UNIVERSITEIT VAN AMSTERDAM

MSc Artificial Intelligence
Master Thesis

Knowledge Generation

by
Florian Wolf
12393339

January 11, 2021

48 Credits
April 2020 - December 2020

Supervisor:
Dr Peter Bloem
Thiviyan Singam
Chiara Spruijt

Asessor:
Dr Paul Groth

Contents

List of Tables

Abstract

This thesis investigates the idea of having a VAE learn latent features of the raw data from KGs. Building on successful approaches of prior work, an embedding based, a fully-connected and a convolutional model are evaluated on the two most popular KGs. The influence of a permutation invariant loss function on the models capability to generalize on sparse subgraphs is analyzed. We compare the performance of our models on the task of link prediction to sate of the art scores. The results compare ??? The model x outperforms the others, indicating that convolutions are [/not] necessary. Interpolation of the latent space shows that the model learns to featurize latent dimensions??? When generating subgraphs with up to n nodes, we see ??? Finally we filter generated triples for type constrained predicates on subject or object. A $x\%$ of the generated 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

To begin with, we should clarify the intended ambiguity of this work's title. We are without a doubt the most knowledgeable generation in the history of humanity. While we continuously keep researching and accumulating knowledge, we neither share our knowledge nor apply it for the better of any other species in our Universe. All scientific milestones are from us, for us.

The rise of Artificial Intelligence (AI) marked a turning point in history. For the first time we invest in sharing knowledge and in creating systems which can take our knowledge to superhuman dimension. While machine learning models might not be considered a specie, they have in fact surpassed human intelligence in certain fields (yes, DeepMind's AlphaGO) [1]. This is not only seen as progress but also as danger. The world's richest man, Elon Musk, both build his fortune on AI and respects it as humanity's biggest risk.

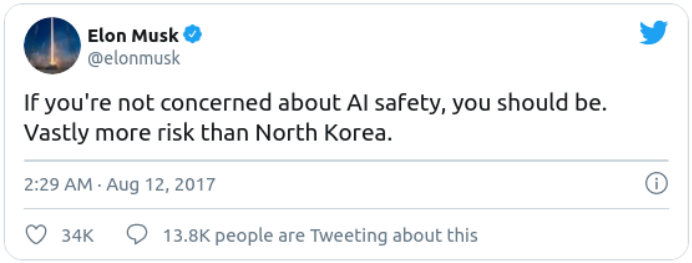


Figure 1: Elon Musk in a tweet on AI. Source [2]

Closing the circle to the ambiguity of the title and relating to the popular concern, that AI might reach a point where it does not need humans anymore to keep evolving, we ask the crucial question: Can AI generate knowledge?

1.1 Motivation

Turning away from philosophy, we will now introduce this thesis in scientific context. A key area of AI is representation learning, where the model learns to identify and disentangle causalities and underlying features of the data. Understanding the semantics of the data is specifically useful for unsupervised learning.

The task of generating data has been widely explored for images. Computer vision has reached a point where a simple image can be semantically segmented, where objects can be detected and classified and even relations between entities inferred [3].

Advances in the parallel field of graph generation have received less attention, yet showed promising results. Data stored as graph has a high density of information and rich semantics, which makes it attractive for variational inference. The recent success of Simonovsky et al. [4] on the generation and completion of molecules represented in graph structure, initially inspired our research. Next to molecules, graphs can be used to store knowledge. While real world Knowledge Graphs (KG) have a far higher complexity than molecule

graphs, the proposed generative model, Variational Auto-Encoder (VAE) also has proven its capacity to learn from huge datasets with high variance. Inspired by Simonovsky work and motivated by the vision of generating knowledge, we will explore the possibilities and limitation of KG generation with VAEs.

1.2 Expected Contribution

The main contributions, which we anticipate for this thesis are firstly the proof of concept that a graph VAE can capture, disentangle and reproduce the underlying semantics of a real world KG. While Simonovsky used small subgraphs with multiple edges, we proof our hypothesis on generating the smallest possible graph of two nodes, also representable as single triple. The VAE will be tested and evaluated in several experiments, including link prediction, latent space interpolation and accuracy of generating valid triples.

We will compare our results to related methods in the filed of KGs and investigate the impact of different hyperparameter. The main focus will be on the influence of the graph matching loss function, the encoding through graph convolutions and stochastic inference. Further, we aim to mimic the success of to molecule generation and will, thus, point out similarities and differences to our work.

On a lower level, we hope to contribute with our implementation of the max-pooling graph matching algorithm for tensor batches. While the algorithm has been cited and implemented numerous times, a working implementation, compatible with deep learning tasks, has to the best of our knowledge not yet been published.

Lastly we introduce a high-level method for evaluating the validity of generated data, which compares the type constrain of the generated triple's predicate with its entity types and reports accuracy. This is made possible by expanding the existing dataset FB15K-237 with entity types from its original KG Freebase. While this scoring method is error prone, it does give an insight into the level representational potential of the model and the syntax coherence of the generated triples. Future work can use this evaluation method to track progress and compare to the baseline.

1.3 Research Question

`How successful is a VAE in representation learning on real world KG compared to molecule graph data and what is the impact of each major hyperparameter?`

Without further ado -

2 Related Work

This section presents previous work which inspired and layed the fundamentals for this thesis. Relevant papers to three topics related to this thesis will be presented. We will present their method and results in the fields of relational graph convolutions, graph encoders, and embedding based link prediction.

2.1 Relational Graph Convolutions

We define a graph as $G = (\mathcal{V}, \mathcal{E})$ with a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. The set of edges, with each edge connecting node x and y, is defined by $\{(x, y) \mid (x, y) \in \mathcal{V}^2 \wedge x \neq y\}$ while the constrain $x \neq y$ prohibits self-connections or self-loops, which is optional depending on the graphs function. Moreover, nodes and edges can have describing features, which contribute additional information about the nodes and their connection. Using graph convolutions, we make use of these properties holding spectral information about their neighboring nodes and relations. The two main standards to evaluate the performance of a neural network on graphs, are node classification and link prediction. Node classification is a classification problem where the model predicts a probability distribution over all classes for each node. During link prediction the model scores a set of one real and corrupted triples and aims to score the highest on the real triple. A more in-depth explanation follows later on in chapter 4.5.

In Thomas Kipf's and Max Welling's first paper on graph convolutions [5] a novel Graph Convolution Network (GCN) for semi-supervised classification is introduced. This method acts directly on the graph structure and shows to be linearly scalable with the number of nodes. While the authors compare different propagation models for the graph convolutions, their propagating rule using a first-order approximation of spectral graph convolutions, outperforms all other implementations. The so-called renormalization trick normalized the adjacency matrix and adds it to an identity matrix of same size. This keeps the eigenvalues in a range between $[0, 2]$ which again leads to a stable training, avoiding numerical instabilities and vanishing gradients during learning. Additionally the feature information of neighboring nodes is propagated in every layer what shows improvement in comparison to earlier methods, where only label information is aggregated. Kipf and Welling perform node classification on the three citation-network datasets, Citeseer, Cora and Pubmed as well as on the KG dataset NELL. In all classification tasks, their results outperform other recently proposed methods in this field and proves to be computationally more efficient than its competition. For more details on the implementation of graph convolutions we refer to the background section 3.2.3.

In their publication *Modeling Relational Data with Graph Convolutional Networks* Schlichtkrull, Kipf, Bloem, v.d. Berg, Titov and Welling propose a relational graph convolutional network (RGCN) and evaluate it on link prediction on the FB15K-237 and WN18 dataset and node classification on the AIFB, MUTAG, BGS and AM datasets [6]. While the RGCN, with its encoder properties, is used by itself as node classifier, yet for link prediction it is coupled with a DistMult model acting as decoder which scores triples encoded by the RGCN see 2. More details on DistMult can be found in 2.3.

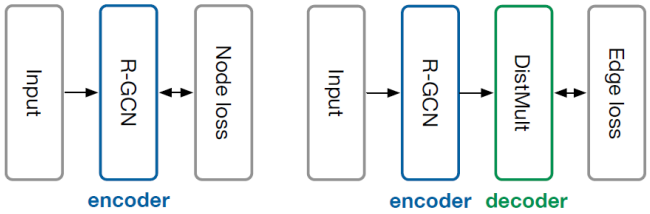


Figure 2: RGCN pipeline for node classification and link prediction experiments. The first pipeline only uses an encoder to classify the input nodes. The second pipeline additionally uses a decoder to score the input and predict the correct link, Source [6].

The RGCN works on graphs stored as dense triples, creating a hidden state for head and tail of each triple. A novel message passing network is layer-wise propagated with the hidden states of the entities. As regularization the authors propose a *basis-* and *blockwise* decomposition. while the first aims at an effective weight sharing between different relation types, the second can be seen as a sparsity constraint on the relation type's weight. The model outperforms embedding based model on the link prediction task on the FB15K-237 dataset and scores competitive on the WN18 dataset. In the node classification task, the model sets state of the art results on the datasets AIFB and AM, while scoring competitive on the remaining. The authors conclude, that the model has difficulties encoding higher-degree hub nodes on datasets with many entities and low amount of classes.

2.2 Graph VAE

We have seen how graph convolutional neural networks can be combined to a encoder-decoder architecture, resulting in a generative model suitable for unsupervised learning. We will present three recent publications with different methods and use cases of a graph generative model, in particular a VAE.

In yet another publication of Kipf and Welling they introduce the Variational Graph Autoencoder (VGAE), a framework for unsupervised learning on graph-structured data [7]. This generative model uses a GCN as encoder and a simple inner product module as decoder. Similar to th GCN the VGAE incorporates node features, what significantly improves its performance on link prediction tasks compared to related models. The VGAE uses a two-layer GCN to encode the mean and the logvariance for the stochastic module to sample the latent space representation. The activation of the inner product of this latent vector yields the reconstruction of the adjacency matrix. Figure 3 shows how the model learns to represent the underlying data structure by grouping the Gaussian latent representations of the datapoints according to their class, without these labels being provided to the model during training.

The VGAE with added features outperforms state of the art models in the task of link prediction on

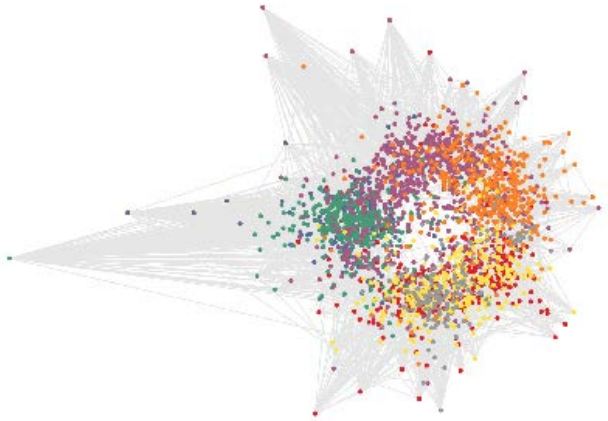


Figure 3: Colorized vizualization of the latent representation of the VGAE trained on the Core citation network with colors differentiating document classes and gray links indicating citations. This shows that the model implies and featurizes the document classes, without them being provided during training. Source [7]

the datasets Cora, Citeseer and Pubmed. The authors point out, that a Gaussian prior might be a poor choice combined with the inner-product decoder.

