UNIVERSITEIT VAN AMSTERDAM

MSc Artificial Intelligence
Master Thesis

Knowledge Generation

by
Florian Wolf
12393339

December 27, 2020

48 Credits
April 2020 - December 2020

Supervisor:
Dr Peter Bloem
Chiara Spruijt

Asessor:
Dr Paul Groth

Contents

Abstract

We generate Knowledge! [1]

We apply genius idea of having a VAE learn latent features of the data to KGs. Building on successful models of prior work, a linear, a convolutional and a architecture combining both methods are tested on the two most popular KGs. To best possible train out models on sparse graph representation, we implement a permutation invariant loss function. We compare the performance of our models to sate of the art link predictors, as well as link predictors including the variational module. The results compare ??? The model x outperfoms the others, indicating that convolutions are [not] necessary. Interpolation of the latent space shows that the model learns features??? When generating subgraphs with up to x nodes, we see ??? Finally we filter generated triples for predicates which imply the subject or object to be entity of the class location. x out of these triples adhere to this axiom. Thus we can say that VAE's are to a certain extend able to capture the underlying semantics of a KG.

1 Introduction

Here comes a beautiful introduction. Promise!

1.1 Motivation

Computer vision reached a point, where semantics, entities and relations can be infered in an simple image. Would it not be fantastic to be able to apply this to text too? Could we train a model to learn the semantics of a KG?

1.2 Expected Contribution

This thesis is aimed to be at fundamental research and provide insight, into if further research in this direction would be meaningful.

1.3 Research Question

`How well can a VAE learn the underlying semantics of a KG?`

2 Related Work

This section presents previous work which inspired and layed the fundamentals for this thesis.

2.1 Graph VAE

Present different papers with graph VAEs

- Belli recurrent VAE
- GraphVAE paper
- some more

The model architecture presented in the GraphVAE paper is the starting point of our model.

2.2 RGCN

Present the relational graph convolution model paper by Kipf and maybe others

2.3 Embedding Based Link Prediction

Present RASCAL and one or two more.

Embeddings
TransE represents entities in in low-dimensional embedding. The relationships between entities are represented by the vector between two entities [2]. (How are different relation between the same entities represented?)

OntoUSP
This method learns a hierarchical structure to better represent the relations between entities in embedding space.

3 Background

In this section we will go over related work and relevant background information for our model and experiments. The depth of the explanation is adopted to the expected prior knowledge of the reader. The reader is supposed to know the basics of machine learning and deep learning, including probability theory and basic knowledge on neural networks and their different architectures. Basic principles such as forward pass, backpropagation and convolutions are expected to be understood. Further the use and functionality of deep learning modules such as the model, the optimizer and the terms target and prediction should be known. This also includes being familiar with the training and testing pipeline of a model in deep learning.

Should all these boxes be checked, then we can expect to get a deeper understanding of the magic behind the VAE and its differences to a normal autoencoder. After that we will present how convolutional layers can be used on graphs. Of course where there is a layer there is a model, thus we are presenting the graph convolutional network (GCN). Closing the circle we show how we can adopt the VAE to graph convolutions. Wrapping things up, we present the state of the art algorithms for graph matching, which will be util to allow permutation invariance when matching prediction and target [3].

3.1 Knowledge Graph

Knowledge graph has become a popular key phrase. Yet, the term is so broad, that it can have various definitions. In this thesis we are going to focus on KGs in the context of relational machine learning.

A KG is a database and as all other databases it is used to store data. The main difference to tabular databases is that KGs store data in a relational fashion. A standard KG structure, introduced by the semantic-web community, is the Resource Description Framework (RDF). It is a so called schema-based approach, meaning that every entity has a unique identifier and all possible relations are stored in a vocabulary. The opposite schema-free approach is used in OpenIE models for information extraction. Here any type of triple can be extracted, eg. (Michelangelo, painted, Sixtine Chapel). Where as a triple in RDF format from the Freebase KG has the form

$$(/m/02mjmr, /people/person/born-in, /m/03gh4) \tag{1}$$

equation (s, r, o)

with:

- Subject $s : /m/02mjmr$ Barack Obama
- Relation/Predicate $r : /people/person/born-in$

- Object o : $/m/03gh4$ Hawaii

For all triples s and o are part of a set of so called entities, while r is part of a set of relations. This is enough to define a basic KG.

