{x}_u \in \mathcal{X}$, while every edge $(u, v) \in \mathcal{E}$ is associated to a feature vector $\mathbf{a}_{u,v} \in \mathcal{A}$. Assuming $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{A} \subseteq \mathbb{R}^{d'}$, the two terms $d, d' \in \mathbb{N}$ are the number of node and edge features, respectively. Assign at each element of a graph a feature vector is beneficial for learning algorithms. Feature vectors model the features characterizing the nodes in a graph through low-dimensional space representations, allowing to approach learning on structured data as any problem in the field of representation learning.

In order to describe the algorithms used by Deep Graph Networks (Section 2.3) to learn a representation of nodes and edges that compose a *directed graph* through feature vectors, it is useful to define the concepts of *walk*, *path* and *cycle*. One of the main contributions introduced in Deep Learning for Graph is the *cycle* handling in representational learning, an issue described more specifically in Section 2.3.2.

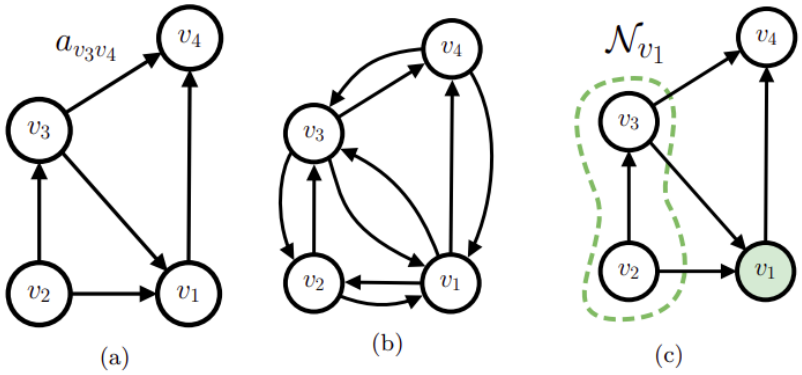


Figure 1: (a) Directed acyclic graph. (b) Directed graph with pairs of opposite oriented edges that model a undirected graph. Every undirected graph can be transformed in a directed one. (c) Open neighborhood of node $v1$. *Image credits*: Bacciu et al. [21]

Definition 2.1 *Walk*
*Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, a **walk** W is a finite non-null sequence $W = v_0e_1v_1e_2v_2 \dots e_kv_k$, whose terms are alternately vertices $v_i \in \mathcal{V}$ and edges $e_i \in \mathcal{E}$, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i. The vertices v_0 and v_k are called the origin and terminus of W.*

Definition 2.2 *Path*
*Let $W = v_0, e_1, v_1, e_2, v_2 \dots e_k, v_k$ be a walk. W is a **path** if and only if its vertices $v_0, v_1 \dots v_k \in W$ are distinct.*

Definition 2.3 *Cycle*
*Let $W = v_0, e_1, v_1, e_2, v_2 \dots e_k, v_k$ be a path. W is a **cycle** if and only the origin and the terminus are the same vertex v_0, i.e $W = v_0, e_1, v_1, e_2, v_2 \dots e_0, v_0$*

Given a graph $\mathcal{G}$, if there exists a (non-empty) path from a node u_i to itself with no other repeated nodes, the graph has a *cycle*. Structures that don't present *cycle* are called *acyclic*.

Definition 2.4 *Directed acyclic graph*
Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a (directed) graph. $\mathcal{G}$ is a Directed Acyclic Graph (DAG) if and only if it does not contains cycles.

For *DAGs* a topological order is always defined. (Figure 1(a)). Given a *DAG* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{A})$, the skeleton of $\mathcal{G}$, $skel(\mathcal{G})$, is the topological structure of the graph ignoring the sets of feature vectors $\mathcal{X}$ and $\mathcal{A}$. It is possible to define the *structured domain* for an acyclic directed structure $\mathcal{G}$ as $\mathbf{G}^{\#^{(i,o)}}$, where $\mathbf{G}$ is the local universe domain and and $\#^{(i,o)}$ is the skeleton where all vertex has a number of i incoming edges (*in-degree*) and o outgoing edges (*out-degree*). One of the earliest techniques for generating graph representations with neural networks, involves structured domains (Frasconi et al. [22], see also 2.3). To give an example, the structured domain of a binary tree is $\mathbf{B}^{\#^{(1,2)}}$.
Unlike acyclic structures, graphs with cycles model more complex relationships between entities, increasing expressiveness on structured data.
Each entity in a graph is characterized by a set of features but also by the connected entities for which there is an directed edge. Two connected vertex are named *adjacent*.

Definition 2.5 *Node adjacency*
*Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, two nodes $u, v \in \mathcal{V}$ are defined **adjacent** if exists a directed edge $(u, v) \in \mathcal{E}$.*

Definition 2.6 *Neighborhood*
*Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph. Given a node $v \in \mathcal{V}$, the **neighborhood** of v, $\mathcal{N}_v$, is the sub-graph induced by all adjacent nodes to v and their edges:*

$$\mathcal{N}_v = \{u \in \mathcal{V} | (u, v) \in \mathcal{E}\} \tag{2.1}$$

When the neighborhood $\mathcal{N}_v$ includes the node v is called *closed*, on the contrary is called *open* (Figure 1(c)). If the domain of edge feature vectors $\mathcal{A}$ is discrete and finite, i.e., $\mathcal{A} = \{c_1, \ldots, c_m\}$, we define the subset of neighbors of v with arc label c_k as:

$$\mathcal{N}_v^{c_k} = \{u \in \mathcal{N}_v \mid \mathbf{a}_{uv} = c_k\} \tag{2.2}$$

where $\mathbf{a}_{uv}$ is the feature vector of the edge $(u, v) \in \mathcal{E}$. The arc label c_k identify a k-type relation between the node v and $u \in \mathcal{N}_v$.

2.2 Knowledge Graph

The term was coined as early as 1972 by Schneider [23], but the modern idea of *Knowledge Graph* (KG) was introduced by Google in 2012 (Singhal [24]) and is intended for any graph-based knowledge base. KGs represent networks of real-world entities, represented by the nodes, and illustrates the relationship between them, represented by the edges. Knowledge graphs have emerged as one of the best functional abstractions for organizing knowledge across large-scale, diverse and dynamic data collections. Some examples of well known KGs are WordNet (Fellbaum [6]), YAGO (Suchanek and Weikum [11]), DBpedia (Auer et al. [10]), Wikidata (Matthew [9]), Freebase (Bordes et al. [7]).
The main focus of this thesis is to exploit the symbolic information represented by knowledge graphs to enrich language models. It is therefore appropriate to introduce the concepts underlying knowledge graphs. In Färber [5], a *knowledge graph* is define as an RDF graphs.

The Resource Description Framework (RDF, Cyganiak et al. [25]) is a framework for representing information in the Web proposed by the World Wide Web Consortium (W3C). According to Cyganiak et al. [25], the RDF framework is composed of three parts:

1. **IRIs**: Internationalized Resource Identifier (IRI), an internet protocol standard which builds on the Uniform Resource Identifier (URI). An IRI can be defined as unique sequence of characters that identifies a logical or physical resource used by web technologies.

2. **Literals**: constant values like strings, number, or date assigne to a costant variables or properties of an object or entity. (LIT).

3. **Blank nodes**: arbitrary elements, as long as they differ from IRIs and literals and have no internal structure (BN). Blank nodes identify anonymous entities for which an IRI is not defined.

The elements in the above listed sets compose RDF triples, a set of three entities that codifies a statement about semantic data.

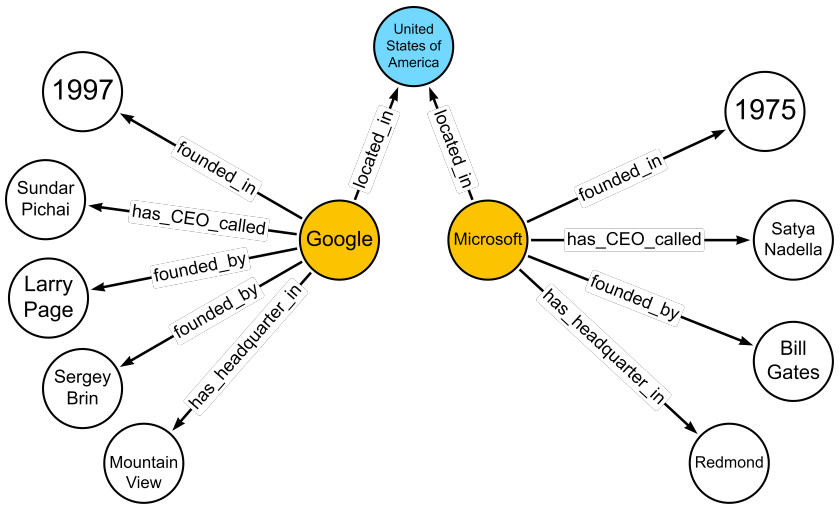


Figure 2: The RDF graph representing Google and Microsoft. Colored nodes are IRIs, white nodes are literals.

Definition 2.7 *RDF Triple*
RDF triple is a tuple $< e_i, r_k, e_j >$ *where:*

1. ***Subject*** e_i***:*** $e_i \in IRI \cup BN$
2. ***Predicate*** r_k***:*** $r_k \in IRI$
3. ***Object*** e_j***:*** $e_j \in IRI \cup BN \cup LIT$

In Figure 2, the fact "Google was founded by Sergey Brin and Larry Page in 1997" is defined by 3 RDF triple where *Google* is the subject, *founded by* and *founded in* are predicates and *Sergey Brin*, Larry Page and *1997* are the objects. In practice, RDF triples are useful for modeling facts and structured semantic data. IRIs identify entities, which defines subjects and objects of any semantic relationship with other entities. Furthermore, RDF triples can be used to model abstract concepts defined over real-word entities, such as language. Subject, predicate, and object are the three different components when breaking down a sentence. A set of *RDF triples* define an *RDF graph*. The set of nodes of an RDF graph is the set of subjects and objects of triples in the graph. The set of edges is the set of predicates. In general, RDF triples and RDF graph model semantic relationships between real or abstract concepts.

A widely known application for RDF that originated from semantic networks is in capturing ontologies. An *ontology* is formal specification of the conceptualization of a domain (Chaudhri et al. [26]). Ontologies play important roles in information exchange and in capturing the background knowledge of a domain that could be used for reasoning and answering questions. The concepts that belong to the domain related to an ontology can be modeled as a *knowledge graph*.

Definition 2.8 *Knowledge Graph*
A knowledge graph (KG) is an RDF graph, representing an ontology with a domain $\mathcal{D}$. Every KG is composed by triples $t^k_{i,j} = (e_i, r_k, e_j)$ where e_i and e_j are entities, $e_i, e_j \in \mathcal{D}$ and r_k is a $k-$type predicate that model the relationship k.

Not every *RDF graph* is a KG. A graph representation of data is useful, but it might be unnecessary to capture the semantic knowledge of the data. Let's take statistical data as an example: let $t^k_{i,j} = (e_i, r_k, e_j)$ a RDF triple. In Figure 2, it is not possible to consider the represented RDF graph as a KG, but only a graph with entities and related attributes.
Moreover, not every *knowledge base* is a *KG*. A key feature of a KG is that entity descriptions should be interlinked to one another. The definition of one entity includes another entity. For example, QA knowledge base do not represent a KG. It is possible to have an expert system that has a collection of data organized in a format that is not a graph but applied in automated processes.

The domain of a KG dependencies varies depending on the domain of the related ontology. For example, in WordNet (Fellbaum [6]) edges define semantic relationships between words of the english language (*synonyms, hypernyms, hyponyms, meronyms*), while Freebase (Bordes et al. [7]) model semantic relationships between more complex real-world entities and concepts (*location, author, kinship, influence*. An example of knowledge graph with the semantic relationships over cited is brought back in Figure 3.

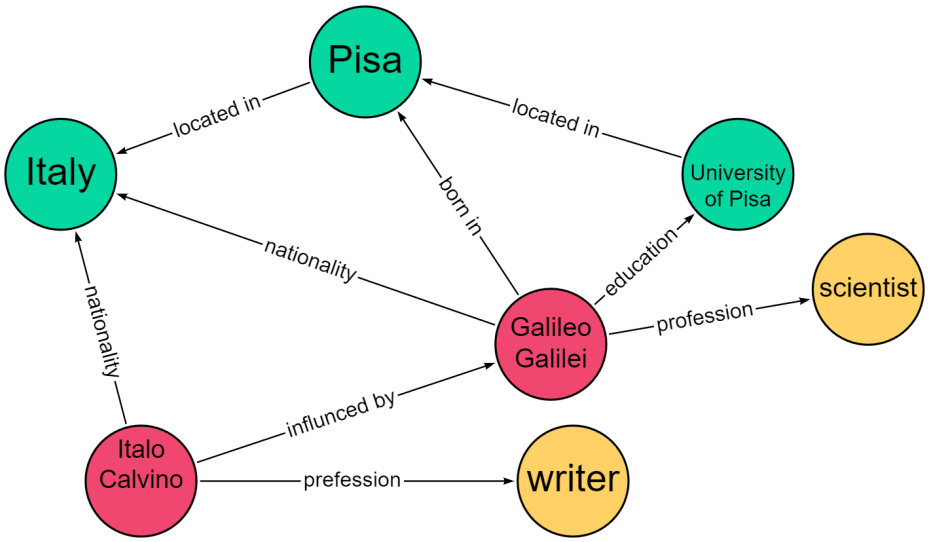


Figure 3: Example of Knowledge Graph modeling the semantic relationships between real word entities. Green nodes are locations, red nodes are people and yellow nodes are concepts. This simple KG view already contains several cycles.

2.3 Deep Learning for Graphs

2.3.1 Early Recursive Approaches

Although there has been a steady increase in publications related to automatic graph processing in recent years (Zhang et al. [27], Figure 2), neural networks and deep learning for graphs have a long history dating back to before the 2000s. The very first applications of neural networks on structured data were Recursive Neural Networks (**RecNNs**) on *DAGs* (Sperduti and Starita [28], Frasconi et al. [22]). The main issue of RecNN is the cycle handling. Sperduti and Starita [28] introduce a neural network composed by *generalized recursive neurons*, which is essentially a generalization to structures of a recurrent neuron. A single recursive neuron defines the representation of directed acyclic graphs in a recursive fashion. The whole structure is computed in the output node $O(v_s)$, where v_s is the *origin* vertex of the DAG. Given a vertex v, with the related feature vector $\mathbf{x}_v$, the output of the

i-th recursive neuron in ℓ-th layer for the vertex v is computed as follow:

$$O_\ell^i(v) = f\left(\mathbf{z}^T \mathbf{w}\mathbf{x}_v + \sum_{j=1}^{V} \omega_{ju} O\left(\{u \mid u \in \mathcal{N}_v\}\right)\right) \tag{2.3}$$

where $\mathbf{z} = O_{\ell-1}(v)$ is the output vector of recursive neurons in the layer $\ell - 1$ with the related weights vector $\mathbf{w}$, V is the the number of neurons in the ℓ-th layer and ω_{ju} is defined as the *recursive weight* of the j-th neuron in the ℓ-th layer associated with neighbor u. f is a sigmoidal function.
Due to the recursive nature of Equation 2.3, it follows that the neural representation for an acyclic graph is computed by a feedforward network obtained by replicating the same generalized recursive neuron and connecting these copies according to the topology of the structure. For cyclic structure this process lead to a non-converging loop. Cycles generate dependencies between the variables in the recursive units of RecNNs.
The same issue is highlighted by Frasconi et al. [22] Authors define the process of computing a neural representation as *isomorphic transduction* between two structured domain (2.1).

Definition 2.9 *Isomorphic Transduction*
A transduction $\tau(\cdot)$ is isomorph if:

$$\text{skel}(\tau(\mathcal{G})) = \text{skel}(\mathcal{G}) \quad \forall \mathcal{G} \in \boldsymbol{G}^{\#} \tag{2.4}$$

An isomorphic transduction $\tau(\cdot)$ from a structured domain $\mathbf{G}_1^{\#^{(i,o)}}$ to another $\mathbf{G}_2^{\#^{(i,o)}}$ can be naturally implemented by recursive neural networks defined by Sperduti and Starita [28]. Therefore, for isomorphic transductions the problem of cycles handling given by the recursive nature re-emerges.

2.3.2 Handling Cycles with Deep Graph Networks

In graph representation learning, Deep Graph Networks (**DGNs**, [29], Scarselli et al. [30], Gallicchio and Micheli [31], Li et al. [32], Kipf and Welling [33], Bacciu et al. [21]) are today the state-of-art for learning representations of elements that compose graphs and whole graphs as well. Social networks, chemistry, logistics, and automobiles have always had networks in common, but such complex networks usually contain several cyclic dependencies between the node-entities. DGNs, unlike RecNNs, solve the problem of cycle

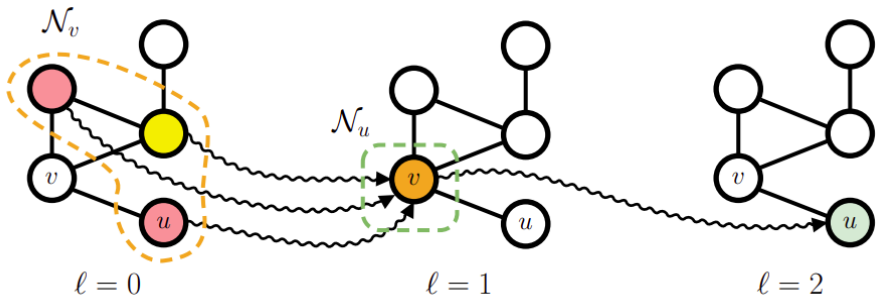


Figure 4: Message passing visualization. ℓ represents iteration (RecNNs) or network depth (GCNs). *Image credits* Bacciu et al. [21]

handling. Two initial solutions, Neural Network for Graph model (**NN4G**, Micheli [29]) and Graph Neural Network model (**GNN**, Scarselli et al. [30]) were proposed, setting the standard for all the modern architecture (Kipf and Welling [33], Velickovic et al. [17], Schlichtkrull et al. [34], Nathani et al. [35]). Both works define the update of a node representation $\mathbf{h}_v^{+1}$ in terms of neighbour representations at the previous step $\mathbf{h}_v^{\ell}$. In Micheli [29] the node state at step ℓ it represented by a layer of a deep architecture, while in Scarselli et al. [30] ℓ refers to the iteration step applying an iterative approach. In Bacciu et al. [21] this approach is defined as *context diffusion*.

Let $c_\ell(v)$ be the *context* of node $v \in \mathcal{V}$, i.e., the set of all nodes that directly or indirectly contribute to the representation $\mathbf{h}_v^{\ell}$ of the node at the ℓ-th iteration. The most common applied technique to apply context diffusion is defining a *message passing* architecture composed by 3 main steps:

1. **Message computation**: for each node $v \in \mathcal{V}$, generate the node representation $\mathbf{h}_v^{\ell}$, the *message*, as a function of its previous state $\mathbf{h}_v^{\ell-1}$ and edges information.
2. **Message dispatching**: for each node $v \in \mathcal{V}$, compute the neighborhood $\mathcal{N}_v$ and send the message $\mathbf{h}_v^{\ell}$ to all the nodes $u \in \mathcal{N}_v$.
3. **State update**: for each node v, compute the new state $\mathbf{h}_v^{\ell+1}$ as a function of its current state $\mathbf{h}_v^{\ell}$ and *all incoming messages* for the node v.

In Micheli [29] the initial node state $\mathbf{h}_v^0$ is computed as follow:

$$\mathbf{h}_v^0 = \psi\left(\sum_{x_i \in \mathbf{x}_v} w_{0i} x_i\right) \tag{2.5}$$

where x_i is the i-th feature of the vector $\mathbf{x}_v$ and w_{0i} the associated weight in the initial neuron. Originally, ψ represents a linear or sigmoidal function, but in most of modern application, DGNs compute the initial state as a non-linear transformation of node feature vector $\mathbf{x}_v$ (Bacciu et al. [21]).
The *message passing* perform by DGNs is referred as *neighborhood aggregation*. Through message passing it is possible to define for each node v a state $\mathbf{h}_v^{\ell+1}$ as a function of the previous state of the node $\mathbf{h}_v^{\ell}$ and the state of the neighborhood $\mathbf{h}_{\mathcal{N}_v}^{\ell+1}$:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^{\ell}, \mathbf{h}_{\mathcal{N}_v}^{\ell+1}\right) \tag{2.6}$$

DGNs compute the neighborhood aggregation $\mathbf{h}_{\mathcal{N}_v}^{\ell+1}$ as a function Ψ, defined as *neighborhood aggregation function*:

$$\mathbf{h}_{\mathcal{N}_v}^{\ell+1} = \Psi\left(\left\{\psi^{\ell+1}\left(\mathbf{h}_u^{\ell}\right) \mid u \in \mathcal{N}_v\right\}\right) \tag{2.7}$$

Given Equations 2.6 and 2.7, DGNs general formula to compute the state of the node v at layer $\ell + 1$:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^{\ell}, \Psi\left(\left\{\psi^{\ell+1}\left(\mathbf{h}_u^{\ell}\right) \mid u \in \mathcal{N}_v\right\}\right)\right) \tag{2.8}$$

where ϕ and ψ are continuous mapping functions that implement arbitrary transformations of the input data. Equation 2.8 proposed in Micheli [29] and Scarselli et al. [30] has given rise to a series of modern architectures, which today represent the state of the art in representation learning: Graph Convolutional Networks (**GCN**, Kipf and Welling [33]). For the purpose of this work, it is useful to analyze different neighborhood aggregation approaches applied in the proposed framework.
An extension of Equation 2.8 proposed by Velickovic et al. [17] introduces *attention mechanisms* with Graph Attention Network (**GAT**), described in section (3.1.2).