While citation networks represent basic graph structures, work has also been done on more complex KGs. Simonovsky and Komodakis introduce the GraphVAE, a generative model which outputs a probabilistic fully-connected graph of a predefined maximum size in a one-shot approach [4]. The model includes a standard graph matching algorithm to align the predicted graph to the ground truth. In contrast to the previously presented publications, the input to this model is a threefold and sparse graph, defined as $G = (A, E, F)$ with A being the adjacency matrix, E the edge attribute matrix and F the node attribute matrix, with E and F being one-hot encoded. Considering that this method lays the foundation for this thesis, we will adopt this notation for our own methods in 4.5. Figure 4 shows the architecture of the GraphVAE. The encoder is a feed forward network with edge-conditioned graph convolutions. After the convolutions the result is flattened and conditioned on the node labels y. A simple neural network then encodes the stochastic latent space, as we know it from previous works. The decoder is again conditioned on the node labels y and in form of a fully-connected neural network reconstructs the latent representation to the graph prediction. The threefold decoder output is matched with the target using graph matching algorithm, which we will discuss further in 3.3. The best matching permutation is then used for the reconstruction loss term of the GraphVAE. It should be noted, that the size of the target and prediction graph do not necessarily have to match. While this approach seems promising, the maximum graph size is limited to a node count of 100 by computational memory requirements.

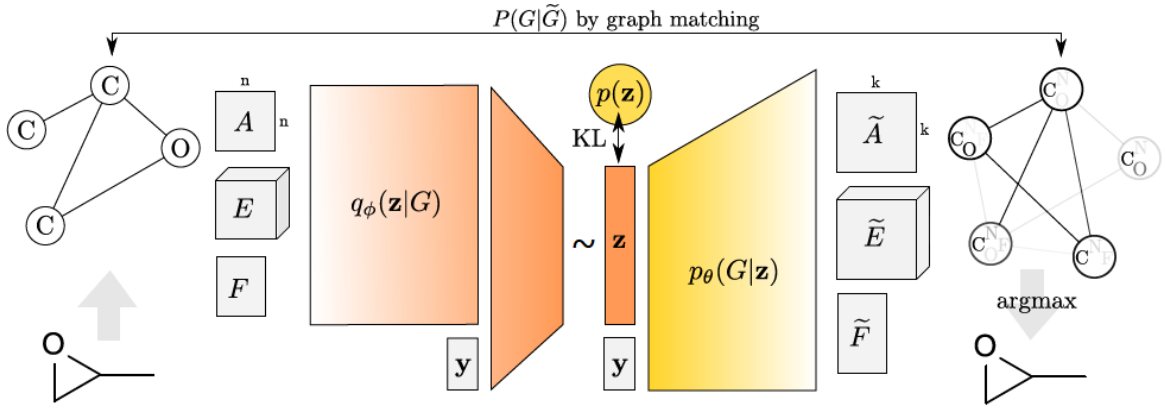


Figure 4: Model architecture of the GraphVAE. The target graph with n nodes is encoded and conditioned on the node labels y. The KL divergence ensures a Gaussian prior to the decoder, which reconstructs the latent representation to a graph with k nodes. Target and prediction graphs are matched and permuted before the reconstruction loss. To sample, the argmax is taken directly from the prediction. Source [4].

The model is trained on the QM9 dataset, containing the graph structure of 134k organic molecules

with experiments on latent space dimension in the range of $[20, 80]$. On the free generation task, about 50% of the generated molecules are chemically valid and thereof remarkably 60% are not included in the trainings dataset. When testing the model for robustness, it showed little disturbance when adding Gaussian noise to the input graph G. The authors conclude that the problem of generating graphs from a continuous embedding was addressed successfully and that the GraphVAE performs better on small molecules, thus worse on larger graphs.

In a little sidestep, we present the idea of supervised graph generation in an autoregressive fashion by Belli and Kipf and their publication *Image-Conditioned Graph Generation for Road Network Extraction* [8]. While we will focus on the generative model, their contribution ranges wider, namely the introduction of the graph-based roadmap dataset *Toulouse Road Network* and the task specific distance metric *StreetMover*. The authors propose the Generative Graph Transformer (GGT) a deep autoregressive model that makes use of attention mechanisms on images, to tackle the challenging task of road network extraction from image data. The GGT has a encoder-decoder architecture, with a CNN as encoder, taking the grayscale image as input signal and predicting a conditioning vector. The decoder is a self-attentive transformer, which takes as input the encoded condition vector and a hidden representation of the adjacency matrix A and feature vector X of the previous step. The adjacency matrix here indicates the links between steps and the features are normalized coordinates. A multi-head operator outputs the hidden representation of A and X which finally are decoded by a MLP to the graph representation. For the fist step, a empty hidden representation is fed into the decoder. The model terminates the recurrent graph generation by predicting a end-of-sequence token, which signalizes the end of the graph. During learning, the generated graphs are matched to the target graphs using the *StreetMover* metric, based on the Sinkhorn distance. The authors attribute *StreetMover* as a scalable, efficient and permutation-invariant metric for graph comparison. The successful results of the experiments performed, show that this novel approach is suitable for the task of road network extraction and could yield similar success in graph generation task of different fields. While this publication does not directly align with the previously presented work, we find it of added value to present alternative approaches on our topic.

2.3 Embedding Based Link Prediction

Finalizing this chapter, we will look at embedding based methods on KGs. This approach was inspired by the success of word-embeddings in the field of NLP. Compared to the previously presented research, embedding models have a much simpler architecture and can be trained computationally very efficiently on large graphs. Embedding based models can only operate on simple triples, meaning a KG is represented as a set of triples with indices pointing to the unique entity and relation in the graph. In disregard of their simplicity, they achieve great results on relational prediction tasks such as link prediction. In particular, this means to rank a correct triple the highest between a set of corrupted triples. In order to generate corrupt triples, each triple from the test set is modified by replacing the head or tail with all remaining entities occurring in the KG. Link prediction will be explained in more detail in 4.5.

Already in 2013 Bordes et al. introduced in their paper *Translating Embeddings for Modeling Multi-relational Data* the low-dimensional embedding model TransE [9]. The core idea of this model is that relations can be represented as translations in the embedding space. Entities are encoded to a low-dimension embedding space and the relation is represented as vector between the head and tail entity. The assumption is, that correct triples have a lower norm of the relational vector than corrupted triples, thus, the distance between head and tail entity in embedding space is less. The loss function for learning of the model takes a set of corrupted triples for every triples in the training set and subtracts the translation vector of the corrupted triple in embedding space from the translation vector of the correct triple with added margin. To minimize the loss, the model has to place entities of correct triples closer together in embedding space. We think of a triple as (s, r, o) and the bold notation as its embedded representation, $d()$ the distance measure, γ the positive margin and S and S' as sets of correct and corrupt triples, the loss function of TransE is

$$\mathcal{L} = \sum_{S} \sum_{S'} \left[\gamma + d(\boldsymbol{s} + \boldsymbol{r}, \boldsymbol{o}) - d\left(\boldsymbol{s}' + \boldsymbol{r}, \boldsymbol{o}'\right)\right] \tag{1}$$

The model is trained on the two KGs Freebase and Wordnet, which will also be the source for the datasets used in this thesis. TransE's link prediction results on both head and tail outperformed other competing methods of the time, such as RESCAL [10].

In 2015, Yang et al. proposed a similar, yet better performing KG embedding method [11]. Their model

DistMult captures relational semantics by matrix multiplication of the embedded entity representation and uses a bilinear learning objective. The main difference to TransE is the bilinear scoring function $d_r^b()$, where bilinear denotes the score invariance of swapping head and tail entity. For the embedding space representation of subject and object $\boldsymbol{s}$ and $\boldsymbol{o}$ and a diagonal matrix $\mathbf{M}_r \in \mathbb{R}^{n \times n}$ as embedding of r, the scoring function is

$$d_r^b(\boldsymbol{s}, \boldsymbol{o}) = \boldsymbol{s}^T \mathbf{M}_r \boldsymbol{o} \tag{2}$$

The publication goes on to explore the options of embedding based rule extraction from KGs. Concluding, the authors state that that embeddings learned from the bilinear objective not only outperform the state of the art in link prediction but can also capture compositional semantics of relations and extract Horn rules using compositional reasoning.

In a more recent publication by Ruffinelli et al. re-trained outdated KG embedding models such as TransE and DistMult with state of the art techniques in deep learning. The authors start by pointing out the similarities and differences of most models. While all methods share the same embedding approach, they differ in their scoring function and their original hyperparameter search. The authors perform a quasi-random hyperparameter search on the five models RESCAL, TransE, DistMult, ComplEx and ConvE using the two datasets FB15K-237 and WN18. For evaluation the MRR and Hits@10 are used. Since these metrics and datasets will be used later on in our research, they are explained in 5.1(datasets) and 4.4(metrics). The tuned models report a higher MRR score of up to 24% compared to their first reported performance. The authors conclude that simple KG embedding methods can show strong performance when trained with state-of-the-art techniques and score competitive to or even outperforme more recent architectures. The improved model configurations were found by exploring relatively few random samples from a large hyperparameter space.

3 Background

This section derives and explains the techniques, which form the backbone of this research. For fundamental background on machine learning, deep learning or including probability theory we refer the reader to Bishops book [12]. We begin with introducing the VAE and its differences to a normal autoencoder. Building on this base model, we will answer the question how convolutional layers can be used on graphs. On a higher level we present model to this layer, 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 graph [13].

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.

KGs are a subclass of databases, with its main function being data storage. 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, e.g. (Michelangelo, painted, Sixtine Chapel). In contrast a triple which we wil denote as (s, r, o) in RDF format from Freebase, one of the largest open-source KGs, has the form

```
[['/m/02mjmr'], ['/people/person/place_of_birth'], ['/m/02hrh0_']]
```

with the *id2text* translation of Wikidata [**WIKIDATA**]:

- Subject s: `['/m/02mjmr Barack Obama']`
- Relation/Predicate r: `[/people/person/born-in]`
- Object o: `['/m/02hrh0_ Honolulu']`

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 constraints. 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 the 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 constraints for relations and many more are defined. Again, we have to differentiate between KGs with open-world or closed-world assumption. In a closed-world assumption all constraints 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 again 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 [14].

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, indicates 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 benefit of this method is the representation of subsets of triples, also referred to as subgraphs, with more than one relation and the possibility to perform graph convolutions.

3.2 One Shot Graph VAE

3.2.1 VAE

The VAE as first presented by [15] is an unsupervised generative model in form of an autoencoder, consisting of an encoder and a decoder. Its architecture 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 $p_\theta(\mathbf{z} \mid x)$ with x being the variable we want to inference and z being the latent representation given an observed value of x. The encoder parameters are represented by θ. Similarly, we denote the decoder as $p_\theta(\mathbf{x} \mid z)$, 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 reparametrization trick. When sampling from the latent prior distribution, we get a stochastic module inside our model, which can not be backpropagated and makes learning impossible. By placing the stochastic module outside the model, 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.

