École Polytechnique Fédérale de Lausanne

Graph-based text representations

by Wentao Feng
Section of Electrical Engineering

Master Thesis

Approved by the Examining Committee:

Prof. Dr. Robert West
Thesis Advisor

Dr. Andreas Spitz
External Expert

Dr. Maxime Peyrard, Akhil Arora
Thesis Supervisor

EPFL IC DLAB
INN 311 (Bâtiment INN)
Station 14
CH-1015 Lausanne

June 25, 2021

God’s in his Heaven
All’s right with the world
— Neon Genesis Evangelion

To my family and friends.

Acknowledgments

I am very grateful to the Data Science Lab for providing me excellent academic atmosphere and vital infrastructure to complete this project. I enjoy participating dMonday, where I learn much knowledge from colleagues.

Especially, I would like to express my appreciation to the following people for their continuous support in the past four months:

> **Prof. Robert West** for advising this project. I cultivate good research habits from his on-boarding guidance. During the semester, he provides insightful advice and guides this work moving in the right direction.
>
> **Dr. Maxime Peyrard** and **Akhil Arora** for being my supervisors. They always timely, patiently, and comprehensively answer my questions. Their theoretical and technical support helps my project run smoothly. Their guidance and suggestions are an indispensable part of this thesis.

I am also thankful to all my colleagues and friends in China, Switzerland, Australia, and other countries. They share with me numerous delightful and memorable moments in the past few years.

Most importantly, I would like to show gratitude to my family. They support me not only financially but also mentally under no condition. Their love and encouragement are essential to my completion of the Master's program.

Lausanne, June 25, 2021 冯文韬/FENG, Wentao

Abstract

Text representation is the fundamental procedure of natural language processing (NLP) tasks. It aims to convert text into a mathematically computable form and preserve linguistic information. The quality of this process significantly affects downstream tasks, e.g., machine translation.

The popular approaches define unsupervised tasks to learn a vector in the low-dimension space for each word. These word vectors provide numerical representation but lack interpretability. Researchers cannot explain why that word should locate at that point. We propose a novel representation model based on the knowledge graph to address this issue. Considering the volume of the modern knowledge graph, we introduce relaxations to make our method concise, efficient, and scalable like Word2Vector from Mikolov et al. [25].

Our method benefiting from a large-scale knowledge base can generate highly interpretable word representation even on a small dataset. We observe that our results have much higher interpretability than the state-of-the-art methods through quantitive and qualitative experiments. Moreover, various datasets show that our model can capture good syntactic and semantic information.

Résumé

Nous abordons, dans cet article, sur la procédure fondamentale de Traitement Automatique du Langage (abrégé en NLP), qui permettant aux machines de transformer le texte en langue informatique et de préserver les informations. La clé du problème réside donc dans ce processus puisqu' il affecte directement la qualité des tâches suivantes, par example, la traduction automatique.

Pour les méthodes populaires, des tâches non supervisées servent à apprendre les vecteur de mots en dimension faible. Ils nous offrent une représentation numérique mais l'interprétabilité fait defaut. Cependant, les chercheurs n'ont pas trouvé le mécanisme par lequel les mots devraient se trouver à cet endroit. Pour résoudre ce problème, nous introduisons un nouveau modèle de représentation qui repose sur le graphe de connaissances. Vu le volume du graphe de connaissances, nous y mettons en place des relaxations pour rendre notre méthode plus claire et efficace comme Word2Vector [25].

Reposé sur une base de connaissances à vaste échelle, nous pouvon obtenir une représentation des mots hautement interprétable, même sur un petit jeu de données. Par des expérimentations quantitatives et qualitatives, nous trouvons que l'interprétabilité de notre méthode surpasse les autres méthodes dans de nombreux cas. De même, différents jeux de données montrent que notre modèle peut capturer des informations syntaxiques et sémantiques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural language processing (NLP) is flourishing in recent years thanks to the success of artificial intelligence in academia and industry. Among the tasks of NLP, text representation is the most fundamental one. The naive approach uses one-hot vector to numerically represent text where 1 is the word's location in the dictionary. The upgrade of one-hot vector is a co-occurrence matrix that can include contexts. These two approaches are interpretable. However, a large encoding matrix consumes much memory. The popular approaches [25–27, 32] use dense vector in low-dimensional space to encode text. The word vectors come from maximizing the accuracy of the phrase completion task. So, these approaches are advanced in terms of efficiency and effectiveness. Like many other NLP models, the results are not interpretable. For the human, the word vectors are meaningless numbers.

An interpretable text representation model is significant in many NLP tasks. Reasoning the embedding of out-of-vocabulary word becomes feasible. The researcher can remove less meaningful dimensions for the usage of low-end devices. Counter-intuitive embedding helps to locate bias in the corpus or bug in the pipeline. The workload of learning embedding in another language can reduce, as the interpretability of common words can share across languages.

The major challenge of achieving understandable embedding is to find a proper space to place words. Dictionary can be a discrete space with each word as an axis. The word vector of one-hot and co-occurrence model is a point in such a space and thus understandable to humans. The classical Word2Vec [25] use Euclidean space without definition of coordinates or points. Therefore, we cannot understand the embedding of Word2Vec. Finding a proper space is hard. The dimension to represent a word out of total words is $\mathcal{O}(1)$ preferably. Defining the axes or points should not require much manual work. The knowledge in the space needs to be large enough to explain vocabulary.

The research on interpretable neural embedding is limited. Many previous works [5, 25, 32, 41] focus on encoding the linguistic information well and efficiently without considering the interpretability. Ryabinin et al. [37] embed the word as a vertex in a graph. Although they change the space where words locate, their graph is still not meaningful. Faruqui et al. [11] and Subramanian et al. [38] augment pre-trained word vectors by sparsity and non-negativity constraints. They both try to improve the interpretability but use word vectors for self-interpretation, which has poor performance on a small dataset and bad word vectors.

To solve the issue above and build a well interpretable model, we introduce Word2GMM. This model uses the workflow of Skip-gram but embeds the word on the knowledge graph (specifically, Wikidata). The word embedding is a Gaussian mixture distribution on the graph rather than a vector. Under our relaxations, the knowledge graph meets all standards of proper space and provides interpretation through probability density on nodes. Experiments show that our method achieves comparable performance with popular methods and owns interpretability. To sum up, our major contributions contain:

- Proposing a new text representation architecture that embeds a word on the knowledge graph. The embedding is explainable because of the knowledge in the graph structure.
- Giving relaxations in practical usage. The relaxations ensure that learning embedding is still feasible when the graph is huge.
- Instantiating an interpretable embedding model and comparing it with the state-of-the-art baselines. The results show the model has good interpretability as well as capturing syntactic and semantic information.