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^{\ell}, \Psi\left(\left\{\alpha_{uv}^{\ell+1} * \psi^{\ell+1}\left(\mathbf{h}_u^{\ell}\right) \mid u \in \mathcal{N}_v\right\}\right)\right) \tag{2.9}$$

where $\alpha_{uv}^{\ell+1}$ is the *attention coefficient* (Velickovic et al. [17], Bacciu et al.

[21]). However, it can be seen that in all of the above equations the relationships between nodes are not considered for node state computation. In many practical applications, the arc labels assigned to the edges of the graph contain key information to make the most of the structured information: in protein structures, know what relationship exists between the different amino acids that compose them, as in the same way, in a social network the interactions between individuals, are the most significant information (Fout et al. [36], Wu et al. [37]). For this same work, the relationships between the words (nodes) of a corpus are crucial to generate meaningful node representation. Handling edges was originally introduced already in NN4G (Micheli [29]) and later applied to graph convolutional layers by Schlichtkrull et al. [34], with the Relational Graph Convolutional Network model (**R-GCN**). Taking into account *arc labels* in Equation 2.8:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^{\ell}, \sum_{c_k \in \mathcal{A}} \left(\Psi\left(\left\{\psi^{\ell+1}\left(\mathbf{h}_u^{\ell}\right) \mid u \in \mathcal{N}_v^{c_k}\right\}\right) * w_{c_k}\right)\right) \tag{2.10}$$

where w_{c_k} is a learnable scalar parameter that weighs the contribution of arcs with label $\mathbf{a}_{uv} = c_k$, and $*$ multiplies every component of its first argument by w_{c_k}.
Introducing the edge feature vectors in Equation 2.10, a more general solution can be defined as follows:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^{\ell}, \Psi\left(\left\{e^{\ell+1}\left(\mathbf{a}_{uv}\right)^T \psi^{\ell+1}\left(\mathbf{h}_u^{\ell}\right) \mid u \in \mathcal{N}_v\right\}\right)\right) \tag{2.11}$$

where $e^{\ell+1}$ can be any function. In this way is possible to assign a weight to the state of each neighbors $u \in \mathcal{N}_v$ as a function of the feature vector $\mathbf{a_{uv}}$ related to the edge $(u, v) \in \mathcal{E}$.
Nathani et al. [35] propose an attention based feature embedding that captures both entity and relation features in any given entity's neighborhood, introducing edge feature vectors (Equation 2.11) to the attention based approach (Equation 2.9):

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^{\ell}, \Psi\left(\left\{\alpha_{uv}^{\ell+1} * \psi^{\ell+1}\left(\mathbf{h}_u^{\ell}, \mathbf{a}_{uv}\right) \mid u \in \mathcal{N}_v\right\}\right)\right) \tag{2.12}$$

where the *attention score* α_{uv} defined in KBAT (Equation 2.12) is an extension of the GAT score (Equation 2.9). Attention scores and **KBAT** model proposed by Nathani et al. [35] is discussed in 3.1.2 as it has been used in practice by the framework proposed in this thesis.

2.3.3 Topological Ordering Issue in Neighborhood Aggregation

As pointed out in the previous section, DGNs solve the problem of cycles, but must handle another additional issue: *topological neighborhood ordering*, i.e. in which order to aggregate neighborhood nodes. Recalling Equation 2.7, the *neighborhood aggregation function* Ψ takes as inputs the neighbour representations $\mathbf{h}_u$. If the output of the aggregation function depends on the order of the input elements, without an ordering of the neighborhood nodes $u \in \mathcal{N}_v$, Ψ could generate different representation $\mathbf{h}_{\mathcal{N}v}$ for the same neighborhood $\mathcal{N}_v$. This issue was solved in literature defining Ψ as an *invariant permutation function*, a function whose output is independent of the order of the input elements (e.g. sum, average). The mathematical definition of invariant permutation function is given in E. Wagstaff and Osborne [38] as follow:

Definition 2.10 *Invariant permutation function*
A function $\Psi(\mathbf{x})$ *is permutation-invariant if* $(x_1, \ldots, x_M) = f\left(x_{\pi(1)}, \ldots, x_{\pi(M)}\right)$ *for all permutation* π*.*

A function Ψ defined on sets of size M is *sum-decomposable* via Z if there are functions $\sigma : \mathcal{X} \to Z$ and $\rho : Z \to \mathcal{Y}$ such that:

$$\Psi(X) = \rho\left(\sum_{x \in X} \sigma(x)\right) \tag{2.13}$$

where $\mathcal{X}$ is the feature space of the input element x and $\mathcal{Y}$ it is the output space. We refer to Z here as the latent space. Since summation is permutation-invariant, a sum-decomposition is also permutation-invariant.

In the context of neighborhood aggregation, Equation 2.13 can be interpreted as follows:

$$\Psi(\{\mathbf{h}_u \mid u \in \mathcal{N}_v\}) = \rho\left(\sum_{u \in \mathcal{N}_v} \sigma(\mathbf{h}_u)\right) \tag{2.14}$$

To summarize, the major contribution introduced by DGNs is to treat graphs as a topological-order independent composition of the vertices and edges, applying two main approaches: compute the node representation $\mathbf{h}^{\ell}$ as a function of the previous representation $\mathbf{h}^{\ell-1}$ and handle neighborhood aggregation through the application of invariant permutation functions. These architectures thus define a level of granularity useful for the proposed framework,

being able to interact at a node-level on the processed graphs. In practice, within language models, node representations correspond to representations of individual words. This allows complete word-level control during the language model training. From now on, this thesis will refer to high-dimensional representations of nodes in a graph by the term *node-embeddings*.

2.4 BERT

Bidirectional Encoder Representation from Transformer (**BERT**) introduced by Devlin et al. [16], is a deep contextual language representation model. It is designed to pre-train deep bidirectional representations of words from unlabeled text by jointly conditioning on both the left and right contexts in all its layers. Attention mechanisms adopted in the model are described below. In Section 2.4.2 are described Transformers (Vaswani et al. [13]) and BERT architecture, the input structure and the learning parameters involved during BERT training. Finally, the *masked language modeling* (MLM) setup is presented, the BERT training task in which the regularization of the framework proposed was applied.

2.4.1 Self-Attention and Multi-Head attention

Attention mechanisms (Bahdanau et al. [14]) are used to generate representations that incorporate long-term dependencies among the element in a sequence. An attention function is defined as mapping a query and a set of key-value pairs to an output. Given a sequence of N word, for each word representation $\mathbf{x}_i \in \mathbb{R}^{d_{\text{model}}}$, , formally named *word embedding*, *self-attention* define a contextualized vector $\mathbf{z}_i$ according to the following steps:

1. Compute a query vector $\mathbf{q}_i$, a key vector $\mathbf{k}_i$ both of dimension d_k and a value vector $\mathbf{v}_i$ with dimension and d_v. All these vector are computed by multiplying the input term $\mathbf{x}_i$ with learnable matrices $\mathbf{W}^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ respectively (Figure 5):

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q \tag{2.15}$$

$$\mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K \tag{2.16}$$

$$\mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V \tag{2.17}$$

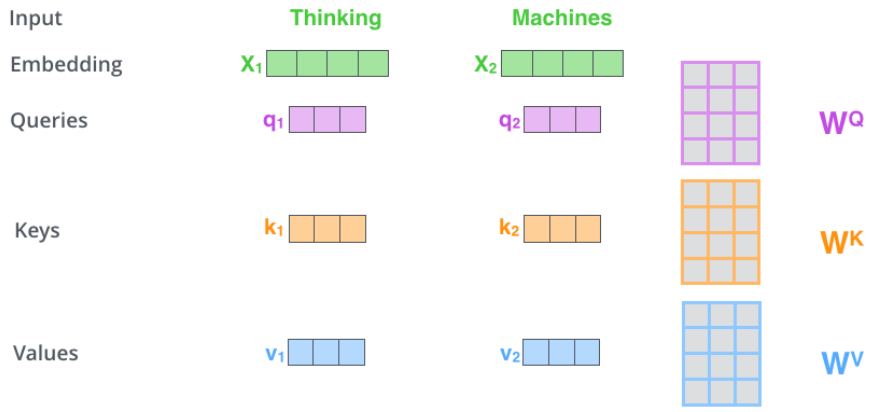


Figure 5: Visualization of queries, keys and values vectors for words *"Thinking"* and *"Machines"* represented by word embeddings $\mathbf{x}_1$, $\mathbf{x}_1$ and related learnable matrix parameters. *Image credits*: Alammar [39]

2. To contextualize the i-th term with the other elements in the sequence, compute the score as the dot-product between the query $\mathbf{q}_i$ vector related to the input term with the key vectors $\mathbf{k}_j$ of all the remaining elements:

$$s_{ij} = \mathbf{q}_i \mathbf{k}_j \quad \forall j \in [1, \ldots, N] \tag{2.18}$$

3. The score s_{ij} is divide by the square root of the key vector dimension d_k:

$$s'_{ij} = \frac{s_{ij}}{\sqrt{d_k}} \quad \forall j \in [1, \ldots, N] \tag{2.19}$$

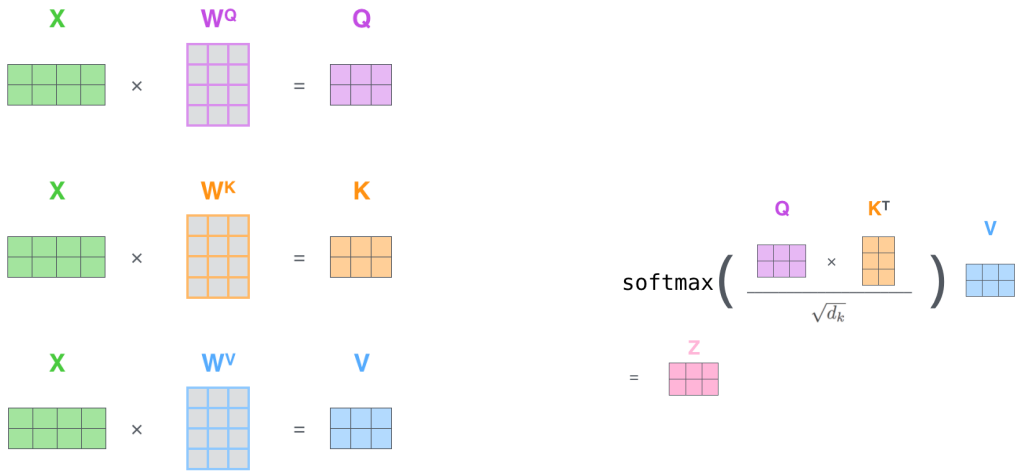
4. Apply the *softmax* function to normalize the score s'_{ij}.

$$s''_{ij} = \frac{e^{s'_{ij}}}{\sum_{j=1}^{N} e^{s'_{ij}}} \quad \forall j \in [1, \ldots, N] \tag{2.20}$$

   For large values of d_k applying softmax to score without the division applied in step 3, the gradient vanishing problem may occur (Vaswani et al. [13]).

5. Each value vector $\boldsymbol{v}_j$ of the elements in the sequence is multiplied by the respective score s''_{ij}:

$$\boldsymbol{v}'_{ij} = s''_{ij} \boldsymbol{v}_j \quad \forall j \in [1, \ldots, N] \tag{2.21}$$



(a) (b)

Figure 6: (*a*) Dot-product between the set of word embeddings X and the learnable parameters. W^K, W^Q and W^V refer to the learnable parameters reported in Equation 2.23. (*b*) Visualization of matrices in Equation 2.24.

6. The weighted values vector are aggregated in order to compute the self-attention output vector for the input term $\mathbf{x}_i$:

$$\mathbf{z}_i = \sum_{j=1}^{N} \boldsymbol{v}'_{ij} \tag{2.22}$$

This process is executed in parallel for all N elements in input sequence, packing the the three vectors computed for each word-embedding into matrices $\mathbf{Q} \in \mathbb{R}^{N \times d_k}$, $\mathbf{K} \in \mathbb{R}^{N \times d_k}$ and $\mathbf{V} \in \mathbb{R}^{N \times d_v}$ (Figure **??**). The input sequence can be represented by matrix $\mathbf{X} \in \mathbb{R}^{N \times d_{\text{model}}}$ and the resulting attention output by matrix $\mathbf{Z} \in \mathbb{R}^{N \times d_{\text{model}}}$. The attention function for $\mathbf{X}$ as follow:

$$\mathbf{Z} = \text{Attention}(\mathbf{X}\mathbf{W}^{\mathbf{Q}}, \mathbf{X}\mathbf{W}^{\mathbf{K}}, \mathbf{X}\mathbf{W}^{\mathbf{V}}) = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \tag{2.23}$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.24}$$

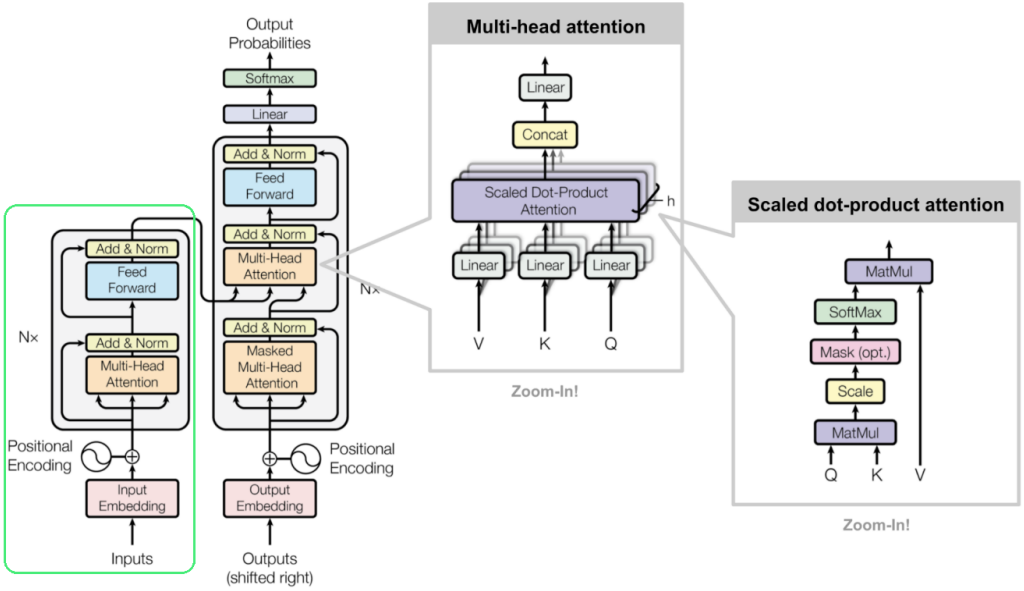


Figure 7: Transformers architecture and detail of attention mechanism. *Image credits*: Rameez [40]

where the term in the softmax function represents the *attention scores matrix* $\mathbf{A} \in \mathbb{R}^{N \times N}$:

$$\mathbf{A} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \tag{2.25}$$

The described mechanism is called Scaled Dot-Product Attention (Figure 7, right). In order to allows the model to jointly attend to information from different representation sub-spaces at different positions, [13] introduce h Scaled Dot-Product layers, defining a Multi-Head attention mechanism (Figure 7, center), defined as follow:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{ Concat } (\text{ head }_1, \ldots, \text{ head }_\text{h})\, \mathbf{W}^O \tag{2.26}$$

$$head_\text{i} = \text{ Attention } \left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right) \tag{2.27}$$

where $\mathbf{W}_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$, $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_\text{model}}$ are matrices of learnable parameters.

2.4.2 Architecture Overview

After its introduction in Vaswani et al. [13], in last years *Transformer architecture* has achieved impressive results in many tasks with different domains, starting from language modeling (Devlin et al. [16], Brown et al. [41]) up to finance, image processing and medical applications (Ramos-Pérez et al. [42], Chen et al. [43], Rasmy et al. [44]).
As early as Recurrent Neural Networks (**RNNs**, Jordan [45]), a common feature of all sequence transduction models is the *encoder-decoder* architecture. The encoder maps an input sequence $\mathbf{x}_i = (x_i)_{i=1}^n$ of symbol representations to a sequence of continuous representations $\mathbf{z}_i = (z_i)_{i=1}^n$. Given $\mathbf{z}_i$, the decoder then generates an output sequence $\mathbf{y}_i = (y_i)_{i=1}^m$ of symbols one element at a time. At each step the model consumes the previously generated symbols as additional input when generating the next. *Transformers* are composed by a stack of *encoder* and *decoder* layers. The two building blocks are described below:

• **Encoder**: structured in two sub-layer. The first is a multi-head self-attention mechanism, which compute an enriched contextualized word embedding 2.4.1. The second is a simple, position-wise fully connected feed-forward network FFN:

$$\text{FFN}(\mathbf{z}_i) = \max\left(0, \mathbf{z}_i\mathbf{W}_1 + \mathbf{b}_1\right)\mathbf{W}_2 + \mathbf{b}_2 \tag{2.28}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}, \mathbf{b}_1 \in \mathbb{R}^{d_{\text{ff}}}, \mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}, \mathbf{b}_2 \in \mathbb{R}^{d_{\text{model}}}$ are two linear transformation and d_{ff} represents the number of hidden units. The encoder applies a residual connection (He et al. [46]) on the two sub-layers, followed by layer normalization (Ba et al. [47]):

$$LayerNorm(\mathbf{x}_i + Sublayer(\mathbf{x}_i)) \tag{2.29}$$

where $Sublayer(\mathbf{x}))$ is the function implemented by the sub-layer itself, described by Equation 2.22 and Equation 2.28 respectively.

• **Decoder**: in addition to the two Encoder's sub-layers, introduces a third sub-layer, which performs multi-head attention over the output of the encoder stack. This third sub-layer is called masked multi-head self attention. The masking is performed by transforming the matrix of attention scores $\mathbf{A}$ (defined in Equation 2.25) used in the sub-layer as follow:

$$\forall s_{ij} \in \mathbf{A}, i < j < N \quad s_{ij} = -\infty \tag{2.30}$$

Applying the softmax function to the transformed matrix A, the resulting scores matrix is then *lower triangular*. For example, with $N = 3$:

$$A = \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix} \rightarrow \begin{pmatrix} s_{11} & -\infty & -\infty \\ s_{21} & s_{22} & -\infty \\ s_{31} & s_{32} & s_{33} \end{pmatrix}$$

$$softmax \begin{pmatrix} s_{11} & -\infty & -\infty \\ s_{21} & s_{22} & -\infty \\ s_{31} & s_{32} & s_{33} \end{pmatrix} = \begin{pmatrix} s''_{11} & 0 & 0 \\ s''_{21} & s''_{22} & 0 \\ s''_{31} & s''_{32} & s''_{33} \end{pmatrix}$$

The resulting triangular matrix is the multiplied by the values matrix $\mathbf{V}$. For the i-th term, zero scores have the effect of masking the i+1 terms and only considering the terms preceding i-th row:

$$\begin{pmatrix} s''_{11} & 0 & 0 \\ s''_{21} & s''_{22} & 0 \\ s''_{31} & s''_{32} & s''_{33} \end{pmatrix} \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix} =$$

$$\begin{pmatrix} s''_{11}v_{11} & s''_{11}v_{12} & s''_{11}v_{13} \\ s''_{21}v_{11} + s''_{22}v_{21} & s''_{21}v_{12} + s''_{22}v_{22} & s''_{21}v_{13} + s''_{22}v_{23} \\ s''_{31}v_{11} + s''_{32}v_{21} + s''_{33}v_{31} & s''_{31}v_{12} + s''_{22}v_{22} + s''_{33}v_{32} & s''_{31}v_{13} + s''_{32}v_{23} + s''_{33}v_{33} \end{pmatrix}$$

BERT consists only in a stack of Encoder building blocks (Figure 7, highlighted in green).

2.4.3 Input Representation

Every input sequence is constructed as the sum of 3 vectors (Figure 8), composed by embeddings collected as lookup tables:

1. *Token embeddings*: convert the input sentence into a sequence of tokens. First step of tokenization process is performed through WordPiece Embedding described in Wu et al. [48], basically defining a lookup table that stores embeddings of a fixed size dictionary. In addition, two special tokens are appended at the beginning `[CLS]` and end `[SEP]` of the input sequence. Each tokens is finally translated into a unique integer id.

2. Segment embeddings: in case of multiple input sequences, a segment embedding is associated with each input id to indicate the sequence

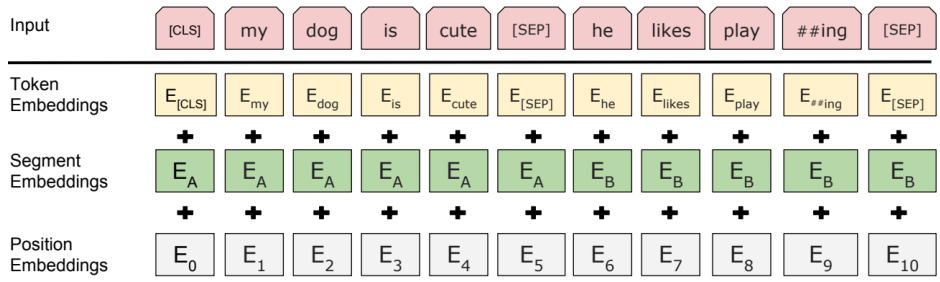


Figure 8: Sequence input preprocessing. The input of the encoders stack is the summation of token, segment and positional embeddings . Image credits: Alammar [39]

to which it belongs. At each step, BERT handles at most two input sequence which means two embedding type in the related lookup table.