Schema-based KG can include type hierarchies and type constrains. Classes group entities of the same type together, based on common criteria, e.g all names of people can be grouped in the class 'person'. Hierarchies define the inheriting structure of classes and subclasses. Picking up our previous example, 'child' would be a subclass of 'person' and inherit its properties. At the same time the class of an entity can be key to a relation with type constrain, since some relations can only be used in conjunction with entities fulfilling the constraining type criteria.

These schema based rules of a KG are defined in its ontology. Here properties of classes, subclasses and constrains for relations and many more are defined. Again we we have to differentiate here between KGs with open-world or closed-world assumption. In a closed-world assumption all constrains must be sufficiently satisfied before a triple is accepted as valid. This leads to a huge ontology and makes it difficult to expand the KG. On the other hand open-world KGs such as Freebase, accept every triple as valid, as long as it does not violate a constrain. This leads inevitably to inconsistencies within the KG, yet it is the preferred approach for large KGs. In context of this thesis we refer to the ontology as semantics of a KG, we research if our model can capture the implied closed-world semantics of an open-world KG [4].

Lastly, we point out one major difference between KGs, namely their representation. RDF KGs are represented as set of triples, consisting of a unique combination of numeric indices. Each index linking to the corresponding entry in the entity and relation vocabulary. This is called dense representation and benefits from fast computation due to an optimized use of memory.

In contrary the dense representation of a triple is the sparse representation. Here a binary square matrix also called the adjacency matrix, indicated a link between two entities. To identify the node, each node in the adjacency matrix has a one-hot encoded entity-vocabulary vector. All one-hot encoded vectors are concatenated to a node attribute matrix. In simple cases, like citation networks this is a sufficient representation. In the case of Freebase, we need an additional edge-attribute matrix, which indicates the relation of each link. The main benefits of this method are the representation of subsets of triples, also subgraphs, with more than one relation and the computational possibility to perform graph convolutions.

3.2 Graph VAE – one shot method

3.2.1 VAE

The VAE as first presented by [5] is an unsupervised generative model in form of an autoencoder, consisting of an encoder and a decoder. We will The architecture of the VAE differs from a common autoencoder by having a stochastic module between encoder and decoder. The encoder can be represented as recognition model with the probability $p_{\boldsymbol{\Theta}}(\mathbf{z} \mid x)$ with x being the varbiable we want to inference and z being the latent representation given an observed value of x. The encoder parameters are represented by Θ. Sinilarly, We denote the decoder as $p_{\boldsymbol{\Theta}}(\mathbf{x} \mid z)$, which given a latent representation z produces a probability distribution for the possible values, corresponding to the input of x. This will be the base architecture of all our models in this thesis.

The main contribution of the VAE is the so called reparameterization trick. By sampling from the latent prior distribution, we get stochastic module inside our model, which can not be backpropagates through and makes machine learning not possible. By placing the stochastic module outside the model FIGURE!!!, we can again backpropagate. We use the predicted latent space as mean and variance for a Gaussian normal distribution, from which we then sample ϵ, which acts as external parameter and does not need to be updated.

This makes the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$ intractable. Thus, we assume that the prior to the decoder to be Gaussian with an approximately diagonal covariance, which gives us the approximated posterior.

$$\log q_{\phi}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) = \log \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\mathbf{I}\right) \tag{2}$$

This gives us computational freedom. meaning we can compute the posterior probability. Using the

Monte Carlo estimation of $q_\phi(\mathbf{z} \mid \mathbf{x})$ we get the so called estimated lower bound (ELBO):

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{3}$$

We denote the first term the regularization term, as it forces the model into using a Gaussian normal prior. The second term represents the reconstruction loss, matching the prediction with the target.

While we use a discrete input space, the output space is a continuous probability. To generate final result, the prediction is used as binomial probability distribution, from which we then sample. Once training of a VAE is completed, the Decoder can be used on its own to generate new samples by using latent input signals [5].

3.2.2 MLP

The Multi-Layer Perceptron (MLP) was the beginning of machine learning models. Its properties as universal approximator has been discovered and widely studied since 1989. The innovation it brought to existing models was the hidden layer between the input and the output.