Chapter 2

Background

2.1 Skip-gram model

Skip-gram is an embedding architecture firstly introduced by Mikolov et al. [25]. The core idea is using the current word w_i (target word) to predict words in the range $[i - R, i + R]$ (context words). The training objective is maximizing logarithmic probability (Eq. 2.1) for N samples, which is from passing pair-wise similarity $\mathcal{S}$ between the target word and all words into softmax function (Eq. 2.2). The original paper uses the dot product as $\mathcal{S}$. Fig. 2.1 demonstrates the model architecture. The radius R is a parameter balancing the computation complexity and embedding quality. Increasing radius improves quality but also requires additional training time. However, larger R does not always bring better quality, as faraway words are less likely to have similar semantics. Two optimizations from Mikolov et al. [27] contribute to the success of this concise model.

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{i-R\leq j\leq i+R, j\neq i}\log p(w_j|w_i) \tag{2.1}$$

$$p(w_j|w_i) = \frac{\exp\left(\mathcal{S}(w_j, w_i)\right)}{\sum_{j'=1}^{W}\exp\left(\mathcal{S}(w_{j'}, w_i)\right)} \tag{2.2}$$

Negative sampling. Computing softmax on the entire dictionary is inefficient. A solution from Mikolov et al. [27] is randomly drawing K noise samples for each context word from the dictionary. The probability distribution of drawing is a variant of unigram distribution (Eq. 2.3), where W is the size of the dictionary and f is frequency. . By this approach, the model is to distinguish the context word from K noise samples. With negative sampling, the $\log p(w_j|w_i)$ in the objective function becomes Eq. 2.4 where σ is sigmoid function.

$$P_n(w_i) = \frac{U(w_i)^{0.75}}{\sum_i^W U(w_i)^{0.75}}, \quad U(w_i) = \frac{f(w_i)}{\sum_i^W f(w_i)} \tag{2.3}$$

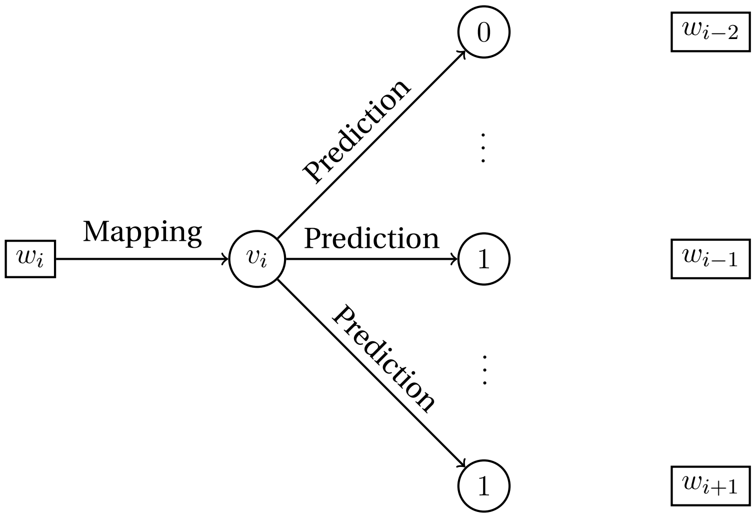


Figure 2.1 – Skip-gram model with $R = 1$. w_i is the word at i and v_i is corresponding projection in continuous space. The predictions from v_i and context embeddings should be 1 for $w_{i\pm1}$ and 0 for the rest.

$$\log \sigma(\mathcal{S}(w_j, w_i)) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim p_n(w)}[\log \sigma(-\mathcal{S}(w_k, w_i))] \tag{2.4}$$

Subsampling. According to Zipf's law, a word's frequency is inversely proportional to its rank. Hence, frequent words can be dominant to weight update. As a result, infrequent words are underfitting. Moreover, stop words are frequent but less informative. To mitigate the imbalance, Mikolov et al. [27] randomly discard part of samples with the frequency into account. Eq. 2.5 [24] gives the formula of computing rejection probability, where t is the subsampling factor.

$$P(w_i) = 1 - \sqrt{\frac{U(w_i)}{t} + 1} \times \frac{t}{U(w_i)} \tag{2.5}$$

Although Mikolov et al. [25] propose skim-gram model for learning word representations, researchers extended the usage to other scenes, e.g., BioVec [3], Node2Vec [15].

2.2 Knowledge graph

A graph $\mathcal{G}$ contains a set of nodes V and edges E. When we view a node $v_i \in V$ as an item and a directed edge $e_{ij} \in E$ as a property, this specific graph is called knowledge graph (KG), which is also known as the knowledge base. The directed network structure (Fig. 2.2) encodes knowledge and can be of arbitrary size. The triplet $< v_i, e_{ij}, v_j >$ is another common form of knowledge representation, e.g., $<$ London, is the capital of, The UK $>$. Since we use Wikidata in this thesis, Table 2.1 explains terms in the Wikidata context.

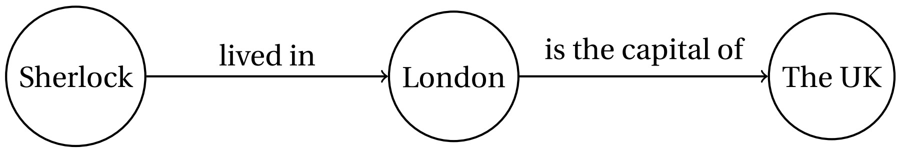


Figure 2.2 – A small example of KG.

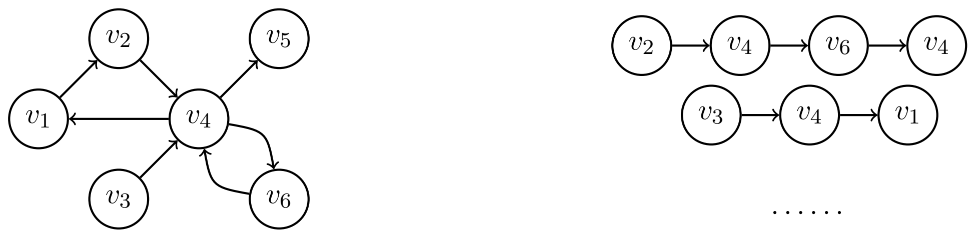
Term	**Explanation**
Item	The node in the Wikidata.
Property	The edge in the Wikidata.
QID	The unique identifier of item. e.g., Q42.
PID	The unique identifier of property. e.g., P65.
Label	The name of an item, e.g., the label of Q42 is Douglas Adams
Aliases	Alternative names of an item.
Description	A short text to disambiguate items with same label.
Degree	The number of links to a node.

Table 2.1 – Terms and explanations in Wikidata context

2.3 Node embedding

Similar to word embedding, node embedding is a technique that maps nodes in a graph into low-dimensional continuous space while preserves basic structural information and node similarity. This continuous representation has friendly properties. Distance computation in Euclidean space is fast and straightforward. In deep learning, backpropagation is well-defined for continuous function. The naive approach for learning node vectors contains two steps. Firstly, random walk on the graph generates enough paths as training data (Fig. 2.3). Then, skip-gram model distinguishes the target node from negative samples (Fig. 2.1). In this thesis, we use DeepWalk [33] in our model.

DeepWalk. Deepwalk has well-known efficiency, scalability, and quality. Perozzi, Al-Rfou, and Skiena [33] improve naive node embedding by introducing parallel execution. They prove the sparse update of embedding that makes the parallel feasible. Their experiments also show that the quality of results is still satisfying with the increased number of workers.