3. Positional embeddings: BERT model structures the input according to the order of the tokens in the sequence through *positional embeddings*, computed as sine and cosine function of token position and dimension d_{model}.

The resulting sum of these 3 vectors is a tensor with 512 word-embeddings of size d_{model}. If the input sequence consists of less than 512 tokens, as many special tokens `[PAD]` are added to reach the fixed length. In practice, the aggregation of the the embeddings vectors is performed by a single *input embedding layer* of BERT's architecture.

2.4.4 Masked Language Modeling

BERT is trained following two training tasks: Masked Language Modeling (**MLM**) and Next Sentence Prediction (**NSP**). The only task considered by the proposed framework is MLM, since the word-embedding regularization is not executable for a sentence-level task like NSP. The MLM task is outlined in Figure 9 Given an input sequence of tokens $\mathbf{x}_i \in \mathbf{X}$, MLM consists into masking k input tokens embedding randomly, replacing them with another special token `[MASK]`.
Let $\Pi = (\pi_i)_{i=1}^{K}$ denote the indexes of the masked tokens in the sentence $\mathbf{X}$, where k is the number of masked tokens. Let $\mathbf{X}_\Pi$ denote the set of masked tokens in $\mathbf{X}_\Pi$, and $\mathbf{X}_{-\Pi}$ denote the set of observed (unmasked) tokens.

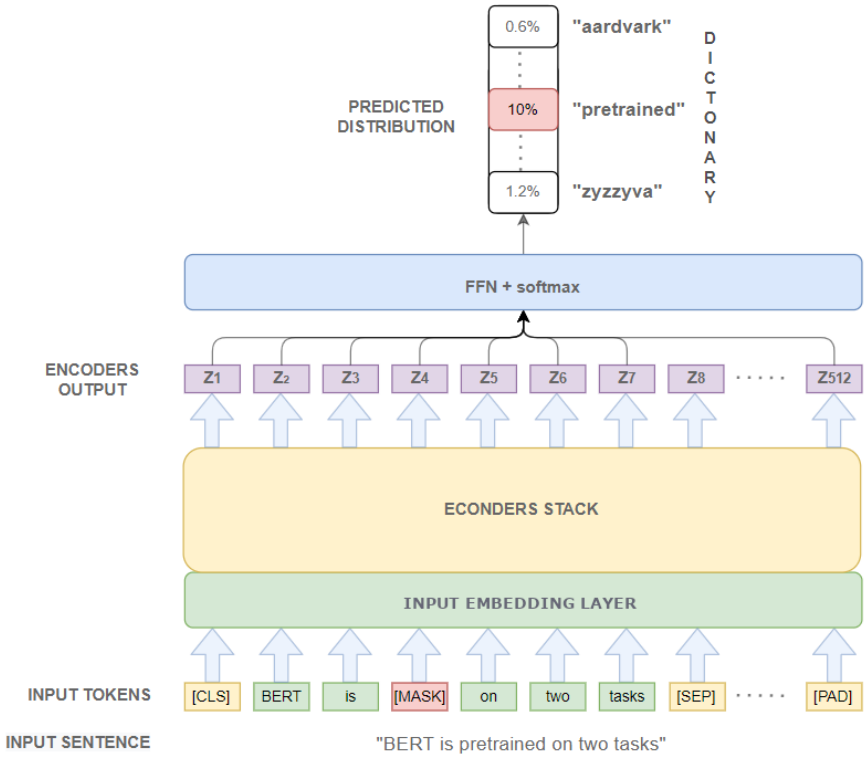


Figure 9: MLM schema for sentence *"BERT is pre-trained on two tasks"*.The token with index 3 is replaced with the special token [MASK], then the input tokens are fed into input embedding layer to compose the encoder's input described in Section 2.4.3. The output is a vector of D probabilities, where D is the dictionary dimension.

The objective of MLM is the log-likelihood:

$$\mathcal{L}_{\text{MLM}}\left(\mathbf{X}_{\Pi} \mid \mathbf{X}_{-\Pi}, \theta\right) = \frac{1}{K} \sum_{k=1}^{K} \log p\left(\mathbf{x}_{\pi_k} \mid \mathbf{X}_{-\Pi}; \theta\right) \tag{2.31}$$

where θ are BERT learnable parameters. In other words, BERT's MLM task eventually learns the optimal parameters to predict the original masked input given the sequence of unmasked embeddings, also referred as cloze test (Taylor [49]). In practice, for each sentence the 15% of the input tokens are

masked with `[MASK]`, random tokens or *original tokens*: the latter two avoid masking the same word too often and misalign pre-training data compared to fine-tuning where masking is not present. MLM exploits word variance in different contexts, learning probability distribution over sequences of words. BERT and Transformers-based models are most widely used language models (LMs) in natural language processing, extensively analyzed model with several already optimized and robust implementations that allowed to focus the implementation of this thesis on the proposed framework.

3 Related Work

Before analyzing the framework, the following sections review work that has approached NLP tasks through the use of structured information contained in knowledge graphs applying different DGN models. In addition, existing works use node-embeddings in fine-tuning pre-trained models and address the parallelism between BERT word-embeddings and knowledge graph node-embeddings. Finally, works aimed at transfer information from different embeddings employing regression is reviewed.

3.1 DGNs and KGs in Language Modeling

DGNs are the state-of-the-art for learning the node-embeddings of a KG. To perform a regularization of a language model with symbolic knowledge, it is necessary that the information of the KG is represented through Knowledge Graph Embeddings. The following describes work that has defined some of the main concepts for learning node-embeddings and relation-embeddings in KGs. Next, the context diffusion mechanisms applied in KGs and the main models of DGNs are described and analyzed. Existing applications that combine DGNs and structured data in NLP tasks are presented in the last part of this section.

3.1.1 Knowledge Graph Embeddings

Formerly DGNs models, shallow knowledge graph encoders apply *translational scoring function* f_s in the objective to minimize in order to learn *Knowledge Graph Embeddings* (KGE). Given a KG triple (h, r, t), different models (Bordes et al. [50], Wang et al. [51], Lin et al. [52],) employ a transitional characteristic to model relationships between entities, in which it assumes that if (h, r, t) is a valid fact, the embedding of head entity h plus the embedding of relation r should be close to the embedding of tail entity t. The closeness, also referred as the *energy*, between two embeddings is defined by the *scoring function* f_s.
TransE (Bordes et al. [50]) has been one of the first shallow knowledge graph encoder to learn KGE. The model objective is to define entities and relationships embeddings of a KG in low structure vector spaces minimizing the

pairwise-margin objective function:

$$\mathcal{L} = \sum_{i=1}^{N} \left[\tau + f_s\left(h_i, r_i, t_i\right) - f_s\left(h'_i, r_i, t'_i\right)\right]_+ \tag{3.1}$$

where $[x]_+$is the hinge function $[x]_+ = \max(0, x)$ and scalar $\tau \in \mathbb{R}$ is a threshold (called margin), with (h_i, r_i, t_i) denoting a *positive triplet* and (h'_i, r_i, t'_i) denoting a *negative triplet*. *Positive triplets* are defined as the elements that compose the knowledge graph, i.e. a triple (h,r,t) such that $h, t \in \mathcal{V}_{KG}$ and $(h, t) \in \mathcal{E}_{KG}$, where $\mathcal{V}_{KG}$ is the set of the subjects and objects of the knowledge graph and $\mathcal{E}_{KG}$. the set of relationships. Conversely, a *negative triple* is a triple not contained in the knowledge graph. The set of negative triplets is constructed from the positive triplets and replacing one of the head and tail entities or the relationship between them without redefining a positive triplet. Assuming that vector representations of the triplet elements $\mathbf{v}_h, \mathbf{v}_r, \mathbf{v}_t$ coexist in the same latent space, the *score function* f_s is defined as follow:

$$f_s(h, r, t) = \|\mathbf{v}_h + \mathbf{v}_r - \mathbf{v}_t\| \tag{3.2}$$

i.e the $L1$ (or in other cases, $L2$-norm) between the triplet elements vector representations $\mathbf{v}$, implying that $\mathbf{v}_t$ should be the nearest neighbor of $\mathbf{v}_h + \mathbf{v}_r$. The objective function (Equation 3.1) requires score $f_s\left(h'_i, r_i, t'_i\right)$ to be greater than score $f_s\left(h_i, r_i, t_i\right)$ by at least τ. The pairwise-margin objective encourages the network to assign a high score to positive triples and penalize negative ones.
TransH (Wang et al. [51]) extends Equation 3.1 to handle $1-N$, $N-1$ and $N-N$ relations. As in the previous model, embedded entities and relations still coexist in the same space:

$$f_s\left(h, r, t\right) = \|\mathbf{v}_{h\perp} - \mathbf{v}_{t\perp} + \mathbf{v}_r\|_2^2 \tag{3.3}$$

where $\mathbf{v}_{h\perp}$ and $\mathbf{v}_{t\perp}$ are respectively projections of $\mathbf{v}_h$ and $\mathbf{v}_t$ on the hyperplane containing $\mathbf{v}_r$. Thus, given $\mathbf{w}_r$ a normal vector to the hyperplane of $\mathbf{v}_r$ such that $||\mathbf{w}_r||_2 = 1$, the above equation can be written as

$$f_s\left(h, r, t\right) = \left\|\left(\mathbf{v}_h - \mathbf{w}_r^T \mathbf{v}_h \mathbf{w}_r\right) - \left(\mathbf{v}_t - \mathbf{w}_r^T \mathbf{v}_t \mathbf{w}_r\right) + \mathbf{v}_r\right\|_2^2$$

Taking advantage of the projections, it is possible to model different roles that each entity plays in different relations. Further generalizations are included

in *TransR* model (Lin et al. [52]) where entities and relations representations are not elements of the same latent space: for each relation r, there exist a projection matrix $\mathbf{W}_r$ such that entities are projected into its specific relation space. Hence, the score function of TransR is defined as

$$f_s(h, r, t) = \|\mathbf{v}_h \mathbf{W}_r - \mathbf{v}_t \mathbf{W}_r + \mathbf{v}_r\|_2^2 \tag{3.4}$$

Convolutional Neural Network (CNN, Lecun et al. [53]) based models like **ConvE** (Dettmers et al. [54]) **ConvKB** (Nguyen et al. [55]) extend shallow knowledge graph encoders to generate richer and more expressive KGE. However, in contrast to context diffusion in DGN, treating triples independently fails to cover the complex and hidden information that is inherently implicit in the local neighborhood surrounding a triple

3.1.2 Context Diffusion in Knowledge Graphs

DGNs can represent KGs triplets as a set of contextualize feature vectors. Given a triplet (h, r, t), let $\mathcal{KG} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{A})$ be a knowledge graph with $h, t \in \mathcal{V}$, $(h, t) \in \mathcal{E}$. The KGE triplet $(\mathbf{h}_h, \mathbf{a}_{h,t}, \mathbf{h}_t)$ is composed by head and tail node representations $\mathbf{h}_h, \mathbf{h}_t \in \mathcal{X}$ and $\mathbf{a}_{h,t} \in \mathcal{A}$. From this perspective, it is possible to apply *context diffusion* described in 2.3 to KGs in order to learn KGE. In Bacciu et al. [21] are described 3 main architectures to perform *context diffusion*: *recurrent*, *feedforward* and *constructive* architectures.
Recurrent architecture employ an iterative process in order to construct the graph representation as the result of a dynamical system. Feedforward architectures are composed by multiple layers to compute the local context learned at each step, as in Convolutional Neural Networks (CNN, Lecun et al. [53]). For this reason, feedforward architectures are often referred to as Graph Convolutional Neural Networks (Kipf and Welling [33]). Constructive architecture (Micheli [29], Bacciu et al. [12]) is a special case of feedforward architecture in which training is perfomed layerwise, solving sub-problem in a *divide-et-impera* fashion in each layer. This section will not be deepened on constructive architecture as not useful for the proposed work.
The first *recurrent* architecture that introduced context diffusion was the Graph Neural Network model (**GNN**, Scarselli et al. [30]), using recurrent dynamics similar to RecNNs and introducing constraints on the supervised loss funcion to induce convergence. Soon after, Gallicchio and Micheli [31] proposed the Graph Echo State Network model (**GraphESN**), a recurrent

architecture that reach convergence through the contractivity of untrained reservoir dynamic. A more modern recurrent model, **GatedGNN** Li et al. [32], achieves convergence setting a priori number of iterations. All of these approaches allows cycles in the state computation within a contractive setting of the recurrent system.

Model	Neighborhood Aggregation $\mathbf{h}_v^{\ell+1}$
GNN [30]	$\sum_{u \in \mathcal{N}_v} MLP^{\ell+1}\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{a}_{uv}, \mathbf{h}_u^{\ell}\right)$ (a)
NN4G [29]	$\sigma\left(\mathbf{w}^{\ell+1^T}\mathbf{x}_v + \sum_{i=0}^{\ell}\sum_{c_k \in \mathcal{C}}\sum_{u \in \mathcal{N}_v^{c_k}} w_{c_k}^i * \mathbf{h}_u^i\right)$ (b)
GCN [33]	$\sigma\left(\mathbf{W}^{\ell+1}\sum_{u \in \mathcal{N}(v)} \mathbf{L}_{uv}\mathbf{h}_u^{\ell}\right)$ (c)
R-GCN [34]	$\sigma\left(\sum_{c_k \in \mathcal{C}}\sum_{u \in N_v^{c_k}} \frac{1}{\lvert\mathcal{N}_v^{2k}\rvert}\mathbf{W}_{c_k}^{\ell+1}\mathbf{h}_u^{\ell} + \mathbf{W}^{\ell+1}\mathbf{h}_v^{\ell}\right)$ (d)
GAT [17]	$\sigma\left(\sum_{u \in \mathcal{N}_v} \alpha_{uv}^{\ell+1} * \mathbf{W}^{\ell+1}\mathbf{h}_u^{\ell}\right)$ (e)
KBGAT [35]	$\sigma\left(\sum_{u \in \mathcal{N}_v} \beta_{uv}^{\ell+1} * \left[\mathbf{W}^{\ell+1}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell+1}\mathbf{h}_v^{\ell}, \mathbf{W}^{\ell+1}\mathbf{a}_{uv}^{\ell}\right]\right)$ (f)

Table 1: The neighborhood aggregation rules defined for some of the models analyzed. (c) $\mathbf{L}$ is the normalized graph Laplacian, (d) $[\cdot,\cdot]$ denotes concatenation, $\mathbf{W}^{\ell}$ are learneable parameters and c_k the arc labels. $(a)(f)$ $\mathbf{a}_{uv}$ is the edge feature vector s.t $\mathbf{a}_{uv} = c_k$. σ is a non-linear activation function such as the sigmoid. ℓ represents the iteration (recurrent architecture) or the or the model depth/layer (feedforward and constructive architectures).

To laverage the full KGs symbolic knowledge, DGN models compute entity representations $\mathbf{h}_v$ as a function of its neighborhood $\{\mathbf{h}_u \mid u \in \mathcal{N}_v\}$ and the corresponding edges feature vector $\mathbf{a}_{uv} \in \mathcal{A}$, where u, v are nodes of a KG. Both GNN and NN4G handle edge feature vectors in their iterative neighborhood aggregation process. For example, GNNs were applied to address an embedding-base *Knowledge Base Completion* (KBC, Socher et al. [56]) task, i.e., predicting missing information in KGs, outperforming TransE (Bordes et al. [50]) baseline and reaching accuracy score of 87.8% over test triplets set

(Hamaguchi et al. [57]). *Feedforward* architecture was introduce by Micheli [29] with the Neural Network for Graph model (**NN4G**), which exploits the idea that mutual dependencies can be managed by leveraging the representations from previous layers in the architecture. This model laid the foundation for one of the most popular architecture in modern Deep Learning for Graphs: Graph Convolutional (Neural) Network (**GCN**, Kipf and Welling [33]). The node state $\mathbf{h}^l$ is defined by the l-th *graph convolutional layer*.

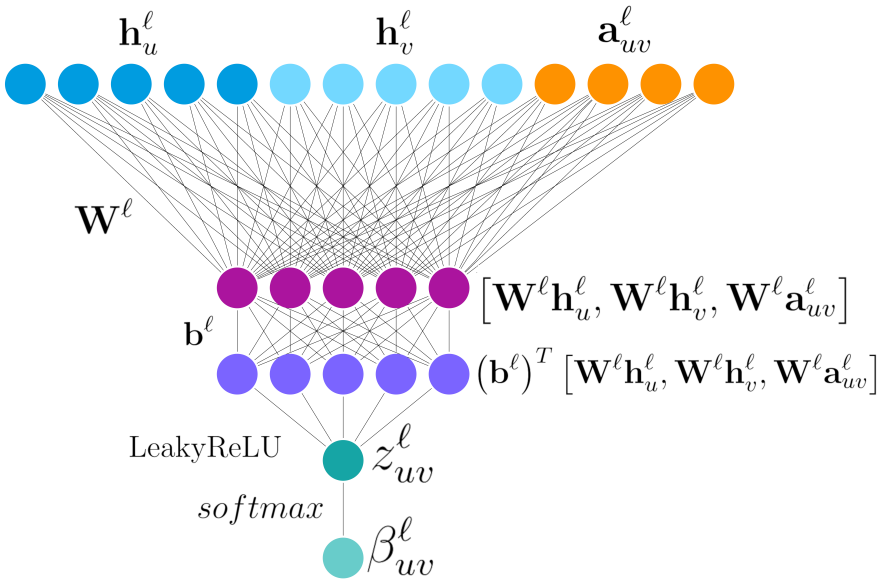


Figure 10: Extension of graph attention layer proposed by Nathani et al. [35]. The layer considers the edge feature vector $\mathbf{a}_{uv}$ in the computation of attention score, described in Equation 3.11.

Attention mechanisms applied to graphs, as with input sequences, assign different weights to different nodes in a neighborhood. Models which leverage attention for node representation are called Graph Attention Networks (**GATs**, Velickovic et al. [17]). Both GCN and GAT models don't consider relational-knowledge during neighborhood aggregation. Whereas in recent years, Relational Graph Convolution Networks (**R-GCNs**, Schlichtkrull et al. [34]) reach positive results in modeling KGs, demonstrating a large improvement on FB15k-237 (Toutanova et al. [8]) over TransE baseline. **KB-**

GAT (Nathani et al. [35]) extends graph attention mechanisms to capture both entity and relation features in a multi-hop neighborhood of a given entity. Authors employed the model for relation prediction task between entities in several KGs: WN18RR (Dettmers et al. [54]), FB15k-237 (Toutanova et al. [8]), NELL-995 (Xiong et al. [58]), Unified Medical Language Systems (UMLS) (Kok and Domingos [59]).
In the of this section, KBGAT is described in detail as the model is applied in the experimental section of this thesis to compute KGEs.

Formally, GAT neighborhood aggregation rule is define as follow (Velickovic et al. [17]):

$$\mathbf{h}_v^{\ell+1} = \sigma\left(\sum_{u\in\mathcal{N}_v} \alpha_{uv}^{\ell+1} * \mathbf{W}^{\ell+1}\mathbf{h}_u^{\ell}\right) \tag{3.5}$$

where $\mathbf{W}^{\ell}$ are the learnable parameters and $\mathbf{h}_i^{\ell}$ is the representation of node i in the layer ℓ.
Then, the *attention score* α_{uv}^{ℓ} is defined as:

$$\alpha_{uv}^{\ell} = \frac{\exp\left(w_{uv}^{\ell}\right)}{\sum_{u'\in\mathcal{N}_v}\exp\left(w_{u'v}^{\ell}\right)} \tag{3.6}$$

where w_{uv} is the *attention coefficient*, which measure some form of similarity between the current node v and each of its neighbors u:

$$w_{uv}^{\ell} = a\left(\mathbf{W}^{\ell}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\mathbf{h}_v^{\ell}\right), \tag{3.7}$$

a is a shared attention function, is implemented as:

$$a\left(\mathbf{W}^{\ell}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\mathbf{h}_v^{\ell}\right) = \text{LeakyReLU}\left(\left(\mathbf{b}^{\ell}\right)^T\left[\mathbf{W}^{\ell}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\mathbf{h}_v^{\ell}\right]\right) \tag{3.8}$$

where $\mathbf{b}^{\ell}$ is a learnable weights vector, $[\cdot,\cdot]$ denotes concatenation, and LeakyReLU is the non-linear activation function. KBGAT (Nathani et al. [35]) expands Equation 3.8, introducing edge feature vector $\mathbf{a}_{uv}$:

$$a\left(\mathbf{W}^{\ell}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\mathbf{h}_v^{\ell}, \mathbf{W}^{\ell}\mathbf{a}_{uv}^{\ell}\right) = \text{LeakyReLU}\left(\left(\mathbf{b}^{\ell}\right)^T\left[\mathbf{W}^{\ell}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\mathbf{h}_v^{\ell}, \mathbf{W}^{\ell}\mathbf{a}_{uv}^{\ell}\right]\right) \tag{3.9}$$

Consequently another *attention coefficient* z_{uv} is defined:

$$z_{uv}^{\ell} = a\left(\mathbf{W}^{\ell}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\mathbf{h}_v^{\ell}, \mathbf{W}^{\ell}\mathbf{a}_{uv}^{\ell}\right) \tag{3.10}$$

where $\mathbf{a}_{uv}^{\ell}$ is a simple ℓ-th linear transformation of the edge feature vector, for $\ell > 0$. Moreover, the *extended attention score* β_{uv}:

$$\beta_{uv}^{\ell} = \frac{\exp\left(z_{uv}^{\ell}\right)}{\sum_{u' \in \mathcal{N}_v} \exp\left(z_{u'v}^{\ell}\right)} \tag{3.11}$$

In Figure 10 Finally, KBGAT neighborhood aggregation rule is defined:

$$\mathbf{h}_v^{\ell+1} = \sigma\left(\sum_{u \in \mathcal{N}_v} \beta_{uv}^{\ell+1} * \left[\mathbf{W}^{\ell+1}\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell+1}\mathbf{h}_v^{\ell}, \mathbf{W}^{\ell+1}\mathbf{a}_{uv}^{\ell}\right]\right) \tag{3.12}$$

The computation of $\mathbf{h}_v^{\ell+1}$ is performed by a single *graph attention layer*. A model with L graph attention layers aggregates the information from L-hop distance neighbors. KBAT training is described in 4.1 in details, applied to learn KGE in the regularization process.
The node-aggregation rules for the main models mentioned above are summarized in Table 1.