The mathematical definition of the MLP is rather simple. It takes linear input vector of the form $x_1, ..., x_D$ which is multiplied by the weight matrix $\mathbf{w^{(1)}}$ and then transformed using a non-linear activation function $h(.Duetoitssimplederivative, mostlytherectifiedlinearunit(ReLU)functionisused.Thisresultsinthehiddenlayer, consistin$ and finally transformed by a sigmoid function $\sigma(\dot{})$, which produces the output.Grouping all weight and bias parameter together we get the following equation for the MLP:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{4}$$

for $j = 1, \ldots, M$ and $k = 1, \ldots, K$, with M being the total number of hidden units and K of the output.

Since the sigmoid function gives us a probability output, the main function of the MLP is as a classifier. Instead of the initial sigmoid function, it was found to also produce good results for multi label classification transforming the output with a softmax function instead. Images or higher dimensional tensors can be processed by flattening them to a one dimensional tensor. This makes the MLP a flexible and easy to implement model [**bishop2006pattern**].

3.2.3 Graph convolutions

Convolutional neural nets (CNN) have the advantage to be invariant to permutations od the input. Convolutional layers exploit the property of datapoints which are close to each other and thus, have a higher correlation.

CNNs have shown great results in the field of images classification and object detection. Neighboring pixel contain information about each other which can let the model detect local features which can then be merged to high-level features [**bishop2006pattern**].

Similar conditions hold for graphs. Neighboring nodes contain information about each other and can used to infer local features.

Let us shortly go over the definition and math behind graph convolutions. Different approaches have been published on this topic, here we will present the graph convolution network (GCN) of [6]. We consider $f(X, A)$ a GCN with an undirected graph input $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $v_i \in \mathcal{V}$ is a set of N nodes and $(v_i, v_i) \in \mathcal{E}$ the set of edges. The input is a sparse graph representation, with X being a node feature matrix and $A \in R^{N \times N}$ being the adjacency matrix, defining the position of edges between nodes. In the initial case, where self-connections are not considered, the adjacency's diagonal has to be one resulting in $\vec{A} = A + I_N$. The graph forward pass

through the convolutional layer l is then defined as

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{5}$$

Here $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ acts as normalizing constant. $W^{(l)}$ is the layer-specific weight matrix and contains the learnable parameters. H returns then the hidden representation of the input graph.

The GCN was first introduced as node classifier, meaning it returns a probability distribution over all classes for each node in the input graph $\mathcal{V}$. Assuming that we preprocess $\vec{A}$ as $\hat{A} = \tilde{D}^{-\frac{1}{2}}\vec{A}\tilde{D}^{-\frac{1}{2}}$, the equation for a two-layer GCN for z classes is

$$Z = f(X, A) = \text{softmax}\left(\hat{A}\,\text{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right). \tag{6}$$

3.2.4 RGCN

GCN which takes further input of edge attribute matrix.

Either present
Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs
or nixx

3.2.5 Graph VAE

Now we have all the building bocks for a Graph VAE. The encode can either be a MLP, a GCNN or an RGCN. The same holds for the decoder with the addition that model architecture needs to be inverted. An version of a Graph VAE presented in [7]. This model combines both the previous methods. The input graph undergoes relational graph convolutions before it is flattened and projected into latent space. After applying the reparametrization trick, a simple MLP decoder is used to regenerate the graph. In addition the model concatenates the input with a target vector y, which represents ???. The same vector is concatenated with the latent tensor. ***Elborate why they do that***.

Graphs can be generated recursively or in an one-shot approach. This paper uses the second approach and generates the full graph in one go. ***Cite?***

3.2.6 One Shot vs. Recursive

The concept of the VAE has been used to generate data for various usecases. When using the VAE as generator, by sampling from the approximated posterior distribution $q_\phi(\mathbf{z}$, we can reconstruct the data in a singular run or recursive manner.

The one shot method is the used in the popular example of the VAE generator on the MNIST dataset [cite], as well as on [the faces dataset]. each sample is independent from each other.

Recursive methods take part of the generated datapoint as input for the next datapoint, thus continuously generating the sample. This has been applied to [voice] and to reproduce videogame environments [8]. In [9] variation of the GraphVAE has been used to recursively construct a vector roadmap, which has been presented in [link].