(a) Original graph (b) Random walk generation

Figure 2.3 – Any node can be the start point. The walk randomly selects the next node according to transient matrix. If the length of path is unlimited, the training set can be arbitrarily large.

2.4 Gaussian mixture model

Mixture model represents a probability distribution as a weighted sum of other distribution. Gaussian mixture model (GMM) is a special case that every component is a normal distribution. Assuming we have M multivariate normal distributions, we can calculate the probability density at $x \in R^d$ by Eq. 2.6 and Eq. 2.7 where $\mu_m \in R^d$ is the mean vector and $\Sigma_m \in R^{d\times d}$ is the covariance matrix. A valid Σ should be symmetric and positive semi-definite. If any two dimensions of x have no covariance and variances are same, Σ becomes diagonal matrix, i.e., spherical normal distribution.

$$P(x) = \sum_{m=1}^{M} a_m \mathcal{N}(x|\mu_m, \Sigma_m) \quad s.t. \sum_{m=1}^{M} a_m = 1, a_m \geq 0 \tag{2.6}$$

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{|\Sigma|^{\frac{1}{2}}(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right) \tag{2.7}$$

2.5 Statistical distance

The distance between two statistical objects is statistical distance. Statistical distance is not the metric necessarily. For example, Kullback–Leibler divergence violates $d(P,Q) = d(Q,P)$ (symmetry). For distributions P and Q, $d(P,Q)$ satisfies at least $d(P,Q) \geq 0$ (non-negativity), $d(P,Q) \iff P = Q$ (identity of indiscernibles). In deep learning, statistical distance works as the objective function.

Chapter 3

Method

3.1 Problem modeling

The most intuitive way to take advantage of existing knowledge is learning a value for each item. We can translate the value into item relevance to the word. A set of related triplets explains a word. To achieve this goal, we can treat the word embedding e as a signal on $\mathcal{G}(V, E)$ (Def. 1) if the dimension of e equals to the number of nodes $|V|$. Then, the problem transforms into learning a signal on the graph for each word. For words w_i, w_j, and graph $\mathcal{G}$, the similarity function between two graph signals is $\mathcal{S}(w_j, w_i|\mathcal{G})$. Our graph-based text representation is to maximize Eq. 3.1 over training samples.

Definition 1 (Graph signal). *A vector defined on $V \to R$ is a signal on the graph $\mathcal{G}(V, E)$, where the value at i^{th} dimension is the value of v_i.*

$$\log \sigma(\mathcal{S}(w_j, w_i|\mathcal{G})) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim p_n(w)}[\log \sigma(-\mathcal{S}(w_k, w_i|\mathcal{G}))] \tag{3.1}$$

3.2 Relaxation

Two obstacles halt the application of our model. First, applying similarity function $\mathcal{S}$ and back-propagation to signal in the discrete space is non-trivial. Most statistical distances cannot apply to the graph signal directly. Wasserstein metric is fully capable of graph scenario. Nevertheless, we cannot implement differentiable linear programming within a reasonable time. Although we can get a fully differentiable Wasserstein with the tools crafted by Amos and Kolter [2], the time complexity of solving quadratic programming will be the bottleneck of the model. The upper limit of $|V|$ we test on a 12GB-memory GPU is 30 nodes that are too little in our case.

Another is that the coupling between $|V|$ and $|e|$ brings scalability issue to our model. The preliminary experiments show that 200 nodes are the limit for a high-performance computer. However, modern KG has millions of nodes (e.g., Wikidata has 94 million items [1]). Moreover, this coupling is also against the purpose of word embedding, which is mapping words into low-dimension space. To address these problems, we propose three relaxations to Eq. 3.1.

3.2.1 Node embedding

Instead of $\mathcal{G}(V, E)$, we use node embedding matrix $\mathcal{E}_V \in R^{|V| \times d}$ as the word embedding space, where d is the cardinality of node embedding space. Though $\mathcal{E}_V$ preserves the node's similarity approximately, we can benefit from automatic differentiation and numerous similarity measurement defined in continuous space.

3.2.2 GMM

Thanks to the node embedding, we now can learn a parameterized function $f(\theta) : \mathcal{E}_V \to e$ for each word. Therefore, the complexity of the model is controllable with $|\theta|$ and irrelevant to $|V|$. The $f(\theta)$ we choose is GMM. GMM has the following advantages:

- Efficient. Two mixtures of Gaussian have closed-form statistical distance. We can improve efficiency by avoiding $\mathcal{E}_V \to e$ during training.
- Expressive. Although GMM is less expressive than free graph signal, it is enough for encoding a word's syntactic and semantic similarity with sufficiently large M (number of components).
- Sparse. From the three-sigma limit, only a small portion of nodes will have positive probability density from GMM. e will be a sparse vector. This sparsity meets our expectation, as the meaning of a word is single in most cases.

Our Word2GMM model represents a word with $M \times \{\mu, \Sigma, a\}$. Ensuring the positive semi-definite of all Σ involves eigenvalues decomposition that is a computation-intensive task. Meanwhile, we care more about the center than the divergence of each Gaussian. Therefore, we use a diagonal covariance matrix and treat it as a hyperparameter. All GMMs share one diagonal Σ for further simplicity. Finally, The learnable parameters for each word reduce to $M \times \{\mu, a\}$.

[1]`https://www.wikidata.org/wiki/Wikidata:Statistics`

Previous works[14, 18] prove that properly initializing a bunch of parameters helps the convergence of the neural network. We borrow the idea of weight initialization and use truncated normal to generate Σ instead of using the same value. Concretely, we shift and scale points from standard truncated normal distribution (i.e., $\mu = 0, \sigma = 1, -1 \leq x \leq 1$). In this way, the variances will normally distribute in $[c - r, c + r]$ where c and r are the amounts of shifting and scaling. Gaussian is elliptical when $r > 0$ and spherical when $r = 0$. We will discuss how c and r affect the quality of embedding.

3.2.3 Anchor

Since we compute the distance of two GMMs only with parameters, the node similarity of the KG is missing. Therefore, we use anchors (Def. 2) to re-import the knowledge.

Definition 2 (Anchor)**.** *An anchor is a carefully selected node (item) whose name (label) is existing in the dictionary, aiming to regulate the word's GMM.*

Without anchors, GMM can move freely in the node embedding space. As most of the points in that space are non-meaningful, the model is unlikely to be interpretable. Through anchoring nodes in word embedding space, the GMM of words can respond to them. Concretely, GMM will move close to an anchor if they have similar semantics. Although w_i may not have anchoring words in the context $[i - R, i + R]$, indirect response will help regulate their GMM. For anchors, we set a single normal distribution centered at the anchoring node. Their μ, Σ are fixed during training.