3.1.3 Applications of DGNs in Natural Language Processing

Over the years, Natural Language Processing (NLP) applications have increasingly approached the use of symbolic knowledge of graphs.
In text classification task, **Text-GCN** (Yao et al. [60]) models an entire corpus as a heterogeneous graph and learns word and document embeddings with jointly GCNs. Given the dataset of documents, the authors define a graph of *document-nodes* v_d and *word-nodes* v_w. The weight w_{dw} related to the edge (v_d, v_w) is computed as the term frequency-inverse document frequency (TF-IDF) of the word in the document (*document-word* relationship). Given two word-nodes v_w, $v_{w'}$ the weight $w_{ww'}$ related to the edge (v_w, v'_w) is computed as the point-wise mutual information (PMI) (*word-word* relationship). The model is trained in a node-classification fashion, in order to classify the document-nodes with a specific label.

Moreover, knowledge graphs are seen as another form of structured relational knowledge representation in generating word-embeddings. Distributed representations like word-embeddings learned from Transformers models manage to preserve part of the semantic and syntactic information present in large corpora. In Wang et al. [61], language models are exploit to construct KGs

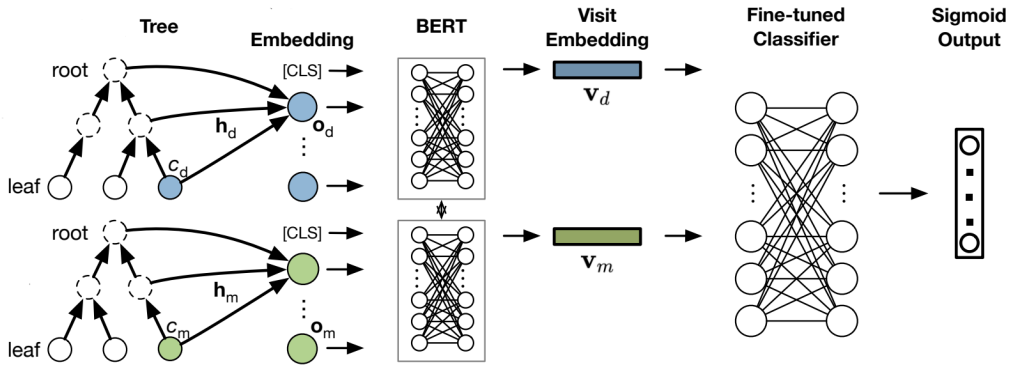


Figure 11: G-BERT architecture. Visit trees are encoded in a single node-embedding that composes the model input. The blue elements define the embeddings of the diagnosis visit trees, while the green elements represent the medication tree embeddings. The sequence of diagnosis and medications are employed to fine-tune the BERT model in order to predict the next medication, (represented by a medical code) given a patient's medical history. *Image credits*: Shang et al. [62]

with a single forward pass of the pre-trained LMs over the corpora without fine-tuning. On the other hand, BERT and Transformers-based models struggle to diversify real-word entities often present in the same context, as they are mapped into similar word-embeddings. The work proposed in Zhou et al. [15] aim to derive word-embeddings only from a dataset derived from WN18RR (Dettmers et al. [54]) exploiting one state-of-art DGN. Authors applied KBGAT (Nathani et al. [35]) to obtain representations of words and relations, both semantic and syntactic, contained in WordNet. The retrieved word-node-embeddings are evaluated over the word similarity dataset MEN (Bruni et al. [18], reporting positive results, but not competitive with language models and reaching a spearman correlation $\rho = 0.270$ on the test set.

Some recent studies have defined *hybrid* models that combine transformer architectures and symbolic knowledge.

G-BERT (Shang et al. [62], Figure 11) approaches medication recommendation task, which predict a medical code given the electronic health records (EHR), i.e., a tree-structured sequence of patient's visits. The root of the

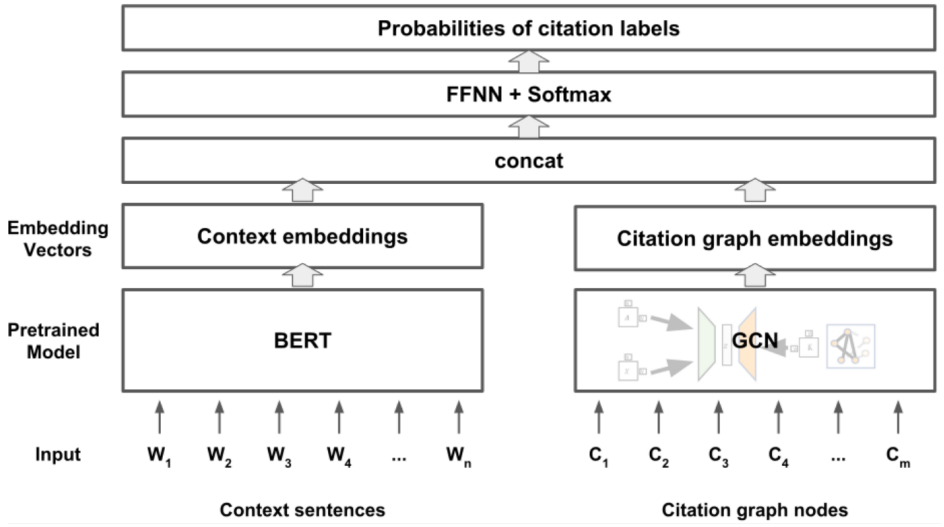


Figure 12: Hybrid model for citation recommendation BERT's fine tuning task, employing VGAE to encode the citation graph. *Image credits*: Jeong et al. [63]

tree corresponds to the first recorded visit, while the leaves the more recent visits. Every visit in the node is defined by a medical codes c_*. After computing initial embeddings for each medical code, G-BERT use GAT (Velickovic et al. [17]) to define visit node-embedding $\mathbf{h}_{c*}$, representing the embedding for a given medical code c_*:

$$\mathbf{h}_{c*} = \sigma\left(\sum_{j \in \mathcal{N}_{c*}} \alpha_{c_*,j} \boldsymbol{W} \boldsymbol{h}_j\right) \tag{3.13}$$

where $\mathbf{h}_j$ is the parent node representation of medical codes c_*, the the previous visit. Instead token embeddings vector defined in 2.4.3, G-BERT's input consists in concatenating the `[CLS]` with visit node-embedding $\mathbf{h}_{c*}$ and a final 0. G-BERT compute through Transformer's Encoder building-block the visit embedding $\mathbf{v}_{c_*}$ as follow:

$$\mathbf{v}_{c*} = Encoder_{BERT}([\texttt{[CLS]}, \mathbf{h}_{c*}, 0]) \tag{3.14}$$

With this setup, it is possible to train G-BERT in an MLM fashion, to learn the medical codes that compose visit embedding $\mathbf{v}_{c*}$. BERT model can be

fine-tuned to predict the next word given an input text, in the same way G-BERT is fine-tuned on a visit embedding sequence (diagnosis and medications) to predict the next medication code. However, the goal of this thesis is to enrich word-embeddings to improve language modeling, while G-BERT uses transformers and MLM for a domain specific fine-tuning task. In addition, G-BERT employs GAT model that does not consider the relationships between nodes in the visit tree, using only the topological information defined by the medical code sequences. The proposed framework in section **??** employs the entire multi-domain symbolic knowledge of KGs, both entities and relationships, rather than structural information.

Fine-tuning BERT by adding symbolic knowledge via node-embeddings of graphs is a topic addressed by several recent works. Jeong et al. [63] proposed a model for context-aware paper citation recommendation task using paper citation graphs. The model takes as input text sections retrieved from scientific papers with related citation of references. The citation encoder conducts unsupervised learning for citation with the GCN-based Variational Graph Auto-Encoders model (**VGAE**, Kipf and Welling [64]) using citation relationships between papers as input values. VGAE applies the unsupervised learning method of the Variational Autoencoder (Jimenez Rezende et al. [65]) to the graph structure using GCN. The model computes sentence embeddings with BERT and concatenates the BERT's output with the citations graph-embedding composed by the citations in the input text.

Similar models like **VGCN-BERT** (Lu et al. [66]) combines the capability of BERT with a Vocabulary Graph Convolutional Network (VGCN) for text classification. For each input sentence, it is constructed the related vocabulary graph using normalized point-wise mutual information (NPMI): given two words in the input sentence, if their NPMI is larger than a fixed threshold, the two words are connected in the vocabulary graph. The obtained graph is fed into a GCN (Kipf and Welling [33]) in order to learn vector representation for the vocabulary graph. Sentence-embeddings and vocabulary-graph embedding are given as input to BERT. The word embedding and graph embedding then interact with each other through the self-attention mechanism while learning the classifier.

All of the cited works support the combination of BERT with graph symbolic knowledge, outperform classic BERT model in the downstream tasks, but

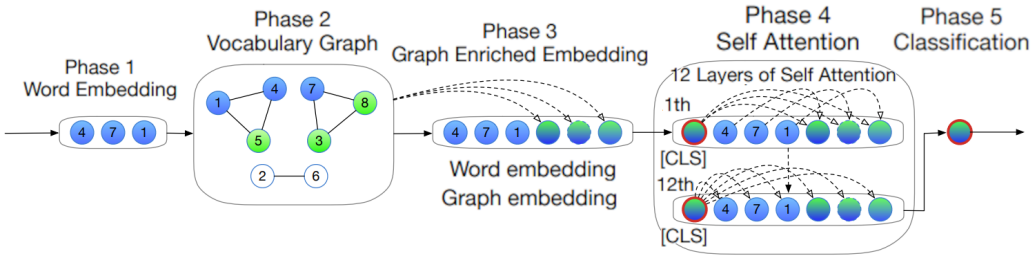


Figure 13: VGCN-BERT architecutre with all the steps described to enrich word-embedding using GCN and NPMI embeddings. The model leverage the BERT's attention mechanism to transfer information during the fine-tuning. *Image credits*: Lu et al. [66].

none of them apply DGNs in pre-training for language modeling. Moreover, none of the works use knowledge graphs that contain global information, but only constructed graphs according to the input sentence. At the same time, these existing combinations of BERT with node and graph embeddings have contributed significantly to a new NLP approach through graphs. Some of the methodologies described were the inspiration for this thesis.

3.2 Adapts Different Embeddings Spaces

After defining the methodologies for representing symbolic knowledge as KGEs and presenting some practical applications that leverage graph representation in NLP, in the following section the information transfer problem between embeddings posed as a regression problem is presented. In the first part the approaches on word decomposition to enrich word embeddings are summarized, while in the second part, the works introducing embedding regression are described. In this section it is introduced the concept of embedding regression as regularization term.

3.2.1 Enriching Word Representations

Improving learning of word representations has always been a challenge in NLP and it has already been analyzed since before the advent of transformers-based language models such as BERT and GPT-3. Morphological Recursive

Neural Network (**morphoRNN**, Luong et al. [67]), combine RecNNs (described in 2.3) with Neural Language Models (**NLMs**, Bengio et al. [68], Alexandrescu and Kirchhoff [69]) to consider contextual information in learning morphologically aware word representations. In morphology, a morpheme is defined as a word or a part of a word with meaning. Given a word composed by multiple morpheme, a *stem* is any morpheme to which an additional morpheme, called an *affix*, may be added at the head or tail. First, the model gradually builds up vectors of morphologically complex words from their morphemic representations. Once vectors of all complex words have been built, the NLM assigns a score for each input words sequence $\mathbf{n} = \{w_i\}_{i=1}^{N}$ as follows:

$$s\left(n_i\right) = \mathbf{v}^\top f\left(\mathbf{W}\left[\mathbf{v}_1; \ldots; \mathbf{v}_n\right] + \mathbf{b}\right) \tag{3.15}$$

Here, $\mathbf{v}_i$ is the vector representing the word w_i, which in the case of complex words are computed by RecNNs. $\mathbf{W} \in \mathbb{R}^{h \times nd}, \mathbf{b} \in \mathbb{R}^{h \times 1}$, and $\mathbf{v} \in \mathbb{R}^{h \times 1}$ are parameters of the NLM, and f is an element-wise activation function. The model aims to minimize the following objective function:

$$\mathcal{L} = \sum_{i=1}^{N} \max\left\{0, 1 - s\left(n_i\right) + s\left(\bar{n}_i\right)\right\} \tag{3.16}$$

where $\bar{n}_i$ is a corrupted word sequence generated from n_i.
Another approach employs n-gram instead of morphene (Bojanowski et al. [70]). Given a large text corpus $\mathcal{C}_\mathcal{V}$ and a subcorpus $\mathcal{C}_w \subset \mathcal{C}_\mathcal{V}$, for a given focal word w in dictionary $\mathcal{V}$, let be $c \in \mathcal{C}_w$ the context of w, defined as the sequence of words $w' \in c$ surrounding w e.g. a fixed-size window around the word w. Let be $\mathcal{G}_w \subset \{1, \ldots, G\}$ the set of n-grams in w. As done by morphoRNN, for a given word w is computed a score function s based on the context $c \in \mathcal{C}_w$:

$$s(w, w') = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_{w'} \tag{3.17}$$

where $\mathbf{z}_g, \mathbf{v}_{w'} \in \mathbb{R}^d$ are the initial vector representations of the n-grams and context word w'. Like Equation 3.16, the objective function is defined on positive samples, the context words $w' \in c$, and negative samples $w'' \in \mathcal{N}_c$, randomly sampled from the vocabulary $\mathcal{V}$. Minimizing the negative log-likelihood:

$$\mathcal{L} = \log\left(1 + e^{-s(w,w')}\right) + \sum_{w'' \in \mathcal{N}_c} \log\left(1 + e^{s(w,w'')}\right) \tag{3.18}$$

the problem of predicting context words can be framed as a set of independent binary classification tasks. The defined objective aims to independently predict the presence (or absence) of context words.
In literature, other many different works apply enrichment techniques at *word-level* (Padó et al. [71], Lazaridou et al. [72], Herbelot and Baroni [73]). Similarly, this thesis attempts fine-grained *word-level* embedding enrichment to improve BERT pre-training MLM task. In contrast, language models defined by transformer architectures are usually enhanced with *sentence-level* fine-tuning on specific domain, not by acting directly on single word representations.

3.2.2 Embeddings Regression

Recalling the notation described in the previous section, it is possible to learn high-quality embedding for a given word w given its context c.
ALC embedding method (*à la carte*, Khodak et al. [74]), approximates the word pre-trained representation $\mathbf{v}_w$ as a context vector $\mathbf{u}_w$ (for simplicity consider a single context c for each word w):

$$\mathbf{v}_w \approx \mathbf{u}_w = \mathbf{A}\left(\sum_{w' \in c} \mathbf{v}_{w'}\right) \tag{3.19}$$

where $\mathbf{A} \in \mathbb{R}^{d \times d}$ is *transformation matrix*, learned solving the follow *linear regression* problem:

$$\hat{\mathbf{A}} = \arg\min_{\mathbf{A}} \sum_{w \in \mathcal{V}} \|\mathbf{v}_w - \mathbf{A}\mathbf{u}_w\|_2^2 \tag{3.20}$$

Given $\hat{\mathbf{A}}$, we can introduce new embeddings $\hat{\mathbf{v}_w}$ for any word by averaging the existing embeddings for all words in its context to create $\mathbf{u}_w$ and then applying the transformation such that $\hat{\mathbf{v}}_w = \hat{\mathbf{A}}\mathbf{u}_w$.

Recently, an expansion of the equation has been proposed with a slightly modified linear regression model that reweights words with a non-decreasing function $\alpha(\cdot)$ of the total instances of each word w in the corpus (Rodriguez et al. [75]). Reweighting addresses the fact that words that appear more frequently have embeddings that are measured with greater certainty.

$$\hat{\mathbf{A}} = \arg\min_{\mathbf{A}} \sum_{w \in \mathcal{V}} \alpha\left(w\right) \|\mathbf{v}_w - \mathbf{A}\mathbf{u}_w\|_2^2 \tag{3.21}$$

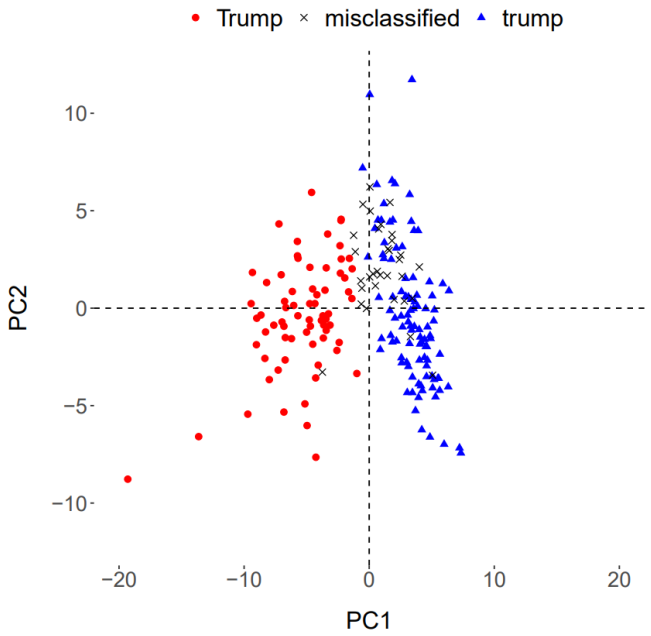


Figure 14: Example of clusters composed by applying k-means clustering to evaluate ALC embeddings. *Image credits*: Rodriguez et al. [75].

Regression embedding allows context information to be inferred in the focal word. The results proposed by the authors demonstrate how ALC word-embeddings manage to distinguish the different senses associated with a word (e.g `Trump` the person and `trump` as a verb or a noun). Furthermore, ALC embeddings perform on par with transformer-based RoBERTa (Liu et al. [76]) on the average cluster homogeneity task, i.e. apply k-means clustering with two clusters to the set of embedded instances for selected words (`Trump` and `trump`) and evaluate whether the clusters partition the two senses (Figure 14). This may be a result of RoBERTa being optimized for sentence-level embeddings more than individual word-level embeddings.

Deep Multi-Network Embedding (**DMNE**, Ni et al. [77]) coordinates multiple neural networks with a co-regularized loss function to manipulate cross-graph relationships. To give an example, DMNE learns node-embeddings for two different social networks (e.g., Instagram and Twitter) co-inferring the user's relationships from both social networks in the node representations.

Let be $\mathcal{G} = \{\mathbf{G}^i\}_{i=0}^g$ the set of g input graphs. Given two graphs $\mathbf{G}^i, \mathbf{G}^j \in \mathcal{G}$, $\mathbf{h}_v^i \in \mathbb{R}^d$ represents the embedding for a given node $v \in \mathbf{G}^i$, where d is the embeddings dimension. After initializing node-embeddings for each graph $\mathbf{G}^i$, the model partially incorporates AutoEncoder (Baldi [78] to minimize the node-embeddings matrix reconstruction error $\mathcal{L}_{AE}^i$. Intuitively, if a node $v \in \mathbf{G}^i$ is mapped to a node $u \in \mathbf{G}^j$, then the the node-embedding representations $\mathbf{h}_v^i$ and $\mathbf{h}_u^j$ should be similar, where $\mathbf{h}_u^j$ denote the node $u \in \mathbf{G}^j$ mapped to $v \in \mathbf{G}^i$. $\mathcal{N}^{(i \to j)}(v)$ denote the set of nodes $u \in \mathbf{G}^j$ that are mapped to a single node $v \in \mathbf{G}^i$ with positive weights. To incorporate cross-network relationships, authors propose two loss function:

- *Embedding Disagreement* (ED): measures the inconsistency of cross-network node-embeddings.

$$\mathcal{L}_{ED}(i,j) = \sum_{v \in \mathbf{G}^i} \left\| \mathbf{h}_v^{(i)} - \mathbf{h}_v^{(i \to j)} \right\|_F^2 \tag{3.22}$$

  where

$$\mathbf{h}_v^{(i \to j)} = \frac{1}{\sum_{y \in N^{(i \to j)}(v)} \mathbf{S}_{vu}^{(ij)}} \sum_{u \in \mathcal{N}^{(i \to j)}(v)} \mathbf{S}_{vu}^{(ij)} \mathbf{h}_u^{(j)} \tag{3.23}$$

  is the weighted mean of the embeddings of nodes in network $\mathrm{G}^{(j)}$ that are mapped to v. Recall that $\mathbf{S}_{vu}^{(ij)}$ is the weight on the crossnetwork relationship between node v in $\mathrm{G}^{(i)}$ and node u in $\mathrm{G}^{(j)}$.
  Let $\mathbf{H}^{(i)} = \left[\left(\mathbf{h}_1^{(i)} \right)^T, \ldots, \left(\mathbf{h}_{n_i}^{(i)} \right)^T \right]^T \in \mathbb{R}^{n_i \times d_i}$, packing together the nodes $v \in \mathbf{G}_i$ in Equation 3.22, embedding disagreement loss function can be

$$\mathcal{L}_{ED}^{(i,j)} = \left\| \mathbf{O}^{(ij)} \mathbf{H}^{(i)} - \mathbf{S}^{(ij)} \mathbf{H}^{(j)} \right\|_F^2 \tag{3.24}$$

  where we introduce a diagonal indicator matrix $\mathbf{O}^{(ij)} \in \{0,1\}^{n_i \times n_i}$, with $\mathrm{O}_{xx}^{(ij)} = 0$ if the x-th row of $\mathbf{S}^{(ij)}$ is all-zero; and $\mathrm{O}_{xx}^{(ij)} = 1$ otherwise.