For this thesis, we will use the one-shot method, predicting each datapoint independent from each other. The predictions will sparse graph representation with n nodes. A single triple being $n = 2$ and a subgraph representation $2 < n < 100$. [TODO]

3.3 Graph Matching

Intro to graph matching on sparse graphs.

3.3.1 Permutation Invariance

Permutation invariance refers to the invariance of a permutation of an object. An visual example is the the image generation of numbers. If the loss function of the model would not be permutation invariant, the generated image could show a perfect replica of the input number but being translated by one pixel the loss function would penalize the model. Geometrical permutations can be translation, scale or rotation around any axis.

In the context of sparse graphs the most common, and relevant permutation for this thesis, is be the position of a link in the adjacency matrix. By altering its position with help of a permutation matrix, the link will connect different nodes. When matching graphs with more than two nodes, we can partially match the graphs by permuting only a subgraph instead of the full graph. This way also graphs of different sizes, can be matched.

A model is permutation invariant, when it includes a function to detect and match such permutations between target and prediction.

3.3.2 Max-Pool Graph matching algorithm

While there are numerous graph matching algorithms, we will focus on the max-pool algorithm, which can be effectively implement and used in the VAE setting. First presented in [10] in the context of computer vision and successfully trained to match graphs from feature points in an image. It uses a max-pooling graph matching approach, which is resilient to deformations and highly tolerant to outliers. The output is a reliable cost matrix. Notable is, that also graphs with different number of nodes can be matched.

Let us introduce a new sparse representation for a sampled subgraph, where the discrete target graph is $G = (A, E, F)$ and the continuous predicted graph $\widetilde{G} = (\widetilde{A}, \widetilde{E}, \widetilde{F})$. The A, E, F are store the discrete data for the adjacency, for node attributes and the node attribute matrix of form $A \in \{0,1\}^{n \times n'}$ with n being the number of nodes in the target graph. $E \in \{0,1\}^{n \times n' d_e}$ is the edge attribute matrix and a node attribute tensor of the shape $F \in \{0,1\}^{n \times d_n}$ with d_e and d_n being the size of the entity and relation dictionary. For the predicted graph with k nodes, the adjacency matrix is $\widetilde{A} \in [0,1]^{k \times k}$, the edge attribute matrix is $\widetilde{E} \in R^{k \times k \times d_e}$ and the node attribute matrix is $\widetilde{F} \in R^{k \times d_n}$.

GIven these graphs the algorithm aims to find the affinity matrix $S : (i,j) \times (a,b) \to R^+$ where $i, j \in G$ and $a, b \in \widetilde{G}$. The affinity matrix expresses the similarity of all nodepairs between the two graphs and is calculated

$$S((i,j),(a,b)) = \left(E^T_{i,j,.}, \widetilde{E}_{a,b,.}\right) A_{i,j} \widetilde{A}_{a,b} \widetilde{A}_{a,a} \widetilde{A}_{b,b} [i \neq j \wedge a \neq b] + \left(F^T_{i,.} \widetilde{F}_{a,.}\right) \widetilde{A}_{a,a} [i = j \wedge a = b] \tag{7}$$

where the square brackets define Iverson brackets [7].

The next step is to find the similarity matrix $X^* \in [0,1]^{k \times n}$. Therefore we iterate a fist-order optimization framework and get the update rule

$$\mathbf{x}_{t+1} \leftarrow \frac{1}{\|\mathbf{S}\mathbf{x}_t\|_2} \mathbf{S}\mathbf{x}_t. \tag{8}$$

To calculate Sx we find the best candidate $x_{i,a}$ from the possible pairs in the affinity matrix. Heuristically taking the argmax over all neighboring nodepair affinities yields the best result. Other options are sum-pooling or average-pooling, which do not discard potentially irrelevant information. Thus we get can

pairwise calculate

$$\mathbf{S}\mathbf{x}_{ia} = \mathbf{x}_{ia}\mathbf{S}_{ia;ia} + \sum_{j \in \mathcal{N}_i} \max_{b \in \mathcal{N}_a} \mathbf{x}_{jb}\mathbf{S}_{ia;jb}. \tag{9}$$

Depending on the matrix size, the number of iterations are adjusted. The resulting similarity matrix $X*$ yields a normalized probability of matching nodepairs. In order to get a discrete translation matrix, we need to find the optimal match for each node.