We provide four strategies for selecting $|A|$ anchors:

- k-means++. This algorithm is probably the most simple but effective way to selecting landmarks from big data. To get $|A|$ anchors, we set the number of clusters to $|A|$. After convergence of k-means++, we use k-nearest neighbors (k-NN) to find the nodes closest to cluster centers.
- Sampling with density. k-means++ does not consider the density of data. It fails to give representative centers if some clusters only have one member. To choose anchors according to node density, we use minus distances to neighbors as sampling weight. Specifically, k-NN sums distance to $\lfloor |V|/|A| \rfloor + 1$ nearest neighbors for every node. Small total distance means high density, and this area should have extra anchors. Then, we sample $|A|$ anchors without replacement from V using minus total distance as weight. We do not simply choose nodes with the highest weights, as they may close to the other. Weighted sampling can avoid that, as it turns into pseudo uniform sampling among similarly weighted individuals.
- High-degree nodes. In a network, a node with an extremely high degree is called a hub. Hub is naturally a good choice for anchor, as it connects many nodes, and the average distance

from the hub to the other is relatively small. Therefore, we select the first $|A|$ nodes sorted by degree decreasingly as anchors.

- Sampling with the degree. This strategy is similar to sampling with density. The weight used here is the degree. The purpose is also to avoid close anchors, though hubs tend to distribute in the network evenly.

Not all items are valid candidates for anchors because we can only choose the item with the label existing in the training corpus. Moreover, ambiguous items exist in the Wikidata. To filter valid candidates, we first create a sub-graph with the single token entity. We keep the item with the highest degree to avoid ambiguity. Then, anchor selection algorithms run on the sub-graph.

3.3 Squared Euclidean distance

Suppose we have two GMMs $P(x|a_m, \mu_m, \Sigma_m, m \in [1, M])$ and $Q(x|b_n, \eta_n, \Lambda_n, n \in [1, N])$. To save time and memory, the statistical distance between two GMMs should only take $a, \mu, \Sigma, b, \eta, \Lambda$ as inputs and output the similarity. The choices are limited. Kullback-Leibler divergence only gives an approximate result for two GMMs. Wasserstein distance only has a closed-form for two Gaussian distributions. The well-defined distance measurement for GMM we find is Euclidean distance (ℓ_2). As gradient explodes when $\ell_2 \to 0$, we use squared Euclidean distance (ℓ_2^2) in our model. Here, we directly give the formulas (Eq. 2.7 and Eq. 3.2) to compute ℓ_2^2 without proving.

$$\begin{aligned}\ell_2^2(P, Q) = \sum_{m,m'} a_m a_{m'} \mathcal{N}(\mu_m|\mu_{m'}, \Sigma_m + \Sigma_{m'}) + \sum_{n,n'} b_n b_{n'} \mathcal{N}(\eta_n|\eta_{n'}, \Lambda_n + \Lambda_{n'}) \\ - 2\sum_{m,n} a_m b_n \mathcal{N}(\mu_m|\eta_n, \Sigma_m + \Lambda_n)\end{aligned} \tag{3.2}$$

3.4 Word2GMM

After applying the relaxations, our Word2GMM model maximizes the objective function Eq. 3.3 under the constrains Eq. 3.4. Since lower ℓ_2^2 means better similarity, every ℓ_2^2 already times -1 in Eq. 3.3.

$$\log \sigma(-\ell_2^2(a^i, \mu^i, \Sigma, b^j, \eta^j, \Sigma)) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim p_n(w)}[\log \sigma(\ell_2^2(a^i, \mu^i, \Sigma, b^k, \eta^k, \Sigma)))] \tag{3.3}$$

$$s.t. \quad a_m^i = \frac{1}{M}, \mu_m^i = A_i, b_m^{\{j,k\}} = \frac{1}{M}, \eta_m^{\{j,k\}} = A_{\{j,k\}}, \forall \{i, j, k\} \in A \tag{3.4}$$

Chapter 4

Experiment

4.1 Experimental setup

4.1.1 Data and pre-processing

We use $300,000$ sentences from Wikipedia (Wiki-$300k$) as training data. The training dataset is small, but it is enough for experiments. The tokenizer we use is `en_core_web_trf` model from spaCy[1]. After tokenization, we replace rare words, which appear less than 12 times in the corpus, and stop words, which are provided by `en_core_web_trf`, with `<UNK>`. The actual dictionary size is 23190. For avoiding invalid probability, we slightly modify Eq. 2.5 and use Eq. 4.1 to subsample. The negative sampling occasionally gives two or more same negative samples. We apply rejection sampling simultaneously to counter duplication.

$$P(w_i) = 1 - \min(1, \sqrt{\frac{U(w_i)}{t}} \times \frac{t}{U(w_i)}) \tag{4.1}$$

4.1.2 Model settings and training configuration

We build our pipeline with PyTorch v1.8 [31] and PyTorch-lightning v1.3 [10]. Due to time limitations, we only conduct a grid search on the optimizer's hyperparameters. For the others, we tune them independently on the validation dataset. The best combination of parameters we find is in Table 4.1. Without special announcement, the Word2GMM in this thesis uses the best combination. The code and data[2] to reproduce results are available online.

[1]https://spacy.io/

[2]https://go.epfl.ch/word2gmm

Category	Name	Value
Node embedding	Algorithm	DeepWalk
	d	10
Data	R	2
	K	5
	t	10^{-5}
	Batchsize	128
Anchor	Sub-graph $\|V\|$	8869
	$\|A\|$	128
	Selection	Sampling(Density)
GMM	M	25
	c	1.2
	r	0.2
Optimization	Optimizer	Adam [20]
	Learning rate	0.005
	Scheduler	Step learning rate
	Step size	10
	Decay ratio	0.75
	Epoch	50

Table 4.1 – The best combination of parameters.

4.1.3 Baseline models

Word2Vec-S. The Word2Vec [25] trained on Wiki-$300k$ is called Word2Vec-S. The data configurations are the same as Table 4.1. Optimizer is Adam with fine-tuned cosine annealing learning rate. The dimension of the word vector is 275 in order to have the same number of trainable parameters as Word2GMM.

SPINE-S. The SPINE [38] is a state-of-the-art and open-source interpretable neural embedding. This model converts existing word vectors to interpretable ones. We train our SPINE model with the official implementation[3] on Word2Vec-S to get SPINE-S. We use the hyperparameters from Subramanian et al. [38] and increase the epochs to 6000 for good performance on our dataset.

For reference, we also provide the results of the state-of-the-art Word2Vec model, which is called Word2Vec-L. This model learns from Google News with about 100 billion tokens and has 3 million words in the dictionary. We use a pretained model from Gensim [35]. Corresponding to Word2Vec-L, we have SPINE-L. Since Subramanian et al. [38] have trained a SPINE on Google News Word2Vec, we directly use the model they publish online[3].