In 5 we see, that the true posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ is intractable. Thus, we make the assumption that the prior to the decoder is Gaussian with an approximately diagonal covariance, which gives us the approximated posterior

$$\log q_\phi\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) = \log \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\mathbf{I}\right). \tag{3}$$

Now variational inference can be performed, which allows both θ the generative and ϕ the variational parameters to be learned jointly. Using the Monte Carlo estimation of $q_\phi(\mathbf{z} \mid \mathbf{x})$ we get the so called estimated

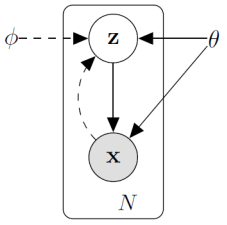


Figure 5: Representation of the VAE as bayesian network, with solid lines denoting the generator $p_\theta(z)p_\theta(\mathbf{x} \mid z)$ and the dashed lines the posterior approximation $q_\phi(\mathbf{z} \mid \mathbf{x})$ [15].

lower bound (ELBO)

$$\mathcal{L}\left(\theta, \phi; \mathbf{x}^{(i)}\right) = -D_{KL}\left(q_\phi\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\phi\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\left[\log p_\theta\left(\mathbf{x}^{(i)} \mid \mathbf{z}\right)\right]. \tag{4}$$

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.

The VAE is a unsupervised learning model where the input s]pace is discrete and the output space a continuous logits. To generate final results, the prediction activated via Sigmoid and is used as mean for a 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 [15].

3.2.2 MLP

The Multi-Layer Perceptron (MLP) is the mother of all neural networks. 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 activated using a non-linear function $h(\dot{)}$, which results in the hidden representation of $\mathbf{x}$. Due to its simple derivative, mostly the rectified linear unit (ReLU) function is used as activation. The hidden units get multiplied with the second weight matrix, denoted $\mathbf{w^{(2)}}$ and finally transformed by a sigmoid function $\sigma(\dot{)}$, which produces the output. Grouping weight and bias parameter together we get the following equation for the MLP

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{5}$$

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 returns a probability distribution for all classes, the MLP can have the function of a classifier. Instead of the initial sigmoid function, it was found to also produce good results for multi label classification activating the output through 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 [12].

3.2.3 Graph convolutions

Convolutional neural nets (CNN) have the advantage to be invariant to permutations of 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, thus by detecting local features, the model can then be merged those to high-level

features, e.g a face in an image [12]. Similar conditions hold for graphs. Neighboring nodes contain information about each other and can be 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 [5]. We consider $f(X, A)$ a GCN with an undirected graph input $G = (\mathcal{V}, \mathcal{E})$, 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 $A \in \mathbb{R}^{N \times N}$ 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

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{6}$$

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. 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

$$Z = f(X, A) = \text{softmax}\left(\hat{A}\,\text{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right). \tag{7}$$

3.2.4 Graph VAE

So far we have understood all the principles needed for a Graph VAE. While there are many approaches with mayor differences in terms of the model and graph representation, e.g sparse or dense, we will focus here on the graph VAE architecture presented in [4]. A sparse graph model with graph convolutions.

Simonovsky's GraphVAE follows the characterizing encoder decoder architecture. The encoder $q_\phi(\mathbf{z} \mid G)$ takes a graph G as input, on which graph convolutions are applied. After the convolutions the hidden representation is flattened and concatenated with the node label vector y. A simple MLP encodes the mean and logvariance of the latentspace distribution. Using the reparametrization trick we sample the latent representation.

For the decoder $p_\theta(\mathbf{x} \mid G)$ the latent representation is again concatenated with the node labels. The decoder architecture for this model is a MLP with inverted dimensions, which outputs a flat prediction of $\tilde{G}$, which is split and reshaped in to the sparse matrix representation.

Simonovsky's work was mainly focused on molecule data. To adopt this model for multi-relational KGs and to reduce it to a minimum viable product, we drop the conditioning on the node labels and use the node attributes as pointers towards the corresponding entity in $\mathcal{E}$. By using node attributes as unique pointers, we hide additional information about the entity, such as type and topic from our model. Simplifying further we drop the convolutional layer and directly flatten G as input for the MLP encoder.

TODO Make Illustration

3.2.5 One Shot vs. Recursive

The concept of the VAE has been used to generate data for various use cases. 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 used in the popular example of the VAE generator on the MNIST dataset [15]. 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 generate audio and to reproduce videogame environments [16].

In [8] variation of the GraphVAE has been used to recursively construct a vector roadmap, which has been presented in [link to related work].

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

3.3 Graph Matching

In this subsection we will explain the term permutation invariance and present an algorithm to find the optimal permutation in order to match two graphs. We then derive the loss term of the graph VAE 3.2.4 applying the optimal matching matrix to the models prediction.

3.3.1 Permutation Invariance

Permutation invariance refers to the invariance of a permutation of an object. A visual example is 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 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.

In context of this thesis, a model or a function is called permutation invariant, if it can match any permutation of the original. For the case of matching between prediction and target graph, from all possible permutations of the predictions, the one with highest similarity to the target should be considered. The similarity can be full or partly as its possible that only parts of the graph match or the graphs in size differ.

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 implemented and used in the VAE setting. First presented in [17] 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. $E \in \{0,1\}^{n \times n' \times d_e}$ 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 $\widetilde{E} \in \mathbb{R}^{k \times k \times d_e}$ and the node attribute matrix is $\widetilde{F} \in \mathbb{R}^{k \times d_n}$.

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

$$S((i,j),(a,b)) = \left(E^T_{i,j,\cdot}, \widetilde{E}_{a,b,\cdot}\right) A_{i,j} \widetilde{A}_{a,b} \widetilde{A}_{a,a} \widetilde{A}_{b,b} [i \neq j \wedge a \neq b] + \left(F^T_{i,\cdot} \widetilde{F}_{a,\cdot}\right) \widetilde{A}_{a,a} [i = j \wedge a = b]. \tag{8}$$

Here the square brackets define Iverson brackets [4].

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

$$\mathbf{x}_{t+1} \leftarrow \frac{1}{\|\mathbf{S}\mathbf{x}_t\|_2}\mathbf{S}\mathbf{x}_t. \tag{9}$$

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 node pair affinities yields the best result. Other options are sum-pooling or average-pooling, which do not discard potentially irrelevant information, yet have shown to perform worse. Thus, using the max-pooling approach, we can pairwise calculate

$$\mathbf{S}\mathbf{x}_{ia} = \mathbf{x}_{ia}\mathbf{S}_{ia;ia} + \sum_{j \in \mathbb{N}_i} \max_{b \in \mathbb{N}_a} \mathbf{x}_{jb}\mathbf{S}_{ia;jb}. \tag{10}$$

Depending on the matrix size, the number of iterations are adjusted. The resulting similarity matrix $X*$ yields a normalized probability of matching node pairs. 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 node pair 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 $i \in \mathbb{R}^n$ to $j \in \mathbb{R}^n$ is stored in x_{ij} of the quadratic cost matrix $x \in \mathbb{R}^{n \times n}$. 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 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 [18].

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 $C \in \mathbb{R}^{n \times n}$ where $V \in \mathbb{R}^n$ and $U \in \mathbb{R}^n$ are sets of nodes and $E \in \mathbb{R}^{n \times n}$ 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 the following short summary [**mills2007dynamic**].

1. Initialization:

   (a) Initialize the 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 a Hungarian tree with 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.

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

(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 VAE Loss

Coming back to our generative model, we now explain how the loss function needs to be adjusted to work with graphs and graph matching, which results in a permutation invariant graph VAE.

The normal VAE maximizes the evidence lower-bound or, in a practical implementation, minimizes the upper-bound on negative log-likelihood. Using the notation of 3.2.1 the graph VAE loss is

$$\mathcal{L}(\phi, \theta; G) = \quad = \mathbb{E}_{q_\phi(\mathbf{z}|G)} \left[-\log p_\theta(G \mid \mathbf{z}) \right] + \mathrm{KL} \left[q_\phi(\mathbf{z} \mid G) \| p(\mathbf{z}) \right]. \tag{11}$$

The loss function $\mathcal{L}$ is a combination of reconstruction term and regularization term. The regularization term is the KL divergence between a standard normal distribution and the latent space distribution of Z. This term does not change when adopting to graphs. The reconstruction term is the binary cross entropy between prediction and target, which in the sparse graph representation are threefold with A, E, F.

The predicted output of the decoder is split in three parts and while $\tilde{A}$ is activated through sigmoid, $\tilde{E}$ and $\tilde{F}$ are activated via edge- and nodewise Softmax. The graph matching permutation for A is applied to the target $A' = XAX^T$ and for E and F on the prediction, $\widetilde{F}' = X^T \widetilde{F}$ and $\widetilde{E}'_{\cdot,\cdot,l} = X^T \widetilde{E}_{\cdot,\cdot,l} X$, l being the one-hot encoded edge attribute vector which is permuted. These permuted subgraphs are then used to calculate the maximum log-likelihood estimate [4]:

$$\begin{aligned} \log p\left(A' \mid \mathbf{z}\right) = & 1/k \sum_a A'_{a,a} \log \widetilde{A}_{a,a} + \left(1 - A'_{a,a}\right) \log \left(1 - \widetilde{A}_{a,a}\right) + \\ & + 1/k(k-1) \sum_{a \neq b} A'_{a,b} \log \widetilde{A}_{a,b} + \left(1 - A'_{a,b}\right) \log \left(1 - \widetilde{A}_{a,b}\right) \end{aligned} \tag{12}$$

$$\log p(F \mid \mathbf{z}) = 1/n \sum_i \log F^T_{i,\cdot} \widetilde{F}'_{i,} \tag{13}$$

$$\log p(E \mid \mathbf{z}) = 1/\left(\|A\|_1 - n\right) \sum_{i \neq j} \log E^T_{i,j,}, \widetilde{E}'_{i,j,\cdot} \tag{14}$$

Note that k can be equal n if target and prediction graphs have the same number of nodes. The normalizing constant $1/k(k-1)$ takes into account the no self-loops restriction, thus an edge-less diagonal. In the case of self loops this constant would be $1/k * k$. Similar $1/\left(\|A\|_1 - n\right)$ for $\log p(E \mid \mathbf{z})$ where $-n$ accounts for the edge-less diagonal and in case of self-loops would be discarded leaving us with $1/\left(\|A\|_1\right)$.

3.4 Ranger Optimizer

Finalizing this chapter, we introduce the novel deep learning optimizer Ranger. An optimizer, which in 2020 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 popular adam optimizer. It improves the learning by dynamically rectifying Adam's adaptive momentum. This is done 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 [19].

The Lookahead optimizer was inspired by recent advances in the understanding of loss surfaces of deep neural networks, thus proposes an approach where, a second optimizer 'looks ahead' on a set of parallel trained weights. while the computation and memory cost are negligible, the learning improves and the variance of the main optimizer is reduced [20].

The last and most novel optimization technique, Gradient Centralization, acts directly on the gradient by normalizing it to a zero mean. Especially on convolutional 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 [21].

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.

4 Methods

This section describes the methods used in the experiments of this thesis. We will begin with data formatting and preprocessing, then the presentation the model, mainly the encoder and decoder and their variations. Moving on we explain our implementation of graph matching the loss function of the model and its evaluation metrics. To the end of this chapter we describe the link prediction pipeline which is the main experiment to evaluate our model. Our implementation is written in Python using PyTorch, a high-performance deep-learning library [22]. All experiments are aimed to be fully reproducible and the model meant to be used in future work, thus the code is openly available on Github [1].

4.1 Knowledge graph data

All our presented methods will operate on KG data. While data from other graph domains is possible, this work focuses solely on datasets in triple format. We will explain the sparse graph representation, which is the input format for our model and how to preprocess the original KG triples to match that format.