- *Proximity Disagreement (PD)*: measures a proximity score for each node-embeddings pair $\mathbf{h}_v^i, \mathbf{h}_w^i \in \mathbf{G}^i$. Then, the same proximity score is computed for all the related pairs $\mathbf{h}_v^{i \to j}, \mathbf{h}_w^{i \to j} \in \mathbf{G}^j$. Proximity score is

defined as the inner product between two paired node-embeddings.

$$\begin{aligned}\mathcal{L}_{PD}^{(ij)} &= \sum_{x=1}^{n_i}\sum_{z=1}^{n_i}\left[\mathbf{h}_x^{(i)}\left(\mathbf{h}_z^{(i)}\right)^T - \mathbf{h}_x^{(i\to j)}\left(\mathbf{h}_z^{(i\to j)}\right)^T\right]^2 \\ &= \left\|\mathbf{O}^{(ij)}\mathbf{H}^{(i)}\left(\mathbf{O}^{(ij)}\mathbf{H}^{(i)}\right)^T - \mathbf{S}^{(ij)}\mathbf{H}^{(j)}\left(\mathbf{S}^{(ij)}\mathbf{H}^{(j)}\right)^T\right\|_F^2\end{aligned} \tag{3.25}$$

where n_i represents the number of nodes in graph $\mathbf{G}^i$. *PD* loss function aims to distance node-embeddings that are very different from each other and bring similar node-embeddings closer, while trying to maintain the local structure of the single graph in the latent space. Applying only ED loss, which tries to make all $h_v^{(i\to j)}$ nodes similar to the mapped node h_v^i, could cause all nodes $h_v^{(i\to j)}$to be clustered together despite they may be very different in G^j.

The complete loss function can be derived as the sum between node-embeddings matrix reconstruction error and the regularization losses:

$$\mathcal{L}_{DMNE} = \sum_{i=0}^{g}\sum_{j=0}^{g}\mathcal{L}_{AE}^i + \mathcal{L}_{ED}^{(i,j)} + \mathcal{L}_{PD}^{(i,j)} \tag{3.26}$$

where g represent the total index of graphs considered in the co-regularizing process (intuitively, $g > 2$). Both regularization loss functions $\mathcal{L}_{ED}^{(i,j)}$, $\mathcal{L}_{PD}^{(i,j)}$ can be viewed the residue of non-linear regression between node-embeddings from multiple graphs, where the regression residue is applied as regularization terms.

Regression between embeddings is a common solution to induce information from a context embedding to a focal embedding. In Bojanowski et al. [70] and Rodriguez et al. [75] the context information is an aggregate vector of the words surrounding the focal word, while in Ni et al. [77] the context is defined by cross-network node-embeddings mapped to the focal node. Similarly, this thesis applies linear regression in order to infer part of knowledge graphs context, represented by KG's node-embeddings, to BERT's word-embeddings.

4 Proposed Framework

In previous sections, it has been examined how transformers-based language models, such as BERT, apply Masked Language Modeling to learn the representations of words in a corpus. As demonstrated by several cited works (Yao et al. [60], Zhou et al. [15], Jeong et al. [63]), words in a text can be represented by nodes in a graph. This framework attempts to combine the word-embeddings of BERT and the node-embeddings of two main knowledge graphs, WN18RR and FB15K-237 by extending concepts described in section 3: map BERT's input sentence into node-embeddings (KGEs) named *KGE sentence dictionary*, similar to Vocabulary Graph proposed in Lu et al. [66] and apply embeddings regression with word-embeddings (ALC embeddings, Khodak et al. [74], Rodriguez et al. [75]) to compute a regularized term comparable to Embedding Disagreement loss (Ni et al. [77]).
The first section describes *KGE sentence dictionary* employed in the regularization. KGEs are learned applying KBGAT (Nathani et al. [35]), in order to leverage the symbolic knowledge of edge feature vectors during language modeling task. Then, an extension of masked language modeling (MLM) is presented, introducing *word-level* regularization term derived from the embeddings regression between word-embeddings and node-embeddings. Regularization aims to induce the structured context of node-embeddings from a KG into BERT's word-embeddings for which there is at least one related KGE in the KG.

4.1 KGE Sentence Dictionary

In order to regularize BERT's word-embeddings with KGEs, this framework describes the process to retrieve the KGEs related to each BERT's input sentence, collected in the *KGE sentence dictionary* $\mathbf{D_w}$. The dictionary allows to retrieve for each word w_i the structured KGE representation $\mathbf{h}_i$. Given an input sentence $\mathbf{w} = \{w_i\}_{i=0}^{|\mathbf{w}|}$ as a sequence of words w_i in a fixed dictionary $\mathcal{D}_{LM}$, it is possible to define an injective map function $\phi : \mathcal{D}_{LM} \rightarrow \mathcal{V}$:

$$\phi(\mathbf{w}, w_i) = i \tag{4.1}$$

which maps the word $w_i \in \mathbf{w}$ to node $i \in \mathcal{V}$ of the KG, given as context the input sentence $\mathbf{w}$. The retrieved node for word w_i is represented by the i-th row $\mathbf{h}_i$ of the *entity embeddings matrix* $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d_h}$, where d_h is the

embeddings dimension. Applying function ϕ is possible due to the definition of KG, where entities model real-word concepts, present a natural language label l to identify them. There are words in a corpus that may not be associated with entities (e.g. articles, conjunctions, pronouns): in this case, a special KGE $\mathbf{h}_{none} \in \mathbf{H}$ is assign to the word.
For each input sentence $\mathbf{w}$, $\mathcal{D}_{\mathbf{w}}$ is defined as set of pairs $< k, v >$ where the key k is a word $w_i \in \mathbf{w}$ and the value v is the KGE $\mathbf{h}_i$ related to word w_i, retrieved applying the function ϕ:

$$\mathcal{D}_{\mathbf{w}}[k \to v](w_i) = \begin{cases} \mathbf{h}_{\phi(\mathbf{w},w_i)} & \text{for } w_i = k \\ \mathbf{h}_{none} & \text{otherwise} \end{cases} \tag{4.2}$$

However, a word can be assigned several nodes in the graph (different entities can be labeled with the same name), implying that the value v for a given key k can be a set of multiple KGEs $\mathbf{h}_{\phi(\mathbf{w},w_i)}$. Therefore, the framework include *word-sense disambiguation* (WSD) mechanism to make the mapping function injective. For simplicity, we will only consider ϕ to be injective for now. Before describing WSD mechanism, the steps to compute KGEs are analyzed.

4.1.1 Learning KGEs

Following Nathani et al. [35], this work employs KBGAT as the encoder model to learn the values v of dictionary $\mathcal{D}_{\mathbf{w}}$. The KGEs of the knowledge graph are computed using the following steps:

1. For each node i, for each edge (i, j), compute the initial KGEs $\mathbf{h}_i^0$ and $\mathbf{a}_{ij}^0$ employing shallow knowledge graph encoder TransE (Bordes et al. [50])

2. Given N entities and R relationships in a KG, *entity embeddings matrix* $\mathbf{H}^\ell$ is constructed as follow:

$$\mathbf{H}^\ell = \left[\left(\mathbf{h}_1^\ell\right)^T, \ldots, \left(\mathbf{h}_N^\ell\right)^T \right]^T \in \mathbb{R}^{|\mathcal{V}| \times d_h} \tag{4.3}$$

   where $\mathbf{h}_i^\ell$ is the node-embeddings of the node i in the the ℓ-th *graph attention layer* in KBGAT model. The *relation embeddings matrix* $\mathbf{G}^\ell \in \mathbb{R}^{|\mathcal{E}| \times d_g}$ similarly contains all input relationships, represented by KGEs $\mathbf{a}_{ij}^\ell$:

$$\mathbf{G}^\ell = \left[\left\{ \left(\mathbf{a}_{ij}^\ell\right)^T \right\} \right]^T \in \mathbb{R}^{|\mathcal{E}| \times d_g}, (i, j) \in \mathcal{E} \tag{4.4}$$

3. For every KGEs $\mathbf{h}_i^\ell \in \mathbf{H}^\ell$, the KGE $\mathbf{h}_i^{\ell+1}$ is defined as the output of the single graph attention layer $\ell + 1$, computed using the neighborhood aggregation rule described by Equation 3.12:

$$\mathbf{H}^{\ell+1} = attention(\mathbf{W}^{\ell+1}, \mathbf{H}^\ell, \mathbf{G}^\ell) \tag{4.5}$$

where $\mathbf{W}^{\ell+1} \in \mathbb{R}^{d_{aggr} \times d'_h}$ are the learnable parameters of the graph attention layer, where d_{aggr} is the dimension of aggregated vector $[\mathbf{h}_j, \mathbf{h}_i, \mathbf{a}_i j]$ and d'_h the final entity KGEs dimension.
Each layer ℓ compute the entities embeddings matrix $\mathbf{H}^{\ell+1}$. $\mathbf{G}^{\ell+1}$ is computed as a linear transformation of $\mathbf{G}^\ell$:

$$\mathbf{G}^{\ell+1} = \mathbf{G}^\ell \mathbf{W}_r^{\ell+1} \tag{4.6}$$

where $\mathbf{W}_r^{\ell+1} \in \mathbb{R}^{d_g \times d'_g}$ is a weights matrix, where d'_R is the final dimension of relation embeddings matrix.
Let $\mathbf{H}^L$ and $\mathbf{G}^L$ be the outputs of the last graph attention layer. In addition, a linear projection of the input entity embedding matrix $\mathbf{H}^0$ is added to matrix $\mathbf{H}^L$ in order to preserve the initial information during KGE learning:

$$\mathbf{H}^L = \mathbf{H^0} \mathbf{W}_E^L + \mathbf{H}^L \tag{4.7}$$

where $\mathbf{W}_e^L \in \mathbb{R}^{d_h \times d'_h}$ is the weights matrix for initial entity embeddings.

4. Every triplet t_{ij}^r of the KG can be represent as KGE triplet $\mathbf{t}_{ij}$:

$$\mathbf{t}_{ij} = (\mathbf{h}_i^L, \mathbf{a}_{ij}^L, \mathbf{h}_j^L) \in \mathcal{T} \quad \mathbf{h}_i^L, \mathbf{h}_j^L \in \mathbf{H}^L, \mathbf{a}_{ij}^L \in \mathbf{G}^L. \tag{4.8}$$

The set of triplet $\mathcal{T}$ is defined as the set of *positive triplets*. Denote by $\mathcal{T}'$ the set of *negative triplets*, obtained replacing the head or tail entities in a valid triplet of the KG:

$$\mathcal{T}' = \underbrace{\left\{\mathbf{t}_{i'j} \mid \mathbf{h}'_i \in \mathbf{H}^L \backslash \mathbf{h}_i^L\right\}}_{\text{replace head entity}} \cup \underbrace{\left\{\mathbf{t}_{ij'} \mid \mathbf{h}'_j \in \mathbf{H}^L \backslash \mathbf{h}_j\right\}}_{\text{replace tail entity}}. \tag{4.9}$$

5. The model learns KGEs triplets as values $v \in \mathcal{D}_{\mathcal{KB}}$ minimizing the pairwise-margin objective loss defined in Equation 3.1:

$$\mathcal{L}_{KBGAT} = \sum_{\mathbf{t}_{ij} \in \mathcal{T}} \sum_{\mathbf{t}'_{ij} \in \mathcal{T}'} \left[\tau + \|\mathbf{h}_i + \mathbf{a}_{ij} - \mathbf{h}_j\| - \left\|\mathbf{h}'_i + \mathbf{a}_{i'j'} - \mathbf{h}'_j\right\|\right]_+ \tag{4.10}$$

4.1.2 Word Sense Disambiguation

As stated in the previous section, the first issue to build a KGE sentence dictionary $\mathcal{D}_{\mathbf{w}}$ for an input sentence $\mathbf{w} = \{w_i\}_{i=0}^{|\mathbf{w}|}$is to define an injective mapping function $\phi : \mathcal{D}_{LM} \rightarrow \mathcal{V}$, where D_{LM} is the language model dictionary and $\mathcal{V}$ the set of the KG's nodes. In other words, the function ϕ identifying the *sense* related to the input word w_i in a single specific node $i \in \mathcal{V}$ of the KG. After having defined ϕ, it is possible to construct the key-value pair $< k, v >$, taking the related KGE $\mathbf{h}_i$ of the output node as value v. The sense identification performed by ϕ can be approached as a *Word-sense disambiguation* (WSD) problem. WSD is an open problem in computational linguistics concerned with identifying which sense of a word is used in a sentence. Graph-based WSD methods have gained much attention in years (Mihalcea [79], Agirre and Soroa [80], Navigli and Velardi [81], Navigli and Lapata [82], Ponzetto and Navigli [83], Wei et al. [84]). Graph-based techniques are performed over the graph underlying a specific knowledge base; they first consider all the sense combinations of the words in a given context and then try to search for the relations among senses based on the whole graph. The main disadvantage of graph-based methods is their computational expense (Navigli and Lapata [82]).

Let $\mathbf{w} = \{w_i\}_{i=0}^{|\mathbf{w}|}$ the input sentence. Let $\mathbf{c}_i = \{c_{ik}\}_{k=0}^{K}$ the set of all K *senses* associated to the word w_i, identified by the set of K nodes $\mathcal{S}_i = \{i_k\}_{k=0}^{K} \subset \mathcal{V}$. $\mathcal{S}_i$ is considered the set of *candidate nodes* for a single word w_i The set $\mathcal{S}_i$ is compute applying topological search and morphological comparison between the word w_i and all the nodes i with labels l_i. This thesis addresses only the issue to handle multi-senses for a single word input. For example, in the case of the word `bank`, WordNet has defined several entity nodes with different semantic definitions, few of them listed below:

- `bank.n.01`: *sloping land (especially the slope beside a body of water)*
- `depository_financial_institution.n.01`: *a financial institution that accepts deposits and channels the money into lending activities*
- `bank.n.03`: *a long ridge or pile*
- `bank.n.09`: *a building in which the business of banking transacted*

For each input word $w_i \in \mathbf{w}$, the set of *candidate nodes* $\mathcal{S}_i$ is computed. Then, the injective mapping function ϕ introduced in Equation 4.1 is defined as follow:

$$\phi(\mathbf{w}, w_i) = \max_{i \in \mathcal{S}_i} \sum_{w_j \in \mathbf{w}} \max_{j \in \mathcal{S}_j} \psi(i, j) \tag{4.11}$$

The function ϕ compute the single sense-node i for word w_i given the context sentence $\mathbf{w}$ as the candidate node in set $\mathcal{S}_i$ which maximize the node similarity function $\psi : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ with all other sense-nodes sets $\mathcal{S}_j$ of the other words in the input sentence. To reduce the computational cost during BERT's regularization, the similarity function is computed between candidate nodes of target word w_i and words w_j included in a context window of fixed size M, $0 < M < |\mathbf{w}|$ centered on w_i. In addition, a threshold s, $0 < s < 1$ is introduced such that a candidate node is chosen as optimal and assigned to a word embedding only if the function *psi* returns a score $> s$.
The node similarity function ψ employed in this framework is the *Wu-Palmer similarity* (WUP, Wu and Palmer [85]). Given two distinct entity nodes $i, j \in \mathcal{V}$, Wu-Palmer similarity, denoted by δ_{wup}, is computed based on the *lowest common subsumer* (LCS) in the path between i and j:

$$\delta_{wup}(i, j) = \frac{2d}{L_i + L_j + 2d} \tag{4.12}$$

where L_i and L_j are the path length between LCS and nodes i and j, respectively. d is defined as the distance between the LCS and the *root node* of the ontology underlying the KG. On the other hand, if the KG doesn't present a root node (`entity.n.01` in WordNet) and the attributes and the relationships are not suitable to compute the LCS, it is possible to employ different similarity measures like cosine similarity between the related KGEs or class similarity, where the set of common parents for two nodes is compared (Ilievski et al. [86]).

4.2 MLM KGE Regression Regularization

As described in Section 2.4, BERT applies the bidirectional training of Transformers to language modeling through two self-supervised learning tasks without labelled data: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). Since the goal of this work is to define a fine-grained regularization at a word-level, the proposed framework only extends the

MLM task. The main objective of this work is to evaluate the impact of a regularization loss that forces word-embeddings to be good regressors of node-embeddings. BERT parameters are updated to compute word-embeddings similar to KGEs, as per embedding regression (Section 3.2.2), by projecting word-embeddings in the latent space of node-embeddings. Having performed language model pre-training, language model probing tasks are performed to assess whether the word-embeddings have acquired the symbolic knowledge of the KG.
Let be $\mathcal{C}$ a corpus consisting of a sequence of K words w_i belonging to a fixed dictionary $\mathcal{D}_{LM}$, i.e $\mathcal{C} = \{w_i\}_{i=0}^{K}$, $w_i \in \mathcal{D}_{LM}$. BERT's input embedding layer transforms an input sub-sequence of N words $\mathbf{w} = \{w_i\}_{i=0}^{M}$, $s \in \mathcal{C}$, in a matrix $\mathbf{X} \in \mathbb{R}^{N \times d_{\text{hidden}}}$ where each i-th row is the initial distributed input word representation $\mathbf{x}_i \in \mathbb{R}^{d_{\text{hidden}}}$ for the input word w_i, d_{hidden} is the model hidden dimension and N the number of input tokens:

$$\mathbf{x}_i = Tokenizer(w_i) \tag{4.13}$$

Then, the word representations matrix is fed into the encoders stack to compute the matrix $\mathbf{Z_w} \in \mathbb{R}^{N \times d_{\text{hidden}}}$ of $\mathbf{z}_i \in \mathbf{Z}$ contextualized word-embeddings related to input sentence $\mathbf{w}$:

$$\mathbf{z}_i = Encoder(\mathbf{x}_i) \tag{4.14}$$

In the general case of transformers architecture, $\mathbf{z}_t$ is the output of the transformers encoder-decoder building blocks:

$$\mathbf{z}_i = Decoder(Encoder(\mathbf{x}_i)) \tag{4.15}$$

specifically, BERT uses only the encoders stack, where decoder is the identity function. Therefore this framework considers word-embedding $\mathbf{z}_i$ as defined in Equation 4.14.

For the same input sentence $\mathbf{w}$, the KGE sentence dictionary $\mathcal{D}_{\mathbf{w}}$ is computed, then the node-embeddings for the word w_i is retrieved:

$$\mathbf{h}_i = \mathcal{D}_{\mathbf{w}}(w_i) \tag{4.16}$$

Since all the KGEs $\mathbf{h}_i$ are stored in the entity embeddings matrix $\mathbf{H}$, the regularization computational impact is only given by the mapping injective function $\phi(\mathbf{w}, w_i)$.

For each sentence $\mathbf{w}$ in the input batch, the model encodes in two embedding matrices, the sentence word-embedding matrix $\mathbf{Z}_w$ (Equation 2.23):

$$\mathbf{Z_w} = \left[(\mathbf{z}_1)^T, \ldots, (\mathbf{z}_N)^T \right]^T \in \mathbb{R}^{N \times d_{hidden}} \tag{4.17}$$

and the sentence KGE matrix $\mathbf{H}_w$:

$$\mathbf{H_w} = \left[(\mathcal{D}_{\mathbf{w}}(w_1))^T, \ldots, (\mathcal{D}_{\mathbf{w}}(w_N))^T \right]^T \tag{4.18}$$

$$= \left[(\mathbf{h}_1)^T, \ldots, (\mathbf{h}_N)^T \right]^T \in \mathbb{R}^{N \times d_{KGE}} \tag{4.19}$$

The regularizing loss is then defined as follow:

$$\mathcal{L}_R(\mathbf{w}) = \sum_{w_i \in \mathbf{w}} \|\mathbf{h}_i - \mathbf{z}_i \mathbf{A}\|_2^2 \tag{4.20}$$

where $\mathbf{A} \in \mathbb{R}^{d_{hidden} \times d_{KGE}}$ the linear projection matrix. In practice $\mathbf{A}$ is a single linear layer.

$$\mathcal{L}_R(\mathbf{w}) = \|\mathbf{H_w} - \mathbf{Z_w}\mathbf{A}\|_2^2 \tag{4.21}$$

The concept of inducing information from one embedding to another described in 3.2.2 is extended on embeddings computed by two different types of encoder: BERT, aimed to learn the local information in the input sentence, and KBGAT which capture relationships between node-embeddings employed to regularize word-embeddings. Unlike other proposed works (Lu et al. [66]), the node-embeddings added in the BERT's training process contain global information of the whole KG thanks to neighborhood aggregation and not only structured information derived from the single input sentence.