[3]`https://github.com/jacobdanovitch/SPINE`

4.2 Word similarity

Word similarity is evaluating how accurately the model predicts the similarity of two words. To get the similarity, Word2GMM computes $-\ell_2^2$ of two mixtures of distributions, and vector-based models compute cosine similarity (Eq. 4.2). Spearman's rank correlation coefficient (r_s) between predicted and true scores measures prediction quality. r_s firstly gets the rank of each observation and computes the correlation coefficient (r_ρ). Therefore, r_s is independent of the range of similarity function and ground truth. High r_s means our model well preserves the word's semantics. The range of r_s is $[-1, 1]$. In this task, we have 13 datasets from various sources. Since our training set is relatively small, not all words from the evaluation set are covered. We skip the pairs with one or two words missing. Table 4.2 lists the number of total pairs and good pairs for every dataset. The overall performance r_s^{all} is computed by Eq. 4.3.

$$\cos(\overrightarrow{v_1}, \overrightarrow{v_2}) = \frac{\overrightarrow{v_1} \cdot \overrightarrow{v_2}}{||\overrightarrow{v_1}||||\overrightarrow{v_2}||} \tag{4.2}$$

$$r_s^{all} = \frac{\sum_i r_s^i \times valid_i}{sum(valid)} \tag{4.3}$$

Dataset	SPINE-S	Word2Vec-S	Word2GMM	Total	Valid	Ratio	SPINE-L	Word2Vec-L
MEN-TR-3k [7]	0.0791	0.2960	**0.5041**	3000	2315	77.17%	0.6899	0.7636
MC-30 [28]	-0.0008	0.1150	**0.4886**	30	25	83.33%	0.5874	0.8075
MTurk-771 [17]	0.0891	0.2173	**0.3834**	771	719	93.26%	0.5764	0.6699
SIMLEX-999 [19]	0.0862	0.0873	**0.1714**	999	882	88.29%	0.3606	0.4564
VERB-143 [4]	-0.0239	0.0438	**0.3497**	144	118	81.94%	0.3956	0.4326
YP-130 [39]	-0.1174	0.1505	**0.2797**	130	89	68.46%	0.0854	0.4553
RW-STANFORD [22]	0.1305	0.0586	**0.1657**	2034	429	21.09%	0.4389	0.5839
RG-65 [36]	-0.0191	0.3388	**0.4110**	65	51	78.46%	0.6411	0.7229
WS-353-ALL [12]	0.2714	0.3014	**0.5212**	353	323	91.50%	0.5969	0.7038
WS-353-SIM [1]	0.2539	0.2974	**0.5455**	203	185	91.13%	0.6717	0.7773
WS-353-REL [1]	0.2245	0.2978	**0.4942**	252	233	92.46%	0.5587	0.6375
MTurk-287 [34]	-0.0002	0.3008	**0.4582**	287	272	94.77%	0.6290	0.6785
SimVerb-3500 [13]	0.0334	0.0104	**0.0742**	3500	2306	65.89%	0.1012	0.3542
Total	0.0791	0.1648	**0.3076**	11768	7947	67.53%	0.4368	0.5753

Table 4.2 – Evaluation on word similarity. Ratio is $\frac{valid}{total}$.

Results. From Table 4.2, Word2GMM outperforms Word2Vec-S and SPINE-S in all datasets. Word2Vec has proved its capability in recent years. The winning of our model means it serves competitively well for downstream tasks. The gap to Word2Vec-L is non-negligible. However, that difference is mainly due to small training data. Our word representations expect to have a comparable quality if we increase training data. Meanwhile, we observe that both SPINE-S and SPINE-L hurt the linguistic information in the original word embedding.

4.3 Interpretable embedding

	Word2Vec-S	**Word2Vec-L**
people	bassists, litre, nc dispense, daytona	capt, astronomers lakers, nec, shootout
government	wafer, quark, ibsen ounces, eocene	jacket, consortium vaccine, coupe. cigar
water	hotter, newark, bohr modernisation, lysander	microsoft, sr, bt malaysia, jan
	SPINE-S	**SPINE-L**
people	mgm, nudity, tensile semitone, secretion	viewers, readers, listeners travelers, commuters
government	katz, bess, tampa nearing, salisbury	envoy, minister, ministers parliament, ambassador
water	tarot, repel, pepys voltaire, prematurely	dam, dams, river rivers, tributary
	Word2GMM	
	Node	**Description**
people	Schumacher	football player
	Wilde	Argentine city
	Jonas	football player
	Alexis	football player
	goalkeeper	position in football
government	Valencia	electoral district
	Alameda	municipality
	Monaco	country
	Arenas	municipality
	Algeria	country
water	salmon	fish
	lettuce	plant
	fish	aquatic animal
	tea	drink
	rack	gadget

Table 4.3 – The explanation for `people`, `government`, `water` from baselines and our model. The description is from searching QID in Wikidata.

Previous works [11, 29, 30, 38] use word intrusion test to evaluate the interpretability quantitatively. The participants need to find out the weird word from several choices. The percentage of correct answers reflects the interpretability. Here, we only include qualitative assessment and leave human intelligence tasks as future work. For our graph-based model, we pick the top 5 activated nodes as the word's explanation. For the vector-based model, we use the method mentioned in Subramanian et al. [38]. Specifically, we pick 5 most participating words on the top activated dimension.

Results. From Table 4.3, we can find the GMM will have high probability density on nodes closely related to the word. The top-5 activated nodes of `people` are profession and city. The profession is all about football probably because of the bias in the training corpus. `Government` refers to a group of people who manage country, city, community, so it has activated nodes about the targets it serves. The relation between `water` and nodes is more abstract and more persuasive than `people` and `government`. `water`'s top activated nodes include creature living water, the variant of water, the tool to drain water. Although these three examples only provide a qualitative analysis, they can clarify why word embedding ends at that location. Meanwhile, given the top activated nodes, we can easily determine the source from these three words, which also implies the strong interpretability of our model.

The Word2Vec model is uninterpretable no matter the training data is large or small. Our model is much more interpretable than SPINE-S. We notice that the training data of SPINE-L are only frequent words. The infrequent words may harm the performance of the self-explanation model, as we can see in the interpretations from SPINE-S. Our model incorporates external knowledge and thus is robust to the training dataset.

Error analysis. Our model fails to interpret when the anchor's contextual meaning is far from Wikidata's description. For example, London usually refers to the capital city of the United Kingdom and England. However, the anchor `London` in our best model is a city in the United States, which causes two American cities (Table 4.4) to appear as the top activated nodes of `England`. The reason behind this failure is the mismatch between the knowledge and the text. Although anchor and word are identical, they have different meanings.

	Node	**Description**
England	garner	American town
	astros	American football team
	blind	type of bet in poker
	linebacker	position in American football
	rochester	American borough

Table 4.4 – A case with wrong interpretation.