4.1.1 Sparse representation

In this work, we opt for the sparse graph representation $G(A, E, F)$, where A denotes the adjacency matrix, E the edge feature matrix and F the node feature matrix. This allows models architectures as discussed in 3.2.4. The graph is binary and each matrix is stored in a separate tensor.

The adjacency matrix A takes the shape $(n \times n)$ with n being the number of nodes in our graph/subgraph. While most of previous work would only allow edges on the upper triangular adjacency matrix and fill the diagonal with ones, we chose a less constrained representation, which we assume is a better fit for representing KGs. In particular, we allow self-loops, meaning a triple where object and subject are the same entity and our relations are directed and can be inverted. Thus A can have a positive signal at any position $A_{i,j}$ $i, j \in \mathbb{R}^{n \times n}$, indicating a directed edge between node of index i and node of index j, while $A_{i,j}$ differs from $A_{j,i}$.

[1] `https://github.com/INDElab/rgvae`

The edge attribute matrix E takes the shape $(n \times n \times d_e)$ with d_e being the number of unique entities in our dataset. For each possible edge in the adjacency matrix we have a one hot encoded vector pointing to the unique relation in the dataset. Stacking these vectors leads to the three dimensional matrix E.

The shape of node attributes matrix F is $(n \times d_e)$ with d_e being the number of node attributes describing the nodes. Considering that we will split the KG in subgraphs, we use the entity index as node attribute, making it possible to assign every node in a subgraph to the entity in the full KG. Thus, the number of node attributes d_e equals the number unique entities in our dataset. Again the node attributes are one hot encoded vectors, which stacked result in the two dimensional F matrix.

4.1.2 Preprocessing

Our datasets consist of three tab separated value files full of triples for training, evaluation and final prediction. The preprocessing steps do not only account for generating subgraphs in the right format, but also ensure valuable research by withholding the triples from the test set until the final run.

From all three sets, we create a set of all occurring entities and similar set for the relations. Now we can define our dimensions d_e and d_r. For both sets we create two dictionaries *index-2-entity* and *entity-2-index* which map back and forth between numerical index and the string representation of the entity (similar for the relation set). These dictionaries are used to create a train and test set of triples with numeric indices. Depending if we are in the final testing stage or not, we include all triples from the training and evaluation file in the training set and use the triples in the testing file as test set, or we ignore the triples in the test file and use the evaluation file triples as test set.

Further we create two dictionaries, *head* and *tail* which for all occurring subject and relation combination, contain all entities which would complete it to a real triple in our dataset (similar for all relation and object combinations). This will allows us to filter true triples, an important part of link prediction and helpful in graph generation.

The final step of preprocessing is a function, which takes a batch of numerical triples and converts them to a batch of binary, multidimensional tensors A, E and F. While this might sound easy for only one triple per graph, it proves more complex for graphs with $n > 2$ facing exemption cases such as self loops or an entity occurring in two triples. We solve this by creating a separate set for head and tail entities, then storing the indices od both in a list, starting with the subject set and finally using this list as keys for a dictionary with values in the range to n. In both edge cases, this results in an adjacency matrix with a rank lower than n. A similar approach, with less edge cases to consider, is used to apply the inverted translation from tensor matrix to triple.

4.2 RGVAE

The principle of a graph VAE has been explained in 3.2.4, which also covers the foundation of our model, the Relational Graph VAE (RGVAE). Therefore we will focus on the implementation as well as parameter and hyperparameter choice. Since this work is meant to be a prove of concept rather than aimed at outperforming the state of the art, our model is kept as simple as possible and only as complex as necessary. Our approach is module based for both experiment pipeline and model, meaning independence between sequential modules and compatibility with parallel modules. For the encoder we implemented two variations, a fully connected and a convolutional, while for the decoder we opted for a single fully connected network.

4.2.1 Initialization

The RGVAE is initialized with a set of hyperparameter, which define the input shape. Table **??** shows a complete list of those parameters and their default values. It is left to mention that we use the Xavier uniform method with a gain of $0,01$ to initialize the weight parameter.

Hyerp.	Default	Description
n	2	Number of nodes
d_e	-	Total number of entities
d_r	-	Total number of relations
d_z	100	Latent space dimension
d_h	512	Hidden dimension
dropout	0.2	Dropout
β	1	β value for regularization
perminv	**True**	Permutation invariant loss function
clipgrad	**True**	Learning w/ gradient clipping
encoder	**MLP**	Choice of encoder architecture

Table 1: The initial hyperparameter of the RGVAE with default value and description.

4.2.2 Encoder

The prove-of-concept encoder is a MLP as described in 3.2.2, which takes the flattened concatenated threefold graph $x = G(A, E, F)$ as batch input. We use the initial parameters to calculate the input

$$d_{input} = n \times n + n \times n \times d_r + n \times d_e \tag{15}$$

The main encoder architecture is a 3 layer fully connected network, with both layers using ReLU as activation function. The choice for two hidden layers is based on the huge difference between d_{input} and d_z. The first layer has a dimension of $2 * d_h$ and the option to use dropout, which by default is set to 0.2. The second (hidden) layer has the dimension d_h which is by default set to 1024. After the second ReLU activation, the encoder linearly transforms the hidden state to an output vector of $2 \times d_z$. This vector is split and makes the mean and log-variance of size d_z for the reparametrization trick. Sampling ϵ from an autonomous module, we get the latent representation z of x, the final output of the encoder.

The second option for our RGVAE encoder is a GCN as described in 3.2.3. We adopt the architecture from [5] namely two layers of graph convolutions with dropout in between. To match the encoder output to the base model, we then add a flattening and a final linear transformation layer. To substitute the feature matrix used in Kipf's work, we reduce the edge attribute matrix E by one dimension and concatenate it with F resulting in $x_{GCN} \in \mathbb{R}^{n \times (d_e + n * d_r)}$. The forward pass of the adjacency matrix A and x_{GCN} through the first GCN layer with a hidden dimension of d_h and ReLU as activation function is followed by a dropout layer. It should be mentioned that dropout is only applied during learning, not on evaluation. The second GCN layer takes the output from the previous layer, the two dimesntional hidden state and again A as input. Now, instead of having the GCN predict on a number of classes, we use it to output a logits vector of dimension 2_z. Therefore we pass the GCN output through a flattening and a linear transformation layer. Similar to the above described encoder we use the reparametrization trick to output the latent reparametrization z. Table 2 shows the two encoder architectures side by side.

MLP	Graph Conv.
Flatten(d_{input})	Concatenate
Linear($d_{input}, d_h \times 2$)	Convolution(A, X)
ReLu()	ReLU()
Dropout(0.2)	Dropout(0.2)
Linear($d_h \times 2, d_h$)	Convolution($H^{(1)}, X$)
ReLu()	Flatten
Linear(d_h, d_z)	Linear($d_{H^{(1)}}, d_z$)

Table 2: Comparison of the two variants for the encoder of the RGVAE.

4.2.3 Decoder

For our RGVAE decoder, we use the same minimal approach as in [4], namely an MLP with inverted dimensions. The decoder architecture is similar with inverse dimensions to the MPL encoder described in 2. Since we are

decoding the latent space, the input dimension is d_z and the output dimension is d_{input} as calculated in equation 15. The flat logits output tensor is split threefold and reshaped to the original input shape of $G(A, E, F)$.

To sample from the generated graph we apply the Sigmoid activation function to the logits prediction for the adjacency matrix and use the normalized output as weights for binomial distributions, from which we can sample the discrete $\tilde{A}$. For $\tilde{E}$ and $\tilde{F}$ we take the argmax on the last dimension of both matrices. Each node and edge can have only one attribute, referring to its index in $\mathcal{E}$ and $\mathcal{V}$, thus only the highest predicted value is relevant. The generated sample is a discrete graph $\tilde{G}(\tilde{A}, \tilde{E}, \tilde{F})$.

4.2.4 Limitations

The main limitation of the RGVAE is the parabolic increase of model parameters with the increase of nodes per input graph $\mathcal{O}(n^2)$. The number of parameters to train is directly linked with the GPU memory requirements. Even more computationally expensive is the use of permutation invariant graph matching, with a complexity of $\mathcal{O}(n^3)$. Thus, we propose this model only for generating small graphs with $n < 30$.

4.3 RGVAE learning

In this section we will present our implementation of how to fit the model to the data. Learning a model on data is a mostly standardized procedure, which includes training and evaluation per epoch. During training, the model forward passes the data, computes the loss, then does a backwards pass and updates its parameters. During evaluation, it is presented a split of the dataset unseen during training. Only the forward pass is done and the loss tracked during evaluation. Up to this point the RGVAE does not differ from the vanilla VAE training. Special is the graph matching function which is applied to the predicted graph and the loss function which takes into account the optimal permutation. Thus, we will look deeper into graph matching and derive RGVAE loss. The training and all experiments are performed on the GPU cluster LISA on a single node. The GPU equipped on this node is a Nvidia titan RX 25GB. To log our experiments results, we use *Weights Biases*, a cloud-based experiment tracking tool [23].

4.3.1 Max pooling graph matching

While the pseudocode presented in [17] is simple and straight forward, it proves complicated to implement this in algorithms for batches and thus, without looping over the indices. Yet, our batch implementation solves these challenges and is more efficient than the direct implementation, which we use for validating our results. Given the target graph G and the predicted graph $\tilde{G}$, the algorithm can be divided in three steps, calculating the five dimensional affinity matrix (the first being the batch dimension), max-pool matching the soft (continuous) similarity matrix X^* and discretizing X^* to our final permutation matrix X.

We use equation 8 for the first step but instead of adding the two terms to a single output, we return S twofold as S_r, five dimensional holding the information of edge affinity and S_e, three dimensional with the affinity information of the nodes. In a preprocessing step we zero out the diagonal of A, $\tilde{A}$ and for E and $\tilde{E}$ the diagonal of the second and third dimension, to compile with the constrain $[i \neq j \wedge a \neq b]$ of the first term. For the second term we only take into account the diagonal of $\tilde{A}$ to compile with the constrain $[i = j \wedge a = b]$. Pseudocode 1 shows the implementation, here *diag()* stands for a vector with only the diagonal entries. For the dot product of E and $\tilde{E}$ over the last dimension we implement our own version of `torch.matmul()` to cope with higher dimensions. The operator $\odot$ denotes element-wise matrix multiplication.

The next step is the graph matching algorithm is the max-pool loop presented in [17]. We initialize the similarity matrix as ones $X^* \in 1^{bs \times n \times n}$ with bs denoting the batch size. For a certain number of iterations, Cho proposes 40 but the number should be adjusted to the number of nodes in the graph, we multiply X^* with a reduced version of S and use its Frobenius norm as normalizer. The algorithm 2 shows our implementation for batches.

To the best of our knowledge, this is the first time this algorithm is implemented in batch style. Thus, we would like to believe that laying out the implementation in detail will contribute to the academic value of this thesis.