3.3.3 Hungarian algorithm

Picking up on the nodepair probabilities X^* of last chapter, we reformulate it as a linear assignment problem. Here comes into play an optimization algorithm, the so called hungarian algorithm. It original objective is to optimally assign n resources to n tasks. The cost of assigning task $i \in R^n$ to $j \in R^n$ is stored in x_{ij} of the quadratic cost matrix $x \in R^{n \times n}$. By assuming tasks and resources are simple nodes and taking $C = 1 - X^*$ we get the cost matrix C for the optimal translation. This algorithm is has a complexity of $O\left(n^4\right)$, thus is not applicable to complete KGs but only to subgraphs with limited number of nodes per graph [11].

The core of the hungarian algorithm consist of four main steps, initial reduction, optimality check, augmented search and update. The now presented algorithm is a slight alternative of the original algorithm and improves the complexity of the update step from $O\left(n^2\right)$ to $O\left(n\right)$ and thus, reduces the total complexity to $O\left(n^3\right)$. It takes as input a bipartie graph $G = (V, U, E)$ and the cost matrix $C \in R^{n \times n}$ where $V \in R^n$ and $U \in R^n$ are sets of nodes and $E \in R^n$ the set of edges between the nodes. The algorithm's output is a discrete matching matrix M. To avoid two irrelevant pages of pseudocode, the steps of the algorithm are presented in a short summary [12].

1. Initialization:

   (a) Initialize empty matching matrix $M_0 = \emptyset$.

   (b) Assign α_i and β_i as follows:

   $$\begin{aligned} &\forall v_i \in V, && \alpha_i = 0 \\ &\forall u_i \in U, && \beta_j = \min_i \left(c_{ij}\right) \end{aligned}$$

2. Loop n times over the different stages:

   (a) Each unmatched node in V is a root node for an Hungarian tree with when completed results in an augmentation path.

   (b) Expand the Hungarian trees in the equality subgraph. Store the indices i of v_i encountered in the Hungarian tree in the set $I*$ and similar for j in u_j and the set J^*. If an augmentation path is found, skip the next step.

   (c) Update α and β to add new edges to the quality subgraph and redo the previous step.

   $$\theta = \frac{1}{2} \min_{i \in I^*, j \notin J^*} \left(c_{ij} - \alpha_i - \beta_j\right)$$
   $$\alpha_i \leftarrow \begin{cases} \alpha_i + \theta & i \in I^* \\ \alpha_i - \theta & i \notin I^* \end{cases}$$
   $$\beta_j \leftarrow \begin{cases} \beta_j - \theta & j \in J^* \\ \beta_j + \theta & j \notin J^* \end{cases}$$

   (d) Augment M_{k-1} by flipping the unmatched with the matched edges on the selected augmentation path. Thus M_k is given by $(M_{k-1} - P) \cup (P - M_{k-1})$ and P is the set of edges of the current augmentation path.

3. Output M_n of the last and n^{th} stage.

3.3.4 Graph Matching Loss

The loss is described in [7] as.

In contrast to his implementation we assume, that a node or edge can have none, one or multiple attributes. Therefore our attributes are also not sigmoided and do not sum up to one. This leads o the modification of term logF and logE where we do not matrix multiply over the attribute vector but take the BCE as over the rest of the points. KG can have multiple or no attributes vs molecular graphs can be one hot encoded.

Okay, further we need to treat the log_pE and log_pF just like log_pA and subtract the inverse. Otherwise the model learn to predict very high values only.

A note to the node features, these stay softmaxed and one-hot encoded since we will use them as node labels.

3.4 Ranger Optimizer

Finalizing this chapter, we introduce the novel optimizer Ranger. A optimizer, which placed itself on the top of 12 FastAI leaderboards.Ranger combines Rectified Adam (RAdam), lookahead and optionally gradient centralization. let us shortly look into the different components.

RAdam is based on the very popular adam optimizer. It improves the learning by dynamically rectifying Adam's adaptive momentum. It does this by reducing the variance of the momentum, which is especially large at the beginning of the training. Thus, leading to a more stable and accelerated start [13].

The Lookahead optimizer proposes an approach where, a second optimizer 'looks ahead' on a set of parallel trained weights. while the computation and memory cost are negligible, it learning improves and the variance of the main optimizer is reduced [14].