4.4 Graph signal analogy

The analogy of graph signals from GMM is vital to interpretability because resembling words should have similar signal patterns. Precisely measuring similarity between two graph signals can use earth mover's distance (EMD). The node-to-node distance is the moving cost. However, computing EMD takes much time and memory on a large graph. Instead, we binarize the value with threshold 0.01 and compute the Jaccard index (J) (Eq. 4.4) between sets of activated nodes. High Jaccard index means A resembles B. We pick the first 10% and last 10% (ordered by ground truth score decreasingly) of word pairs from datasets in Table 4.2 and get the Jaccard index for their activated nodes.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.4}$$

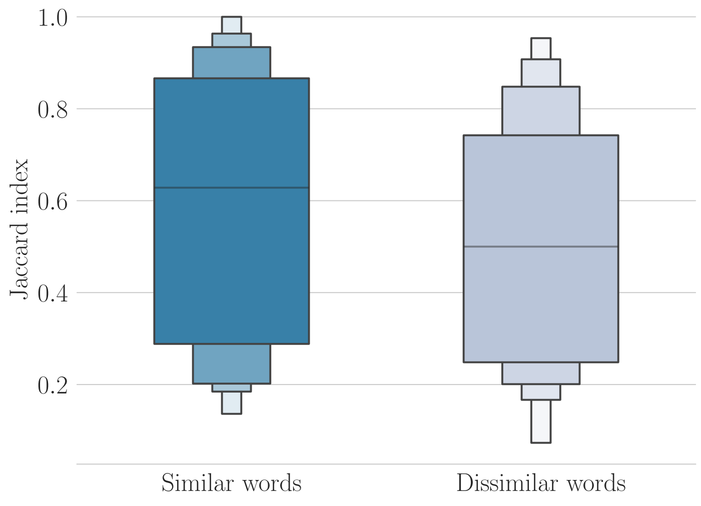


Figure 4.1 – Jaccard index distribution for similar words and dissimilar words. The area of box is determined by the span and the number of observations. From bottom to top, the box represents $3.125\%, 6.25\%, 12.5\%, 25\%, 25\%, 12.5\%, 6.25\%, 3.125\%$ of observations. The median is the border between two largest boxes.

Word	Top activated nodes
chemistry	tyrannosaurus/Q14332, antlers/Q834007, lettuce/Q83193
inorganic	tyrannosaurus/Q14332, lettuce/Q83193, antlers/Q834007
chemical	antlers/Q834007, lettuce/Q83193, tyrannosaurus/Q14332
covalent	rack/Q2928735, antlers/Q834007, celestine/Q407221

Table 4.5 – Three most similar words of chemistry. They share one common activated node.

Results. In Fig. 4.1, the median of J_{similar} is larger than 0.6 and the same statistic of $J_{\text{dissimilar}}$ is around 0.5. Some word pairs can have exactly the same activation pattern ($J_{\text{similar}} = 1$). An example in Table 4.5 also shows words in the same context share commonly activated nodes. We can conclude that similar words have similar patterns of graph signal.

4.5 Parameter influence

In this section, we evaluate how each parameter affects the quality of embedding. The metric we use to compare the performance of different configurations is Eq. 4.3. Default configuration (Table 4.1) functions as baseline. For clear visualization, scores of the varied model are relative to the baseline. The baseline is always 1.

4.5.1 Anchor

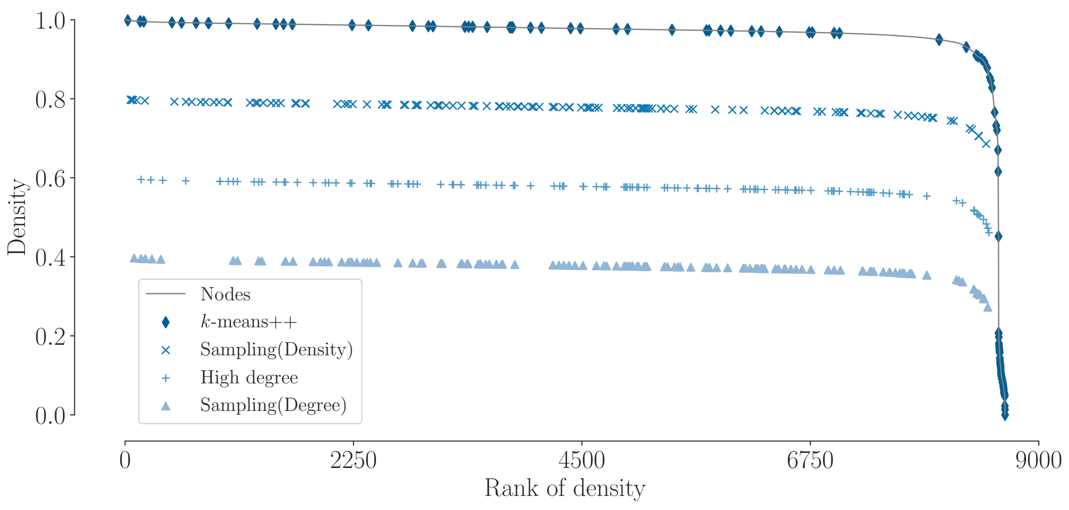


Figure 4.2 – Surrounding density estimation for nodes and anchors. The densities of anchors from `Sampling(Density)`, `Sampling(Degree)`, `High degree` are shifted down for comparison. Estimation method is the same as `Sampling(Density)`.

Anchors are critical as they are connections between GMM and KG. We prefer evenly selected because anchors give a good summary of KG. We compare anchors from four algorithms in Fig. 4.2. As k-means++ does not consider the density of point, it picks isolated nodes as anchors. These nodes contain terms, abbreviations, family names, foreign words. The other three methods all ignore isolated nodes and pick anchors more evenly than k-means++. From the result of `High degree`, we can find isolated nodes have limited connection to the network, which implies they are not a good choice for anchors. Another finding is that hubs spread out in the network.

We run several models varying the number of anchors and the anchor selection method. Each time, only one parameter differs from the default configuration. From Fig. 4.3 left, increasing the number of anchors worsens the quality of word GMMs. The impact is inversely proportional to $|A|$ between 64 and 192. The influence of adding anchors becomes marginal when $|A| = 500$. The comparison tells us that $|A|$ is a parameter balancing the interpretability and word embedding quality. More anchors hurt word embeddings as they give less freedom to GMM.

Density is an important aspect for good anchors. In Fig. 4.3 right, sampling with density results in the best performance. Degree is not a very influential factor. High degree anchors yield result close to k-means++ cluster centers.

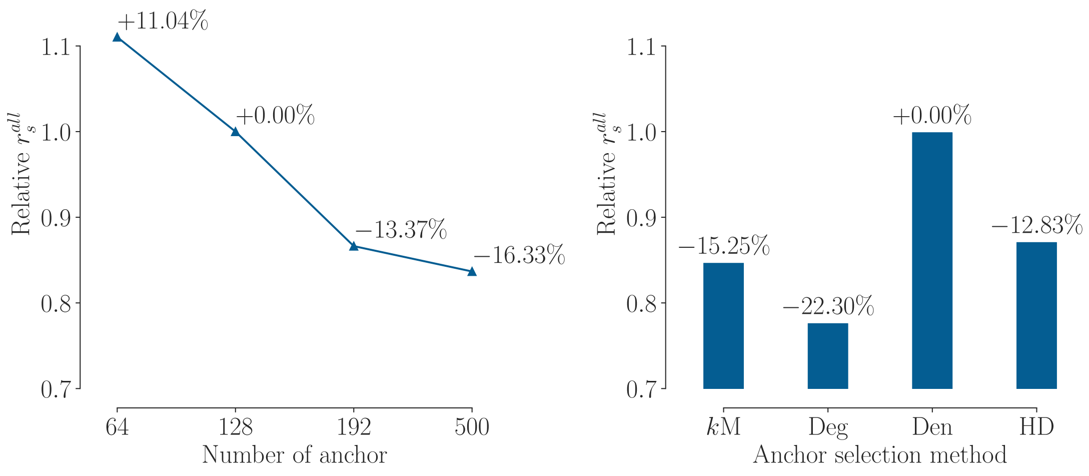


Figure 4.3 – Different versions of model by changing the settings of anchor selection. The labels on x-axis are short names for k-means++, `Sampling(Degree)`, `Sampling(Density)`, `High degree`.