Algorithm 1 Batch implementation for the affinity between two graphs

Input: $G(A, E, F$ and $\tilde{G}(\tilde{A}, \tilde{E}, \tilde{F})$

First term: $[i \neq j \wedge a \neq b]$

$E_{term1} = E^T \tilde{E}$ ▷ Dot product over the last dimension

$A_{term1} = A \,.\, \text{unsqueeze}(-1)^T (\tilde{A} \odot (\tilde{A}\tilde{A}^T)) \,.\, \text{unsqueeze}(-1)$ ▷ Dot product over the last (empty) dimension

$S_r = E_{term1} \odot A_{term1}$

Second term: $[i = j \wedge a = b]$

$A_{term2} = \text{ones\_like}(\text{diag}(\tilde{A}))^T \,\text{diag}(\tilde{A})$

$F_{term2} = F^T \tilde{F}$ ▷ Dot product over the last dimension

$S_e = F_{term2} \odot A_{term2}$

return (S_r, S_e)

Algorithm 2 Max-pool graph matching for batches

Input: (S_r, S_e)

Init $X^* \in 1^{bs \times n \times n}$

for *iteration* $= 1, 2, \ldots$ **do**

  $S_{max} = \text{sum}(\max(S_r \odot X^* \,.\, \text{unsqueeze}([1, 1])))$ ▷ Sum and max over the last dimension. Unsqueeze two times on the second dimension

  $X^* = X^* \odot S_e + S_{max}$

  $X^* = X^* / \,\text{frobenius\_norm}(X^*)$

end for

return X^*

The last step in the graph matching pipeline is the discretization of X^*. We chose the Hungarian algorithm as presented in 3.3.3. To our disappointment and resulting in a bottleneck, no batch nor tensor implantation of the named algorithm has been published so far. Thus, we convert X^* to `numpy.array()` format and make use of the *Scipy* package [24]. First we create the cost matrix $X_{cost} = 1 - X^*$ and then, iterating over the batch size, use `scipy.optimize.linear_sum_assignment` which returns the optimal assignment matrix. This is our final permutation matrix X, indicating the best match between target and prediction. If no permutation is needed, X is the identity matrix of A.

4.3.2 Loss function

The RGVAE uses the ELBO loss from equation 4, consisting of two terms, the regularization loss and the reconstruction loss. We will present our implementation of both loss terms with graph matching and an alternative loss without graph matching for comparative evaluation of our results

TODO derive why A is permuted on target (BCE) and E and F on prediction (Categorical) when $n \neq k$.

We implement $\log p\,(A' \mid \mathbf{z})$ from equation 12 with the second normalizing constant as $1/k * K$ since we allow self-loops. The permutation matrix X is applied to to the target adjacency, resulting in A'. For $\log p\,(E' \mid \mathbf{z})$ and $\log p\,(F \mid \mathbf{z})$ the permutation is applied to the prediction, which in the case of E' requires our own implementation of matrix multiplication of $d > 2$. Taking into account self-loops we change the normalization constant of $\log p\,(E' \mid \mathbf{z})$ to $1/\left(\|A\|_1\right)$. It is left to mention that when implementing $\sum_{i \neq j} \log E^T_{i,j,}, \widetilde{E}'_{i,j,\cdot}$ in matrix multiplication style, we have to account for the zero values before taking the logarithm. We implement `torch.sum(torch.sum(torch.log(no_zero(`E `* `$\hat{E}$`)),-1)-1)` with `no_zero()` being a function which replaces 0 values with 1. This implementation of the loss function can be backpropageted with exception of the graph matching part, where the `numpy` implementation of the hungarian algorithm prevents backpropagation.

The regularization loss is given by the KL divergence between the approximated posterior $\log q_\phi\,(\mathbf{z} \mid \mathbf{x})$ and the Gaussian prior $p(\mathbf{z})$. The only modification we make to the original loss, is adding a β parameter which in values $100 < \beta < 500$ has shown great results in factorizing the latent space [25]. By setting $\beta = 1$ we return to the original loss function. This hyperparameter will be explored in the experiments.

Alternatively and as ground truth we implement the VAE loss from equation 4 for graphs without graph matching. we use binary cross entropy (BCE) as reconstruction loss of the adjacency, and categorical cross entropy (CE) for the attribute matrices. The regularization loss is similar to the above presented and also

includes the hyperparameter β. The equation for the ELBO with $\sigma()$ indicating Sigmoid activation is

$$\mathcal{L}(\phi, \theta : G) = BCE(A, \sigma(\tilde{A})) + CC(E, \tilde{E}) + CC(F, \tilde{F}) - D_{KL}\left(q_\phi\left(\mathbf{z} \mid G\right) \| p_\theta(\mathbf{z})\right) \tag{16}$$

We train the model on the negative ELBO. Further we use *Ranger* presented in 3.4 as optimizer combining three learning optimization methods. Out of the various optimization parameters, we achived good performance with the default values and only adjusted the learning rate and the number of lookahead steps. Missing a publication to cite, we refer to the source code *Ranger* [2]

4.4 Link prediction and Metrics

Our main experiment will be link prediction. It is intended as proof of concept rather than an attempt to set the state of the art. The results will let us draw conclusions on the impact and function of different parameters. Besides the final link prediction experiment, we will let the model perform link prediction on a randomly drawn small subset of the test set during training. This gives us a broader view on the models performance, which otherwise would only be evaluated by the ELBO loss.

Link prediction on multi-relational KGs is the task of predicting unobserved triples, based in the information acquired during training. To evaluate a model on this task, the most common method is entity ranking in the form of triple completion of unseen triples from the test set. Given a KG $G(\mathcal{E}, \mathcal{V})$ we want our model find the right entity out of $\mathcal{E}$ which completes the unseen triple $(s, r, ?)$ or $(?, r, o)$ for heads or tail prediction. Thus, the model scores the triple for all possible combination with the entities from $\mathcal{E}$. The rank of the true triple, in descending order, defines the performance of the model [26].

In the preprocessing step we created a dictionary with all occurring combinations for all possible triples with missing head or tail. These dictionaries we use to filter out real triples from the scoring. Unfiltered scores are referred to as *raw scores*. Per link prediction run, the model has to score the number of triples in the test set times the d_e the number of entities in $\mathcal{E}$ times two for head and tail, which mostly results in a number much larger than the size of the actual dataset.

Finally, the metrics for link prediction are the mean reciprocal rank (MRR) of the score for the true triple and the average HITS@k with $k \in [1, 3, 10]$. We denote $\mathcal{K}_{test}$ the unseen test set and $|\mathcal{K}_{\text{test}}|$ the number of triples in the test set. The operator rank() returns the in descending order the by the model scored position of the true triple. Head prediction of a triple given relation and object is denoted $(s|r, o)$ and likewise $(o|s, r)$ for tail prediction. The Iverson brackets $[\text{rank}(s \mid r, o) \leq k]$ return 1 if the scored rank is equal or lower than k, else o.

$$\begin{aligned} \text{MRR} &= \frac{1}{2\,|\mathcal{K}_{\text{test}}|} \sum_{(s,r,o)\in\mathcal{K}_{\text{test}}} \left(\frac{1}{\text{rank}(s \mid r, o)} + \frac{1}{\text{rank}(o \mid s, r)}\right) \\ \text{Hits@ } k &= \frac{1}{2\,|\mathcal{K}_{\text{test}}|} \sum_{(s,r,o)\in\|_{\text{lest}}} (\mathbb{1}[\text{rank}(s \mid r, o) \leq k] + \mathbb{1}[\text{rank}(o \mid s, r) \leq k]) \end{aligned} \tag{17}$$

4.5 Variational DistMult

In the context of link prediction, we implement control model for better interpretability of our results. From the wide range of embedding based models, DistMult reports both good results as well as an efficient architecture. Its bilinear property aligns with the permutation invariance of the RGVAE. additional to the original model we implement a variational version to isolate the effect of variational inference on multi-relational link prediction.

The DistMult encoder has a linear embedding layer for both sets of entities and relations. The embedded or latent representation is passed through the bilinear scoring function, which is equation 2. During training the model scores the triples in the training set among a number of corrupted triples. Loss function and optimizer are tunable hyperparameter. Table REF shows the optimal hyperparameter settings by Ruffellini at al. for the FB15k-237 dataset.

[2] `https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer`

TODO DistMult parameter table

The model implementation is adopted from Peter Bloem's work [3]. Additionally a variational module is implemented, which uses the embedding vector representation as mean and logvariance for a latent distribution from where the latent representation is sampled using the reparametrization trick. When this method is not selected, the stochastic coefficient ϵ is set to 1 rather than a random sample from the standard normal distribution. Lastly we implement the ELBO loss, which adds a regularization term, including β parameters to the BCE loss, thus we can only use the variational DistMult (VDistMult) in combination with BCE loss. The scoring function of the VDistMult stays identical to the original.

5 Experiments & Results

This section presents the experiments and results aimed at evaluating our proposed graph generative model. First we run a grid-search on the hyperparameter space to find the optimal configuration of the RGVAE. We use the ELBO and MRR as evaluation metric. The best configurations are used to perform link prediction. Here we compare the model performance with vs. without convolutions. We use the VDistMult as control model for link prediction. Finally we run two proof-of-concept experiments. The first generating triples and filter on a entity class constraining relation, thus we get an insight of how much percent of the generated triples are valid. Secondly we analyze the results of a RGVAE trained on subgraphs with $n = 10$. For the experiments we use two multi-relational KG datasets.

5.1 Data

For this sake of comparison with state of the art results, we chose the two most popular dataset used in this field of KG link prediction, FB15k237 and WN18rr.

Fb15k-237

Present the concept of FreeBase!

Entities have types, types have classes. Show most common types/classes

Highly incoherent, not following the same notation and inconsistent, contradicting triples.

Everybody was allowed to add facts. Maintainance of Freebase is discontinued.

textwidth for figures: 6.69423in

linewidth for figures: 6.69423in

Entity types

triple topics

WN18rr

Based on the wordnet KG.

Only 18 different relations

triples are synsets (synonym, hypernym)

TODO Table comparing numbers

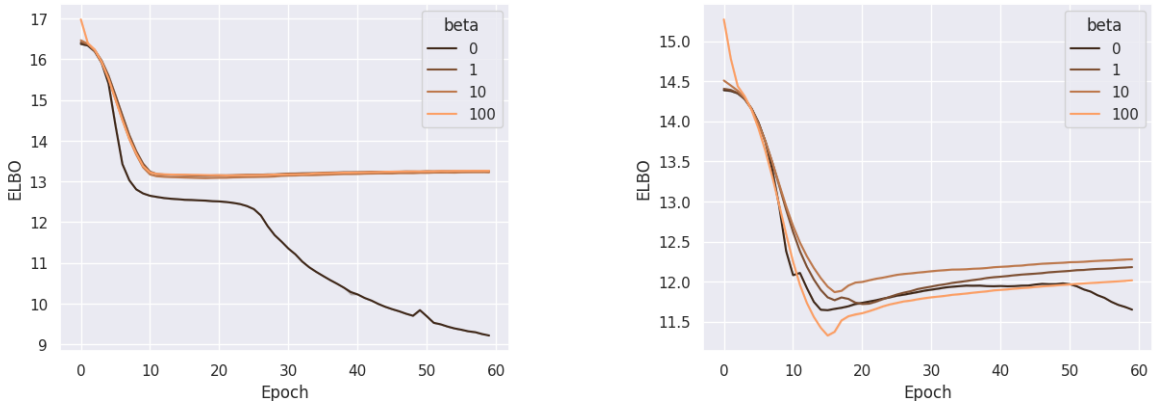
[3] `https://github.com/pbloem/embed`

5.2 Hyperparameter Tuning

In this section we run a grid search for the three hyperparameters β, d_z and d_h for a set of contrastive values. To reduce the computational expenses we train each model for 60 epochs and evaluate link prediction on a subset of 50 triples.