The last and most novel optimization technique, Gradient Centralization, acts directly on the gradient by normalizing it to a zero mean. Especially on Deep Neural Networks, this helps regularizing the gradient and boosts learning. This method can be added to existing optimizers and can be seen as constrain of the loss function [15].

Concluding we can say that Ranger is a state of the art deep learning optimizer with accelerating and stabilizing properties, incorporating three different optimization methods, which synergize with each other. Considering that generative models are especially unstable during training, we see Ranger as a good fit for this research.

LookAhead was inspired by recent advances in the understanding of loss surfaces of deep neural networks, and provides a breakthrough in robust and stable exploration during the entirety of training.

4 Methods

This section describes the methods used in the experiments of this thesis. The first part includes the presentation of the model, the reprocessing of the input and the evaluation metrics. The second part describes the experimental setup and the different experimental runs. The work of this thesis has aimed to be fully reproducible, thus the code is open-sourced and available on Github [1].

4.1 Knowledge graph representation

The first step in our pipeline is the representation of the KG in tensor format. In order to represent the graph structure we use an adjacency matrix A of shape $n \times n$ with n being the number of nodes in our graph. The edge attribute or directed relations between the nodes are represented in the matrix E of shape $n \times n \times d_E$ with

[1] ***Thesis Repo Link***

d_E being the number of edge attributes. Similarly for node attributes we have the matrix F of shape $n \times d_N$ with d_N number of node attributes. The input graph can have less nodes than the maximum n but not more. The diagonal of the adjacency matrix is filled with 1 if the indexed node exists, and with 0 otherwise. The number and encoding of the attributes must be predefined and cannot be changed after training. This way we can uniquely represent a KG.

Graph embeddings? unsupervised approach

4.2 Graph VAE

hi
Convolution part
RCGN relation Convolution neural net
MLP encoder
Latent space
reparametrization trick
MLP decoder
Graph matching
Discretization of prediction

Limitations. The proposed model is expected to be useful only for generating small graphs. This is due to growth of GPU memory requirements and number of parameters (O(k 2)) as well as matching complexity (O(k 4)), with small decrease in quality for high values of k. I [7]

4.3 Loss function

If loss function should be permutation invariant we need to do some kind of graph matching.
Different options for graph matching.
Maxpooling-algorithm:
Assumptions
Node to edge affinity equals 0

Self-loops are possible, adjacency matrix can be zero or one.

Summing over the neighbors means summing over the whole column Normalize matrix with Frobenius Norm

Batch version: Only matmul and dot. keep dimension of S with shape (bs,n,n,k,k) When maxpooling, flatten Xs (n,k) for batch dot multiplication. This way (i think) we sum over all j nad b neighbors instead of taking the max.

Hungarian algorithm for discrimination of X

The hungarian algorithm as presented in section 2 return the shortest path in a matrix. We use this shortest path as bast match between the two graphs. The node paris identified as optimal are masked as 1 and the rest of the matrix as 0. This way we discretizice $X\star$ to X.
The equations

5 Experiments & Results

This section presents the experiments we ran. We covered link and node prediction and compared those to SOTA scores. Further we ran experiments on investigating the coherence of the reproduced graph structure. Lastly we measured the adherence of our model to the KG's underlying syntax.

5.1 Datasets and Training

For this sake of meaningful results, we chose to use the earlier presented, two most popular dataset used in this field, FB15k237 and WN18rr.

Training models on each dataset for 333 epochs, without early stopping.

5.2 Link Prediction

We used link prediction as evaluation protocol and for comparison state of the art models. For each triple in the dataset, we remove the tail and combine it with all possible entities in our dataset. EVen though this is called 'triple-corruption', also correct triples can be generated, which could appear in the trainset. These have bo the filtered out before evaluation. Link prediction on unfiltered test data is termed 'raw'. Our model then computes the ELBO loss for all corrupted triples, which are sorted and stored in ascending order. The same procedure is repeated the triple's head in place of the tail.

The metrics used to evaluate the predictions, is the mean reciprocal rank (MRR) and hits at 1,3 and 10. The MRR ??? explain what the MRR is.
Hits at 1 indicates what percentage of the triples in the test set have been ranked the highest compared to their corruptions. Similar, hits at 3 and 10, give a percentage of triples ranked in the top 3 and top 10. These metrics allow a fair comparison on a dataset between models, regardless of the ranking method of each model.