4.5.2 The number of Gaussian

The complexity of Word2GMM is mainly dependant on the M, because the bottle neck in a training step is $\Omega(M^2)$ computations of probability density function in ℓ_2^2. M also determines the expressiveness. Larger M produces better model, but the training time increases much faster than the quality of model (Fig. 4.4).

4.5.3 Covariance matrix

From Fig. 4.5, our model prefers large variances. Compared to increase variances, random initialization improves much more. For example, $c = 1, r = 0.2$ raises score by 10.82, versus 2.97% by $c = 1.2, r = 0$. Although we run five models to evaluate c and r, they are not rather fine-tuned due to time limitation. It is too early to conclude $r = 0.2$ is the best value, because small r may be reason for the poorness of $r = 0.4$. We leave full rounded grid search on c and r as a future work.

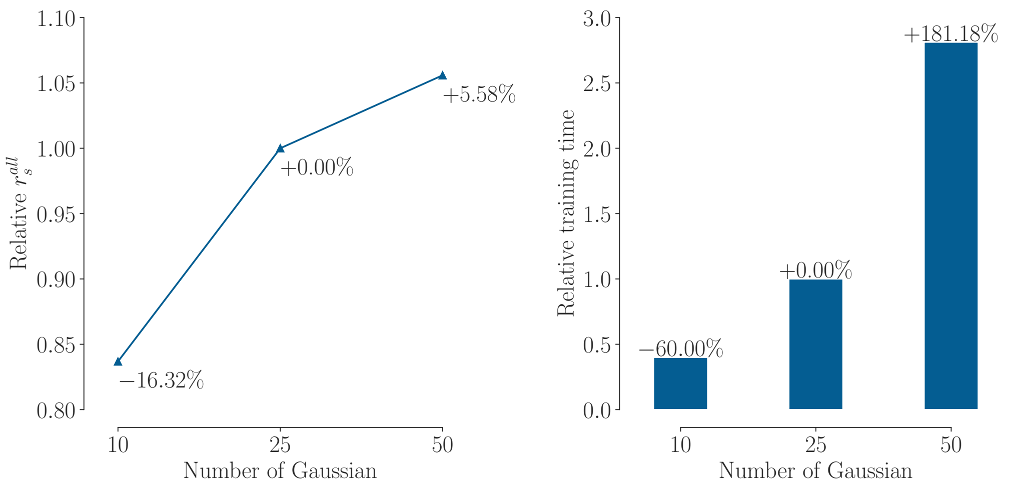


Figure 4.4 – The number of Gaussian controls the trade-off between complexity and expressiveness. $M = 25$ is a compromising choice and takes 42 hours to finish 50 epochs.

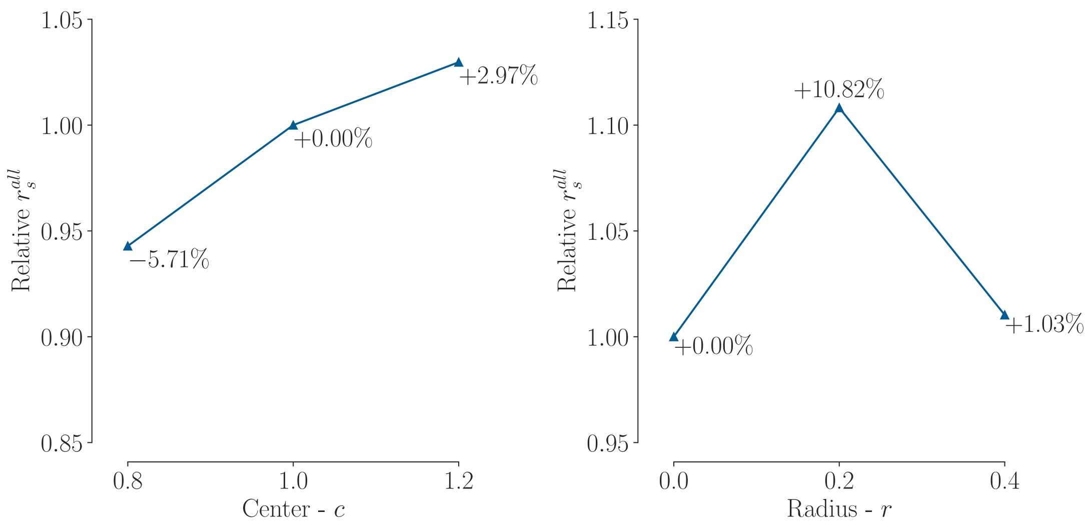


Figure 4.5 – In this figure, the baseline uses $c = 1, r = 0$. Both c and r affect the quality of model. Compared c, Word2GMM is sensitive to the range of Σ.

Chapter 5

Related work and discussion

Interpretable word embedding. Many researchers contribute to the development of word embedding. Mikolov et al. [25] propose Continuous Bag-of-Words and Skip-gram. The GloVe of Pennington, Socher, and Manning [32] factorize word co-occurrence matrix. Liu et al. [21] enhance Skip-gram by assigning topics to words. Bojanowski et al. [6] represent a word as the sum of n-grams vectors. Ryabinin et al. [37] argue that graph can show the better linguistic relationship and create GraphGlove. These methods can encode word similarity in space but do not consider the interpretability of embedding. Several works try to get interpretable embedding from existing word vectors. Murphy, Talukdar, and Mitchell [29] come up with Non-Negative Sparse Embedding to ensure semantic coherence at each dimension. Faruqui et al. [11] add sparsity and non-negativity constraint in the objective function to capture dimension-wise meaning. Subramanian et al. [38] train a neural model to convert word vectors into sparse form. Panigrahi, Simhadri, and Bhattacharyya [30] name each dimension with a sense and present Word2Sense model.

Knowledge incorporation. Integrating existing knowledge reinforces NLP models in many fields. Mihaylov and Frank [23] improve the reading comprehension model by including commonsense knowledge. Chen et al. [8] add semantic relationship of word pairs to enhance natural language inference model. Guan et al. [16] include external knowledge to their story generation model. The ways to incorporate knowledge are varied. Zhang et al. [42] randomly mask entity in a factual statement and use BERT to predict masked position. Chen et al. [9] use graph attention network to encode structural information. Wang et al. [40] encode entity description as the embedding.