Finally, recalling to the MLM loss defined in Equation 2.31, the new BERT loss is defined:

$$\mathcal{L}_{\text{MLM}}(\mathbf{X}_\Pi \mid \mathbf{X}_{-\Pi}, \theta) = \frac{1}{K} \sum_{k=1}^{K} \log p(\mathbf{x}_{\pi_k} \mid \mathbf{X}_{-\Pi}; \theta) + \lambda \mathcal{L}_R(\mathbf{X}_{-\Pi}, \mathbf{H}, \theta) \tag{4.22}$$

Note that $\mathcal{L}_R$ only considers unmasked tokens $\mathbf{X}_{-\Pi}$. For future developments, it is considerable to use a possible KGE associated with masked word-embedding to further help transfer symbolic knowledge into BERT parameters. The λ hyperparameter is added in order to tune the influence

of the regularization term. Moreover, it is to underline that the θ BERT's parameters are taking into account by the regularization term. In the framework, the linear layer's parameter are updated due the back propagation, this is in order not to generate a bias by maintaining static weights in the linear transformation. It is assumed that a simple linear projection fails to generalize the structured knowledge of node-embeddings and consequently, the convergence of the regularization term is mainly due to the regularized BERT's parameters.

5 Experiments

The following section presents the datasets and experiments performed to train and evaluate the proposed language model, enriched by symbolic knowledge of KGs. For learning knowledge graph embeddings, KBGAT model was trained on two knowledge graphs: WN18RR and FB15K-237. Regarding language modeling, two models were trained: the baseline model `BERT-BASELINE` and the proposed regularized model `BERT-KG`. Both models were trained with the same computational resources, input data, and hyperparameters in order to have a fair comparison. The first two sections reports the datasets and the training process for all the models applied in the experiments. The last section describes the datasets and tasks aim to evaluate the symbolic knowledge regularization probing `BERT-KG`'s word-embedding. The proposed framework is implemented in Python using HuggingFace's Transformers [1], PyTorch[2] and PyTorch Geometric[3] libraries. All the experiments were performed with 2 GPUs NVIDIA A100 SMX4 40 GB.

5.1 Training Dataset

5.1.1 WN18RR

WN18RR (Dettmers et al. [54]) is lexical knowledge graph dataset constructed from WN18(Fellbaum [6]), a subset of WordNet. WN18 consists of 18 relations and 40,943 entities. However, many triples are obtained by inverting triples from the training set. Thus the WN18RR dataset is created to ensure that the evaluation dataset does not have inverse relation test leakage. WN18RR dataset contains 93,003 triples with 40,943 entities and 11 relation types (Table 2).

In WordNet and derived datasets, the KG edges model the semantic relationships between word-nodes. WN18RR entities are described by two types of nodes in the KG: *lemma nodes* (lemmas), which represent a specific word from the English dictionary, and *synset node* (synsets), which represent the *sets of synonyms*. As described in the previous section, there are different synsets associated with the word *"bank"*, each of which has multiple lemmas

[1] huggingface.co/docs/transformers
[2] pytorch.org/docs/stable/index
[3] pytorch-geometric.readthedocs.io/en/latest/

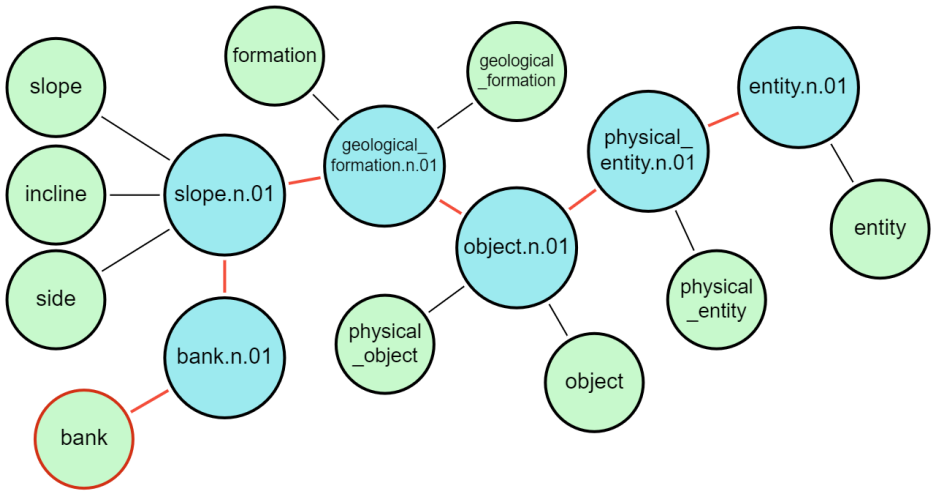


Figure 15: Visualization of the hierarchical structure in WN18RR. The red path represents the hypernym/hyponymy tree. Green nodes represent lemma leaf nodes, while blue one represents synsets. The tree is constructed in a bottom up fashion, given the word *bank* and follow the hypernym relationships to the synsets bank.n.01.

associated with it:

- `bank.n.01`: bank
- `depository_financial_institution.n.01`: bank, banking_concern, banking_company
- `bank.n.03`: bank
- `bank.n.09`: bank, bank_building
- `deposit.v.02`: deposit, bank

Synset identifiers are in the form `<lemma>.<pos>.<number>`, where the `<pos>` is the *part of speech* character identifier (`n`: noun,`v`: verb, `a`: adjective, `s`: adjective satellite, `r`: adverb) and `<number>` is used to disambiguate word meanings for polysemic words, i.e word with several *word senses*.

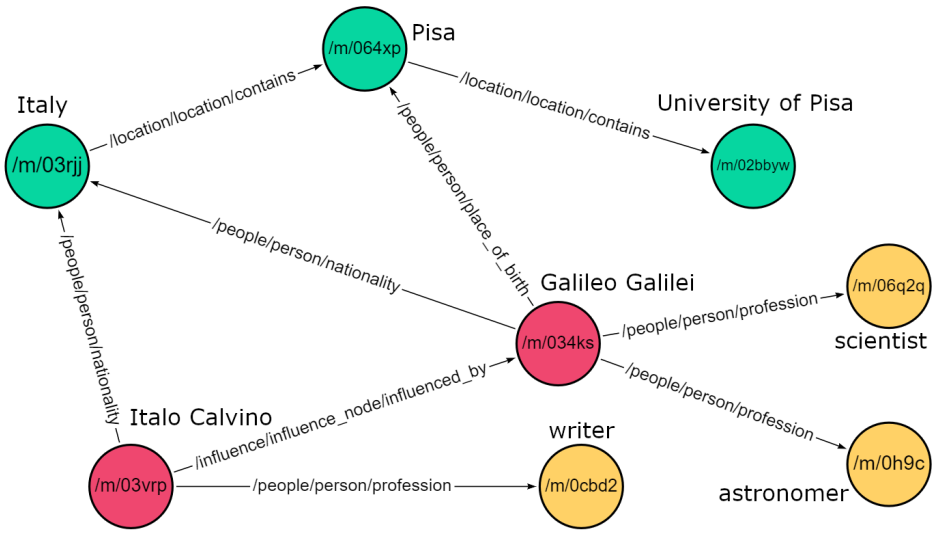


Figure 16: A simple view extracted from FB15K-237.

WN18RR presents a *hierarchical structure* where the different synonyms of a word, represented by lemmas, are linked to a common synset parent node. Each synset is connected to other synsets to model the semantic relationships between different synonym sets. It is possible to group the 11 relationships based on triplets head and tail node types : *lemma-lemma*, *lemma-synset*, *synset-synset*. In addition, a hierarchy between synsets is defined through the `hypernym` relationship. Given a synset it is possible to define a tree which consist in the path of parent hypernym synsets. For each synset, the hypernyms tree presents lemmas as *leaf* nodes and the synset `entity.n.01` as the *root*, considered the root of WN18RR (Figure 15).

5.1.2 FB15K-237

The FB15k dataset (Bordes et al. [7]) contains KG triples and textual mentions of Freebase entity pairs. It has a total of 592,213 triplets with 14,951 entities and 1,345 relationships. FB15k-237 (Toutanova et al. [8]) is a knowledge graph dataset created from FB15k. As for WN18 and WN18RR, in FB15k many triples are inverses that cause leakage from the training to testing and validation splits. FB15k-237 was created to ensure that the testing and evaluation datasets do not have inverse relation test leakage. FB15k-237 dataset contains 310,079 triples with 14,505 entities and 237 relation

Dataset	Entities	Relations	Edges			
			Training	*Validation*	*Test*	**Total**
WN18RR	40,943	11	86,835	3034	3134	93,003
FB15k-237	14,541	237	272,115	17,535	20,466	310,116

Table 2: Number of items which compose the two knowledge graphs used for the experiments.

types (Table 2). Entities in FB15K-237 represent topics about real-world entities such as people, places, and things. In contrast to WN18RR, there is no hierarchical structure among the entities in FB15K-237. Each triple models a real fact defined by the relationship between two topics. Node identifiers are called MIDs (Machine Identifiers) which consist `/m/` followed by a base-32 unique identifier. For example, the unique MID for `Bob Dylan` is `/m/01vrncs`.
Relationships are conceptually arranged in a file directory-like hierarchy `/domain/type/property`. The set of *domains* contains information concerning the same macro-topics: `music/`, `education/`, `sports/`, `people/`. In order to capture the multi-faceted nature of macro-topics, every domain contains different *types* of relationships: `/music/genre`, `/music/artist`, `/music/instrument`. Each *type* contains several relation *properties* that are most commonly needed for describing a particular aspect of information: e.g. `/music/genre/parent_genre`, `/music/genre/artists`.

5.1.3 Wikipedia

HuggingFace datasets[4] include Wikipedia dataset containing cleaned articles of all languages. The dataset is built from the Wikipedia dump with one split per language. For this work, English version `20200501.en` was selected, with $6.4M$ tokenized text rows. Each row contains the corpus of one full Wikipedia article, preprocessed to remove markdown and unwanted sections (e.g. references). To train `BERT-BASELINE` and `BERT-KG`, preprocessing of the dataset was performed to have less computational impact. The tokenization is performed at an earlier stage, before start training process. Each row was tokenized using BertTokenizer based on WordPiece (Wu et al. [48]). Additionally, node labels from WN18RR and FB15K-237 were added to the

[4]https://huggingface.co/datasets/wikipedia

tokenizer dictionary. In this way, if a word in the corpus is associated with an entity in the KG, a single unique integer ID is assigned as input to the BERT embedding layer. Entity names in the text that would be split into multiple tokens are represented by a single token that can be associated with a single KG node, e.g given the sentence *"Joe Biden is the president of U.S."*, the string *"Joe Biden"* is encoded as a single input token `[joe biden]`, rather than the 3 tokens `[joe]`, `[bid]` and `[##en]` following the WordPiece dictionary.

5.2 Experimental Setup

5.2.1 KBGAT

Training Setup

To learn knowledge graph embeddings of WN18RR and FB15K-237, this work employs the same training protocol proposed by Nathani et al. [35] and analyzed in 4.1.1.
The model setup is the same for both KG datasets, except for *batch size* $B = 86835$ for WN18RR and $B = 272115$ for FB15K-237, equal to the number of training triplets.

For a single training step, each node accumulates knowledge from the distant 2-hop neighbors of an entity employing $\ell = 2$ *graph attention layers*. The attention mechanism is composed by 2 *attention heads* in each graph attention layer. A *dropout probability* of $\rho = 0.3$ is applied to attention modules to prevent overfitting.
Initial embedding matrix $\mathbf{H}^0$ is initialized with TransE pre-trained embedding of dimension $d_{h_0} = 50$. The first graph attention layer computes the output matrix $\mathbf{H}^1 \in \mathbb{R}^{N \times d_{h_1}}$, where N is the number of training triplets and $d_{h_1} = 100$. The model output dimension, equals to the KGEs dimension is set to $d_{h_2} = 200$.
The model optimization is performed by Adam optimizer, with *weight decay* $\omega = 5e{-}6$, *learning rate* $lr = 1e{-}3$ and linear scheduler to decay the lr every 500 epochs. The model is trained for 3600 training epochs, equivalent to the iteration steps given the batch size $B = N$. The hinge function (Equation 4.10) is computed with a margin $\tau = 5$ and the valid/invalid triplets ratio is set to 2.

Evaluation Protocol

The KBGAT model is evaluated on the relation prediction task, the aim is to predict a triple (e_i, r_k, e_j) with e_i or e_j missing, i.e., predict e_i given (r_k, e_j) or predict e_j given (e_i, r_k). According to previous works (Bordes et al. [50], Nguyen et al. [55], Dettmers et al. [54], Nathani et al. [35]), a set of $(N-1)$ corrupt triples is generated by replacing for each entity e_i with every other entity $e'_i \in \mathcal{V} \backslash e_i$, and the dissimilarity *score* (presented in Equation 3.2) is assigned to each such triple. Subsequently, the triplets are ordered in ascending order based on the related score and finally the rank of a correct triple $t_* = (e_i, r_k, e_j)$ is computed. The model is evaluated in a *filtered* setting, i.e, during ranking the corrupt triples which are already present in one of the training, validation, or test (except for the triple to test) sets are removed. This whole process is repeated by replacing the tail entity e_j, and averaged metrics are reported. The evaluation metrics used are Mean Reciprocal Rank (MRR), Mean rank (MR) and the proportion of correct entities in the top K ranks (Hits@K) for $K = \{1, 3, 10, 100\}$:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank(t_*^i)} \tag{5.1}$$

$$MR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} score(t_*^i) \tag{5.2}$$

$$Hits@K = \frac{1}{|Q|} \sum_{i=1}^{|Q|} H(K - rank(t_*^i) \tag{5.3}$$

where Q is composed by the N evaluation sets of corrupted triples, *rank* and *score* are the rank and the score of the positive triple in the i-th evaluation set. The function H is a variation of *unit step function*:

$$H(x) = \begin{cases} 1, & 0 \leq x < K \\ 0, & x < 0 \end{cases} \tag{5.4}$$

i.e. given a fact i, Hits@K is 1 if the positive triple t_* is ranked among the top K results, and 0 otherwise. The results are reported in the next section.

5.2.2 BERT-KG and BERT-BASELINE

Training Setup

Both the `BERT-KG` and `BERT-BASELINE` models were trained from scratch on Wikipedia corpus with the same configurations to have a fair comparison. Except for the regularization term $\mathcal{L}_R$, the description below is valid for both models.

The base architecture for both language models is `bert-base-uncased` from HugginFace library, composed by $L = 12$ layers, each containing 12 attention heads. The input and output dimensions for BERT Encoder is $d_{in}, d_{out} = 512$, where d_{in} represents the number of input tokens in the BERT's input embedding layer. The output dimension d_{out} reflects the number of contextualized word-embeddings in $\mathbf{Z_w}$ and the number of KGEs in $\mathbf{H_w}$. The hidden dimension $d_{hidden} = 768$ represents the number of features for a single word-embedding $\mathbf{z}_i \in \mathbf{Z_w}$ and the input dimension for the regression model, implemented by a linear layer (`torch.nn.Linear`). The linear layer parameters represent the transformation matrix $\mathbf{A}$ applied to the word-embeddings matrix $\mathbf{Z_w}$.

For each of the 2 GPUs used in the training process, a batch size $B = 8$ was chosen, or 16 input sentences processed by the model in parallel, for a total of $16 * 512 = 8192$ processed at each iteration. A small batch size slows down the training but at the same time, allows the effect of regularization to be highlighted and be more accurate. Given an input sentence, the word-embeddings that could be associated with entities in the KGs may vary. In some case, a sentence has very few tokens that can be mapped to a node. To prevent an unbalanced ratio between MLM loss $\mathcal{L}_{MLM}$ and regularization loss $\mathcal{L}_R$, it was preferred to update the BERT parameters in small steps. We leave the analysis of competitive batch sizes with pre-trained languages in the literature for future developments of this work.
For `BERT-KG`, the tuning parameter $\tau = 10$ is applied to the regularization term. The WSD threshold $s = 0.3$ is set in order to filter candidate nodes that are not relevant for the given input token.
The language models are trained for 2 epochs (6.4M rows/8 batch size = 800k steps for each GPU). BERT models are optimized by AdamW with learning rate $lr = 5e - 5$.

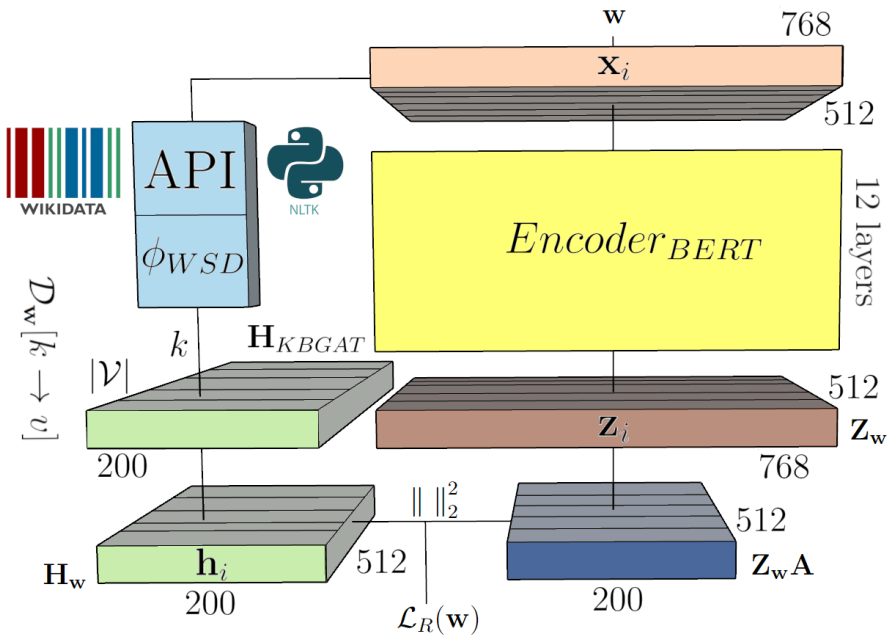


Figure 17: High-level overview of `BERT-KG`. The numbers represent the embedding dimensions for each embeddings matrix. The building block composed by Wikidata/NLTK API extracts candidate nodes, filtered by the WSD mechanism to computed the sentence KGE sentence dictionary given the embeddings matrix $\mathbf{H}_{KBGAT}$ previously computed by KBGAT.

Implementation Details

To build the KGEs sentence dictionary $\mathcal{D}_{\mathbf{w}}$, the framework implement a search mechanism that takes as input a word in string format and returns the set of candidate nodes $\mathcal{S}_i$ (Section 4.1.2). To perform this task, API methods were adopted for each of the two knowledge graphs (Figure 17).
For WN18RR, the framework uses the NLTK library, one of the leading Python toolkits for NLP. NLTK implements a corpus reader for WordNet `nltk.corpus.wordnet`[5], offering several methods to explore and query the KG. In particular, the `synsets()` method returns the set of synsets that can be associated with the text string parameter.

[5]https://www.nltk.org/howto/wordnet.html

Unfortunately, Freebase has been deprecated and has no direct interface. The problem has been remedied thanks to another popular KG, Wikidata[6] containing all the entities present in FB15K-237. Similar to NLTK, Wikidata offers an API web method that returns the set of identifiers given an input word. Each Wikidata entity is called `item` and identified by the prefix `Q`, e.g. Bob Dylan is identified by `Q392`. Each item has properties, identified by the prefix `P`. In particular, for entities common with Freebase, the `Freebase ID` property is defined and identified by `P646`. Before training BERT-KG, an additional lookup table is constructed to associate Wikidata items with FB15K-237 MIDs. In this way, for each input token it is possible to get the Wikidata items with a single API call and retrieve the candidate MIDs querying the lookup table.
For both KGs, the results of each query are cached to minimize API requests that would otherwise have a major impact on training time. In addition to further optimize, for each batch the API requests are executed in parallel by separate threads equal to the number of items in the batch. The regularization procedure is summarized in Algorithm 1.

Algorithm 1 BERT-KG regularization algorithm

Require: sentence tokens matrix $\mathbf{X_w}$, KBGAT embeddings matrix $\mathbf{H}$, linear layer A, BERT's parameters θ

$\mathbf{X}_\Pi, \mathbf{X}_{-\Pi} \leftarrow masking(\mathbf{X_w})$
$\mathbf{Z_w} \leftarrow Encoders(\theta, \mathbf{X_w})$
for each $\mathbf{x}_i \in \mathbf{X_w}$ **do**
    $\mathcal{S}_i \leftarrow API(\mathbf{x_i})$
    $i \leftarrow WSD(\mathbf{X_w}, \mathcal{S}_i)$
    $\mathbf{H_w}.add(\mathbf{H}[i])$
end for
$\mathcal{L}_R \leftarrow ||\mathbf{H_w} - \mathbf{A}\mathbf{X_w}||$
$\mathcal{L}_{tot} = \mathcal{L}_{\text{MLM}}\left(\mathbf{X}_\Pi \mid \mathbf{X}_{-\Pi}, \theta\right) + \mathcal{L}_R$
backward$(\theta, \mathcal{L}_{tot})$

It is possible to exclude overfitting given the limited number of epochs, consequently an evaluation protocol was not considered during language

[6] www.wikidata.org/

models training. The performance of `BERT-KG` and `BERT-BASELINE` was analyzed during comparison on the probing tasks, described below.