5.3 Sanity Checks

Check if adj matrix adheres to edge attribute matrix.

5.4 Interpolate Latent Space

We tale two random triples and interpolate the latent space of these two triples. The interpolations result in: HERE AN EXAMPLE.

Further we go ahead and test what happens if we modify one latent dimension of a triple. HERE AN EXAPMPLE. Can the model assign logical features to latent dimensions?

5.5 Subgraph Generation

Until now our model trained on only one triple per sparse graph. What will happen if we train it on more than one triples?

5.6 Syntax coherence

Check if generated triples follow basic logic.

- Generate triples by random signals
- Filter these triples on a certain relation
- Check if the entities are part of the linked class

Since there is no preset for how to check the semantics of a KG, we will use simple basic logical criteria. The generated triples are filtered for the relation 'is capital of', thus the subject enntity should be a city and he object entity member of the class 'country', Hope this gives good results.

6 Discussion & Future Work

We will discuss certain aspects of our results and give advice on further research.

6.1 Discuss Aspect 1

Well well well

6.2 Future Work

Basing on the believe that the experiments were successful, we recommend:

- Improve the model (deeper)
- link the latent space to word signals e,g text
- prior not normal, e.g NF
- try a GAN

Good luck amigos!

Ideas

The bigger picture of this thesis is to efficiently generate a representation of the information hold in plain text. This representation has the form of a knowledge graph and consists of subject-relation-object triples.
Challenges:

- What knowledge are we looking for? I would like the model to focus on the most important information. This could for example be topic specific.
- Another option would be to extract based on a query or point of interest. Especially if we build the KG incrementally we can use this input as starting point and recursively build from there.

6.3 Use Normalizing Flows

- Is it possible to generate a graph from text using Normalizing flow (NF)?
- Can we train such a NF unsupervised, using the output distributions as loss function?
- NFs can build KG at once or incrementally. No one is using it, could have a reason.

6.4 Recurrent VAE

- Recursively generate graph using a Variational Auto Encoder.
- Use query as start node.
- expand graph on that node. Thiviyan and others work on VAEs as well. Seems an easier approach. Can analyze the latent space.

The encoder decoder strategy has been applied successfully in many cases. Bellis thesis about modeling a graph from images can be applied to text as well by changing the input to a text vector [9]. Note that this is a supervised method and would require a dataset of text labeld with resulting KGs. To the best of my knowledge this does not exist yet.
Same holds for the contrastive world model by Kipf. If we input text vectors instead of an image the model could recognize different objects in the text as it does with pixels [1]. Here the model is trained unsupervised using a loss over the energy function of the graph embedding space TransE [2]. An open question is if this would work for our approach.
A bit more abstract is the idea of the feedback recurrent VAE [16] where sound signals are encoded and decoded. This could also be adopted to text for instance with one sentence at a time, or a fixed number of tokens. Here the text vector would be induced as the latent input to the decoder. This would mean finding an translation of the models latent space to the work embedding. The dataset would consist of positive and negative examples of resulting graphs over timesteps.
While text can be easily vectorized by word embeddings like word2vec, the graph representation seems more tricky. A reasonable approach following Bellis example would be to output a coordinates vector for the nodes and an adjantency matrix for their relationships. Here one node at a time is outputted and the relation is conditioned on the number of previous nodes. Alternatively we could make use of the graph embeddings RASCAL or TransE.
Lastly the question remains how to model the graph when the nodes need to be predefined. A subgraph of DBpedia could be a good starting point.

Open Issues

A section where note will be made on open issus. These aissues are ment to be discussed with Peter or Thivyian and ultimatly solved. No open issues should remain at the end of November.

6.5 Graph matching

Hungarian algorithm: Should we assume the affinity matrix is a cost or a profit matrix. If we assume it is a cost matrix and n != k the algorithm returns an error while padding. If we assume it is a profit matrix and convert it to a profit matrix it can handle any dimension. The shortest path varies with both approaches. Further I assume the shortest path is what we are looking for and mask these entries with 1 and the rest of the matrix with 0.
Diagonal of A is zeros. As soon as I do this, code crashes. Think it is because of divided zero, Martin adds 1e-7 to the X norm
Do we backpropagate through the graph matching? then all torch functions, otherwise np.
Martin takes 300 iterations for hungarian. also not looping over pairs.