Discussion. Our model can jointly optimize word embedding and interpretability. Joint learning avoids the sub-optimality of two-stage method, e.g., SPINE [38] has worse performance than Word2Vec in terms of measuring word similarity. Meanwhile, previous interpretability is dimension-wise. In SPOWV [11] and SPINE, each dimension represents a concept or topic. This setting limits the capacity of meaning to the number of dimensions. Our model's interpretability links to the size of KG. Therefore, our model is more knowledgeable than existing works. Our model also proves its interpretability on a small dataset. This ability will be advantageous to learn professional word embedding. Our work proposes a new application of KG. We use external knowledge to learn an interpretable embedding. As far as we know, our work is the first. The anchors also provide a novel idea to use existing massive knowledge.

Chapter 6

Conclusion

In this thesis, we present our several significant contributions. First, we introduce a new model, called Word2GMM, of learning interpretable word embedding with the aid of KG. This model has the efficiency and conciseness of the Skip-gram model and incorporates the meaningful structure of KG. To instantiate it, we use node embedding to turn the discrete structure into continuous and make backpropagation feasible. Then, GMM detaches the embedding dimension from $|V|$. We summarize the KG and bridge the KG and GMM with anchors. These relaxations successfully solve scalability issues, decrease complexity, and thus make our model practical. To select anchors, we come up with three effective strategies. Our experiments show that Word2GMM can capture syntactic and semantic information well. The interpretability analysis demonstrates the close relationship between words and top activated nodes. The analysis also shows the possibility for picking the correct word from multiples choices with the top activated nodes. Although the analysis is qualitative, our model has more powerful interpretability than simple numerical representation. At last, we systematically investigate how every parameter influences the quality of embedding. The investigation provides direction for parameter tuning.

Future work. Our work needs improvement. Due to the current anchor mechanism, we can only choose anchors from the item with one token. A suitable mechanism should allow items with multiple tokens as anchors. We also need to solve the mismatch between the anchor's description and the word's contextual meaning. Furthermore, interpretability needs quantitative analysis. We hope that we can use human intelligence tasks to help evaluate this function objectively. Other minor aspects, e.g., better node embedding, hyperparameter search, large training corpus, can also enhance our work.

Bibliography

[1] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, and Aitor Soroa. “A study on similarity and relatedness using distributional and wordnet-based approaches”. In: (2009).

[2] Brandon Amos and J Zico Kolter. “Optnet: Differentiable optimization as a layer in neural networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 136–145.

[3] Ehsaneddin Asgari and Mohammad RK Mofrad. “Continuous distributed representation of biological sequences for deep proteomics and genomics”. In: *PloS one* 10.11 (2015), e0141287.

[4] Simon Baker, Roi Reichart, and Anna Korhonen. “An unsupervised model for instance level subcategorization acquisition”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 278–289.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. “A neural probabilistic language model”. In: *The journal of machine learning research* 3 (2003), pp. 1137–1155.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. “Enriching word vectors with subword information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

[7] Elia Bruni, Nam-Khanh Tran, and Marco Baroni. “Multimodal distributional semantics”. In: *Journal of artificial intelligence research* 49 (2014), pp. 1–47.

[8] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Diana Inkpen, and Si Wei. “Neural natural language inference models enhanced with external knowledge”. In: *arXiv preprint arXiv:1711.04289* (2017).

[9] Wenhu Chen, Yu Su, Xifeng Yan, and William Yang Wang. “KGPT: Knowledge-Grounded Pre-Training for Data-to-Text Generation”. In: *arXiv preprint arXiv:2010.02307* (2020).

[10] William Falcon, Jirka Borovec, Adrian Wälchli, et al. “PyTorch Lightning”. In: *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* v1.3 (2021).

[11] Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah Smith. “Sparse overcomplete word vector representations”. In: *arXiv preprint arXiv:1506.02004* (2015).

[12] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. “Placing search in context: The concept revisited”. In: *Proceedings of the 10th international conference on World Wide Web*. 2001, pp. 406–414.

[13] Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. “Simverb-3500: A large-scale evaluation set of verb similarity”. In: *arXiv preprint arXiv:1608.00869* (2016).

[14] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[15] Aditya Grover and Jure Leskovec. “node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[16] Jian Guan, Fei Huang, Zhihao Zhao, Xiaoyan Zhu, and Minlie Huang. “A knowledge-enhanced pretraining model for commonsense story generation”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 93–108.

[17] Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. “Large-scale learning of word relatedness with constraints”. In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, pp. 1406–1414.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[19] Felix Hill, Roi Reichart, and Anna Korhonen. “Simlex-999: Evaluating semantic models with (genuine) similarity estimation”. In: *Computational Linguistics* 41.4 (2015), pp. 665–695.

[20] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[21] Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. “Topical word embeddings”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[22] Minh-Thang Luong, Richard Socher, and Christopher D Manning. “Better word representations with recursive neural networks for morphology”. In: *Proceedings of the seventeenth conference on computational natural language learning*. 2013, pp. 104–113.

[23] Todor Mihaylov and Anette Frank. “Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge”. In: *arXiv preprint arXiv:1805.07858* (2018).

[24] Tomas Mikolov. *word2vec - default*. URL: `https://code.google.com/archive/p/word2vec/source/default/source`.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[26] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. “Advances in pre-training distributed word representations”. In: *arXiv preprint arXiv:1712.09405* (2017).

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. “Distributed representations of words and phrases and their compositionality”. In: *arXiv preprint arXiv:1310.4546* (2013).

[28] George A Miller and Walter G Charles. “Contextual correlates of semantic similarity”. In: *Language and cognitive processes* 6.1 (1991), pp. 1–28.

[29] Brian Murphy, Partha Talukdar, and Tom Mitchell. “Learning effective and interpretable semantic models using non-negative sparse embedding”. In: *Proceedings of COLING 2012*. 2012, pp. 1933–1950.

[30] Abhishek Panigrahi, Harsha Vardhan Simhadri, and Chiranjib Bhattacharyya. “Word2Sense: sparse interpretable word embeddings”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 5692–5705.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *arXiv preprint arXiv:1912.01703* (2019).

[32] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “Deepwalk: Online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[34] Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. “A word at a time: computing word relatedness using temporal semantic analysis”. In: *Proceedings of the 20th international conference on World wide web*. 2011, pp. 337–346.

[35] Radim Řehůřek and Petr Sojka. “Software Framework for Topic Modelling with Large Corpora”. English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. `http://is.muni.cz/publication/884893/en`. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[36] Herbert Rubenstein and John B Goodenough. “Contextual correlates of synonymy”. In: *Communications of the ACM* 8.10 (1965), pp. 627–633.

[37] Max Ryabinin, Sergei Popov, Liudmila Prokhorenkova, and Elena Voita. “Embedding Words in Non-Vector Space with Unsupervised Graph Learning”. In: *arXiv preprint arXiv:2010.02598* (2020).

[38] Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. “Spine: Sparse interpretable neural embeddings”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[39] Peter D Turney. “Mining the web for synonyms: PMI-IR versus LSA on TOEFL”. In: *European conference on machine learning*. Springer. 2001, pp. 491–502.

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

[41] Wei Xu and Alex Rudnicky. “Can artificial neural networks learn language models?” In: *Sixth international conference on spoken language processing*. 2000.

[42] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. “ERNIE: Enhanced language representation with informative entities”. In: *arXiv preprint arXiv:1905.07129* (2019).