5.3 Language Model Probing Tasks

The proposed experiments aim to evaluate the influence of regularization on language models with respect to two knowledge graphs, rather than outperforming the results obtained from pre-trained models. The comparison is made between `BERT-BASELINE` versus regularized `BERT-KG` with WN18RR and FB15K-237. Both models were trained with fewer resources than the standard pre-trained models, given the computational and time constraint and with smaller training datasets (6.4M for BERT-KG versus 3.3B for `bert-based-uncased` proposed in Devlin et al. [16]). To compare `BERT-KG` with the custom baseline, two *probing tasks* are proposed. The first tests the similarity of augmented word integration with symbolic knowledge between word pairs, given a similarity score. The second assesses the ability of the language models in the KB completion task. The first part of this section presents an overview of the datasets used in the two probing tasks, MEN dataset (Bruni et al. [18]) for probing similarity and Stanford Question Answering Dataset (SQuAD, Rajpurkar et al. [87], Rajpurkar et al. [88], F. Petroni and Riedel [19]) for the KB completion task. Finally, the evaluation protocols for the two proposed probing tasks are described. All the results are presented in the next Section.

5.3.1 Datasets

The *MEN Test Collection* (Bruni et al. [18]) is a similarity dataset containing 3000 pairs of words where each pair is a crowdsourced human-assigned similarity score between 0 (low similarity) and 50 (high similarity). In Table 3, the first and the last 5 rows of MEN are reported.

The *Stanford Question Answering Dataset* (SQuAD, Rajpurkar et al. [87]) is a collection of question-answer pairs derived from Wikipedia articles. In SQuAD, the correct answers of questions can be any sequence of tokens in the given text. Because the questions and answers are produced by humans through crowdsourcing, it is more diverse than some other question-answering datasets. SQuAD 1.1 contains 107,785 question-answer pairs on 536 articles. The latest version SQuAD2.0, extends SQuAD1.1 with over

Word 1	Word 2	Similarity Score
sun	sunlight	50
automobile	car	50
river	water	49
stair	staircase	49
morning	sunrise	49
feather	truck	1
festival	whisker	1
muscle	tulip	1
bikini	pizza	1
bakery	zebra	0

Table 3: View of MEN dataset with the first and the last 5 rows composed of triple (word1, word2, similarity_score).

50,000 un-answerable questions written adversarially by crowdworkers in forms that are similar to the answerable ones (Rajpurkar et al. [88]). For KB completion task, the SQuAD subset employed in *LAnguage Model Analysis* probe (LAMA, F. Petroni and Riedel [19]) is selected, composed by a subset of 305 context-insensitive questions from the SQuAD development set with single token answers.

5.3.2 Word Similarity

The similarity probe task using MEN dataset is usually performed by calculating the cosine similarity between two word-embedding pairs in the dataset (Zhou et al. [15]). Since the proposed model cannot compete with pre-trained models, the task is modified by removing cosine similarity. The augmented word-embeddings may not have learned the spatial relationships related to the cosine of the vectors they represent. Transfer symbolic knowledge into embeddings does not ensure that cosine similarity is maintained between word-embeddings.
The probe thus consists of a regression task on the similarity score between two word-embeddings: recalling Table 3, given as input sun and sunlight word embeddings, the regression model tries to predict the similarity score 50. Due to the limited number of training data, the regression model train-

ing and evaluation follow a k-fold cross validation scheme. In particular, the initial setup was to employ Leave One Out (LOOCV, N regression models trained on $N-1$ training data, where N is the size of the training set, and and tested on the single discarded data point in the training set) but it was noted during testing that training 3000 models would be too time consuming. Values of $k = \{10, 20, 50, 100\}$ are then chosen. At first, word-embeddings derived from the full model (last checkpoint) are evaluated and the results for different values of k are compared. Then an incremental comparison is performed for each language models checkpoint keeping the optimal value of k fixed. The metrics for the comparison between `BERT-KG` and `BERT-BASELINE` are the standard regression metrics MSE, R2, and Spearman correlation coefficient ρ:

$$\mathrm{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{5.5}$$

$$\mathrm{R2}(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{5.6}$$

$$\rho(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{6 \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n\,(n^2 - 1)} \tag{5.7}$$

where $\mathbf{y} = \{y_i\}_{i=1}^{n}$ is the ground-truth similarity scores in MEN dataset and $\hat{\mathbf{y}} = \{\hat{y}_i\}_{i=1}^{n}$ are the predicted score by the regression model. n is the number of test scores and $\bar{y}$ is the mean of ground-truth scores. The results are collected for each K regression model, then the average and the standard deviation is computed. In addition, the score achieved by a pre-trained model `bert-base-uncased` is added for the sole purpose of validate the modified probing task.
The word similarity probing task is word-level, generating out-of-context word-embeddings without an input context sentence.

5.3.3 Knowledge Base Completion

In order to evaluate more specifically the amount of symbolic knowledge transferred in word embeddings, the Knowledge Base Completion (KBC, [56]) probing task proposed in F. Petroni and Riedel [19] is performed. The task consists in querying a language model for factual knowledge as cloze test (Taylor [49]), a process similar to the one applied for MLM during BERT pre-training (Section 2.4.4).

Each of the 305 sentence-facts derived from SQuAD is converted into a cloze statement, masking the missing token with the special token [MASK]. The objective of the probed language models is to predict the masked token to complete the sentence fact. The probe input consists of a question-answer pair, where the answer contains a masked token, e.g in the tuple ("Who developed the theory of relativity?","The theory of relativity was developed by `[MASK]`"), the language models should predict the token "Einstein" or "Albert Einstein".
The evaluation follows the same scheme used for standard KBC, the same employed in the KBGAT training evaluation and originally proposed in Bordes et al. [7]. The metrics used are the same as those proposed by the authors in F. Petroni and Riedel [19], MRR (Equation 5.1) and P@K (equivalent to Hits@K described in Equation 5.3):

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}\left(token_*^i\right)} \tag{5.8}$$

$$\text{P@}K = \frac{1}{|Q|} \sum_{i=1}^{|Q|} H\left(K - \text{rank}\left(token_*^i\right)\right) \tag{5.9}$$

where Q is the set of evaluation queries (facts), $token_*^i$ the predicted token for the i-th query and H is the function defined in Equation 5.4. In this setup, a better result is expected from `BERT-KG` regularized with FB15K-237 since the goal of the probe being to predict real-word entities word-embeddings.

6 Results and Analysis

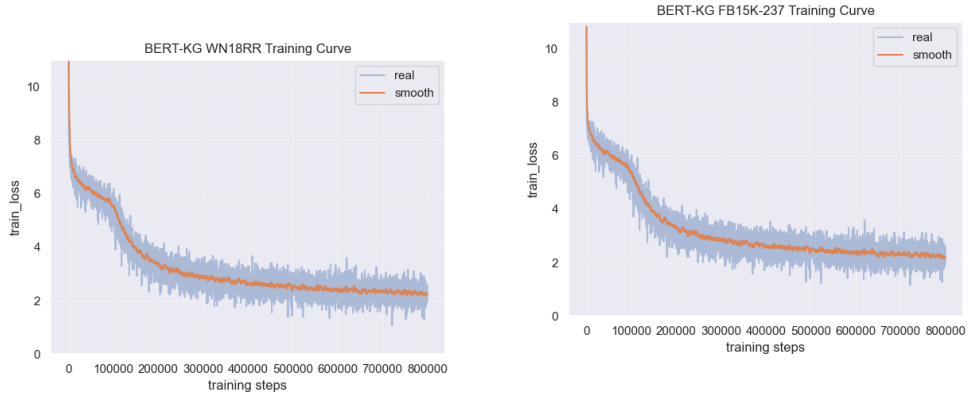
The results of the above experiments are shown below. In the first part, the evaluation results of the DGN KBGAT models and the data collected during BERT-KG training are reported. Next, the evaluation comparison of the regularized model `BERT-KG` against the baseline `BERT-BASELINE` is presented in detail. The results for the pre-trained model `bert-base-uncased` trained by Google are reported, not in order to achieve and compare the same results but to validate the tests. At the same time, the pre-trained results contextualize the proposed work with respect to existing models, helping to consider possible developments of BERT-KG with the same resources.

6.1 KBGAT Evaluation

Following the evaluation protocol defined by KBGAT authors and described in Section 5.2.1, in Table 4 the evaluation metrics for both knowledge graphs employed in the KBGAT training are reported. As expected, the results are the same presented by the authors in Nathani et al. [35] which outperform state-of-the-art models such as ConvE (Dettmers et al. [54]), TransE (Bordes et al. [50]), ConvKB (Nguyen et al. [55]), and R-GCN (Schlichtkrull et al. [34]), as highlighted by the authors. The computed knowledge graph embeddings can be considered well-structured representations of the symbolic knowledge in the KGs, being able to predict the positive test triplets with a high hit rate ($33.5\% - 61.7\%$ WN18RR and $43.9\% - 82.3\%$ FB15K-237) and with a high ranking compared to negative triplets in evaluatioon test sets. This framework employs exactly the same hyperparameters and data in order to compute well-structured node-embeddings and applying an extensively tested and evaluated methodology allowing to focus on regularizing the language models.

Dataset	**Hits@1**	**Hits@3**	**Hits@10**	**Hits@100**	**MRR**	**MR**
WN18RR	0.335	0.431	0.512	0.617	0.397	4411.41
FB15K-237	0.439	0.552	0.654	0.8231	0.514	152.62

Table 4: Evaluation metrics for trained `BERT-KG` on datasets WNRR18 and FB15K-237.



(a) (b)

Figure 18: Training loss of `BERT-KG` during regularization process with (a) WN18RR and (b) FB15K-237. The plots report only the MLM loss without the added regularization term. `BERT-KG` loss converges to reasonable values that exclude possible overfitting with respect to the embedding regression using the linear layer.

6.2 Language Model Training

BERT-KG and BERT-BASELINE were trained for 2 epochs (800k training steps) was approximately 20 days per individual model, both regularized and unregularized. This led to a poor analysis of the hyperparameters and model selection, in particular it did not make it possible to analyze the impact of the tuning of the regularization parameter *tau*. The default value for the tuning parameter was set to $\tau = 10$. The model is able to achieve satisfactory results even without particular attention in the choice of τ. A study of the effect of hyperparameters on model training is left for future development.
During `BERT-KG` training, the percentage of regularized word-embeddings with respect to the total input tokens (512) was monitored. Regularizing the model on WN18RR, the percentage of node-embeddings obtained from the API-WSD process is on average 24.6%, while for FB15k-237 is 17.8%. The first reason is simply that FB15K-237 contains only 15k nodes which are few compared to WN18RR. In addition, in FB15K-237 several nodes are related to famous people, cities, events. Most of the words in the Wikipedia corpus are common names and therefore the word-embeddings are more relatable to a lexical knowledge graph such as WN18RR. In addition, the WSD threshold

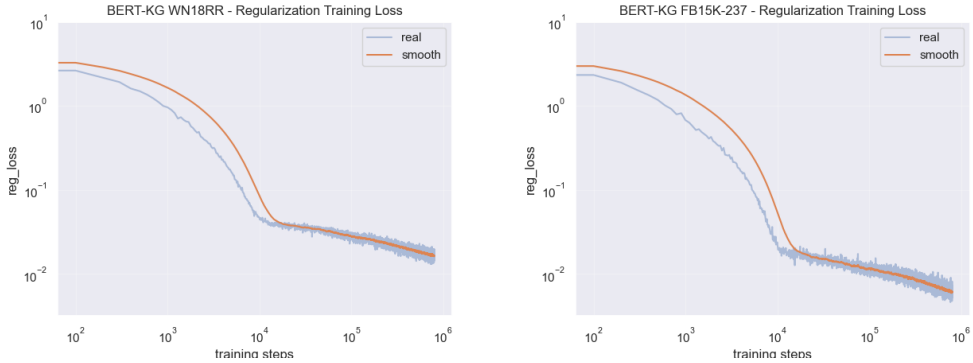


Figure 19: Log-log scale charts reporting the training regularization losses compute by BERT-KG for WN18RR and FB15K-237. The loss is equal to the Mean Squared Error between knowledge graph node-embeddings and the projections of word-embeddings computed by BERT model.

$s = 0.3$ may have been set too high and a further analysis of the optimal value of the hyperparameter is needed.

In order to visualize the impact of the regularization, in Figure 19, it is reported the charts of regularization loss convergence with respect to the training steps for both KGs. First, the regularization loss converges very fast in the first few iterations (10^4), and then stabilizes near 0 for the rest of the training. This may suggest that BERT learns quickly to predict node-embeddings. Another possibility is that the linear layer can learn the optimal parameters for regression, but this conflicts with the assumption that a simple linear layer can model symbolic knowledge of node embeddings. In addition, `BERT-KG` learning curve (Figure 18) shows convergence on MLM training loss values, suggesting that the model is updating its parameters in order to minimize the language modeling loss rather than embedding regression regularization term. Given these considerations it is possible to suppose that BERT succeeds easily to transfer the symbolic knowledge in the word-embeddings. On the other hand, the tuning parameter τ is not updated during training. High values of τ parameter might cause too dramatic an increase in the loss due to the regularization loss, leading to a divergence of the training loss. At the same time, a static value of tau might not adjust the BERT parameters for optimal regression on node-embedding, optimizing the objective over regression to a local minimum. Configuring a scheduler for

Model	metric	10-fold	20-fold	50-fold	100-fold
BERT-BASELINE	**MSE**	0.044	**0.039**	0.044	0.045
	R2	**0.312**	0.305	0.309	0.267
	ρ	0.573	**0.601**	0.572	0.554
BERT-KG WN18RR	**MSE**	0.039	**0.036**	0.041	0.042
	R2	0.364	**0.362**	0.349	0.319
	ρ	0.605	**0.652**	0.623	0.581
BERT-KG FB15K-237	**MSE**	0.043	**0.039**	0.043	0.044
	R2	**0.311**	**0.311**	0.307	0.288
	ρ	0.589	**0.614**	0.583	0.563

Table 5: MEN probing task: k-fold selection results. The regression models are evaluated on MSE, R2 and Spearman ρ. The values in the table highlight how a low number of folds k compared to those evaluated achieve better results on the metrics, in particular the best results are for $k = 20$. The detailed results for $k = 20$ are reported in Table 6. Results in bold are the highest metric values.

the regularization tuning parameter may be future development, to increase the influence of the regularization loss when it approaches 0.
The last significant aspect is the difference in the value reached by the training regularization loss of WN18RR and FB15K-237, as shown in Figure 19. Using WN18RR, the loss stabilizes at a higher level compared to FB15K-237, due to the lower number of tokens regularized by FB15K-237 and consequently a lower, albeit slight, impact in the transfer of symbolic knowledge.

6.3 MEN Word Similarity

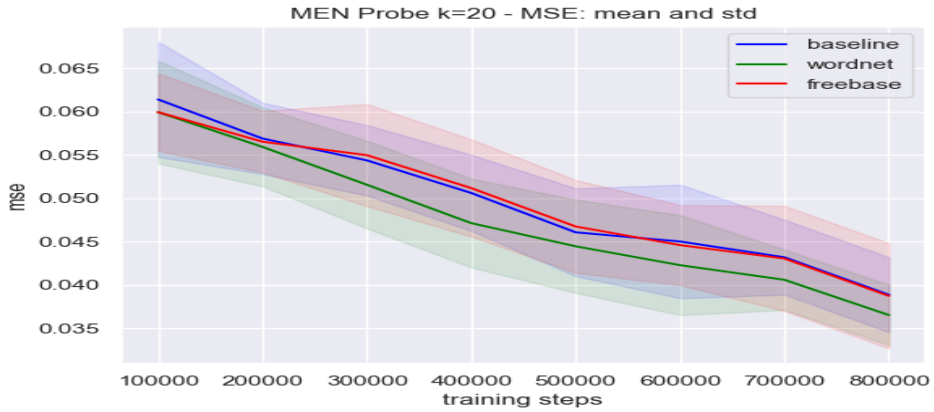
The first step in performing the word similarity probing task is to choose an optimal k for the regression model with k-fold cross validation. An analysis was performed with respect to the values of $k = \{10, 20, 50, 100\}$. The results are reported in Table 5. Considering the obtained results , $k = 20$ was chosen as the parameter for the incremental performance analysis for

Model	MSE		R2		ρ	
20-fold 800k	mean	std	mean	std	mean	std
BERT-BASELINE	0.039	0.004	0.305	0.075	0.601	0.059
BERT-KG WN18RR	**0.036**	0.003	**0.362**	0.076	**0.652**	0.069
BERT-KG FB15K-237	0.039	0.006	0.311	0.112	0.614	0.087
bert-base-uncased	0.017	0.003	0.714	0.079	0.854	0.034

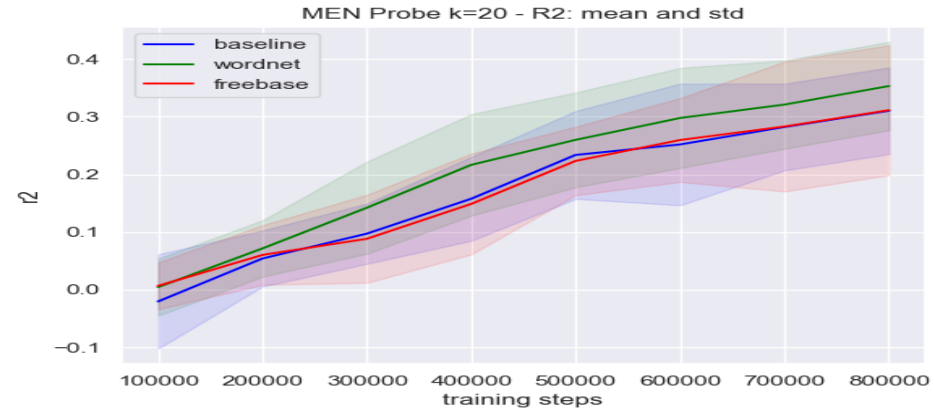
Table 6: Mean and standard deviation of regression metrics for 20-fold cross validation. The best results over all the three metrics are achieved by the regularized model employing WN18RR dataset. In addition, the results for pre-trained `ber-base-uncased` are reported in the comparison.

language models checkpoints. The incremental performance analysis consists into perform the probing tasks at different times during the training. In this way, it was possible to monitor whether the regularization had any effect without having to wait for the completion of the training process. Moreover, analyzing each checkpoint allowed to verify if the regularization could degrade performance or cause overfitting. For the MEN probing task, the incremental performance analysis starts from checkpoint 100k and it was performed every 100k training step. This choice is justified by the fact that the MEN task with k-fold cross-validation generates K different regression models. Probing each checkpoint would be too time-consuming. The same results reported in Table 5 are compared between the different models proposed in this thesis in Table 6, including the standard deviation of the results obtained from the 20 regression models at the 800k iteration. The results obtained by running the same regression test with the pre-trained `bert-base-uncased` model are reported. The pre-trained model was trained on larger datasets for a total on 3.3B rows, extracted from Wikipedia and BookCorpus dataset (Dettmers et al. [89]). In addition, the pre-trained model was trained on more epochs for a total of 1M steps with batch size B=256.

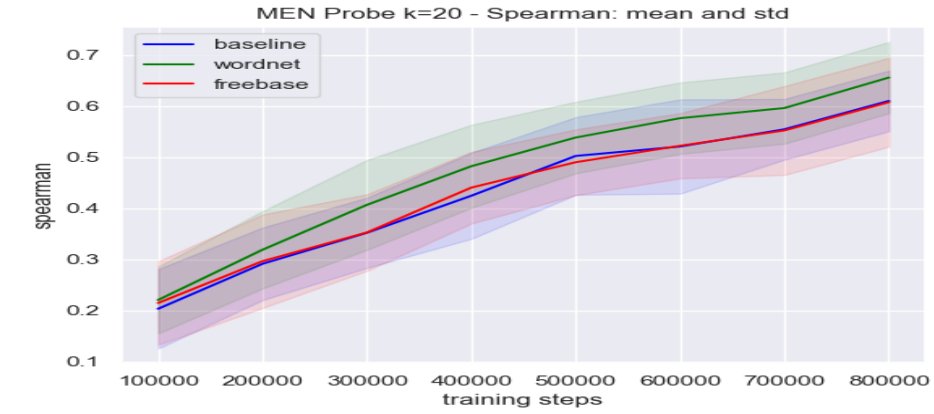
As expected, WN18RR regularization is much more effective than FB15K-237. The higher number of regularized word-embeddings and the KG WN18RR domain show an improvement in single word-embeddings without a sentence context, such as the MEN task. The incremental analysis performed on each checkpoint of the language models, visualized in Figures 20a, 20b and 20c effectively show how regularization with FB15K-237 has no influence on the proposed probing task, while WN18RR achieves competitive results. Results



(a)



(b)



(c)

Figure 20: Mean and standard deviation for (a) MSE, (a) R2 and $(c)\rho$ regression metrics. The incremental analysis was performed starting from checkpoint 100k to last checkpoint at 800k training steps.