References

1. Kipf, T., van der Pol, E. & Welling, M. Contrastive Learning of Structured World Models. *arXiv:1911.12247 [cs, stat]*. arXiv: 1911.12247. http://arxiv.org/abs/1911.12247 (2020) (Jan. 5, 2020).
2. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J. & Yakhnenko, O. in *Advances in Neural Information Processing Systems 26* (eds Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q.) 2787–2795 (Curran Associates, Inc., 2013). http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf (2020).
3. Paulheim, H. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web* **8** (ed Cimiano, P.) 489–508. ISSN: 22104968, 15700844. https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-160218 (2020) (Dec. 6, 2016).
4. Nickel, M., Murphy, K., Tresp, V. & Gabrilovich, E. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE* **104,** 11–33. ISSN: 0018-9219, 1558-2256. arXiv: 1503.00759. http://arxiv.org/abs/1503.00759 (2020) (Jan. 2016).
5. Kingma, D. P. & Welling, M. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*. arXiv: 1312.6114. http://arxiv.org/abs/1312.6114 (2020) (May 1, 2014).

6. Kipf, T. N. & Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*. arXiv: 1609.02907. http://arxiv.org/abs/1609.02907 (2020) (Feb. 22, 2017).

7. Simonovsky, M. & Komodakis, N. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. *arXiv:1802.03480 [cs]*. arXiv: 1802.03480. http://arxiv.org/abs/1802.03480 (2020) (Feb. 9, 2018).

8. Ha, D. & Schmidhuber, J. World Models. *arXiv:1803.10122 [cs, stat]*. arXiv: 1803.10122. http://arxiv.org/abs/1803.10122 (2020) (Mar. 28, 2018).

9. Belli, D. & Kipf, T. Image-Conditioned Graph Generation for Road Network Extraction. *arXiv:1910.14388 [cs, stat]*. arXiv: 1910.14388. http://arxiv.org/abs/1910.14388 (2020) (Oct. 31, 2019).

10. Cho, M., Sun, J., Duchenne, O. & Ponce, J. *Finding Matches in a Haystack: A Max-Pooling Strategy for Graph Matching in the Presence of Outliers* in *2014 IEEE Conference on Computer Vision and Pattern Recognition* 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (IEEE, Columbus, OH, USA, June 2014), 2091–2098. ISBN: 978-1-4799-5118-5. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909665 (2020).

11. Date, K. & Nagi, R. GPU-accelerated Hungarian algorithms for the Linear Assignment Problem. *Parallel Computing* **57,** 52–72. ISSN: 0167-8191. http://www.sciencedirect.com/science/article/pii/S016781911630045X (2020) (Sept. 1, 2016).

12. Mills-Tettey, G. A., Stentz, A. & Dias, M. B. The dynamic hungarian algorithm for the assignment problem with changing costs (2007).

13. Liu, L. *et al.* On the Variance of the Adaptive Learning Rate and Beyond. *arXiv:1908.03265 [cs, stat]*. arXiv: 1908.03265. http://arxiv.org/abs/1908.03265 (2020) (Apr. 17, 2020).

14. Zhang, M. R., Lucas, J., Hinton, G. & Ba, J. Lookahead Optimizer: k steps forward, 1 step back. *arXiv:1907.08610 [cs, stat]*. version: 1. arXiv: 1907.08610. http://arxiv.org/abs/1907.08610 (2020) (July 19, 2019).

15. Yong, H., Huang, J., Hua, X. & Zhang, L. Gradient Centralization: A New Optimization Technique for Deep Neural Networks. *arXiv:2004.01461 [cs]*. arXiv: 2004.01461. http://arxiv.org/abs/2004.01461 (2020) (Apr. 7, 2020).

16. Yang, Y., Sautière, G., Ryu, J. J. & Cohen, T. S. Feedback Recurrent AutoEncoder. *arXiv:1911.04018 [cs, eess, stat]*. arXiv: 1911.04018. http://arxiv.org/abs/1911.04018 (2020) (Feb. 17, 2020).