Empirically we set the learning rate to $3e-5$ and the maximum batchsize fitting on the GPU memory. For d_h we did not see any significant changes for higher values, thus we choose a lower number to reduce the total model parameters. The remaining hyperparameter did influence and the optimal setting vary for each dataset, table 1 shows the results of our hyperparameter tuning.

For the hyperparameter tuning of $\beta \in [0, 1, 10, 100]$ we chose significant values. With $\beta = 0$ we do not constrain our model on the Gaussian prior, thus the latent distribution can take the form of any distribution. This reduces the influence of the variational module and the model becomes closer to an autoencoder. For $\beta = 1$ we get our base model, and for $\beta \in [10, 100]$ the $\beta-$VAE version.



(a) FB15K-237 (b) WN18RR

Figure 6: Validation loss for RGVAE with $\beta \in [0, 1, 10, 100]$ trained on each dataset.

Figure 6 shows the validation ELBO for the different β values and for both datasets. We notice two interesting outcomes.

- For $\beta = 0$ converges further than the rest.
- The remaining values behave quase identical with $\beta = 100$ performing slightly better.

Since setting $\beta = 0$ would undermine our hypothesis of evaluating variational model, we chose $\beta = 100$ as default for the following experiments. This also compares with the β values proposed by Higgins in [25] to achieve a factorization of the latent space.

Especially on the FB15k-237 dataset the $\beta = 0$ configuration converges to a much lower ELBO. Thus, we have the trained models perform link-prediction on a 1% subset of the validation set. Figure 7 indicates an inverse correlation between the ELBO and the MRR score.

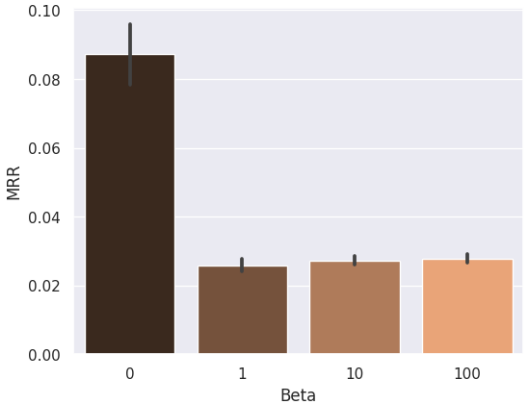


Figure 7: MRR scores for different β values on the dataset FB15k-237.

Barplot IN APPENDIX

Experiments on the impact of d_z on the ELBO show little improvement for $10 < d_z < 100$ and from $100 < d_z < 1000$ insignificant to no improvement. Thus, we chose $d_z = 100$ as default for our experiments.

Lastly, we evaluate the models hidden dimensions d_h and its influence on the ELBO and the (subset)MRR. We compare between $d_h \in [256, 512, 1024, 2048]$, while the lowest configuration performs slightly worse on the ELBO, there is no significant difference between the remaining three configurations. Considering the models parameter count we chose the $d_h = 512$ as default.

5.3 Link Prediction

We now get to the main experiment of this thesis. The results of this experiment will show if the RGVAE architecture is suitable for link prediction and in first place, if it is able to grasp the underlying KG semantics at least significantly better than random by differentiating between real and corrupted triples. First we evaluate the performance of the RGVAE on this experiment, comparing both encoder versions. Then we investigate the influence of the variational inference by comparing the variational and original versions of DistMult on link prediction.

5.3.1 RGVAE

At this point we bring in the convolutional variation of our model, which we will denote as RGCVAE. The experiments reveal if the convolutional architecture holds an advantage compared to the simple MLP baseline. Further a randomly initiated and untrained RGVAE is used as control model.

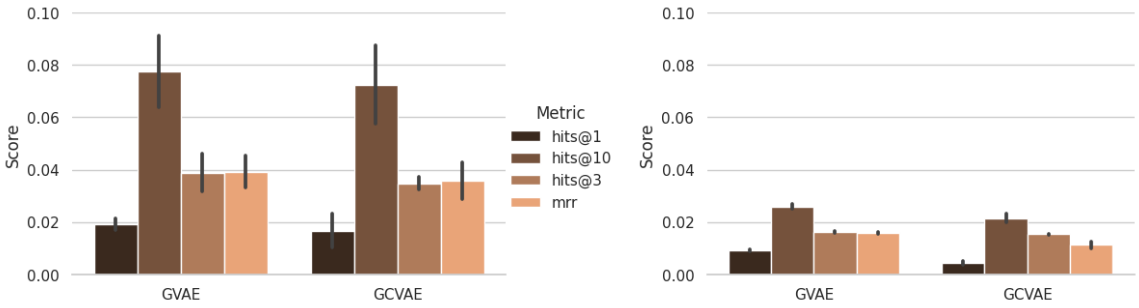
Due to its sparse graph computation, the RGVAE takes about 7 days to evaluate link prediction on the full test set and even 3 days when prediction tasks run parallel on a node of 4 Nvidia Titan RX 25GB GPUs. Since the exemplary link prediction during experimenting with different hyperparameter already gave us an idea of the mediocre performance of our model, we chose to spare computation time and power by running link prediction on a randomly drawn one-third of the complete test set. Each run is repeated three times using a different random seed.

The results are visualized in figure 8. We chose a visualization over a table, to emphasize our observations, and for the mentioned limitation of the test set, which makes these results not suitable for academically valid comparisons. Note that the figures are scales and to a range $[0, 10]$ while all metrics have a maximum of 1.

Comparing a MRR of 0.08 to the DistMult score or 0.4 our model does not perform competitively on link prediction tasks.

Graph convolutions do not yield an advantage over the basemodel. In fact, the RGVAE with convolutions even scores slightly worse.

The model scores about three times better on the FB15k-237 dataset than on WN18RR. FB15k-237 is a richer dataset with more triples and and a more balanced ratio of entities to relations. WN18RR operates on only 18 relations, what makes the relation most crucial when completing a triple. The architecture of the RGVAE puts twice the emphasis entities, described by the adjacency and the node feature matrices, while the relation is only represented by the overly sparse edge attribute matrix. Thus, we could conclude that our model learns to predict based on the hidden types and topics of the entities. All possible conclusions for this are discussed in 6.5. Relevant for this section is solely, that we chose the FB15k-237 dataset to investigate further, how well the RGVAE's grasps the underlying entity types and triple topics.



(a) FB15K-237

(b) WN18RR

Figure 8: Link prediction results in MRR, Hits@1, Hits@3 and Hits@10 for RGVAE and RGCVAE trained on the full trainset of each dataset.

TODO add random!!!

5.3.2 Impact of Variational Inference and Gaussian prior

In order to explain the poor performance of the RGVAE on the task of link prediction, we investigate the impact of the variational inference. Since the RGVAE with relaxed latent space, meaning less variance, indicated higher scores than the version with Gaussian prior, we examine the two variants by means of embedding models. The original DistMult model with optimized parameter serves as control model, while we compare it to the VDistmult, described in section 4.5, learning the full ELBO versus learning only on the reconstruction loss. By not including the regularization term in the loss the model is no longer bound to the Gaussian prior, which results in a relaxation of the latent space.

We train the three models for 300 epochs solely on the FB15k-237 dataset and evaluate MRR, Hits@1, Hits@3 and Hits@10. Table 3 shows the mean scores with $\mu \pm \sigma$ of three runs per model. Note that the exponent on σ holds for the whole term. We can clearly see that both variational versions of the DistMult perform significantly worse than the original model. Learning on the full elbo or only the reconstruction loss does not seem to influence the scores in this setting. This indicates, that the models performance on link prediction suffers from using variational inference.

In the last row we show the results of the RGVAE with relaxed latent space. This model was trained with $\beta = 0$ thus not constraining the latent space on a Gaussian prior. The model outperforms the versions with hyperparameter choice $\beta > 0$ and scores the closest to the DistMult model. The impact of β shows in the regularization, which we tracked separately. The maximum values ofD_{KL} during the experiment are

$$\begin{aligned} D_{reg} &= \beta \times D_{KL}\left(q_\phi\left(\mathbf{z} \mid G\right) \| p_\theta(\mathbf{z})\right) \\ \max_{\beta=0} D_{KL} &= 3506 \\ \max_{\beta=100} D_{KL} &= 0.0154 \end{aligned} \tag{18}$$

While the results of the relaxed RGVAE might seem promising, Distmult is a much simpler and faster link predictor, thus we do not see a justification to keep researching on the RGVAE for this task. Note that due to the high computation cost of the RGVAE we only run the experiment once on the full dataset.

Figure 9: (a) Validation loss RGVAE with vs. w/o permutation invariant loss function. (b) Percentage of permuted nodes during training.

Model	MRR	Hits@1	Hits@3	Hits@3
DistMult	0.2854 ± 0.0025	0.2 ± 0.001	0.3149 ± 0.0038	0.4512 ± 0.0053
VDistMult	$0.517 \pm 0.0197e^{-3}$	$0.2442 \pm 0.1994e^{-4}$	$0.8145 \pm 0.3049e^{-4}$	$0.399 \pm 0.0576e^{-3}$
VDistMult w/ ELBO	$0.6397 \pm 0.0357e^{-3}$	$0.57 \pm 0.3046e^{-4}$	$0.1547 \pm 0.1023e^{-3}$	$0.6351 \pm 0.1992e^{-4}$
RGVAE w/o ELBO	0.1412	0.0981	0.1494	0.2275

Table 3: Link prediction scores of DistMult and RGVAE versions on the FB15k-237 dataset.

5.4 Impact of permutation

Furthermore we examine the influence of the permutation invariant loss function described in 4.3.2. During training and subset link prediction no significant difference was observed between the RGVAE with versus without matching target and prediction graph. Yet, two observations draw our attention, namely:

- The amount of nodes permuted per batch converges during training from 100% to exactly 60
- The RGVAE with permutation invariant loss function learns to predict many variations of adjacency matrix while the standard model predicts similar to the target.

The first point indicates that the model learns a set of adjacency matrices, which can be permuted to match the target while optimizations the loss. Note that the generated matrix representation of the triples either has only one edge on the right upper index $A_{0,n}$ or, in the rare case of self-loops in $A_{0,0}$. The number of nodes per graph for these observations is set to $n = 2$. Curiosity remains why the model converges to steadily permute $\frac{3}{5}$ of the prediction. Secondly, we see that even when converged, the model predicts variations od the adjacency matrix very different to the target. The most common is a single edge on $A_{n,0}$ and on $A_{n,n}$. Less common and with lack of explanation are the predictions of an empty, or multi edge adjacency matrix. In contrast to this and as expected, the RGVAE with the standard loss function learns to solely predict edges on $A_{0.0}$. Finally we will analyze the impact of permutation invariance on the experiment of syntax coherence in section 5.6.

5.5 Interpolate Latent Space

Inspired by the popular results of Higgins, who featurized each latent dimension on a facial features of the FACES dataset [27]. The VAE generates faces controlling feature such as age, gender and emotions by manipulating single latent dimensions [25]. We run this experiment with the RGVAE on the FB15k-237 dataset, using two different interpolation methods. The latent dimension for this experiment is set to $d_z = 10$ in order to analyze each dimension separately and the interpolation is linear with a step count of 10.