Model	MRR	P@10
BERT-BASELINE	0.070	0.172
BERT-KG WN18RR	0.071	0.174
BERT-KG FB15K-237	**0.074**	**0.184**
bert-base-uncased	0.47	0.25

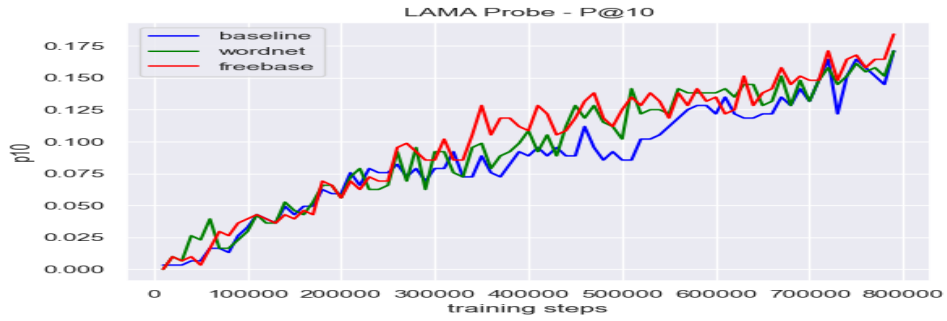
Table 7: Metrics related to KBC SQuAD probing task. The results suggest that the models don't assign high rank for the word-embedding to predicts, but under the same assumptions, the regularized model on KG FB15k-237 achieves better results.

on the MEN task demonstrate how a larger dictionary of knowledge graph embeddings increases the effect of word-level regularization.
The larger number of nodes in WN18RR implies a larger number of regularized word-embeddings with knowledge graph embeddings over the total number of input tokens. The standard deviation values combined with the comparison with `bert-base-uncased` confirms the validity of the probing task. It is possible to state that semantic relations defined in WN18RR improve transformers-based language model's performances in the assignment of recognizing the similarity of two individual words in a dictionary.

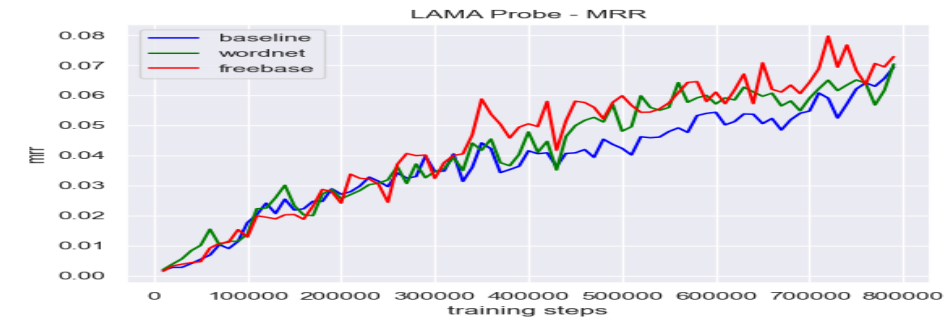
6.4 SQuAD KBC

The proposed KBC task fits better the domain of KG FB15K-237. Contrary to what was expected from the previous probing task, the results obtained invalidate the hypothesis that FB15K-237 did not affect the language modeling process. The results on fully trained models (800k training steps) are reported in Table 7. Regularizing BERT-KG through embedding-regression with real-world entity nodes seems to increment the correct prediction for the masked word-embeddings in KBC queries. P@10 results suggest how the regularized model employing FB15K-237 predicts the correct token for 18% of the test queries, but MRR values underline that the rank of the correct token is low with respect to top 10 predicted elements. The models predict the masked token in the top 10 items for the 18% of test queries, given the competitive values of P@10 but at the same time when this occurs, given the results on the MRR. The incremental analysis performed on the checkpoints

of the 3 compared models further highlights the good performance of BERT-KG regularized with FB15K-237. The results are presented in Figures 21a and 21b. Despite the similar results to BERT-BASELINE, the performance on the KBC task of BERT-KG is also dependent on possible unregularized word-embeddings that make up the in-put question and answer. It is possible to argue that the addition of a KG-based regularization improves the performance of the language models in the KBC task. At the same time, the low performance gap between BERT-KG and BERT-BASELINE suggests further analysis to assess in more detail the effect of regularized word-embeddings while excluding the influence of possible unregularized embeddings.



(a)



(b)

Figure 21: Incremental analysis for (a) P@10 and (b) MRR metrics for LAMA SQuAD probing task. The results underline a prediction percentage in the top 10 rank for about 18% of the test queries. BERT-KG FB15K-237 slightly outperforms the other models in KBC probing task, despite the lower KGEs dictionary dimension.

7 Conclusions

Knowledge Graphs combined with Deep Graph Networks succeed in representing symbolic knowledge defined on an ontology in an effective and resource-efficient way, leading to optimal results in a short time. This thesis work analyzed the possibility of transferring symbolic knowledge in the pre-training process of a transformer-based language model, leading to satisfactory results, leaving several insights for more focused analysis.
From the obtained results it is possible to observe that the MLM regularization term was able to transfer part of the symbolic knowledge into the parameters of the BERT language model. It was possible to employ a dictionary of knowledge graph embeddings previously calculated and then used at a later time for the training of language models. Experiments confirm how KG-based regularization improves the semantic context of word-embeddings due to KGEs representations computed by Deep Graph Networks. The proposed regularization term allows transferring the structured context of knowledge graphs into word-embeddings. This concept can be extended to other deep learning models, encoding symbolic knowledge as knowledge graph embeddings and transferring it into multi-domain embeddings. One key aspect of the proposed framework is the direct addressing of knowledge graph embeddings, allowing symbolic knowledge to be pre-computed, and transferred to multiple models, without particularly impacting the training of the model to be regularized. Another aspect that emerged from the analysis is how regularization through embedding regression is a good method to transfer structured context into word embeddings of language models. From the results, it can be deduced that the number of regularized word-embeddings affects the context of individual words learned by the language model. The number of nodes in the knowledge graph can affect the effect of regularization. At the same time, even with a smaller number of knowledge graph embeddings (FB15K-237), an improvement in the performance in domain-specific tasks was detected. Given the different results obtained from two regularized models on two different knowledge graphs concerning the proposed probing tasks, it is possible to argue that regularization is strongly related to the ontological domain underlying the KG employed in regularization. By using a lexical knowledge graph such as WN18RR, the effects of regularization on a word-level task such as the MEN task were more highlighted. A knowledge graph containing real-world facts such as FB15K-237 allows for reaching better results over the baseline on the KBC SQuAD task,

where regularized word-embeddings are contextualized within multiple sentences. This suggests possible future development to analyze the influence of KG-regularization on multiple graphs in language modeling. Using multiple knowledge graphs with heterogeneous domains in the regularization process is one of the main future developments that emerged from this thesis work. The transfer of symbolic knowledge to representation learning models could allow increasing the performance on different tasks in different domains. On the other hand, transformers are models with a high number of parameters, and performing a thorough analysis of the influence of node embeddings is time-consuming, resource-intensive, and non-trivial. It was observed that graph-driven regularization does not degrade the performance of the language models. Moreover, knowledge graphs are useful to solve the Word Sense Disambiguation problem, one of the subtasks required for the regularization of `BERT-KG`.

Further experiments could use probing task metrics as terms of the BERT regularization itself, with the risk, however, of overfitting word-embeddings on probing tasks, rather than generalizing a language model. Another interesting aspect could be BERT's ability to perform word sense disambiguation for the model itself to choose optimal candidate nodes to map to the input word-embeddings. Analyzing whether BERT's attention mechanism can select node-embeddings given the context of the input sentence to perform the regularization itself is a topic for future research. Finally, the attention mechanisms of language models could be used for context transfer, replacing the embedding regression method applied in this work.

Applications of node-embeddings in NLP are limited to improving fine-tuning of language patterns over specific domains. The ability to transfer structured information between low-dimensional embeddings to improve language models could be the first step in reducing the high number of transformer parameters. The loss of context due to the reduction of language model parameters could be balanced by transferring the symbolic knowledge context of knowledge graphs.

References

[1] Pietro Barbiero, Ramon Viñas Torné, and Pietro Lió. Graph representation forecasting of patient's medical conditions: Toward a digital twin. *Frontiers in Genetics*, 12, 2021. ISSN 1664-8021. doi: 10.3389/fgene.2021.652907. URL https://www.frontiersin.org/article/10.3389/fgene.2021.652907.

[2] Jose M. Gutierrez, Michael Jensen, and Tahir Riaz. Applied graph theory to real smart city logistic problems. *Procedia Computer Science*, 95:40–47, 2016. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2016.09.291. URL https://www.sciencedirect.com/science/article/pii/S1877050916324632. Complex Adaptive Systems Los Angeles, CA November 2-4, 2016.

[3] A. Ruiz-Frau, A. Ospina-Alvarez, S. Villasante, P. Pita, I. Maya-Jariego, and S. de Juan. Using graph theory and social media data to assess cultural ecosystem services in coastal areas: Method development and application. *Ecosystem Services*, 45:101176, 2020. ISSN 2212-0416. doi: https://doi.org/10.1016/j.ecoser.2020.101176. URL https://www.sciencedirect.com/science/article/pii/S2212041620301182.

[4] Támas Hegedűs, Balázs Németh, and Péter Gáspár. Graph-based multi-vehicle overtaking strategy for autonomous vehicles. *IFAC-PapersOnLine*, 52(5):372–377, 2019. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2019.09.060. URL https://www.sciencedirect.com/science/article/pii/S2405896319306822. 9th IFAC Symposium on Advances in Automotive Control AAC 2019.

[5] Michael Färber. *Semantic search for novel information*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2017. URL https://d-nb.info/1137230576.

[6] C. Fellbaum. Wordnet: An electronic lexical database. *MIT Press*.

[7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. 2013. URL https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[8] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1174. URL `https://aclanthology.org/D15-1174`.

[9] R. Matthew. The wikipedia data revolution. *The Wikimedia Foundation*, 2012. URL `https://diff.wikimedia.org/2012/03/30/the-wikipedia-data-revolution/`.

[10] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z Ives. Dbpedia: A nucleus for a web of open data. *The Semantic Web, Springer Berlin Heidelberg*, 2007.

[11] F. M. Suchanek and G. Weikum. Yago: A core of semantic knowledge unifying wordnet and wikipedia. 2007.

[12] D. Bacciu, F. Errica, and A. Micheli. Contextual graph markov model: A deep and generative approach to graph processing. *CoRR*, 2018. URL `http://arxiv.org/abs/1805.10636`.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. 30, 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[14] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409, 09 2014.

[15] W. Zhou, H. Hong, and Z. Zhou. Derive word embeddings from knowledge graph. 2019.

[16] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

[17] P. Velickovic, G. Cucurull, P. Lio A. Casanova, A. Romero, and Y. Bengio. Graph attention networks. *In Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[18] Elia Bruni, Nam Khanh Tran, and Marco Baroni. Multimodal distributional semantics. *J. Artif. Int. Res.*, 49(1):1–47, jan 2014. ISSN 1076-9757.

[19] A. H. Miller P. Lewis A. Bakhtin Y. Wu F. Petroni, T. Rocktäschel and S. Riedel. Language models as knowledge bases? In *In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2019*, 2019.

[20] J. A. Bondy and U. S. R. Murty. Graph theory with applications. *Elsevier, New York*, 1976.

[21] D. Bacciu, F. Errica, A. Micheli, and M. Podda. A gentle introduction to deep learning for graphs. *CoRR*, abs/1912.12693, 2019. URL `http://arxiv.org/abs/1912.12693`.

[22] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks, 9(5): 768–786*, 1998.

[23] E. W. Schneider. Course modularization applied: The interface system and its implications for sequence control and data analysis. *In Association for the Development of Instructional Systems (ADIS), DOI: 10.1037/e436252004-001 1, 158, 161*, 1973.

[24] A. Singhal. Introducing the knowledge graph: Things, not strings. *Google Official Blog*, 2012. URL `blog.google/products/search/introducing-knowledge-graph-things-not`.

[25] Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1: Concepts and abstract syntax. *World Wide Web Consortium (W3C)*, 2014. URL `https://www.w3.org/TR/rdf11-concepts/`.

[26] V.K. Chaudhri, N. Chittar, and M. Genesereth. Knowledge graphs: Data models, knowledge acquisition, inference and applications. *Department of Computer Science, Stanford University*, 2021. URL `https://web.stanford.edu/class/cs520/`.

[27] X. Zhang, L. Liang, L. Liu, and M. Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in Genetics*, 12,

2021. ISSN 1664-8021. doi: 10.3389/fgene.2021.690049. URL https://www.frontiersin.org/article/10.3389/fgene.2021.690049.

[28] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks, 8(3): 714-735*, 1997.

[29] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks, 20(3): 498–511*, 2009.

[30] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks, vol. 20(1): 61-80*, 2009.

[31] C. Gallicchio and A. Micheli. Graph echo state networks. *The 2010 International Joint Conference on Neural Networks (IJCNN), 1-8*, 2010.

[32] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated graph sequence neural networks. *In Proceedings of the 4th International Conference on Learning Representations, (ICLR)*, 2016.

[33] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *In Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[34] M. Schlichtkrull, R. van den Berg I. Titov T. N Kipf, P. Bloem, and M. Welling. Modeling relational data with graph convolutional networks. *In Proceedings of the 15th European Semantic Web Conference (ESWC), 593–607*, 2018.

[35] D. Nathani, J. Chauhan, C. Sharma, and M. Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. *CoRR*, abs/1906.01195, 2019. URL http://arxiv.org/abs/1906.01195.

[36] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. *Advances in neural information processing systems*, 30, 2017.

[37] Y. Wu, D. Lian, Y Xu, L. Wu, and E. Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. *Proceedings of the AAAI Conference on Artificial Intelli-*

gence, 34:1054–1061, Apr. 2020. doi: 10.1609/aaai.v34i01.5455. URL https://ojs.aaai.org/index.php/AAAI/article/view/5455.

[38] M. Engelcke I. Posner E. Wagstaff, F. B Fuchs and M. Osborne. On the limitations of representing functions on sets. *Proceedings of the 36th International Conference on Machine Learning (ICML), 6487–6494*, 2019.

[39] J. Alammar. The illustrated transformers. 2018. URL http://jalammar.github.io/illustrated-transformer/.

[40] M. Rameez. Mitigating unintended bias in masked language models. *Master thesis, Computer Science, Université de Lorraine*.

[41] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

[42] E Ramos-Pérez, P. J. Alonso-González, and José Javier Núñez-Velázquez. Multi-transformer: A new neural network-based architecture for forecasting sp volatility. *Mathematics*, 9(15):1794, Jul 2021. ISSN 2227-7390. doi: 10.3390/math9151794. URL http://dx.doi.org/10.3390/math9151794.

[43] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu, and W. Gao. Pre-trained image processing transformer. *CoRR*, abs/2012.00364, 2020. URL https://arxiv.org/abs/2012.00364.

[44] L. Rasmy, Y. Xiang, Z. Xie, C. Tao, and D. Zhi.

[45] M. I. Jordan. Serial order: A parallel distributed processing approach. *Neural-Network Models of Cognition*, 1986. ISSN 0166-4115. doi: https://doi.org/10.1016/S0166-4115(97)80111-2. URL https://www.sciencedirect.com/science/article/pii/S0166411597801112.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[47] Jimmy Ba, Jamie Kiros, and Geoffrey Hinton. Layer normalization. 07 2016.

[48] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL `http://arxiv.org/abs/1609.08144`.

[49] Wilson L. Taylor. "cloze procedure": A new tool for measuring readability. *Journalism & Mass Communication Quarterly*, 30:415 – 433, 1953.

[50] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf`.

[51] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1112–1119. AAAI Press, 2014.

[52] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2181–2187. AAAI Press, 2015. ISBN 0262511290.

[53] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[54] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476, 2017. URL `http://arxiv.org/abs/1707.01476`.

[55] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. 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 327–333, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2053. URL `https://aclanthology.org/N18-2053`.

[56] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, page 926–934, Red Hook, NY, USA, 2013. Curran Associates Inc.

[57] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *CoRR*, abs/1706.05674, 2017. URL `http://arxiv.org/abs/1706.05674`.

[58] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *CoRR*, abs/1707.06690, 2017. URL `http://arxiv.org/abs/1707.06690`.

[59] Stanley Kok and Pedro M. Domingos. Statistical predicate invention. In *ICML '07*, 2007.

[60] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. *CoRR*, abs/1809.05679, 2018. URL `http://arxiv.org/abs/1809.05679`.

[61] Chenguang Wang, Xiao Liu, and Dawn Song. Language models are open knowledge graphs. *CoRR*, abs/2010.11967, 2020. URL `https://arxiv.org/abs/2010.11967`.

[62] Junyuan Shang, Tengfei Ma, Cao Xiao, and Jimeng Sun. Pre-training of graph augmented transformers for medication recommendation. *CoRR*, abs/1906.00346, 2019. URL `http://arxiv.org/abs/1906.00346`.

[63] Chanwoo Jeong, Sion Jang, Hyuna Shin, Eunjeong Park, and Sungchul Choi. A context-aware citation recommendation model with bert and graph convolutional networks, 2019.

[64] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016. URL `https://arxiv.org/abs/1611.07308`.

[65] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *31st International Conference on Machine Learning, ICML 2014*, 4, 12 2014.

[66] Zhibin Lu, Pan Du, and Jian-Yun Nie. VGCN-BERT: augmenting BERT with graph embedding for text classification. *CoRR*, abs/2004.05707, 2020. URL `https://arxiv.org/abs/2004.05707`.

[67] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://aclanthology.org/W13-3512`.

[68] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. 3(null):1137–1155, mar 2003. ISSN 1532-4435.

[69] Andrei Alexandrescu and Katrin Kirchhoff. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 1–4, New York City, USA, June 2006. Association for Computational Linguistics. URL `https://aclanthology.org/N06-2001`.

[70] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016. URL http://arxiv.org/abs/1607.04606.

[71] Sebastian Padó, Aurélie Herbelot, Max Kisselew, and Jan Šnajder. Predictability of distributional semantics in derivational word formation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1285–1296, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL https://aclanthology.org/C16-1122.

[72] Angeliki Lazaridou, Marco Marelli, and Marco Baroni. Multimodal word meaning induction from minimal exposure to natural text. *Cognitive science*, 41 Suppl 4:677—705, April 2017. ISSN 0364-0213. doi: 10.1111/cogs.12481. URL https://doi.org/10.1111/cogs.12481.

[73] Aurélie Herbelot and Marco Baroni. High-risk learning: acquiring new word vectors from tiny data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 304–309, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1030. URL https://aclanthology.org/D17-1030.

[74] Mikhail Khodak, Nikunj Saunshi, Yingyu Liang, Tengyu Ma, Brandon Stewart, and Sanjeev Arora. A la carte embedding: Cheap but effective induction of semantic feature vectors. *CoRR*, abs/1805.05388, 2018. URL http://arxiv.org/abs/1805.05388.

[75] Pedro L. Rodriguez, Arthur Spirling, and Brandon M. Stewart. Embedding regression: Models for context-specific description and inference. 2022. URL https://github.com/prodriguezsosa/EmbeddingRegression.

[76] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

[77] Jingchao Ni, Shiyu Chang, Xiao Liu, Wei Cheng, Haifeng Chen, Dongkuan Xu, and Xiang Zhang. Co-regularized deep multi-network

embedding. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 469–478, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186113. URL https://doi.org/10.1145/3178876.3186113.

[78] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR. URL https://proceedings.mlr.press/v27/baldi12a.html.

[79] Rada Mihalcea. Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling. USA, 2005. Association for Computational Linguistics. doi: 10.3115/1220575.1220627. URL https://doi.org/10.3115/1220575.1220627.

[80] Eneko Agirre and Aitor Soroa. Personalizing PageRank for word sense disambiguation. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 33–41, Athens, Greece, March 2009. Association for Computational Linguistics. URL https://aclanthology.org/E09-1005.

[81] Roberto Navigli and Paola Velardi. Structural semantic interconnection: a knowledge-based approach to word sense disambiguation. In *Proceedings of SENSEVAL-3, the Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 179–182, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-0844.

[82] Roberto Navigli and Mirella Lapata. An experimental study of graph connectivity for unsupervised word sense disambiguation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):678–692, 2010. doi: 10.1109/TPAMI.2009.36.

[83] Simone Paolo Ponzetto and Roberto Navigli. Knowledge-rich word sense disambiguation rivaling supervised systems. ACL '10, page 1522–1531, USA, 2010. Association for Computational Linguistics.

[84] Tingting Wei, Yonghe Lu, Huiyou Chang, Qiang Zhou, and Xianyu Bao. A semantic approach for text clustering using wordnet and lexical chains. *Expert Systems with Applications*, 42(4):2264–2275, 2015. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2014.10.023. URL https://www.sciencedirect.com/science/article/pii/S0957417414006472.

[85] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. *CoRR*, abs/cmp-lg/9406033, 1994. URL http://arxiv.org/abs/cmp-lg/9406033.

[86] Filip Ilievski, Pedro A. Szekely, Gleb Satyukov, and Amandeep Singh. User-friendly comparison of similarity algorithms on wikidata. *CoRR*, abs/2108.05410, 2021. URL https://arxiv.org/abs/2108.05410.

[87] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL http://arxiv.org/abs/1606.05250.

[88] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL http://arxiv.org/abs/1806.03822.

[89] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476, 2017. URL http://arxiv.org/abs/1707.01476.