The first experiment is linear interpolating between two triples. Therefor two valid triples from the train set are encoded into their latent representation. We chose the two semantically related triples triples to analyze if the linear movement in latent space correlates with an obvious semantic feature.

```
[['/m/02mjmr Barack Obama'], ['/people/person/place_of_birth'], ['/m/02hrh0_ Honolulu']]
[['/m/058w5 Michelangelo'], ['/people/deceased_person/place_of_death'], ['/m/06c62 Rome']]
```

TODO present the results and link to appendix

Obama triple is not reproduced, not even close.

For the second experiment we modify each latent dimension isolated in a 95% confidence interval of the Standard Gaussian distribution. Starting with the encoded representation of the Obama triple, we incrementally add $z_i += j \times$ ... TODO you have to fix this!!!

Can the model assign logical features to latent dimensions?

5.6 Syntax coherence

On closed-world schema-based KGs the approach for testing the validity of a new triple is to add it to the existing KG and run a onthology reasoner on it. A inconsistency in the KG will appear as `null`-Class, but only if an axiom is violated. This approach works only for fully constrained KGs and is not scalable, since the reasoner recursively checks evey triple for every axiom. Thus, we present an alternative and improvised way to estimate the validity of generated triples.

The FB15k-237 is a subset from the FreeBase KG [**FREEBASE**], thus, even thought they are not part of the dataset, - entities have type properties in their original Freebase representation. Querying the last official Freebase dump, we get the types for each entity in the FB15k-237 dataset, With exception of 8 entities, which could not be found in the query.

Our approach is to randomly generate triples, from signals randomly drawn from a Standard Gaussian distribution. Then to filter those triples on predicates which contain the type *people*, which is within the top 10 most common Freebase types. We differentiate between base-class types subclass types, both can contain the word *people*. The entity `['/m/02mjmr Barack Obama']` has between many others the type `[/people/measured_person]`. Here the base class is *people* and the subclass *measured_person*. We could filter directly on subclasses, but we chose to give our model more creative freedom and filter for *people* in the full set of types. Using this choice, the generated triples are scored on logic rather than facts. E.g. any person can hypothetically the a *measured_person*, understanding this implies semantical reasoning, while differentiating between which person is and is not a *measured_person* implies contextual knowledge. Thus, we use the base type *people* to validate triples. Furthermore the relations are inconsistent in their notation, partly not only having a head type constrain. Thus, we check only the head entity for the key type.

- From 14541 entities, 5283 contain the keyword people, or 36.332%.
- From 237 predicates, 25 contain the keyword people, or 10.549%.
- From 310116 triples, 47354 contain the a predicate keyword people, or 15.269%.

Considering these facts, we calculate the marginal probability of guessing a head entity s of type *people*, given a triple which contains the type *people*. Without prior knowledge $p(s_p)$ and $p(r_p)$ are conditionally independent. The probability is calculated as:

$$p(s_p \mid r_p) = p(s_p) = 0.3633 \tag{19}$$

For this experiment we generate triples until $10e^5$ contain the key type. Those filtered triples are validated on the type of the head entity and compared to the full dataset for novelty. Here we again compare the performance between the RGVAE with the two different encoder architectures. Furthermore the models are trained both with regular and permutation invariant loss function. Lastly, the experiments are repeated for sampling the latent signal from $\mathcal{N}(0, 1)$ and for sampling from $\mathcal{N}(0, 2)$. We average the accuracy of three runs of generating a valid triple and of this triple being unseen in the dataset. The results for valid triples are shown in figure 10. The dotted horizontal line indicates the random probability of generating a valid triple, calculated in equation 19 and σ_1^2 and σ_2^2 denote the variance for the different latent space distributions.

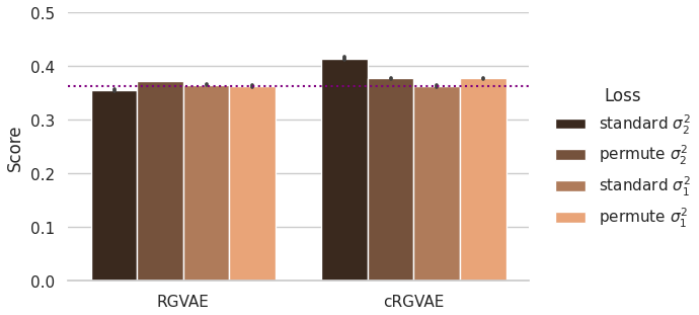


Figure 10: Accuracy of generating valid triples.

To our disappointment the model does not perform significantly better than random. Neither the choice loss function nor the doubled variance show a correlation with the accuracy. The only configuration standing out is the RGVAE with convolutional encoder, standard loss function and $\sigma^2 = 2$, scoring 4% higher than random. From all valid generated triples $100 \pm 0.001\%$ are new and unseen in the dataset. Coming back to `['/m/02mjmr Barack Obama']`, we filter the unseen triples fo the first three appearances of this entity. These are displayed in table 4 for every variation of the RGVAE and in the same order as in figure 10

RGVAE STANDARD σ_2^2
[[Barack Obama] [/people/person/places_lived./people/place_lived/location] [Casablanca]] [[Barack Obama] [/people/person/place_of_birth] [Sarah Silverman]] [[Barack Obama] [/people/person/place_of_birth] [The League of Extraordinary Gentlemen]]
RGVAE PERMUTED σ_2^2
[[Barack Obama] [/people/ethnicity/geographic_distribution] [End of Watch]] [[Barack Obama] [/people/profession/specialization_of] [Montgomery County]] [[Barack Obama] [/people/cause_of_death/people] [WWE Superstars]]
RGVAE STANDARD σ_1^2
[[Barack Obama] [/people/person/place_of_birth] [Academy Award for Best Sound Editing]] [[Barack Obama] [/people/person/places_lived./people/place_lived/location] [Stan Lee]] [[Barack Obama] [/people/person/place_of_birth] [Multiple sclerosis]]
RGVAE PERMUTED σ_1^2
[[James Brolin] [/people/person/places_lived./people/place_lived/location] [Barack Obama]] [[Barack Obama] [/people/person/spouse_s./people/marriage/location] [D.C. United]] [[Jim Sheridan] [/people/person/religion] [Barack Obama]]
cRGVAE STANDARD σ_2^2
None
cRGVAE PERMUTED σ_2^2
[[Pinto Colvig] [/people/deceased_person/place_of_burial] [Barack Obama]] [[Helena Bonham Carter] [/people/person/gender] [Barack Obama]] [[John Buscema] [/people/person/spouse_s./people/marriage/spouse] [Barack Obama]]
cRGVAE STANDARD σ_1^2
[[Suhasini Ratnam] [/people/person/sibling_s./people/sibling_relationship] [Barack Obama]] [[Barack Obama] [/people/person/gender] [Deva]] [[Barack Obama] [/people/person/sibling_s./people/sibling_relationship] [Niagara Falls]]
cRGVAE PERMUTED σ_1^2
[[Jonathan Rhys Meyers] [/people/person/nationality] [Barack Obama]] [[Barack Obama] [/people/person/sibling_s./people/sibling_relationship] [Motherwell F.C.]] [[Pinto Colvig] [/people/deceased_person/place_of_burial] [Barack Obama]]

Table 4: Generated and unseen knowledge.

The generated triples confirm the accuracy results. With exception of the triple

`[[Barack Obama] [/people/person/places_lived./people/place_lived/location] [Casablanca]]`

all remaining triples violate common sense logic. The RGVAE does not differentiate between the types gender, location, movie, person or medicine. It even goes so far to state that Obama was born in `[Multiple sclerosis]`. While this might sound funny it also clearly indicates, that our model did not learn the underlying semantics of this real world KG.

While investigating the model and the generated triple set, we notice two outcomes. Neither the augmentation of the variance nor the enabling of the permutation invariance has an impact on either of the model with two encoder versions. The regularization loss converges during training to zero, meaning that the model learns a latent representation of the dataset as nearly perfect Standard Gaussian distribution. Yet, even when sampling latent signal from the exact same distribution, we notice that each model repeatedly predicts combinations of a small subset of entities and relations, e.g. the RGVAE version which did not predict the Obama entity once in a total of 111583 valid triples. This also aligns with the interpolation results, where we observed a static relation for the full gridsearch of the latent space. If we look at the gradient and parameter values ϕ andθ of the MLP encoder and decoder, we see a much higher variance and gradients for ϕ. The decoder shows higher values and variance for a small subset of neighboring parameter, while the remaining parameters converge to a very similar and low value. This indicates that the encoder learns very well to represent each different triple as Standard Gaussian latent representation. The decoder MLP on the opposite seems to to ignore most of this representation by assigning vanishing values to the connected parameters. Intuitively it seems that the decoder learns to interpret the part of the latent representation corresponding to the adjacency matrix and minimizes as well as stabilizes the loss of the edge and node attribute matrix by uniformly distributing their probability. This leads to the decoder reconstructing edge and node attribute randomly. Further, depending on the values of θ when the model finishes learning, the decoder will keep predicting the same subset of entities and relation independent of the latent signal z. If we look at the flattened representation of our input graph, we see that the part representing the adjacency matrix is way shorter and has a tractable mean of $\frac{1}{4}$ while the mean for the edge and node attribute matrices are $\frac{1}{4\times1345}$ and $\frac{1}{14951}$. The problem of a our decoder partly ignoring the latent input and potential solutions are discussed further in section 6.3.

6 Discussion & Future Work

TODO compare to molecules - point out differences

In this final chapter, we will discuss our results from different experiments, emphasizing the comparison to Simonovsky's work on molecule generation. The fact, that we did not achieve Similarly successful results is obvious, thus we will focus on the analyzing the responsible factors and their cause. Despite the nature of our results we are happy to propose possible solutions. Being of the opinion that questions are often more interesting than answers, we will question the value of these solutions for further research.

6.1 RGVAE Link Predicton Interpolation

Poor results when using ELBO loss.

Variational Inference does not add value to embedding based link predictors.

Relaxation of the latent space allows the RGVAE to score competitively, yet the vast difference in complexity compared to simple embedding based models devalues its useability.

6.2 Research Answer

Higher dimension of freedom when generating graphs using Graph matching.

All valid generated triples are unseen in both train and test set.

Problem seems to be the Gaussian prior.

Based on the results we can confidently answer our research question with

No!

Yet, metaphorically, this is not the end of the book, but rather the beginning of a new chapter. In the remaining part of this discussion we analyze the underlying problem of the RGVAE and propose a variety of solutions.

6.3 Decoder Collapse

To introduce main part of our conclusion we explain a phenomenon observed when training GANs. A GAN is yet another generative model which based on its potential to generate high resolution images, has drawn much attention in recent years. It consists of a generator, who's task it is to decode a latent signal to a an image, and a discriminator, which given the fake image and the real data has to distinguish between them. Training of GANs is unstable and one of the major problem which occur is *mode collape*. The generator learns to fool the discriminator by generating only a single mode with high precision such that the discriminator classifies it was real image.

While VAEs cannot suffer from mode collapse, because they backpropagate over the predicted distributions of all modes, it has a related phenomenon. *Decoder collapse* occurs when the decoder has enough capacity to choose not to consider the latent signal and instead stores the information to reconstruct the data's distribution partly or solely in its parameters. This means that the generated data is not conditioned on the latent signal anymore. The cause of this problem is found in the regularization term $D_{KL}\left(q_\phi\left(\mathbf{z} \mid \mathbf{x}\right) \| p_\theta(\mathbf{z})\right)$, more specifically in the constrain on a standard Gaussian prior. The multidimensional encoding of $d_z > 1$ the approximated posterior is a mixture of Gaussians, which can only match the multivariant Normal distribution, in the case of all $\mu_z = 0$ and $\Sigma_z^2 = \mathbb{I}$, what also implies that no information is encoded. Thus, the VAE has to decide if storing information in the latent vector is necessary to model the dataset distribution $p(x)$. If the penalty for altering the latent Normal distribution outweighs the benefits of the additional information towards the reconstruction loss, the VAE will chose for *decoder collapse*.

Exactly this phenomenon can be observed in the triple interpolation and generation experiments. The RGVAE converges rapidly during training, minimizing the reconstruction loss close to zero. This falsely indicates that the model has learned to reproduce the data. Yet, when sampling using only the decoder, the generated triples repeat combinations of a subset of entities and relations which are more frequent in the dataset. During interpolation of the latent space, the model predicts steadily the same relation. Subject and object seem rather randomly drawn, showing little impact while traversing each single latent dimension. The adjacent matrix ¿¿¿¡¡¡

The last experiment confirms the obvious. The syntax adherence of the different model variations does not differ from random sampling. Further we again notice the predominance of the more frequent data.

Projecting this finding on the link prediction experiment, we question, why the model scored better than random. Possible reasons are for once, that the encoder learned to encode the dataset close to a Normal distribution, but when encountering unseen triples, the mean and variance of the encoding vary. A second reason is, the every triple in the testset is corrupted in all possible combinations, including the less frequent entities. The collapsed decoder subsequently keeps generating frequent triples, thus the reconstruction loss between a less frequent combination of entities and the collapsed prediction will be higher than for a frequent combination. Therefore the model learns score the real triple higher but for the wrong reason. The RGVAE with $\beta = 0$ and therefore unconstrained on the Standard Gaussian prior shows the best results between the different model settings. We can assume that the model learns a latent representation for subject and object. Despite this improvement, is it probable that the decoder still collapses on the reconstruction of the relation index, which explains the remaining scoring gap to the much simpler DistMult model.

6.4 VAE surgery

Similar problems in different fields have been encountered for generative VAE applications. While the author of this thesis whishes to have drawn these parallels earlier, all the proposed solutions imply a significant modification of the VanillaVAE, thus would not align with this work's research question. We present three approaches from different literature which tackle the VAE mode collapse in the filed of image and voice generation.

In a publication of Adobe Research and Google Brain, Hoffman and Johnson propose an elegant modifica-

tion of the variational evidence lower bound [28]. More specifically they change the VanillaVAE's regularization term, where instead of imposing a Standard Gaussian prior on the full latent distribution, the prior is imposed on the mixture of all single latent dimension, giving space for more expressive probability distributions such as truncated Gaussians.

Further, a index-code mutual information term is added, intended to maximize the mutual information between every index of the observation and z. While the reconstruction term enforces to encode every feature of x in a corresponding latent dimension, the information term opposes this by maximizing the mutual information between all x_i and z_i. The information term compares compact to the reconstruction loss, yet it is enough to prevent decoder collapse.

SHOULD WE WIRTE THE NEW LOSS OUT?

Kingma et al. turn towards a autoregressive solution in their publication [29]. Their VAE model generates images recursive per pixel, each conditioned on previous point, using both RNN and RCN as decoder. Their solution to the decoder collapse is a normalizing flow, which predicts the encoder posterior $q_\phi(z \mid x)$. Besides not suffering from decoder collapse, this model can be set to discard irrelevant information in the data. As downside, the autoregressive nature causes slow image generation.

The last approach we present closes the circle to the introduction of this section. The paper *Wasserstein Auto-Encoders* [30] propose a combination between GAN and VAE. A model wich uses the VAE's encoder-decoder architecture but instead of the normal regularization term, the posterior distribution is learned and penalized by its Wasserstein distance tp the data distribution. This is in fact a generalization of the adversarial loss of generator and discriminator. Since our task of generating triples would benefit from a precise prediction, such as GANs achieve on image generation, we question, if employing adversarial loss also on the reconstruction term would yield better generation results in this field than the exact index-wise loss?

Besides these three, numerous other approaches have been proposed. Worth mentioning because of their originality are the idea of adding skip-connections between latent space and the decoder's hidden layer [31] as well as regularizing the amortized inference [32].

6.5 Future Work

There are many avenues to follow for future work. Obviously the first step is to fix the collapsing decoder. The suggested solutions should be compared before making a choice, since none has yet been shown to work on KG VAEs.

Once the reconstruction of triples proved successful, we should look at the data, and question the representation. The adjacency matrix can be inferred from the edge attribute matrix and thus, is obsolete. Since it does not contain additional information, does the model perform worse without it? Approaching this thought from a different angle, we could leave the adjacency as it is and represent the node and edge indices as continuous number, as it is done for color scales of images. This would greatly reduce the number of parameters, but certainly also imply new challenges. Important in this context is that the adjacency is represented in sparse format since this the experiments on triples is but a proof of concept for larger subgraphs.

We noticed the model worst flexibility for predicting the relation index. In the interpolation experiment it kept predicting the very same relation for the entire latent space. Also linked is the variance in number of edges, while the number of nodes is constant. Thus, we wonder if a setup of two VAEs, where the first one predicts the adjacency and node attributes and the second recurrent one, while conditioned on the predicted edges of the first VAE, predicts the edge attribute.

Finally and making use of the batch wise implementation of the max-pooling graph matching alorithm presented in this work, we suggest to explore a more efficient and *intelligent* graph matching approach. Would it be possible drasically reduce complexity by training a neural network to predict the optimal permutation matrix based on target and prediction graph?

We hope to have given enough food for though for everyone willing to continue research in this field and look with excitement towards future findings.

Happy New Year!

References

1. Silver, D. *et al.* Mastering the game of Go without human knowledge. en. *Nature* **550.** Number: 7676 Publisher: Nature Publishing Group, 354–359. ISSN: 1476-4687. https://www.nature.com/articles/nature24270 (2021) (Oct. 2017).
2. *Elon Musk on Twitter* en. https://twitter.com/elonmusk/status/896166762361704450 (2021).
3. 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. 2020).
4. 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. 2018).
5. Kipf, T. N. & Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. en. *arXiv:1609.02907 [cs, stat].* arXiv: 1609.02907. http://arxiv.org/abs/1609.02907 (2020) (Feb. 2017).
6. Schlichtkrull, M. *et al.* en. in *The Semantic Web* (eds Gangemi, A. *et al.*) Series Title: Lecture Notes in Computer Science, 593–607 (Springer International Publishing, Cham, 2018). ISBN: 978-3-319-93416-7 978-3-319-93417-4. http://link.springer.com/10.1007/978-3-319-93417-4_38 (2020).
7. Kipf, T. N. & Welling, M. Variational Graph Auto-Encoders. *arXiv:1611.07308 [cs, stat].* arXiv: 1611.07308. http://arxiv.org/abs/1611.07308 (2020) (Nov. 2016).
8. 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. 2019).
9. 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).
10. Nickel, M., Tresp, V. & Kriegel, H.-P. A Three-Way Model for Collective Learning on Multi-Relational Data. en, 8.
11. Yang, B., Yih, W.-t., He, X., Gao, J. & Deng, L. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. *arXiv:1412.6575 [cs].* arXiv: 1412.6575. http://arxiv.org/abs/1412.6575 (2020) (Aug. 2015).
12. Bishop, C. M. *Pattern recognition and machine learning* en. ISBN: 9781493938438 9780387310732 Publisher: Springer. 2006. https://cds.cern.ch/record/998831 (2020).
13. Paulheim, H. Knowledge graph refinement: A survey of approaches and evaluation methods. en. *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. 2016).
14. Nickel, M., Murphy, K., Tresp, V. & Gabrilovich, E. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE* **104.** arXiv: 1503.00759, 11–33. ISSN: 0018-9219, 1558-2256. http://arxiv.org/abs/1503.00759 (2020) (Jan. 2016).
15. 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 2014).
16. Ha, D. & Schmidhuber, J. World Models. *arXiv:1803.10122 [cs, stat].* arXiv: 1803.10122. http://arxiv.org/abs/1803.10122 (2020) (Mar. 2018).
17. 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* en. in *2014 IEEE Conference on Computer Vision and Pattern Recognition* (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).
18. Date, K. & Nagi, R. GPU-accelerated Hungarian algorithms for the Linear Assignment Problem. en. *Parallel Computing* **57,** 52–72. ISSN: 0167-8191. http://www.sciencedirect.com/science/article/pii/S016781911630045X (2020) (Sept. 2016).
19. 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. 2020).
20. Zhang, M. R., Lucas, J., Hinton, G. & Ba, J. Lookahead Optimizer: k steps forward, 1 step back. *arXiv:1907.08610 [cs, stat].* arXiv: 1907.08610 version: 1. http://arxiv.org/abs/1907.08610 (2020) (July 2019).

21. 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. 2020).

22. Paszke, A. *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv:1912.01703 [cs, stat]*. arXiv: 1912.01703. http://arxiv.org/abs/1912.01703 (2021) (Dec. 2019).

23. Biewald, L. *Experiment Tracking with Weights and Biases* Software available from wandb.com. 2020. https://www.wandb.com/.

24. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17,** 261–272 (2020).

25. Higgins, I. *et al.* beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. en. https://openreview.net/forum?id=Sy2fzU9gl (2021) (Nov. 2016).

26. Ruffinelli, D., Broscheit, S. & Gemulla, R. *You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings* in (Sept. 2019). https://openreview.net/forum?id=BkxSmlBFvr (2020).

27. Ebner, N. C., Riediger, M. & Lindenberger, U. FACES—A database of facial expressions in young, middle-aged, and older women and men: Development and validation. en. *Behavior Research Methods* **42,** 351–362. ISSN: 1554-351X, 1554-3528. http://link.springer.com/10.3758/BRM.42.1.351 (2021) (Feb. 2010).

28. Hoffman, M. D. & Johnson, M. J. *Elbo surgery: yet another way to carve up the variational evidence lower bound* in *Workshop in Advances in Approximate Bayesian Inference, NIPS* **1** (2016), 2.

29. Chen, X. *et al.* Variational Lossy Autoencoder. *arXiv:1611.02731 [cs, stat]*. arXiv: 1611.02731. http://arxiv.org/abs/1611.02731 (2021) (Mar. 2017).

30. Tolstikhin, I., Bousquet, O., Gelly, S. & Schoelkopf, B. Wasserstein Auto-Encoders. *arXiv:1711.01558 [cs, stat]*. arXiv: 1711.01558. http://arxiv.org/abs/1711.01558 (2021) (Dec. 2019).

31. Dieng, A. B., Kim, Y., Rush, A. M. & Blei, D. M. Avoiding Latent Variable Collapse With Generative Skip Models. *arXiv:1807.04863 [cs, stat]*. arXiv: 1807.04863. http://arxiv.org/abs/1807.04863 (2021) (Jan. 2019).

32. Shu, R., Bui, H. H., Zhao, S., Kochenderfer, M. J. & Ermon, S. Amortized Inference Regularization. *arXiv:1805.08913 [cs, stat]*. arXiv: 1805.08913. http://arxiv.org/abs/1805.08913 (2021) (Jan. 2019).